1 Overview

In the last lecture we introduced Mandatory Access Control (MAC) and Discretionary Access Control (DAC).

**Mandatory Access Control**- The system determines what subject owns which object.

**Discretionary Access Control**- The user determines what subject owns which object.

We also discussed the models for confidentiality and integrity.

**Bell-LaPadula Model (Confidentiality)**- Users cannot read a file with higher privileges nor can they write a file with lower privileges.

**Biba's Model (Integrity)**- Users cannot read a file with lower privileges nor can they write a file with higher privileges.

In this lecture we further discuss Access Control, as well as techniques on how to manage a process's privileges.

2 Principle of Least Privilege

The Principle of Least Privilege is to give the least amount of privileges to users and objects when executing processes.

**Problems:** As an example, lets look at login. Login needs privileges in order to read the password database and authenticate users. For this to occur, login will need the same permission level as the owner of the file, root. But what if Login has bugs? Login could delete files, and via buffer overflow, get a shell and run any program as root.

**Question:** How to prevent these kinds of bugs from compromising the system?

We can avoid this by only keeping root access for the tasks that require them, and then immediately dropping privileges when done.
2.1 Login

We looked at login as an example of the "Least Privilege Principle". Login allows users to login as another user which can lead to problems with access control. Login also needs to read the password database to authenticate the user. The passwords are stored in the database as a salted hash, where the password is concatenated with the username, then hashed. We use a salted hash because they are great against powerful machines running dictionary attacks. Moreover, to prevent any users from reading the password database, only root has access.

Question: How can Login read the password database?

Answer: By changing its Effective User ID (euid) to the same as root’s Real User ID (ruid).

2.2 Setuid

- ruid -- Real User ID - Who the user is
- euid -- Effective User ID
- suid -- Save User ID

In Linux, each process has two user id’s: a real user id, and an effective user id. There is also a feature called setuid. Setuid is one of the permission bits that flags a change for privileges. If the setuid bit is set to true, whoever executes the file will inherit the owner’s ruid.

Question: When do we use setuid or change root privilege in our login example?

Answer: By using the least privilege principle, only when a process must access the password file.

The example below shows how to prevent processes from gaining root access, and thus a shell via stack overflow. Let drop() and raise() be functions that drops and raises privileges respectively

Login:
   drop()
   parse()
   raise()
   readpassword_file()
   drop()
   compare()
   execv(...)

Rules:

- Drop privileges at the beginning of a program.
- Drop privileges before every program is called and as soon as possible.
- To prevent escalation, drop privileges in UID and recover permission only when necessary.
- Use the Least Privilege Model.
3 Safe FTP

Privileges

• Read/Write
• Access to file system

Question: How to secure FTP?
user = login()
s = stat(filename)
if(check_permissions(s, user))

Question: How to attack this method?

Answer: Race condition attack. If they have an account and make a file in /temp with a symbolic link to the users own file. Then check permissions will see that link points to the users own file. It is then possible for that user to gain access to other files by changing the destination of the symbolic link after the stat call.

Question: How to prevent this attack?

Answer: Login will spawn a new child process and setuid to the current users permissions.

4 SSH Server

Privileges

• Read/Write
• Change process ID
• Read host private key
• Bind to port 22

If we allow SSH to run as root, we put ourselves in danger. If the crypto libraries have bugs, then the user can take over it. To try and prevent this, the SSH server must start with new privileges and then it can drop them later. After it starts, it will bind the port and then it will receive incoming network connections.

The process of ssh uses a master slave model. Once a connection comes in, the program will spawn a new child process and then drop all privileges. If it goes nowhere, it dies, but the master process still has root privileges.