# COMP 4108

# Assignment 1

Author: Youssuf Hichri

Student ID: 101187757

**Part A:**

1. My user id is → **1001.**
2. My primary group name is: **student,** while its corresponding GID is: **1001.**
3. It is /etc/group/
4. The Linux group files (/etc/group) permissions are:
    1. -rw-r—r—
    2. 0644
5. The root group GID is: **0.**
6. The root group GID is: 42.
7. In code file.
    1. In code file.
    2. There seems to be a total of 422 root owned directories.
    3. In code file.
    4. In code file.
8. In code file.
9. In code file.
10. Parts:
    1. In code file.
    2. Example Binary: **/usr/bin/passwd**
        1. Why the setuid bit is enabled: It allows users to execute the passwd binary. We need to have this as the passwd binary accesses/modifies **/etc/shadow** which requires elevated privileges.
        2. What the setuid bit accomplishes: The setuid bit allows a user to run executables with the permissions of the files owner. This makes it so that

any user can perform specific tasks (changing their passwords) without having to be the root user themselves (or have root privileges).

**Part B)**

1. The principles that ACL helps enforce the most are:
    1. P4 Complete Mediation: Every access attempt is mediated by the ACLs, ensuring that no access occurs without appropriate permission checks.
    2. P6 Least Privilege: ACLs enforce P6 by ensuring that permissions are granted on a need-to-know basis, limiting exposure and minimizing the risk of unauthorized access

**Part C)**

Part a:

1. To start, I needed to figure out how vuln_slow works. So I ran strace ./vuln_slow 0 hello a couple of times, and found out where the input text was being appended to, which turned out to be /home/student/.debug_log. I then checked if that file had the input text appended to it as expected, and it did indeed have it.

2. Second thing I had to do, was figure out what commands I needed to achieve the expected result.
    1. First command I needed to invoke, was rm, as I needed to delete the original file for it to be replaced.
    2. Second command was the ln command. As I needed to “recreate” that same “original” file as a “**symbolic link”** that links it to the file that I need to hack into (root_file).

3. So the following commands I needed to run where the following respectively:
    1. rm /home/student/.debug_log
    2. ln /A1/Racing/Slow/root_file /home/student/.debug_log
    3. See **Part_C_a.sh** for the entire process code.

3. **Why the Exploit Works:** The TOCTOU vulnerability is based on the difference between the time a program checks the permissions of a file (the check) and when it uses that file (the use). In this case, vuln_slow checks if it has the necessary permissions to write to a debug file, then enters a sleep state. During this time, the file is vulnerable because the program assumes the state of the file remains unchanged. By deleting the log file during the sleep and replacing it with a symbolic link to root_file, the program unknowingly writes to a file it shouldn't have access to.

   The **access()** system call is another critical aspect in this vulnerability. This function is used by the program to check whether the real user ID (UID) of the process has permission to perform certain operations on a file. vuln_slow uses **access()** to verify whether the real UID can write to the debug log file.

   The real UID is the ID of the user running the process, whereas the effective UID is the one used by the operating system to determine privileges. In many cases, the program runs with the real UID of the user but may elevate its privileges using the effective UID (e.g., through setuid).

Part b:

**Objective:** The goal is to gain root access by exploiting the vuln_fast program, which modifies the .rhosts file of the root user to allow passwordless login via the rsh and rlogin commands.

1. Add the debug file location (/home/student/.debug_log) to vuln.sh
2. Add the payload string (localhost student) to vuln.sh
3. Add the debug file location (/home/student/.debug_log) to exploit.sh
4. Add the file **you want to exploit** (/root/.rhosts) to exploit.sh
5. Run both scrips simultaneously and then terminate after a minute.
6. Check by running: rsh -l root localhost whoami

**Why does this work?**

The attack exploits a **race condition** vulnerability, which occurs when two processes (or threads) attempt to access the same resource concurrently in a way that leads to unintended behavior. In this case, vuln_fast attempts to modify a debug file while the exploit.sh script simultaneously creates a symbolic link from this file to the target .rhosts file.

The idea is to "win" the race condition by having exploit.sh create the symbolic link just before vuln_fast writes to the debug file, causing it to instead write your payload (username) into the root's .rhosts file. This allows passwordless rsh login for your user, giving you full access to the root account.

By repeatedly running both scripts, the attacker increases their chances of successfully winning the race condition and exploiting the system.

Part c:

1. **P6 - Least-Privilege**: This principle states that users or programs should only have the privileges necessary to complete their tasks and no more. In the case of this exploit, the vulnerable program allows unauthorized privilege escalation by adding an unauthorized user to the root's .rhosts file. This violates the least-privilege principle, as a regular user is gaining root-level access without needing proper authorization.
2. **P20 - Reluctant-Allocation**: This principle emphasizes only granting access or resources when absolutely necessary and in the minimum amount required. The server in this case grants access through insecure rsh and rlogin commands without thorough verification or safeguards, especially since it uses the .rhosts file. This failure to limit access properly breaks the principle of reluctant-allocation, as it allows unwarranted privileges to users who should not have them.