Industrial Workstation Overview of Installation

------------Part 4 Rootly Powers-----------

\*Access control and Rootly Powers

--In the early days, UNIX did not have a single point access control system.

--Objects (Files and Processes) have owners who have privileges over the objects.

--You own new objects that you create

--The special user known as root can have ownership of any file

--Only root can perform security sensitive tasks on the system.

--There is no single black box of control to designate power over the system as the code that implements these features is scattered across the computer.

\*Filesystem access control

--Every file has both an owner and a group sometimes referred to as the “group owner”.

--The owner can set permissions on that file

--Both the kernel and the filesystem track owners and groups as numbers rather than as text names in the most basic form of UIDs. They are mapped to users in the /etc/passwd file. GIDs are mapped to group names in the /etc/group file. The text names that correspond to the UIDs and GIDs are defined only for the convenience for the convenience of the human users.

\*Process Ownership

--The owner of a process can send the process signals and can also reduce the process’s scheduling priority. Processes have multiple identities, real, effective, and saved for UIDs and GIDs as well as under Linux a filesystem UID that is only used to determine file access permissions. Broadly speaking, real numbers are used for accounting and effective numbers are used for the determination of access permissions. The real and effective are normally the same.

--Saved IDs have no direct effect; they allow programs to park an inactive ID for later use, facilitating the parsimonious use of enhanced privileges.

--The filesystem UI is generally explained as an implementation detail of NFS.

\*The Root Account

--Root is the super user account with the username “root” and with the UID of 0. Nothing prevents you from changing the username on this account or from creating additional accounts with UIDs of 0; however, these are bad ideas.

--Traditional design allows the superuser to perform any valid operation on any file or process.

--Changing the root directory of a process with chroot

--Creating device files

--Setting the system clock

--Raising resource usage limits and process priorities

--Setting the system’s hostname

--Configuring network interfaces

--Opening privileged network ports

--Shutting down the system

--An example of super user powers is the ability of a process owned by root to change its UID and GID. The login program and its window system equivalents are a case in point; the process that prompts you for your password when you log in to the system initially runs as root.

\*Setuid and setgid execution

--UNIX access control is complemented by identity substitution that’s implemented by the kernel and file system collaboration. This allows specially prepared executable files to run with elevated permissions, usually those of root.

--When the kernel runs an executable file that has its UID or GID set, the effective ID is changed to that of its new permissions.

------------Part 4.2 Modern Access Control-------------

\*Access Control Systems

--Shortcomings of Linux standard access control

--The Root account represents a potential single point of failure with massive damage if done so.

--The only way to subdivide the special privileges of the root account is by writing setuid programs. With the internet’s steady stream of security updates, it’s difficult to write truly secure software.

--The security model isn’t strong enough for use on a network. No computer to which an unprivileged user has physical access can be trusted to accurately represent the ownerships of the processes its running.

--Many high-security environments enforce conventions that simply can’t be implemented with traditional UNIX security. For example, the US government standards require computer system to forbid privileged users from publishing high security documents at a lower security level.

--Because much access control related rules are embedded in the code of individual commands and daemons, you cannot redefine the system’s behavior without modifying the source code and recompiling

--Because of these shortcomings, Linux systems have adjustments such as PAM.

\*Role-based access control

--Role based access control known as RBAC is a theoretical model made to implement a layer of indirection to access control calculations. Instead of permissions being assigned directly to users, they are assigned to intermediate constructs known as roles. Roles are more useful than groups because the systems that implement them allow them to be used outside the context of the filesystem.

--SELinux: Security Enhanced Linux is an NSA Project that requires Mandatory Access Control where access control is assigned by system administrators

--PAM Pluggable Authentication Modules is an authentication technology rather than an access control technology. It answers the question: How do I know this is really user X?

--Kerberos: Like PAM, Kerberos deals with authentication rather than access control. Where Pam is an authentication framework, Kerberos is the actual authentication method.

--Access Control Lists or ACLs are part of the filesystem implementation and show who and what each person has access to.

----------Part 4.3 Real World Access Control-----------

\*Logging into the Root Account

--Logging directly into the root account is a bad idea due to leaving no record of what operations were performed as root. It also leaves no record of who was doing the work.

--It is possible to disable root login on terminals through window system and across the network.

\*SU: Substitute user identity

--A better way to access the root account is to use the su command. If invoked without arguments, su prompts for the root password and then starts up a root shell. Root privileges remain in effect until you terminate the shell. Sui doesn’t record the commands executed as root but it does create a log entry that states who became root and when.

--When signed in as root, calling the su command into someone else’s account does not need a password.

--Get in the habit of typing the whole path name to us /bin/su or /usr/bin/su as programs can be called su in your search path and try to harvest the root password

\*sudo: limited su

--sudo consults the file/etc/sudoer which lists the people who are authorized to use sudo and the commands they are allowed to run on each host. If successful, sudo prompts for the user’s password to get access

--It is common to use aliases in the sudoer file to keep things more organized

\*\*\*\*\*Review Structure of sudoer file\*\*\*\*\*

--The built in alias ALL allows for any command to be run or for any user to run a command

--Any attempt to allow “all commands except” is doomed to failure.

--To modify /etc/sudoers file, you can use the visudo command, which checks to be sure no one else is editing the file, invokes an editor on it, and then verifies the syntax of the edited file before installing it. This last part is important as changing the sudoers file in an invalid way may prevent you from sudoing again to fix it.

--Sudo advantages

--Accountability is improved with command logging

--Operators can do chores without unlimited access

--The real root password is secure by only a handful of people

--Its faster to use sudo that to use su or log in as root

--Privileges can be revoked without the need to change the root password

--A canonical list of all users with root privileges is maintained

--Less chance of a root shell being left unattended

--A single file can be used to control access for an entire network

--Sudo Disadvantages

--Any breach in the security of a sudoers personal account can be equivalent to breaching the root account itself.

--sudo’s command logging can be subverted by tricks such as shell escapes from within an allowed program or by sudo sh and sudo su if you allow them

\*Password Vaults and Password Escrow

--Basic idea of people and administrators must keep track of their passwords