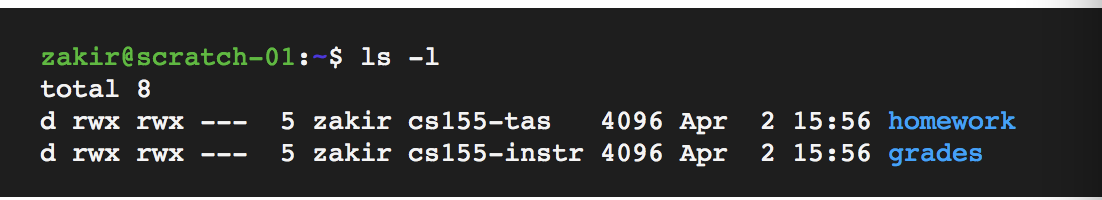
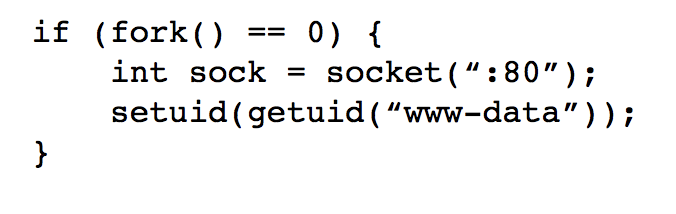
CAS CS 357

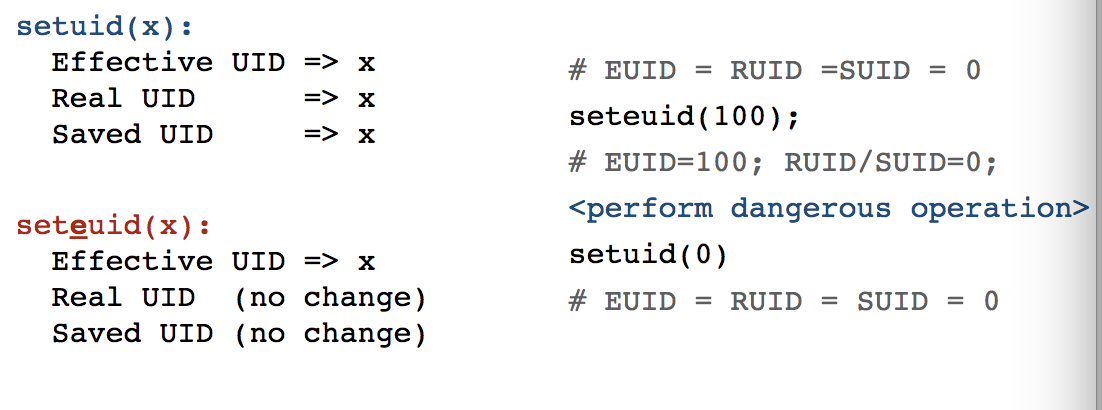
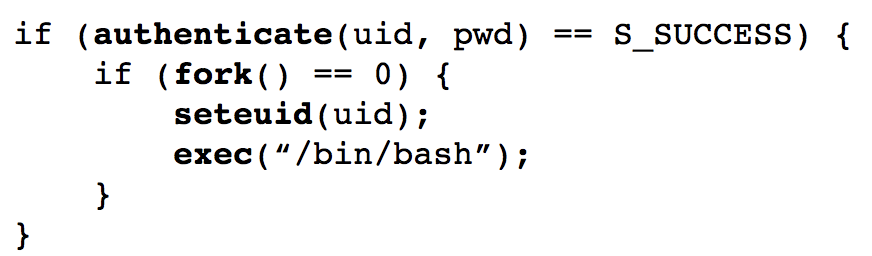
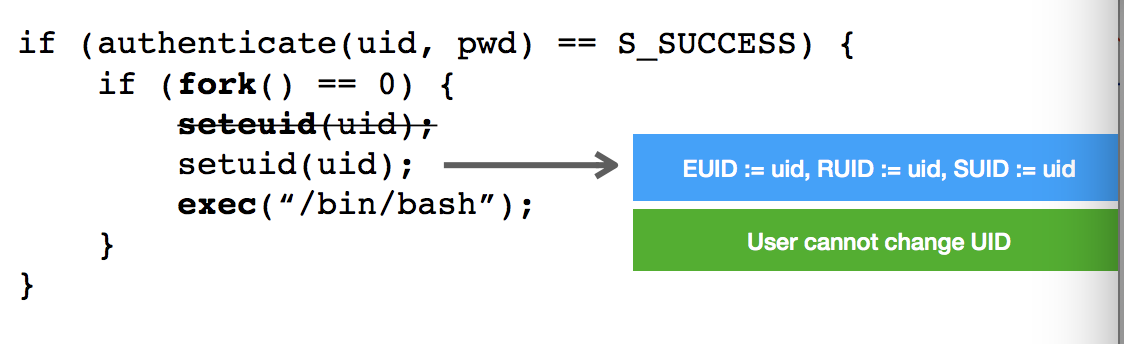
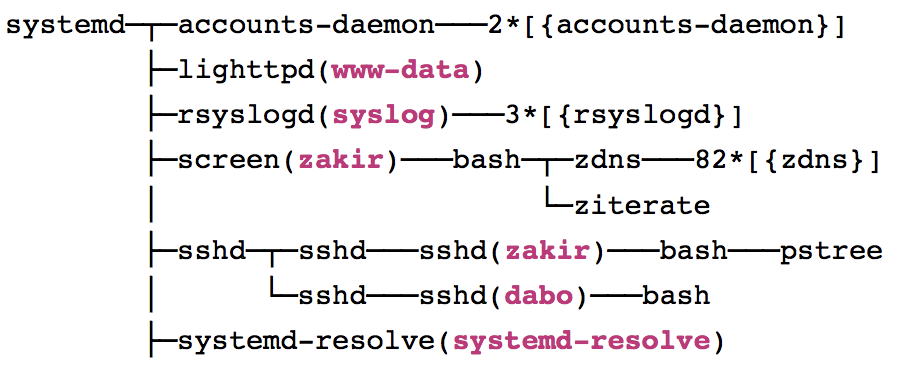
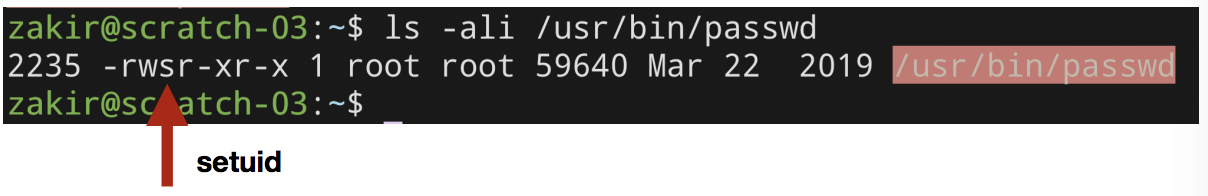
InClass Note 23

1. UNIX Processes
2. Processes are isolated: they cannot access each other’s memory
3. Processes run as a specific user

* When you run a process, it runs with your UID’s permissions
* Process can access any files that the UID has access to and processes run by the same UID have the same permissions

1. Processes started by root can reduce their privileges by changing their UID to a less privileged UID
2. Process example
3. When you run a command, it runs with all of your privileges because your shell runs as your user account and forks to start the command
4. When any process forks, it inherits its parent UID
5. 
6. Process User IDs
7. Every process has three different User IDs
8. Effective User ID (EUID) – determines the permissions for process
9. Real User ID (RUID) – determines the user that started the process
10. Saved User ID (SUID) – EUID prior to change
11. Changing User IDs
12. Root can change EUID/RUID/SUID to arbitrary values
13. Unprivileged users can change EUID to only RUID or SUID
14. Reducing Privilege through setuid
15. Apache Web Server must start as root because only root can create a socket that listens on port 80 (a privileged port)
16. Without any privilege reduction, any Apache bug would result in the attacker having unrestricted server access
17. Instead, Apache creates children using the following scheme:



1. Temporarily changing UID
2. Remember: unprivileged users can change EUID back to the RUID or SUID
3. 
4. SSH Example
5. Suppose SSH runs as root and runs the following code:
6. 
7. Attack: user can call setuid(0) to become root because SUID == 0
8. 
9. UNIX process Tree
10. Main system process starts as root and forks
11. Output of pstree –u
12. 
13. SETUID Bit – Elevating Privileges
14. The passwd utility allows you to change your password by updating password /etc/shadow – a file that only root can read/write
15. Normally, this would not be possible. Remember that executables run with the privilege of the executing user – and your account can’t access
16. UNIX allows you to set EUID of an executable to be the file owner rather than the executing user
17. SETUID on passwd
18. 
19. Setuid vs Setuid
20. Setuid syscall (in code): Allows caller to change User IDs of the process
21. Setuid bit on Executable: Execution runs as owner and group of executable rather than the calling user
22. Becoming Root User
23. System configuration files are owned by root
24. Important system processes run as root
25. Sometimes, you as a user, need to “become” root to fix problems
26. Sudo: run a single command as root (requires you to be blessed)
27. Su: allows you to become root by knowing its password
28. Sudo su: become root without their password
29. Worst privilege separation ever?
30. Traditional UNIX distinguished between privileged processes (EUID ==0) and unprivileged processes (EUID != 0)
31. Privileged processes bypass all kernel permission checks, while unprivileged processes are subject to full permission checking
32. Lots of utilities – like ping – depend on setuid
33. Exceptionally dangerous – a bug in many utilities can lead to compromise
34. Linux Capabilities
35. Capabilities segment root powers into components, such that if a program that has one or more capabilities is compromised, damage is limited
36. Overview of UNIX Security Mechanisms
37. Pros
38. simple model provides protection for most situations
39. flexible enough to make most simple systems possible in practice
40. Cons
41. ACLs are coarse grained – doesn’t account for enterprise complexity
42. ACLs don’t handle different applications within a single user account
43. Nearly all system operations require root access – people are sloppy