Carleton University

# Race Conditions and Access Control

## Assignment 1

Kenji Isak Laguan

101160737

COMP 4108 B Computer System Security

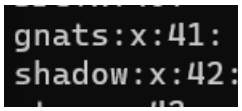
David Barrera

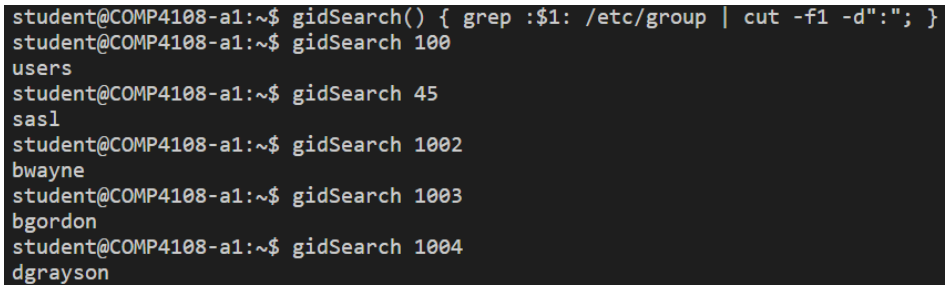
February 5th, 2023

Part A – File system Permissions

1. 1 Mark What is your user id (UID)? How did you determine this?
    a. Used cat /etc/passwd line to view the passwd file contents using cat command.
    b.
```
student:x:1001:1001:,,,:/home/student:/bin/bash
student@COMP4108-a1:~$ cat /etc/passwd
```
    c. My UID for student is 1001
2. 1 Mark What is your primary group name, and the corresponding group id (GID)?
    a. Used id student to display effective user and group ids for a specific user according to the man page which is student
    b.
```
student@COMP4108-a1:~$ id student
uid=1001(student) gid=1001(student) groups=1001(student),27(sudo)
```
    c. The primary group name is student with the corresponding GID is 1001.
3. 0.5 Marks What is the filesystem path of the Linux group file?
    a. The filesystem path of the Linux group file is /etc/group as its in the same directory /etc as passwd file
4. 1 Mark What are the permissions of the Linux group file? Give the permissions as both a *'rwx'* string and the numeric octal representation. You will need to use ls command as well as the stat command.
    a. I used ls -l /etc/group to get the permissions of the group file where ls lists the directory contents, -l flag displays the long listing format according to the man page for the file that shows the permissions in rwx string.
        i. -rw-r—r-- is the rwx permissions for group
    b. I used stat -c “%a” /etc/group to get the permissions of the group file where stat displays the file, -c flag uses a specified format with “%a ” operand that is for displaying access/permission rights in octal according to man page.
        i. 644 is the octal permissions for group (110 100 100 in binary)
    c.
```
student@COMP4108-a1:~$ stat -c "%a" /etc/group
644
student@COMP4108-a1:~$ ls -l /etc/group
-rw-r--r-- 1 root root 890 Nov 30  2016 /etc/group
student@COMP4108-a1:~$
```

5. 0.5 Marks What is the GID of the root group?
    a. Used id root to display effective user and group ids for a specific user according to the man page which is root. We can user the root as a user since we know they have the same name for group and user
    b.
```
student@COMP4108-a1:~$ id root
uid=0(root) gid=0(root) groups=0(root)
```
    c. The GID of root group is 0.

6. 0.5 Marks What is the GID of the shadow group?
    a. Since shadow doesn't have a user, we just cat /etc/group and look for the shadow in those contents which will show its corresponding gid
    b. 

    c. Shadow group has a GID of 42
7. 4 Marks Write a bash function to convert a GID to a group name using the Linux group file. Submit both the function definition and 5 invocations.
   Use the following skeleton as a starting point:
    a. We use the command grep :$1: / etc/group | cut -f1 -d":" and declared it as a bash function in the terminal with gidSearch() { grep :$1: / etc/group | cut -f1 -d":"; }.
        i. Grep is used to find the line that match a pattern we set which is the GID given along semicolons on either side, so we retrieve the line with the gid that's attached to its group name in /etc/group file.
            1. because the group entries in the file follow a "gname:x:gid:" pattern
        ii. I piped it together with cut -f1 -d":" as cut is used to remove the sections we don't want in that line we found. -d":" is the delimiter where we set the character to be the semicolon where this is where we separate the text, and -f1 is us choosing the field of text where the semicolons separate the text. So we get the group name out of that line.
    b. Function definition, invocations, and results
        i. 

8. 2 Marks (total) List **ALL** directories and sub-directories in /A1/Haystack that have the following properties:

    a. 0.5 Marks Are owned by the user comp4108. Include the command used and the resulting list.
        a. Used find /A1/Haystack -type d -user comp4108, used find to list all the directories and sub-directories at the directory /A1/Haystack. -type d is for looking for only types of d which is directories, -user is for displaying only those that are owned by user comp4108
        b.
```
student@COMP4108-a1:~$ find /A1/Haystack -type d -user comp4108
/A1/Haystack
/A1/Haystack/Bar
/A1/Haystack/Bar/Charlie
/A1/Haystack/Bar/Alpha
/A1/Haystack/Foo
/A1/Haystack/Foo/S/S
/A1/Haystack/Foo/P/P
/A1/Haystack/Foo/J/J
/A1/Haystack/Foo/D/D
/A1/Haystack/Foo/E/E
/A1/Haystack/Foo/F/F
/A1/Haystack/Foo/M/M
/A1/Haystack/Foo/B/B
/A1/Haystack/Foo/W/W
/A1/Haystack/Foo/Z/Z
/A1/Haystack/Foo/Q/Q
/A1/Haystack/Foo/O/O
/A1/Haystack/Foo/L/L
/A1/Haystack/Foo/X/X
/A1/Haystack/Foo/U/U
/A1/Haystack/Foo/A/A
/A1/Haystack/Foo/V/V
/A1/Haystack/Foo/I/I
/A1/Haystack/Foo/C/C
/A1/Haystack/Foo/H/H
/A1/Haystack/Foo/N/N
/A1/Haystack/Foo/K/K
/A1/Haystack/Foo/R/R
/A1/Haystack/Foo/G/G
/A1/Haystack/Baz
/A1/Haystack/Baz/Beta
/A1/Haystack/Baz/Charlie
```
    b. 0.5 Marks Have group ownership root. Include the number of directories but **NOT THE ENTIRE LIST**
        a. Used find /A1/Haystack -type d -group root | wc -l, used find to list all the directories and sub-directories at the directory /A1/Haystack.-type d is for looking for only types of d which is directories, -group is for displaying only those that are owned by group root. Piped with wc -l that counts the number of lines from the resulting list  with the -l flag in wc command.
        b.
```
student@COMP4108-a1:~$ find /A1/Haystack -type d -group root | wc -l
422
```
        c. There are 422 directories and subdirectories owned by root
    c. 0.5 Marks Are owned by the user sshd. Include the command used and the resulting list.
        a. Used find /A1/Haystack -type d -user sshd, used find to list all the directories and sub-directories at the directory /A1/Haystack. -type d is for looking for only types of d which is directories, -user is for displaying only those that are owned by user sshd
        b. Refer to PartAq8c.txt for list output

d. 0.5 Marks Have permissions equal to 777. Include the command used and the resulting list.
    a. Used find /A1/Haystack -type d -perm 777, used find to list all the directories and sub-directories at the directory /A1/Haystack. -type d is for looking for only types of d which is directories, -perm is for displaying only those that have 777 octal permission bits
    b.

```
student@COMP4108-a1:~$ find /A1/Haystack -type d -perm 777
/A1/Haystack/Bar/Beta
/A1/Haystack/Foo/F
/A1/Haystack/Foo/U/M
/A1/Haystack/Foo/U/Y
/A1/Haystack/Foo/H/P
/A1/Haystack/Foo/H/H
/A1/Haystack/Baz/Charlie
```

9. 1 Mark Write a command to change all the directories with permissions 777 in /A1/Haystack to have permissions 750 instead.
    a. Used sudo find /A1/Haystack -type d -perm 777 -exec chmod 750 {} \; , used find to list all the directories and sub-directories at the directory /A1/Haystack. -type d is for looking for only types of d which is directories, -perm is for displaying only those that have 777 octal permission bits, -exec chmod 750 {} \; to execute the command chmod 750, {} is replaced with the file names we found in the search and \; to escape according to the man page for -exec in find. And sudo to run as root as we need root perms to do this.
    b.

```
student@COMP4108-a1:~$ sudo find /A1/Haystack -type d -perm 777 -exec chmod 750 {} \;
[sudo] password for student:
student@COMP4108-a1:~$ sudo find /A1/Haystack -type d -perm 777
student@COMP4108-a1:~$ sudo find /A1/Haystack -type d -perm 750
/A1/Haystack/Bar/Beta
/A1/Haystack/Foo/F
/A1/Haystack/Foo/U/M
/A1/Haystack/Foo/U/Y
/A1/Haystack/Foo/H/P
/A1/Haystack/Foo/H/H
/A1/Haystack/Baz/Charlie
```

        i. Ran the command in the first line. 2nd line checked if any directories pop up with 777 perms. 3rd line showing results of directories with 750 perms after the change.

10. 2 Marks (total) In your home directory create the following directory structure using mkdir:

    a. 0.5 Marks Change the permissions of bottom and bottom_two to 664
        a. Used chmod 664 directoryname to change the permissions for both directories bottom and bottom_two to 664 that weve used in junction in 9a. Then verified and checked the output of permissions with stat -c “%a” directoryname that weve used in 4b.

b.
```
student@COMP4108-a1:~/top/middle$ chmod 664 bottom
student@COMP4108-a1:~/top/middle$ stat -c "%a" bottom
664
student@COMP4108-a1:~/top/middle_two$ chmod 664 bottom_two
student@COMP4108-a1:~/top/middle_two$ stat -c "%a" bottom_two
664
```

b. 0.5 Marks Create an empty file foo.txt in middle_three and grant the execute permission for user and group
   a. Created the empty file with touch foo.txt, used a binary to octal converter online to change 001 001 000 into octal with the bits set as 1 is the position for execute perms for the user and group but not other and the octal conversion is 110. Used chmod 110 foo.txt to change the perms for user and group to have execute. And verified with ls -l foo.txt that weve used in 4a.

b.
```
student@COMP4108-a1:~/top/middle_three$ touch foo.txt
student@COMP4108-a1:~/top/middle_three$ chmod 110 foo.txt
student@COMP4108-a1:~/top/middle_three$ ls -l foo.txt
---x--x--- 1 student student 0 Feb  1 23:21 foo.txt
```

c. 0.5 Marks Change the ownership of top to root using the sudo and chown command
   a. Used sudo chown 0 top to change the ownership to root. Sudo as root perms are needed, chown to change the user ownership to 0 which is root UID can use root also but better to use UID, and top as the directory name. then used ls -l from 4a to verify root user ownership and group ownership is still set to student.

b.
```
student@COMP4108-a1:~$ sudo chown 0 top
student@COMP4108-a1:~$ ls -l
total 12
-rw-rw-r-- 1 student student  141 Feb  1 21:55 gidtogroup.sh
-rw-rw-r-- 1 student student 2688 Feb  1 22:26 PartAq8c.txt
drwxrwxr-x 5 root    student 4096 Feb  1 23:00 top
```

d. 0.5 Marks Change the group of top to www-data using sudo and the chgrp command.
   a. Used sudo chgrp www-data top to change the group ownership to www-data. Sudo as root perms are needed, chgrp to change the group ownership to www-data or 33 is the GID, and top as the directory name. then used ls -l from 4a to verify www-data group ownership and user ownership is still set to root.

b.
```
student@COMP4108-a1:~$ sudo chgrp www-data top
student@COMP4108-a1:~$ ls -l
total 12
-rw-rw-r-- 1 student student  141 Feb  1 21:55 gidtogroup.sh
-rw-rw-r-- 1 student student 2688 Feb  1 22:26 PartAq8c.txt
drwxrwxr-x 5 root    www-data 4096 Feb  1 23:00 top
```

11.

a. 1 Mark Find all the binary files in /usr/bin with the setuid bit set.
   i. Used find /usr/bin -type f -perm -u=s , used find to list all the files at the directory /usr/bin,-type f is for looking for only types of f which is files, -perm is for displaying only those that have -u=s where user has the executable setuid bit set. All executable files are binary files so no need for any other options to verify.

ii.
```
student@COMP4108-a1:/usr/bin$ find /usr/bin -type f -perm -u=s
/usr/bin/at
/usr/bin/sudo
/usr/bin/newgrp
/usr/bin/traceroute6.iputils
/usr/bin/chsh
/usr/bin/passwd
/usr/bin/gpasswd
/usr/bin/netkit-rsh
/usr/bin/sudoedit
/usr/bin/mtr
/usr/bin/netkit-rlogin
/usr/bin/netkit-rcp
/usr/bin/chfn
```

b. 1 Mark Using one of the binaries you found as an example, describe why it has the setuid bit enabled, and what the setuid bit accomplishes.
   i. /usr/bin/passwd
      1. As it has the setuid bit enabled it allows the unprivileged user to run that exectuble/binary file with the original file owner permissions which is root and so the effective uid is root during that time and only allows that binary to edit and change that user's password as if the binary has the user permissions it will not be able to change the users password as it's a sensitive file that needs root permissions. This prevents the user from being able to access the password file in malicious ways as we don't give user permissions to access that sensitive file and only the binary that has that setuid bit enable would be able to access it during the time its run.

Part B:

1. 0.5 Marks Give the ACL for the top level Gotham directory.
   a. Used getfacl /A1/Gotham to get the Access Control List of Gotham. Getfacl command is used as its purpose is to get the access control list of a specified file which I gave the location of /A1/Gotham.
   b.
```
student@COMP4108-a1:~$ getfacl /A1/Gotham
getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham
# owner: bwayne
# group: gotham
user::rwx
group::r-x
other::---
```

2. 1 Mark Use chmod to add rx permissions for the other category to Gotham and **ALL** its sub-directories.
    a. Used sudo chmod -R -o+rx /A1/Gotham to accomplish the task. Sudo was used as normal user wasn't permitted to do so, chmod to change the permissions, -R to apply all the changes to all files and directories in the specified location, -o+rx was to set o which is others + to add the rx permissions for the other category, for the directory and subdirectories in /A1/Gotham. And was verified with find /A1/Gotham -perm -o=rx to check which directories had the permissions set to rx for others category.
    b.
```
student@COMP4108-a1:/A1/Gotham$ sudo chmod -R -o+rx /A1/Gotham
student@COMP4108-a1:/A1/Gotham$ find /A1/Gotham -perm -o=rx
/A1/Gotham
/A1/Gotham/WayneManor
/A1/Gotham/WayneManor/MasterBedroom
/A1/Gotham/WayneManor/Batcave
/A1/Gotham/Arkham
/A1/Gotham/GothamPD
```
3. 1 Mark Use setfacl to add read and write permissions to Gotham, Arkham and GothamPD for the user jgordon
    a. Used sudo setfacl -m u:jgordon:rw directory to add read and write perms to the 3 directories. Sudo as user perms weren't sufficient, setfacl to set acl, -m to modify the acl and u:jgordon:rw to specify category:username:permissions that were setting for the acl. And the directories we add the perms to /A1/Gotham, /A1/Gotham/Arkham, /A1/Gotham/GothamPD . so we use this line 3 times for each of the directories. Then verify the acl modification with getfacl for those directories.
    b. To add the rw perms
        i.
```
student@COMP4108-a1:/A1/Gotham$ sudo setfacl -m u:jgordon:rw /A1/Gotham
[sudo] password for student:
student@COMP4108-a1:/A1/Gotham$ sudo setfacl -m u:jgordon:rw /A1/Gotham/Arkham
student@COMP4108-a1:/A1/Gotham$ sudo setfacl -m u:jgordon:rw /A1/Gotham/GothamPD
```
    c. Proof the acl was modified for all directories
        i.
```
student@COMP4108-a1:/A1/Gotham$ getfacl /A1/Gotham
getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham
# owner: bwayne
# group: gotham
user::rwx
user:jgordon:rw-
group::r-x
mask::rwx
other::r-x

student@COMP4108-a1:/A1/Gotham$ getfacl /A1/Gotham/Arkham
getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham/Arkham
# owner: bwayne
# group: gotham
user::rwx
user:skyle:rw-
user:ocobblepot:rwx
user:jgordon:rw-
group::rwx
mask::rwx
other::r-x

student@COMP4108-a1:/A1/Gotham$ getfacl /A1/Gotham/GothamPD
getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham/GothamPD
# owner: bwayne
# group: gotham
user::rwx
user:jgordon:rw-
group::r-x
mask::rwx
other::r-x
```

4. 1 Marks Use setfacl to add read, write, and execute permissions to WayneManor, Batcave and MasterBedroom for the user bwayne.
    a. Used sudo setfacl -m u:bwayne:rwx directory to add read and write perms to the 3 directories. Sudo as user perms weren't sufficient, setfacl to set acl, -m to modify the acl and u:bwayne:rwx to specify category:username:permissions that were setting for the acl. And the directories we add the perms to /A1/Gotham/WayneManor, /A1/Gotham/ WayneManor/Batcave, /A1/Gotham/ WayneManor/MasterBedroom. so we use this line 3 times for each of the directories. Then verify the acl modification with getfacl for those directories.
    b. To add the rwx perms
        i.
```
student@COMP4108-a1:/A1/Gotham$ sudo setfacl -m u:bwayne:rwx /A1/Gotham/WayneManor
student@COMP4108-a1:/A1/Gotham$ sudo setfacl -m u:bwayne:rwx /A1/Gotham/WayneManor/Batcave
student@COMP4108-a1:/A1/Gotham$ sudo setfacl -m u:bwayne:rwx /A1/Gotham/WayneManor/MasterBedroom
```

    c. Proof the acl was modified for all directories
        i.
```
student@COMP4108-a1:/A1/Gotham$ getfacl /A1/Gotham/WayneManor
getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham/WayneManor
# owner: bwayne
# group: gotham
user::rwx
user:bwayne:rwx
group::r-x
mask::rwx
other::r-x

student@COMP4108-a1:/A1/Gotham$ getfacl /A1/Gotham/WayneManor/Batcave
getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham/WayneManor/Batcave
# owner: bwayne
# group: gotham
user::rwx
user:bwayne:rwx
group::r-x
mask::rwx
other::r-x

student@COMP4108-a1:/A1/Gotham$ getfacl /A1/Gotham/WayneManor/MasterBedroom
getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham/WayneManor/MasterBedroom
# owner: bwayne
# group: gotham
user::rwx
user:bwayne:rwx
group::r-x
mask::rwx
other::r-x
```

5. 1 Marks Use setfacl to remove the ACL entries on Arkham for the users skyle and ocobblepot.
    a. Used sudo setfacl -x u:username /A1/Gotham/Arkham to remove acl entries for the specified users. Sudo as user perms weren't sufficient, setfacl to set acl, -x to remove acl entries and u:skyle and u:ocobblepot to specify category:username that were removing acl entries for. So we do this twice for each of user. Then verify the acl modification with getfacl for the Arkham directory.
    b. To remove the acl entries
        i.
```
student@COMP4108-a1:/A1/Gotham$ sudo setfacl -x u:skyle /A1/Gotham/Arkham
[sudo] password for student:
student@COMP4108-a1:/A1/Gotham$ sudo setfacl -x u:ocobblepot /A1/Gotham/Arkham
```

c. Proof the acl entries were removed for Arkham

i.
```
student@COMP4108-a1:/A1/Gotham$ getfacl /A1/Gotham/Arkham
getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham/Arkham
# owner: bwayne
# group: gotham
user::rwx
user:jgordon:rw-
group::rwx
mask::rwx
other::r-x
```

6. 2 Marks Give the ACL for Gotham and all its subdirectories.
   a. We can use getfacl -R /A1/Gotham to get all the ACL for Gotham and all its sub directories. Getfacl to get acl of Gotham with -R flag to do so recursively and find all files and subdirectories. This is a quick and simple approach but as the hint suggests were probably looking only for directories well use a different approach. Both ways produce the same result but just because theres no files.
   b. Used find /A1/Gotham -type d -exec getfacl{} \; , used find to list all the directories and sub-directories at the directory /A1/Gotham. -type d is for looking for only types of d which is directories, -exec getfacl {} \; to execute the command getfacl to get the acl of each directory we find, {} is replaced with the directory names we found in the search and \; to escape according to the man pages for -exec in find.
   c.
```
student@COMP4108-a1:/A1/Gotham$ find /A1/Gotham -type d -exec getfacl {} \;
```
   d. All acl of Gotham and subdirectories

i.
```
getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham
# owner: bwayne
# group: gotham
user::rwx
user:jgordon:rw-
group::r-x
mask::rwx
other::r-x

getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham/WayneManor
# owner: bwayne
# group: gotham
user::rwx
user:bwayne:rwx
group::r-x
mask::rwx
other::r-x

getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham/WayneManor/MasterBedroom
# owner: bwayne
# group: gotham
user::rwx
user:bwayne:rwx
group::r-x
mask::rwx
other::r-x

getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham/WayneManor/Batcave
# owner: bwayne
# group: gotham
user::rwx
user:bwayne:rwx
group::r-x
mask::rwx
other::r-x
```

```
getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham/Arkham
# owner: bwayne
# group: gotham
user::rwx
user:jgordon:rw-
group::rwx
mask::rwx
other::r-x

getfacl: Removing leading '/' from absolute path names
# file: A1/Gotham/GothamPD
# owner: bwayne
# group: gotham
user::rwx
user:jgordon:rw-
group::r-x
mask::rwx
other::r-x
```

7. 2 Marks ACLs help enforce which security principle(s) discussed in Chapter 1.7 of the course textbook? Give the name of the principle(s) and a brief explanation on how ACLs help enforce the principle(s). Use only P1-20 and not HP1 or HP2.
    a. P6 least privilege as access control lists are used to modify ownerships of files or directories and when you add permissions for certain users we just give them the least privilege we want them to have. We can also modify those same permissions and even remove them when they are no longer needed to enforce the shortest duration of those privileges are needed.
    b. P5 isolated compartments as access control lists are limited for only the user to setfacl or with root privileges. This controls the privileges for different users to rwx the files or directories and only allows those who has privileges to prevent information leakage to other users that do not have the privileges for the file/directory as they aren't listed in its ACL for example.

Part C:

A.

1. I used sudo strace -o locout.log ./vuln_slow 5 message to view that log file for system calls that was run by vuln_slow with cat locout.log and find the location of the debug file it writes to, where I saw after the nanosleep() sys call which is where the program sleeps before writing, I see open("/home/student/.debug_log", …….) which shows the name and location of the debug file vuln_slow writes to. Used sudo since I needed elevated perms.
    a.
    b.

```
student@COMP4108-a1:/A1/Racing/Slow$ sudo strace -o locout.log ./vuln_slow 5 this_is_where_my_message_is
Sleeping for 5 seconds.
Writing debug message.
Goodbye!
student@COMP4108-a1:/A1/Racing/Slow$ cat locout.log
nanosleep({5, 0}, 0x7ffd311c38a0)       = 0
write(1, "Writing debug message.\n", 23) = 23
open("/home/student/.debug_log", O_RDWR|O_CREAT|O_APPEND, 0666) = 4
```

2. I invoked vuln_slow again and ran it in the back ground with & to be able to run commands needed for the exploit and a large delay of 15 seconds
    a.

```
student@COMP4108-a1:/A1/Racing/Slow$ sudo strace -o locout.log ./vuln_slow 15 this_is_where_my_message_is &
[2] 6962
student@COMP4108-a1:/A1/Racing/Slow$ Sleeping for 15 seconds.
```

3. While it slept I deleted the log file knowing the location and name of the file from step 1, with sudo rm /home/student/.debug_log and used sudo just to make sure I wouldn't run into any trouble. Then created a symbolic link to the target file which is root_file and the location of the file we want for it to point to the target which is /home/student/.debug_log, using the command ln -s /A1/Racing/Slow/root_file /home/student/.debug_log where ln is the command used to make links between files and -s flag to create a symbolic link. Then I used cat root_file to verify my message was written to that file instead of the debug file.

a.

```
student@COMP4108-a1:/A1/Racing/Slow$ sudo strace -o locout.log ./vuln_slow 15 this_is_where_my_message_is &
[2] 6962
student@COMP4108-a1:/A1/Racing/Slow$ Sleeping for 15 seconds.
sudo rm /home/student/.debug_log
student@COMP4108-a1:/A1/Racing/Slow$ ln -s /A1/Racing/Slow/root_file /home/student/.debug_log
student@COMP4108-a1:/A1/Racing/Slow$ Writing debug message.
Goodbye!
cat root_file
#
# This file is owned by root!
# You can test your ability to exploit a race condition by attempting to
# insert lines into this file. Note: You should *NOT* chown/chmod this
# file. You *MUST* insert lines into it using the setuid program 'vuln'
# included in the same directory
#
#
# Your Lines Should Follow Here:
this_is_where_my_message_is
```

5 Marks Describe the exploitation process in detail, including commands invoked and a log of successfully adding a message to root_file. Include an explanation of why the attack works.

b. As ive explained how the exploitation process works above, I will reference that process with this explanation of why the attack works. The reason this is works is: once vuln_slow is run in the background, this permits us to run the commands needed for exploiting, looking the log file of system calls, once access() is called it verifys that it is able to write to the file as it has the W_OK mode and that call returns a 0 so its able to write. Now once its put to sleep and output that, we have to run our exploit commands before the open() is called as that's when it opens the file to write the message to. So we have to run this exploit after it access() and before it open(). So we remove the debug file (we need to remove because we aren't permitted to make a symlink when the file we are setting a pointer to the target still exists) then make our symbolic link to the root_file. Now when open is called to the debug file it will follow the symlink to the target file we set a symlink to on that debug file and open that instead with no issues as the open syscall didn't have the option to not follow a symbolic link like O_NOFOLLOW. Then proceeds to write to our target root_file instead in the write() syscall.

   The access syscall uses the real uid instead of the effective uid as stated in the man page so for setuid programs it will check if the real uid of the user that invoked the program is permitted to access rather than the effective. So since we invoked the setuid program vuln_slow and the access would be successful as it's a file we have access to as a user which would be the debug file. We can just add a symbolic link to the file were trying to write to even without having the permissions of it like root owned files, since open uses the euid which is set to root as the setuid program gives us an euid of root while we run the program, so it allows us to write to the root owned file.

B.

1. Edit the vuln.sh script (using nano, or another text editor) to provide it the location of the debug file
   i. Used sudo strace -o logfile.log ./vuln_fast fast_message to run the program and checked the log file to verify the location of the debug file with cat logfile.log and saw again access() uses the same location /home/student/.debug_log which is used when editing this script with nano vuln.sh.
   ii.
```
  GNU nano 2.2.    File: vuln.sh

#!/bin/bash

DEBUG_FILE=/home/student/.debug_log
```
2. Edit the vuln.sh script to give it the payload string you want written to the target file (**see Hint!**).
   i. As the hint states we are to add the localhost and our username to roots .rhosts file as the rhosts file has a format of hostname username. We know our username is student and the hostname were using is localhost. So now we can set those as our payload "localhost student" in the script.
   ii.
```
while true
do
  nice -n 20 ./vuln_fast "localhost student"
done
```

3. Edit the exploit.sh script to provide it the location of the debug file
   i. Since we know the location of the debug file previously which is /home/student/.debug_log well use this again when editing this script with nano exploit.sh.
   ii.
```
  GNU nano 2.2.   File: exploit.sh

#!/bin/bash

DEBUG_FILE=/home/student/.debug_log
```
4. Edit the exploit.sh script to give it the target file for your attack (**See Hint!**).
   i. As the hint states we are able to find the location of roots home directory by checking the passwd file so I used cat /etc/passwd and looked at the root user entry to see its home directory which is the portion right before the shell directory at the end so the roots home directory is /root which contains the .rhosts file and we can verify with sudo ls -la /root where -l flag shows hidden files(files with . in the front).
      1.
```
student@COMP4108-a1:/A1/Racing/Fast$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

2.
```
student@COMP4108-a1:/A1/Racing/Fast$ sudo ls -la /root
[sudo] password for student:
total 36
drwx------  4 root root 4096 Jan 22  2017 .
drwxr-xr-x 24 root root 4096 Sep  2  2021 ..
-rw-------  1 root root    5 Jan 22  2017 .bash_history
-rw-r--r--  1 root root 3106 Apr 19  2012 .bashrc
drwx------  2 root root 4096 Dec  6  2013 .cache
-rw-r--r--  1 root root  140 Apr 19  2012 .profile
-rw-r--r--  1 root root    0 Jan 29  2014 .rhosts
```

ii. Then as I found the location which is /root/.rhosts and used that as the target file so our payload which is our localhost and username is where we can add it to.

1.
```
#!/bin/bash

DEBUG_FILE=/home/student/.debug_log
TARGET_FILE=/root/.rhosts
```

5. Run the vuln.sh script in one terminal
   i. Used ./vuln.sh in the /A1/Racing/Fast directory to run the script
   ii.
```
Writing debug message.
Goodbye!
Invoking user does not have access to log file.
Invoking user does not have access to log file.
Writing debug message.
Goodbye!
```

6. Run the exploit.sh script in another terminal
   i. Used ./exploit.sh in the /A1/Racing/Fast directory to run the script
   ii.
```
ln: failed to create symbolic link `/home/student/.debug_log': File exists
ln: failed to create symbolic link `/home/student/.debug_log': File exists
ln: failed to create symbolic link `/home/student/.debug_log': File exists
```

7. Wait ~a minute and terminate both scripts by pressing Ctrl+C in the respective terminals
   i. Waited a bit longer than a minute for better probabilistic chances of letting it run longer to have more chances of timing the loops successfully. Then ctrl+c in both terminals.
   ii.
```
Goodbye!                    ^C
^C                          student@COMP4108-a1:
```

8. Check if your exploit was successful by running rsh -l root localhost whoami. If it was successful you should not be asked for a password and will receive the reply root to the whoami command. Remember, if it did not work you may have to repeat steps 5 onward due to the probabilistic nature of race conditions (if it failed, you will receive a *permission denied* error).
   i. The exploit was successful as running the command rsh -l root localhost whoami returned root as the effective uid
   ii. Checking the .rhosts file as well to verify the successful writes with sudo cat /root/.rhosts shows multiple instances of localhost student entries.

1.
```
student@COMP4108-a1:/A1/Racing/Fast$ sudo cat /root/.rhosts
[sudo] password for student:
localhost student
localhost student
localhost student
localhost student
```

Once the exploit is successful, you can execute rsh -l root localhost bash to spawn a new superuser privileged shell with full control over the system

i. Running rsh -l root localhost bash was successful as well as the shell we spawned has root privileges when we checked with whoami and verified the localhost is indeed COMP4108-a1, and were able to see the contents of .rhosts with cat /root/.rhosts without needing the password as we have the effective uid of root. But main thing is that we can run rsh without needing to sudo or to give the password now, as we weren't permitted to do so since we didn't have our entry in the rhosts file.

ii.
```
^Cstudent@COMP4108-a1:/A1/Racing/Fast$ rsh -l root localhost bash
whoami
root
hostname
COMP4108-a1
cat /root/.rhosts
localhost student
```

8 Marks Describe the exploitation process in detail, including commands invoked and a log of you successfully gaining root privilege on the box. Include an explanation of why the attack works.

To reiterate and go through the steps of the exploitation process that's simplified as the in-depth explanation for each command is already explained above. We use vuln.sh script to run the vuln_fast program with the nice command with the -n flag and integer of 20 adjusts the program to have a lower priority(most default nice values are 0, and the highest priority nice value is -20 so we can set the nice value to be any value 39 or over to make sure we set it to the lowest priority as it gets capped to 19, we can also verify nice value using top command although the vuln_fast program cant be seen as it runs too fast, when running the scripts we can run them with nice -n 19 as the vuln.sh default nice is 0 but we are still able to get our desired exploit with just using nice for vuln_fast in the script) with process scheduling. This makes vuln_fast program have less CPU time, which makes the program run longer, which creates a bigger gap in the timing between access and open to give us better chances to exploit the race condition.

The exploit.sh script removes the debug file which is needed before making the symbolic link to the target file which is the rhosts file. We need to add localhost student as the hostname and username onto the rhosts file as the rsh command allows us to run a command as a specified user which is root on a host which is localhost. The rsh command checks the rhosts file if the the user who invoked the process is allowed to execute the command. So

using this exploit we can add ourselves onto the rhosts file to be able to execute a command as the specified user which is root with rsh without needing the password.

So us running both scripts at the same time in different terminals, they are set to both be in continuous loops, where rerunning the program in a loop allows us to keep restarting the program while the exploit.sh script is also running with a loop to keep trying to do the exploit multiple times as the chance of being able to do it once even if you run both the programs at the exact same time is very low. So once it eventually times it right and the exploit script manages to remove and create the symbolic link after the access syscall and before the open syscall, we would successfully add our payload as an entry into the target file which is the rhosts file.

The same thing as vuln_slow as vuln_fast is also a setuid program, the access syscall uses the ruid instead of the euid so for setuid programs it will check if the ruid of the user that invoked the program is permitted to access. So since we invoked the setuid program vuln_fast and the access would be successful as it's a file we have access to as a user which would be the debug file. We can then just add a symbolic link to the file were trying to write to even without having the permissions to which is the .rhosts file, since open uses the euid which is set to root as the setuid program gives us an euid of root while we run the setuid program vuln_fast, it lets us to write to .rhosts.

All the pictures of results, edited scripts, and proof gaining root privileges.

C.

P5 isolated compartments as roots files are being written over from a different user that shouldn't be allowed to do so as this principle is meant to protect against escalation of privileges which is what the attacker is doing as the end goal is to elevate to root privileges in the rsh command.

P4 Complete mediation as the debug file was checked the attacker had proper permissions, however the rhosts file wasn't checked if the attacker had proper permissions as it was a symbolic link from the debug file to the rhosts file. So not every object that was accessed was verified for proper authorization.

P13 Defense in Depth as after the debug file was checked to see if the attacker had proper permissions to access it, open syscall can use the option to not follow a symbolic link like O_NOFOLLOW to have more layers of defense to make sure its writing to the file instead of following a symbolic link to a different file that its not meant to write to. As the only defence was checking if the file was accessible but not when we write the file, as we were able to exploit the race condition between access and open syscalls.