Race Condition Vulnerability Lab Report

Mudit Vats
mpvats@syr.edu
10/19/2018

Table of Contents

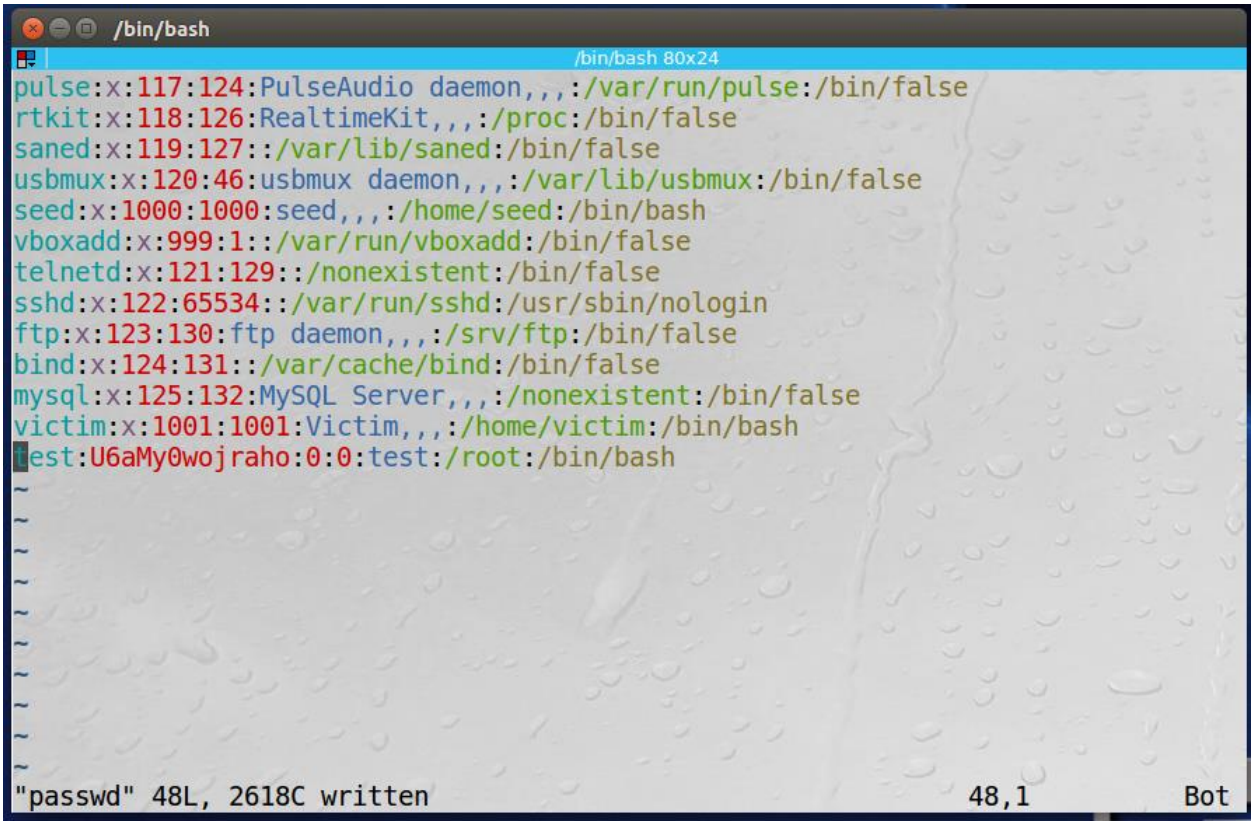
Overview

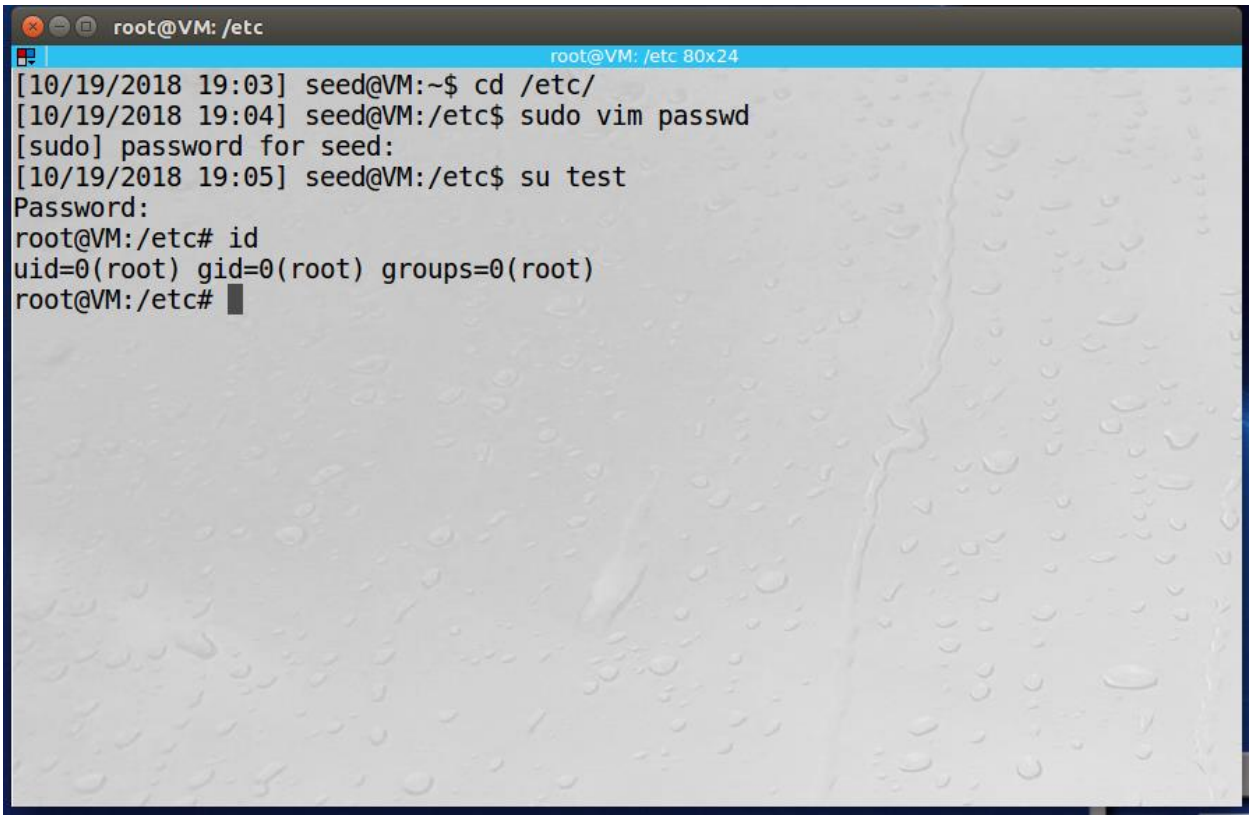
This lab report presents observations and explanations for the tasks described in the Race Condition Vulnerability Lab.

Task 1: Choosing Our Target

Goal: Please report whether you can log into the test account without typing a password, and check whether you have the root privilege.

The file below includes the lower several lines of the /etc/passwd file. I've added the test user account with the magic value for the password which allows no password to be specified when logging in. We also specific, via the third parameter that the user test will have root level (0) privileges.





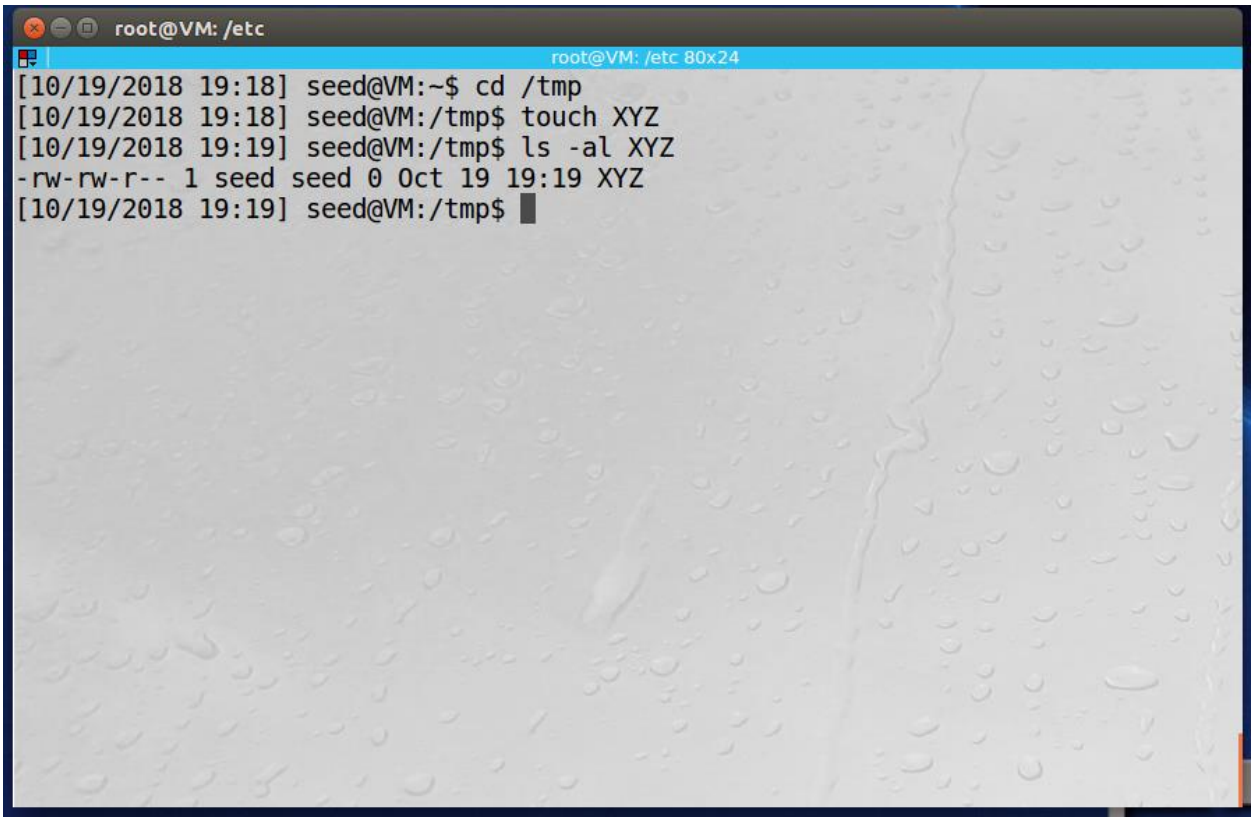
Observations / Explanation

In the figure above, we can see that we were able to switch user to test and login without a password. Additionally, by executing the id command, we can see that the test user has root privileges -- uid=0(root). Success!

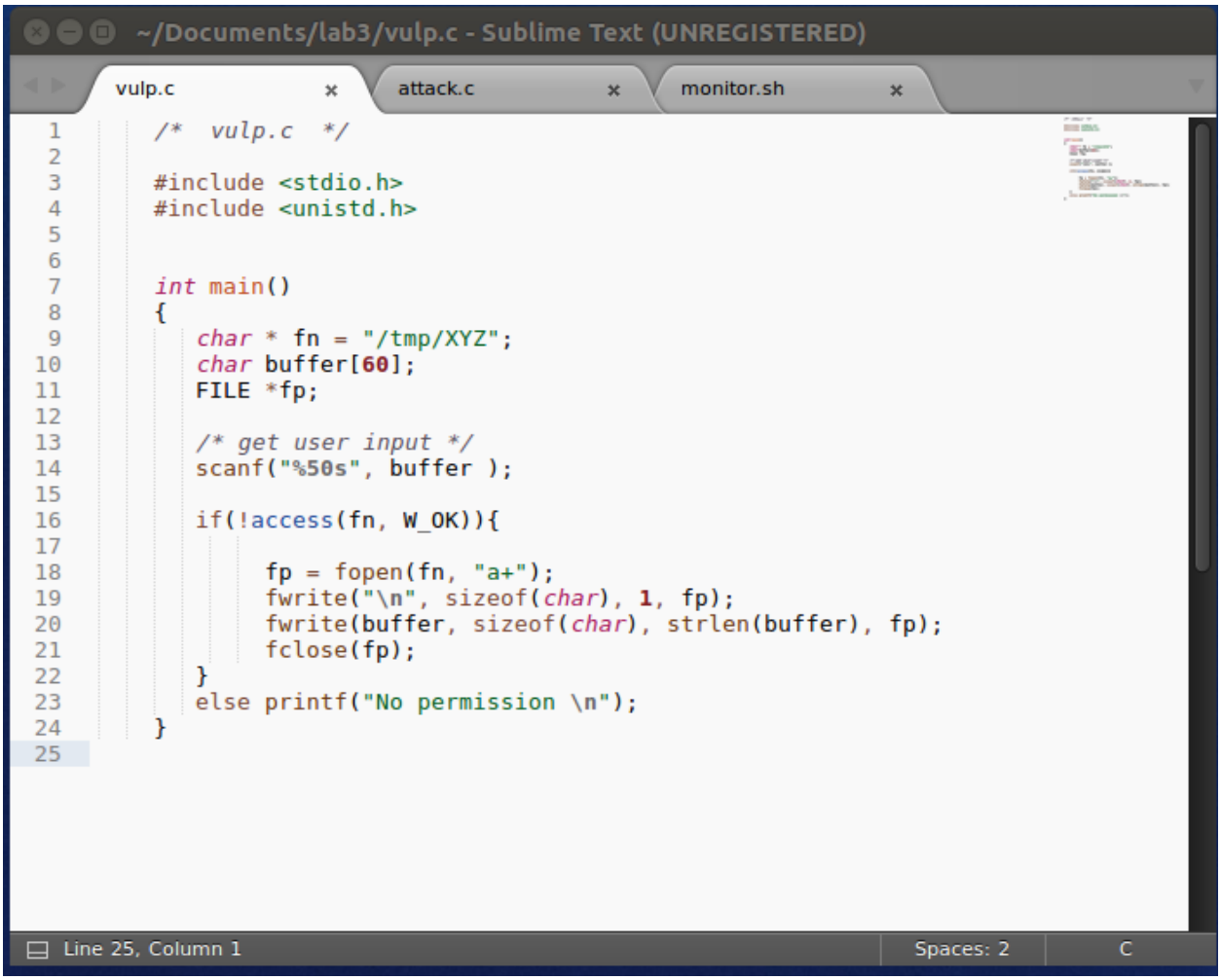
Task 2: Launching the Race Condition Attack

Goal: The goal of this task is to exploit the race condition vulnerability in the vulnerable Set-UID program listed earlier. The ultimate goal is to gain the root privilege.

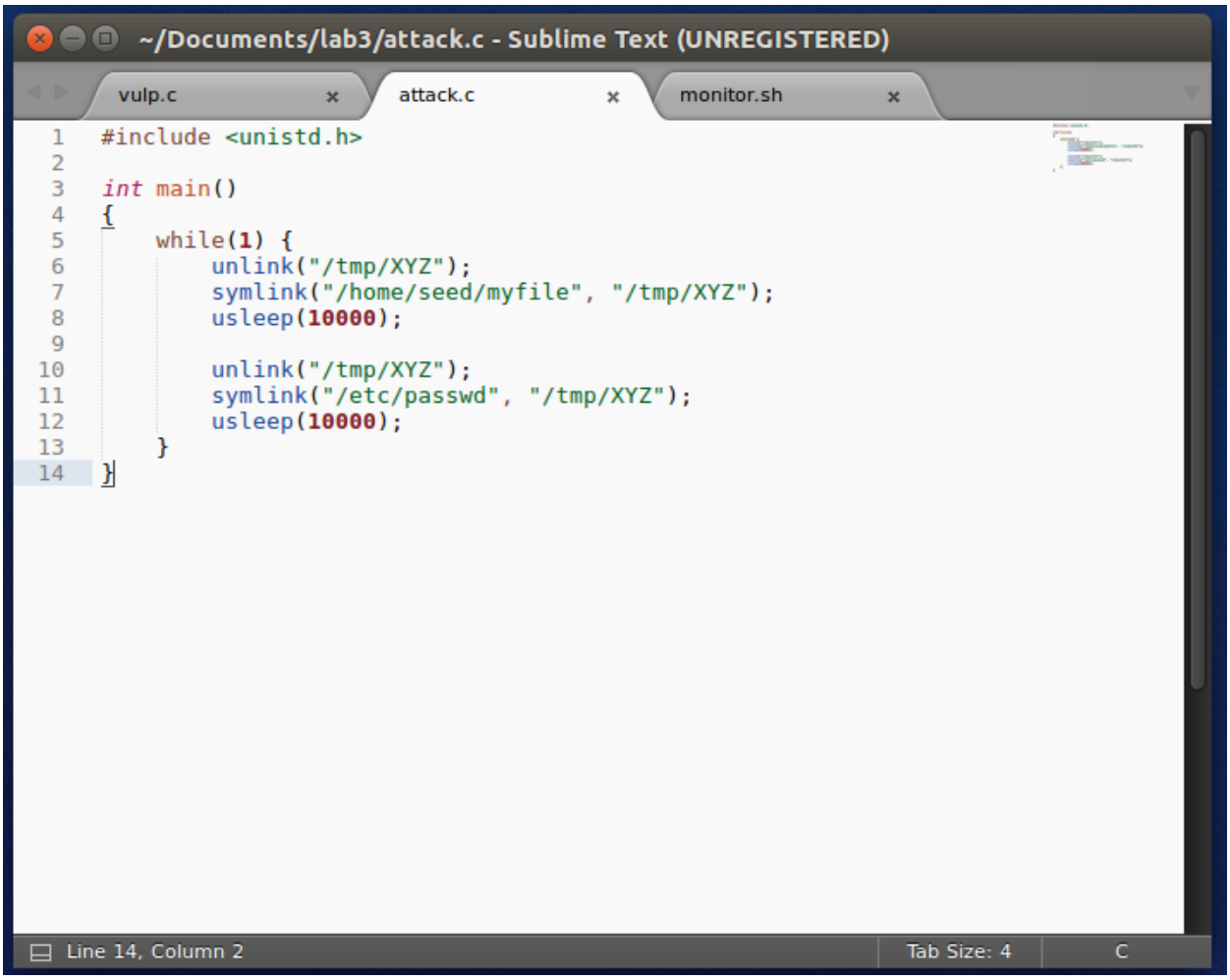
In the figure below, we create the XYZ file as seed. This file is used by the vulp.c program. This is the file we are going to exploit by linking it to a file that vulp has access to and then linking it to a file it doesn't have "access()" to. We will do this repeatedly in the attack.c program.



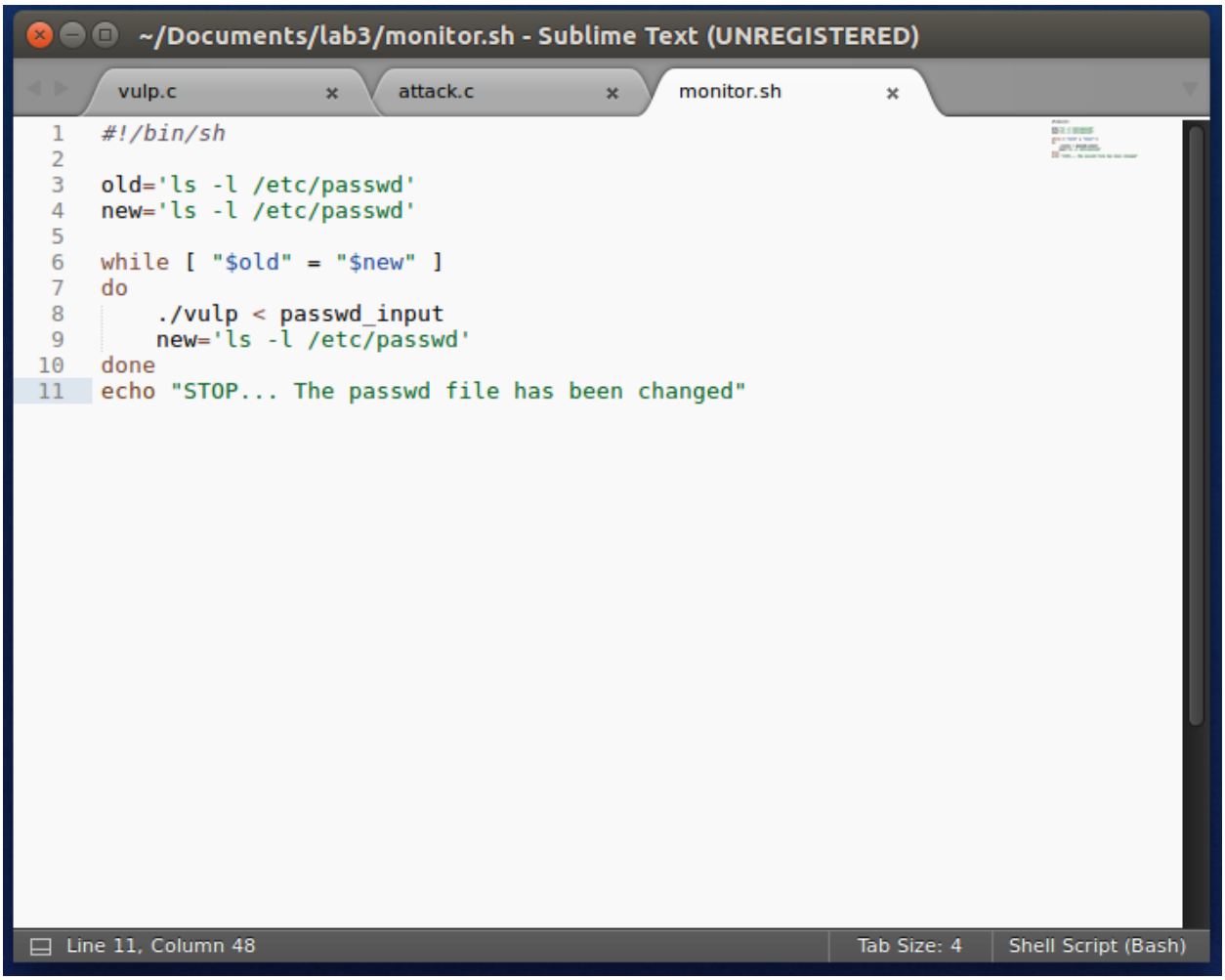
The program below, show the code for the vulp.c, which is the vulnerable program. This program simply checks if the user has access to the XYZ file and, if so, opens the file for write. Please note, that access() checks the real-id of the user and fopen() checks the effective id of the user. This is the window which gets exploited since access() is not checked again within the fopen() and since we are running as a Set-UID program any file on the system, even one that the user should not have access too, can be opened for write.



The program below is the code for the attack. This attack works by interleaving symbolic links to a file that the user has access() to and a file that a user doesn't have access() to. The hope is that the attack switches links to the non-access file after the access() in the vulnerable program code gets pass the access check. Then, since fopen() doesn't access check for access again, it will try to open the second symbolic link. In this case, the second file to open is /etc/passwd which test does not have access to.



The program below is a shell script which checks if the /etc/password file date has changed and if so, stops executing the vulp program. Additionally, it uses input redirection to input the user/password text to the vulp program.



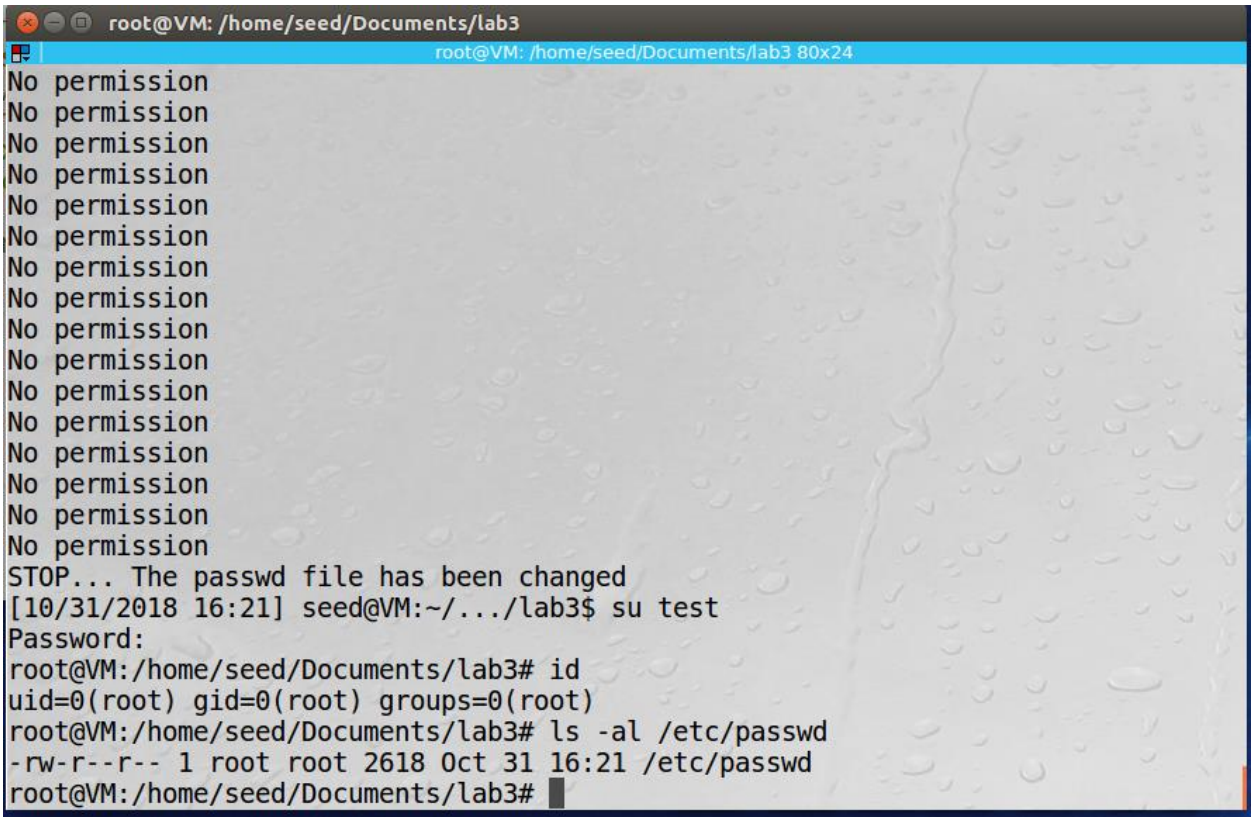
Below we see the execution of "./attack" and "monitor.sh". The monitor.sh stops when the /etc/passwd file is changed (see the "STOP… The password file has changed" message). The figure below this one shows that the /etc/passwd file now has the user/password line we passed into the vulp program.


/bin/bash
/bin/bash 80x24
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
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
[10/31/2018 16:21] seed@VM:~/.../lab3$


/bin/bash
/bin/bash 80x24
avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/bin/false
colord:x:113:123:colord colour management daemon,,,:/var/lib/colord:/bin/false
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/
false
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
victim:x:1001:1001:Victim,,,:/home/victim:/bin/bash
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
~
~
30,1 Bot

After the new user is added to the /etc/passwd, we check to see if we can log in as the new test user. We can and we also see, via the id command, that we have root privileges. The exploit was a success!

The screenshot below shows that the password file with the current data/time of the execution, per the redo.



Observations / Explanation

The exploit works by taking advantage of a race condition which exists in the vulnerable program. It exploits the window between the access() function and the fopen().

The access() only checks for the real id. The fopen() does not check for access again. Since we are running as a Set-UID program, if the file that is being opened is "changed" before the fopen, but after the access(), the vulp program will open that file regardless of the user's real-id.

In the case of this exploit, we use the attack program to switch between a file which seed has access to and the /etc/passwd file. We count on the context switching (via the usleep() in attack) to yield execution to vulp after changing what XYZ points to. We hope that after some amount of runs a the following sequence of events occurs –

1. Attack changes /tmp/XYZ to point to /home/seed/myfile.

2. vulp checks access("/tmp/XYZ") and passes since seed has access to /home/seed/myfile, which is what /tmp/XYZ point to.

3. Attack changes /tmp/XYZ to point to /etc/passwd.

4. Vulp executes fopen(/tmpXYZ) and successfully opens the file for writing. The file /tmp/XYZ now point to /etc/passwd per step 3. Since access() is not checked again, this file successfully opens.

5. The contents of "passwd_input" is written to the /etc/passwd file.

6. Malicious user was able to "su test" and gain root privileges into the system.

By exploiting the race condition and taking advantage of the window between access() and fopen() we were able to successfully gain root privileges.

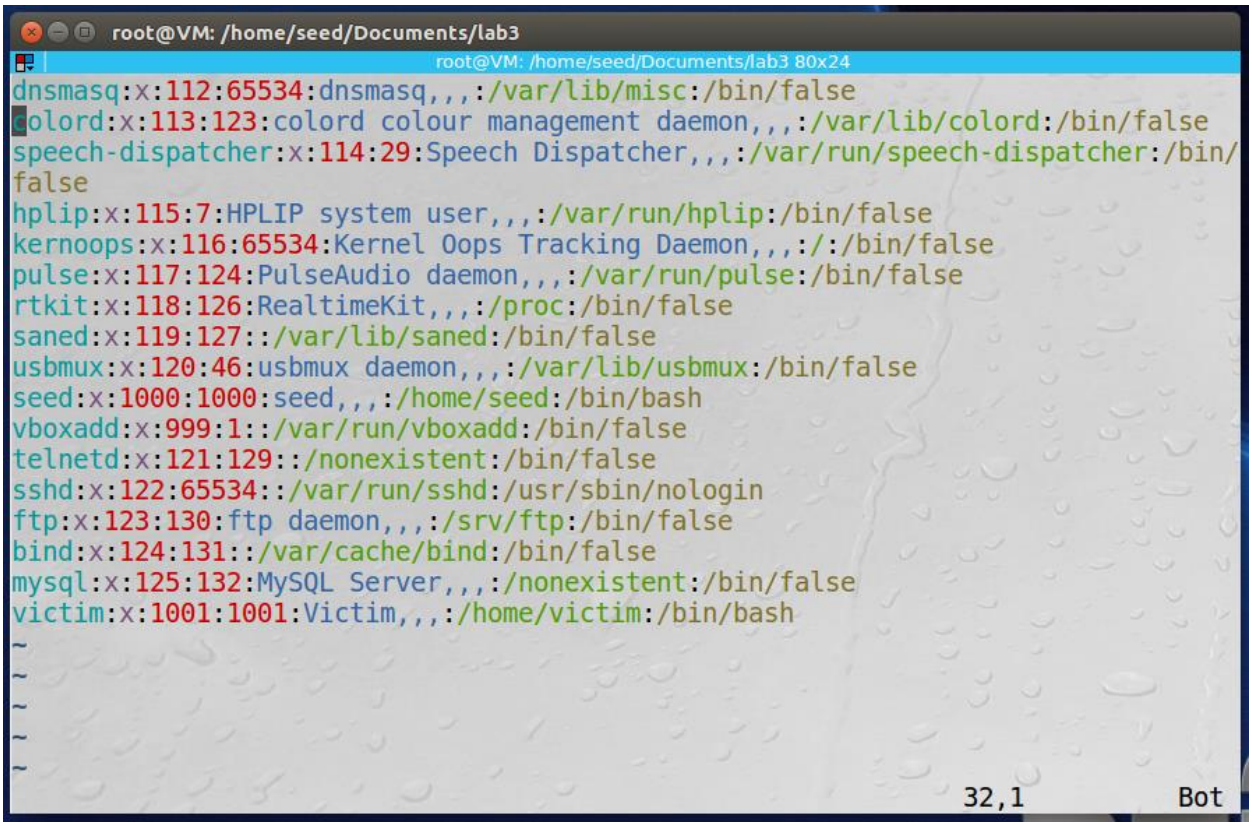
Task 3: Applying the Principle of Least Privilege

Goal: Compile shell_code.c and run it. Document observations.

The figure below shows the vulp program modified to set the effective userid to the real id. By doing this, we essentially lower vulp's privileges to those of the user account and not root.

We then execute the attack again. The first figure shows that we execute the program but it never stops. This means that the /etc/passwd file never changes. We further confirm this by looking at the /etc/passwd file, the second figure, and see that the test user/password is not appended to this file.


root@VM: /home/seed/Documents/lab3
root@VM: /home/seed/Documents/lab3 80x24
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission
No permission


root@VM: /home/seed/Documents/lab3
root@VM: /home/seed/Documents/lab3 80x24
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/bin/false
colord:x:113:123:colord colour management daemon,,,:/var/lib/colord:/bin/false
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/
false
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
victim:x:1001:1001:Victim,,,:/home/victim:/bin/bash
~
~
~
~
~
32,1 Bot

Observations / Explanation

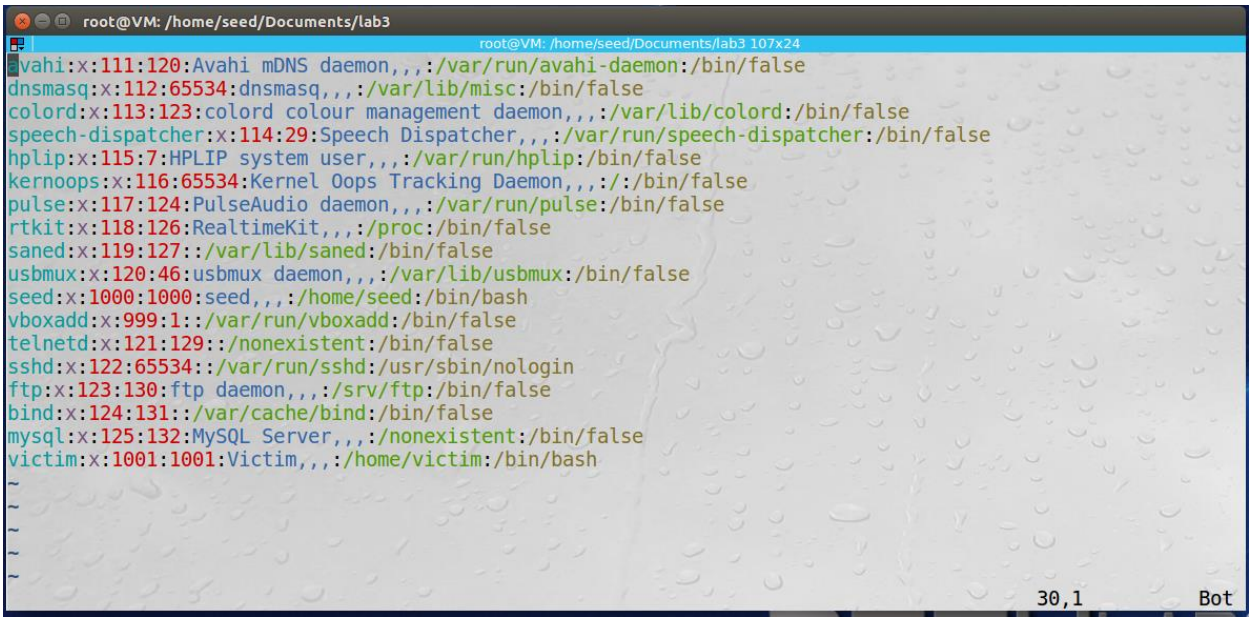
By using least privilege, we gave the vulp the least amount of privilages it needs to execute the operations it needs to do. By doing this, we limit what fopen() can open; i.e. it will only be able to open files which seed has access to. It can no longer open any file that a "root" Set-UID program normally has access to (which is everything). By limiting access, we avert the exploit and thus the attack!

Task 4: Countermeasure – Using Ubuntu's Built-in Scheme

Goal: Conduct attack after the protection is turned on.

We turn on the fs.protected_symlinks=1 protection and re-run the attack. In the figure below, we see that we get No permission and Segmentation faults.

After waiting some time to see if we hit the vulnerability, we end the program. The figure below shows the /etc/passwd. We can see that it does not get appended with the test user/password.



Observations / Explanation

By reenabling the protection mechanism the symbolic link to /etc/passwd no long results in vulp writing to the file. To understand the protection, the following definition exists –

https://sysctl-explorer.net/fs/protected_symlinks/

When set to "1" symlinks are permitted to be followed only when outside a sticky world-writable directory, or when the uid of the symlink and follower match, or when the directory owner matches the symlink's owner.

So, since we are in the /tmp directory and uid of the symbolic link (seed) and follower (root) do not match, the symbolic link is not followed which results in the exploit no longer occurring.