Ready walkthrough

Index

List of pictures

Disclaimer

I do this box to learn things and challenge myself. I'm not a kind of penetration tester guru who always knows where to look for the right answer. Use it as a guide or support. Remember that it is always better to try it by yourself. All data and information provided on my walkthrough are for informational and educational purpose only. The tutorial and demo provided here is only for those who are willing and curious to know and learn about Ethical Hacking, Security and Penetration Testing.

Just to say: I am not an English native person, so sorry if I did some grammatical and syntax mistakes.

Reconnaissance

The results of an initial nMap scan are the following:

```
┌──(k14d1u5㉿kali)-[~/…/Linux/Medium/Ready/nMap]
└─$ nmap -sT -sV -p- -A 10.10.10.220 -oA Ready
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-05-05 04:35 PDT
Nmap scan report for 10.10.10.220
Host is up (0.035s latency).
Not shown: 65533 closed tcp ports (conn-refused)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 48:ad:d5:b8:3a:9f:bc:be:f7:e8:20:1e:f6:bf:de:ae (RSA)
|   256 b7:89:6c:0b:20:ed:49:b2:c1:86:7c:29:92:74:1c:1f (ECDSA)
|_  256 18:cd:9d:08:a6:21:a8:b8:b6:f7:9f:8d:40:51:54:fb (ED25519)
5080/tcp open  http    nginx
| http-title: Sign in \xC2\xB7 GitLab
|_Requested resource was http://10.10.10.220:5080/users/sign_in
|_http-trane-info: Problem with XML parsing of /evox/about
| http-robots.txt: 53 disallowed entries (15 shown)
| / /autocomplete/users /search /api /admin /profile
| /dashboard /projects/new /groups/new /groups/*/edit /users /help
|_/s/ /snippets/new /snippets/*/edit
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 28.77 seconds
```

Figure 1 - nMap scan results

Open ports are 22 and 5080. Therefore, SSH service (22) was enabled. Also, a web application was deployed on port 5080. Lastly, nMap provided Linux as operative system, probably Ubuntu, but any other information about it.

Initial foothold

The only port I was able to analyze on this box was 5080. Therefore, I analyzed the web application running FFUF tool. In this way, I found some paths. Some of them was relative to some user registered on the GitLab application. In addition, I was able to register a new user and I did it. After logged in with the user I just created, I was able to found the GitLab version, as shown in the following:

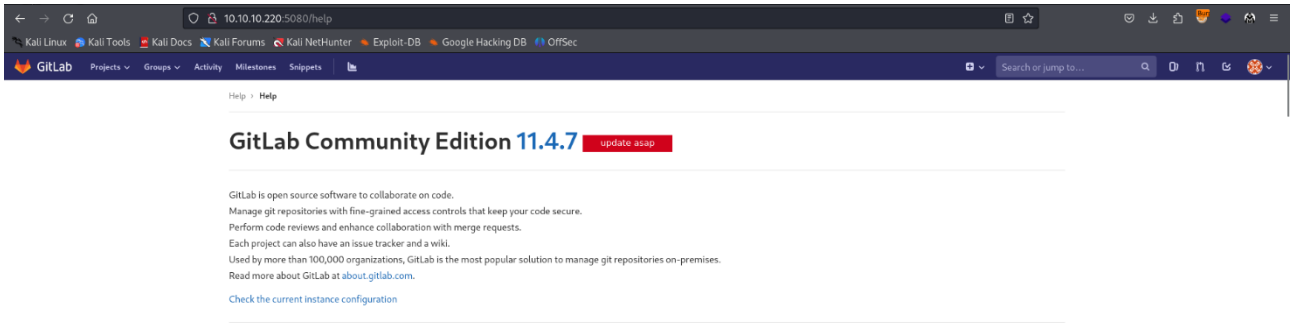

Figure 2 - GitLab version

User flag

At this point, I looked on the Internet if some exploits were available against the GitLab version and I found an interesting one. Since I have all information I needed, I just run the exploit and, luckily, I obtained the user shell, as shown in the following picture:

Figure 3 - User shell

Using this shell, I was already able to retrieve the user flag:

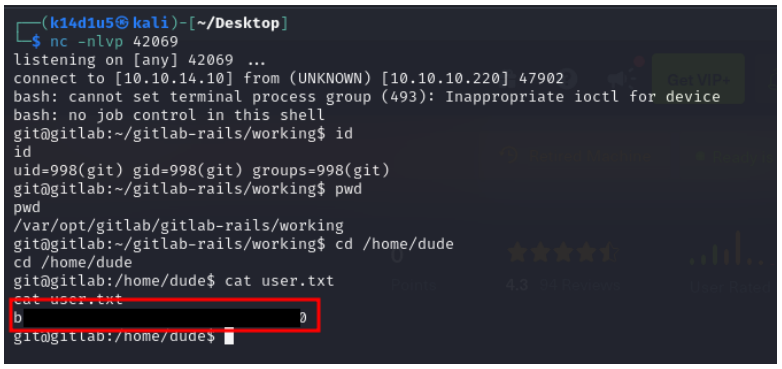

Figure 4 - User flag

Privilege escalation

This was the moment to escalate my privileges. To achieve this goal, I looked for some interesting information on the file system. First of all, I found a file named $root\_pass$. It contained something that looks like a password. I tried to use it running the su command (I needed to upgrade my shell with a python one), but it didn't work. Therefore, I kept to look for other information. This time, I found an interesting database file named $gitlab.rb$. In this file I found another password:

Figure 5 - gitlab.rb file

```
git@gitlab:~$ cat /opt/backup/gitlab.rb | grep pass
cat /opt/backup/gitlab.rb | grep pass
#### Email account password
# gitlab_rails['incoming_email_password'] = "[REDACTED]"
#     password: '_the_password_of_the_bind_user'
#     password: '_the_password_of_the_bind_user'
#   '/users/password',
#### Change the initial default admin password and shared runner registration tokens.
# gitlab_rails['initial_root_password'] = "password"
# gitlab_rails['db_password'] = nil
# gitlab_rails['redis_password'] = nil
gitlab_rails['smtp_password'] = "w[redacted]h"
# gitlab_shell['http_settings'] = { user: 'username', password: 'password', ca_file: '/etc/ssl/cert.pem', ca_path: '
/etc/pki/tls/certs', self_signed_cert: false}
##! `SQL_USER_PASSWORD_HASH` can be generated using the command `gitlab-ctl pg-password-md5 gitlab`
# postgresql['sql_user_password'] = 'SQL_USER_PASSWORD_HASH'
# postgresql['sql_replication_password'] = "md5 hash of postgresql password" # You can generate with `gitlab-ctl pg-
password-md5 <dbuser>`
# redis['password'] = 'redis-password-goes-here'
####! **Master password should have the same value defined in
####!   redis['password'] to enable the instance to transition to/from
# redis['master_password'] = 'redis-password-goes-here'
# geo_secondary['db_password'] = nil
# geo_postgresql['pgbouncer_user_password'] = nil
#     password: PASSWORD
###! generate this with `echo -n '$password + $username' | md5sum`
# pgbouncer['auth_query'] = 'SELECT username, password FROM public.pg_shadow_lookup($1)'
#     password: MD5_PASSWORD_HASH
# postgresql['pgbouncer_user_password'] = nil
```

Figure 6 - Password found in gitlab.rb file

I was lucky this time because I became root using this password running the *su* command:

```
git@gitlab:/$ su root
su root
Password: Y[redacted]w

su: Authentication failure
git@gitlab:/$ su root
su root
Password: Y[redacted]w

su: Authentication failure
git@gitlab:/$ su root
su root
Password: w[redacted]h

root@gitlab:/# cd /root
cd /root
root@gitlab:~# cat root.txt
cat root.txt
cat: root.txt: No such file or directory
root@gitlab:~# ls -la
ls -la
total 20
drwx------ 1 root root 4096 Apr  5  2022 .
drwxr-xr-x 1 root root 4096 Apr  5  2022 ..
-rw------- 1 root root   57 Apr  5  2022 .bash_history
-rw-r--r-- 1 root root 3106 Oct 22  2015 .bashrc
-rw-r--r-- 1 root root  148 Aug 17  2015 .profile
root@gitlab:~# id
id
uid=0(root) gid=0(root) groups=0(root)
root@gitlab:~#
```

Figure 7 – Container root user

At this point I looked for the root flag, but it wasn't there and it was very strange. Honestly, I thought for a while that creator forgot to insert the root flag. Even if *id* command told me I was root, there were something strange. For example, the root flag didn't exist and *sudo* command didn't work. Therefore, I investigated more and I found other interesting signs. In particular, the process with PID 1 wasn't the *init* process and the *init* process didn't exist on the box, as shown in the following pictures:

```
root@gitlab:/proc/1# ps -auxww
ps -auxww
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0  18044  2860 ?        Ss   07:27   0:00 /bin/bash /assets/wrapper
root          24  0.0  0.0   4388  1152 ?        S    07:27   0:00 runsvdir -P /opt/gitlab/service log: ............
.....
.....
.....
.....
root          28  0.0  0.0   4236   652 ?        Ss   07:27   0:00 runsv sshd
root          29  0.0  0.0   4380   704 ?        S    07:27   0:00 svlogd -tt /var/log/gitlab/sshd
root          30  0.0  0.1  65504  5592 ?        S    07:27   0:00 /usr/sbin/sshd -D -f /assets/sshd_config -e
root         433  0.0  0.0   4236   644 ?        Ss   07:27   0:00 runsv redis
root         434  0.0  0.0   4380   708 ?        S    07:27   0:00 svlogd -tt /var/log/gitlab/redis
```

Figure 8 - PID 1 wasn't init process

```
root@gitlab:/# ps -auxww | grep init
ps -auxww | grep init
root       12189  0.0  0.0  11280   976 pts/1    S+   08:53   0:00 grep --color=auto init
root@gitlab:/#
```

Figure 9 - Init process not found

In addition, even Linpeas provided some proofs that I was in a container:

```
                                ╔══════════╗
                                ║ Container ║
                                ╚══════════╝
                Container related tools present (if any):
                Am I Containered?
                Container details
  Is this a container? ........... docker
  Any running containers? ........ No
                Docker Container details
  Am I inside Docker group ....... No
  Looking and enumerating Docker Sockets (if any):
  Docker version ................. Not Found
  Vulnerable to CVE-2019-5736 .... Not Found
  Vulnerable to CVE-2019-13139 ... Not Found
  Rootless Docker? ............... No
```

Figure 10 - Docker container recognized by Linpeas

Linpeas was useful to find a clue on how to exploit the container, as well:

```
                Executing Linux Exploit Suggester 2
  https://github.com/jondonas/linux-exploit-suggester-2

                Protections
  AppArmor enabled? .............. AppArmor Not Found
  AppArmor profile? .............. unconfined
  is linuxONE? ................... s390x Not Found
  grsecurity present? ............ grsecurity Not Found
  PaX bins present? .............. PaX Not Found
  Execshield enabled? ............ Execshield Not Found
  SELinux enabled? ............... sestatus Not Found
  Seccomp enabled? ............... disabled
  User namespace? ................ enabled
  Cgroup2 enabled? ............... enabled
  Is ASLR enabled? ............... Yes
  Printer? ....................... No
  Is this a virtual machine? ..... Yes (docker)
```

Figure 11 - Container exploitation clue

In particular, Linpeas suggested the following two methods:

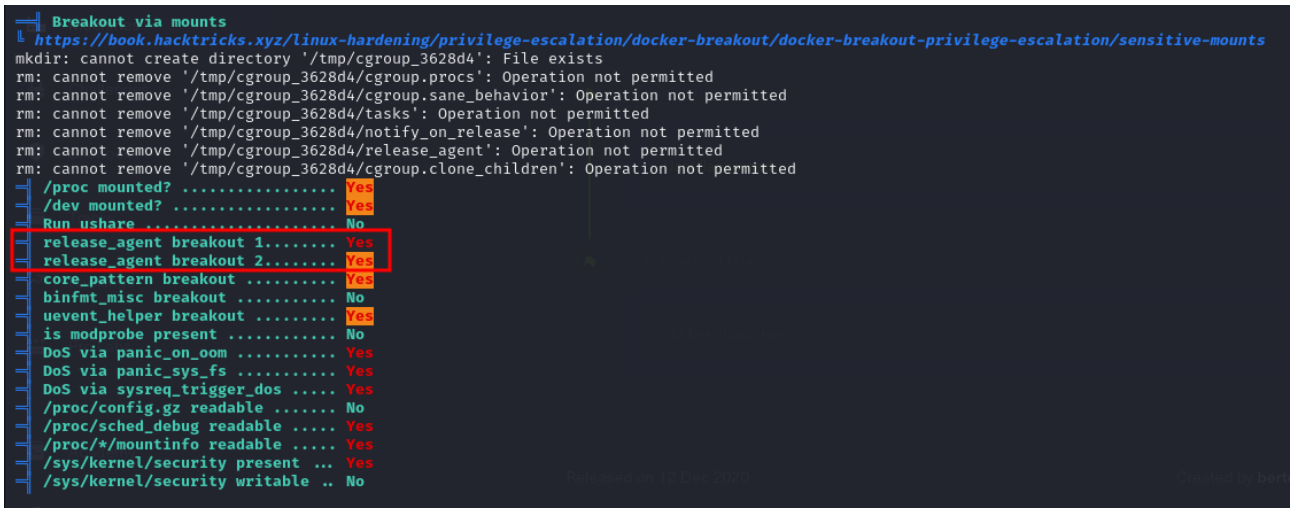

Figure 12 - Docker vulnerabilities

Looking on the Internet for some way to exploit the container, I found some interesting tutorial. First of all, I needed to check which enabled capabilities I had:

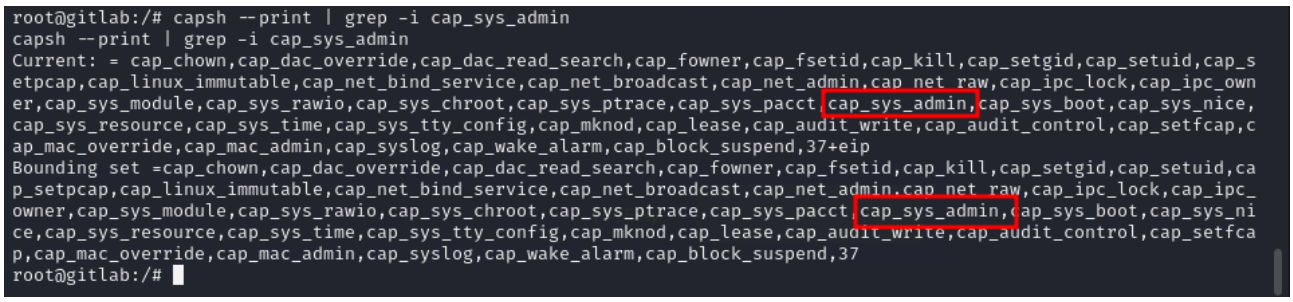

Figure 13 - Enabled capabilities

Luckily, the *cap_sys_admin* capability was enabled. Second, I needed to check if AppArmor was running:

Figure 14 - AppArmor running check

Since the file was empty or didn't exist, AppArmor was not running. This condition was perfect to run the exploit I found. Following the tutorial, I was able to became root on the machine and retrieve the root flag:

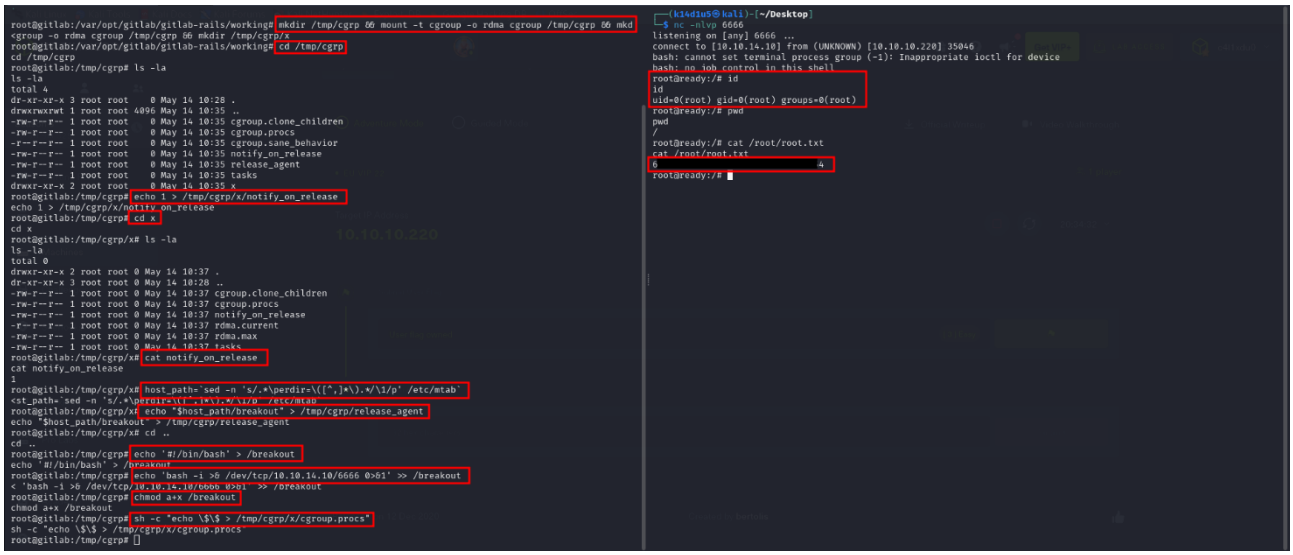

Figure 15 - Privilege escalation and root flag

Personal comments

I was very surprised by this box. The most interesting part was the privilege escalation. This was the first time I needed to exploit a container to retrieve a flag. It was not easy because I had some clues that could make me think I was really root, but it was just in a container. So, I learned about I need to pay attention to some other details and I improved my skills for sure. However, I lost a lot of time because of the GitLab exploit. In fact, some of them I tried it was not correct. They missed a very important part to make them work. In conclusion, it was a very good and interesting box and I evaluate medium on platform (if I remember well).

Appendix A – CVE-2018-19571

This vulnerability affects an unknown code of the *Webhooks* component. The manipulation with an unknown input leads to a server-side request forgery vulnerability. The web server receives a URL or similar request from an upstream component and retrieves the contents of this URL, but it does not sufficiently ensure that the request is being sent to the expected destination.

Appendix B – CVE-2018-19585

This vulnerability affects an unknown code of the component *Project Mirroring*. The manipulation with an unknown input leads to a CRLF injection vulnerability. The product uses CRLF (carriage return line feeds) as a special element, e.g. to separate lines or records, but it does not neutralize or incorrectly neutralizes CRLF sequences from inputs.

References

1. GitLab 11.4.7 Remote Code Execution - https://liveoverflow.com/gitlab-11-4-7-remote-code-execution-real-world-ctf-2018/;
2. GitLab 11.4.7 Remote Code Execution PoC - https://github.com/dotPY-hax/gitlab_RCE;
3. Understanding Docker container escaping - https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/;
4. Container escaper checker - https://github.com/teamssix/container-escape-check;
5. Digging into cgroups escaping - https://0xdf.gitlab.io/2021/05/17/digging-into-cgroups.html.