Infrastructure Security Using Linux

Computer science / Cybersecurity

Software, Shell and User Management

6 – 7 - 8

Software

Software Installation and Management

Operating System Installation

- **Linux distributions** and **FreeBSD** have straightforward procedures for basic installation.
- **For physical machines**, you can boot from a CD, DVD, or USB drive.
- **For virtual machines**, you can boot from an ISO file.
- Installing the base OS from local media is fairly trivial thanks to the GUI apps that shepherd you through the process.

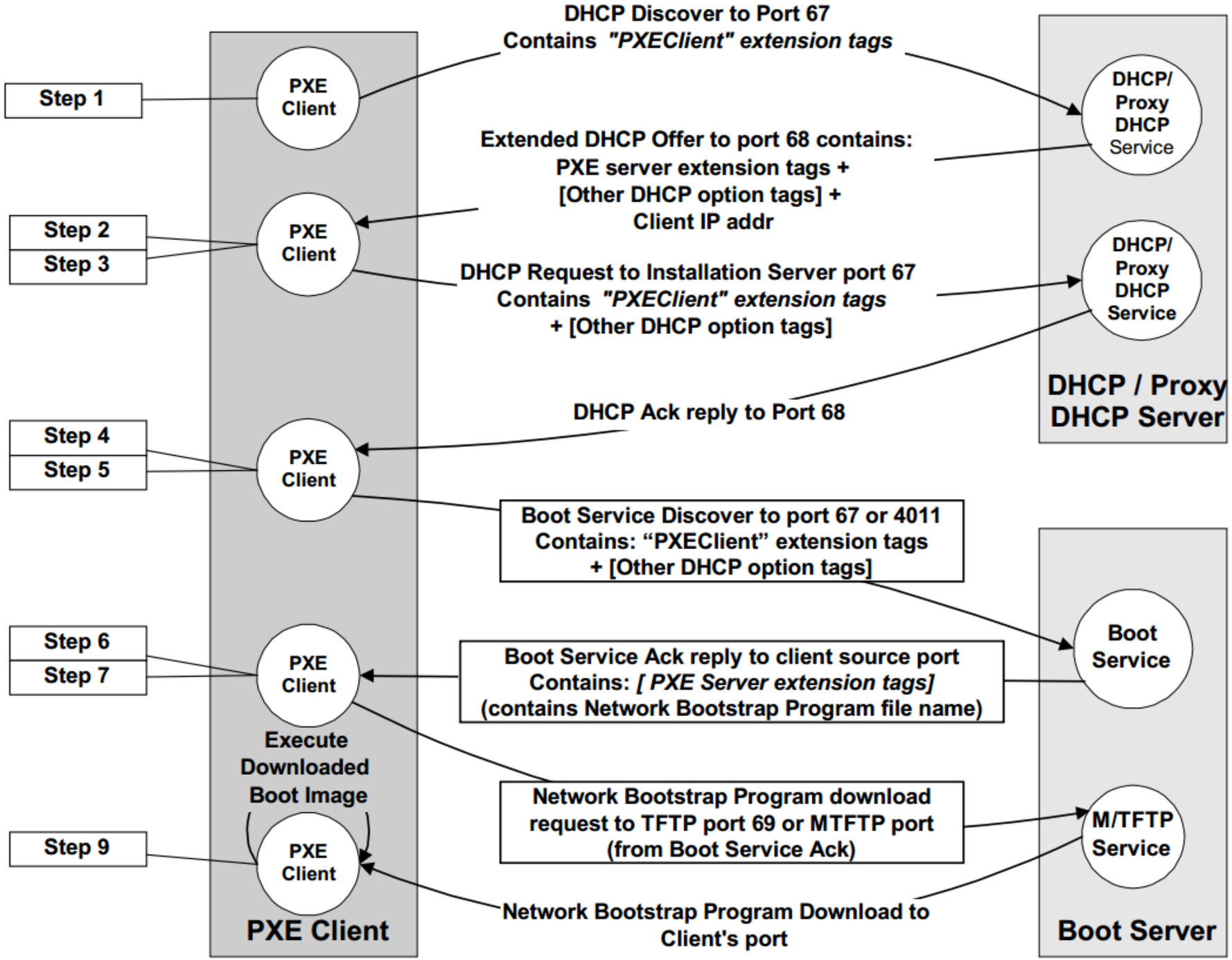
Installation from the network

- **Challenges with Local Media:**
  - Inefficient for multiple systems.
  - Repetitive, time-consuming, and prone to error.
- **Network-Based Approach:**
  - Common in data centers and cloud environments.
  - Systems boot using **DHCP** and **TFTP** without physical media.
  - Installation files retrieved via **HTTP**, **FTP**, or **NFS**.
  - Files may reside on the same server or a separate server.

Installation from the network (**PXE**)

- **Purpose:**
  - Enables hands-free installations.
- **Key Features:**
  - Standardized by Intel.
  - Allows systems to boot from the network interface.
- **How It Works:**
  - PXE is a minimal OS stored in the network card’s ROM.
  - Provides a standard API for BIOS to access network functions.
- **Advantages:**
  - One boot loader can **netboot** any PXE-enabled system.
  - No need for unique drivers for different network cards.

Preboot eXecution Environment (PXE)



Linux Package Management Systems

- **Linux Package Formats:**
  - **RPM:** Used by Red Hat, CentOS, SUSE, Amazon Linux, etc.
  - **.deb:** Used by Debian and Ubuntu.
  - Both formats are functionally similar.
- **Package Management Tools:**
  - **rpm** for RPM-based systems.
  - **dpkg** for .deb-based systems.
  - Both support full configuration management from install to query.

Linux Package Management Systems

- **Higher-Level Package Managers:**
  - Handle package downloads, dependency resolution, and system-wide upgrades.
- **yum (Yellowdog Updater, Modified):**
  - Designed for **RPM**-based systems.
- **APT (Advanced Package Tool):**
  - Originated with **.deb** packages.
  - Now supports both **.deb** and **RPM** formats.

High-Level Package Management

- **High-Level Package Management Tools**
  - Used most frequently
  - Install, remove, and upgrade packages
  - Search for packages
  - List installed packages
- **Package Repositories**
  - Maintained by Linux distributors
  - Integrated with package management systems
  - Default configuration points to official web/FTP servers

High-Level Package Management **(cont'd)**

- **Release:** A self-consistent snapshot of the package universe.
- **Component:** A subset of the software within a release.
- **Architecture:**
  - Represents a class of hardware.
  - Machines within the same architecture can run the same binaries.
  - Example: i386 architecture in the Fedora 20 release.

APT: Advanced Package Tool

- **APT Overview**
  - A set of tools for managing Debian packages
  - Most widely used on Debian-based systems
  - Complete package management system
- **Key Tools in APT:**
  - **apt-get:** Command-line package management (install, remove, upgrade)
  - **apt-cache:** Searches and queries the package cache
  - **apt-file:** Finds files within packages
  - **apt-show-versions:** Displays package versions
  - **aptitude:** High-level interface with additional capabilities beyond apt-get
  - **apt-mirror:** Mirrors a package repository
- **Note:** On Ubuntu, ignore the legacy frontend tool dselect.

yum: Yellowdog Updater, Modified

- **YUM (Yellowdog Updater, Modified)**
  - Package manager for RPM-compatible Linux systems
  - High-level tool for package management
  - Automatically resolves dependencies when installing, updating, or removing packages
  - Works with installed repositories and supports individual package operations via command-line
- **Software Localization and Configuration**
  - Adapting systems to local/cloud environments is crucial for effective administration
  - Structured, reproducible localization helps prevent fragile, hard-to-recover “snowflake” systems
  - Improves reliability and simplifies recovery after incidents

Scripting and Shell

Scripting philosophy

- **Write Microscripts**
  - Most admins maintain short personal scripts, also known as scriptlets, in their **~/bin** directories.
  - These quick, simple scripts help address daily pain points efficiently.
- **Alternative: Shell Functions**
  - Instead of standalone files, you can define functions directly in your shell configuration files (like **.bash_profile** or **.bashrc**).

Learn a few tools well

- **Skim and Learn Quickly**
  - Focus on acquiring foundational knowledge of tools and improving documentation-skimming skills.
  - Adopt a strategic approach: quickly identify relevant sections, key commands, and core principles.
- **Master Core Tools**
  - **Shell:** Bash (know it thoroughly)
  - **Text Editor:** Vim (get comfortable and proficient)
  - **Scripting Language:** Python (understand syntax, libraries, and usage patterns)
- **Read, Study, Repeat**
  - Start by reading official manuals cover-to-cover.
  - Keep up with relevant books and trusted blogs.
  - Leverage the shared experiences and best practices of the broader tech community.

Pick the right scripting language

- **Shell Scripts**
  - Historically defined by the ***sh*** shell
  - Ideal for lightweight tasks like automating command sequences or combining filters to process data
- **Modern Alternatives, Python and Ruby:**
  - General-purpose programming languages
  - Built with decades of language design improvements
  - Offer powerful text processing capabilities beyond what ***sh*** can achieve

Follow best practices

1. When run with inappropriate arguments, scripts should print a usage message and exit. For extra credit, implement a **--help (-h)** option that prints a more detailed message.
2. Validate inputs and sanity-check derived values.
3. Return a meaningful exit code: *zero* for success, *nonzero* for failure.
4. Use appropriate naming conventions for variables, scripts, and routines.
5. Assign meaningful names to variables that reflect their purpose.
6. Start every script with a comment block that includes the script's name, purpose, author, and date of creation.

Follow best practices

7. Use a consistent coding style.
8. Don't clutter your scripts with useless comments; assume intelligence and language fluency on the part of your readers.
9. It's OK to run scripts as root, but avoid making them setuid root. Use sudo instead.
10. Don't script what you don't understand.
11. With **bash**, use **-x** to echo commands before they are executed, and **-n** to check syntax
12. Remember in Python, you are in debug mode unless you explicitly turn it off with a **-O** argument on the command line.

Tom Christiansen's five golden rules:

- Error messages should go to STDERR, not STDOUT.
- Include the name of the program that’s issuing the error.
- State what function or operation failed.
- If a system call fails, include the perror string.
- Exit with some code other than 0

Shell Basics

- **UNIX Shells**
  - UNIX has long provided multiple shell options. The Bourne shell (**sh**) has been a universal standard on UNIX and Linux systems.
- **Modern Versions of sh**
  - The original Bourne shell's source code remained under AT&T licensing.
  - Today, **sh** typically appears as:
    - Almquist shell (ash, dash, or **sh**)
    - Bourne-again shell (**bash**)
- **Almquist Shell (ash, dash)**
  - A minimal, efficient reimplementation of the Bourne shell. Lacks modern usability; primarily used to run **sh** scripts.
- **Bourne-again Shell (bash)**
  - Designed for interactive use. Incorporates features introduced by other shells, making it user-friendly.

Command line editing mode

- **bash** has two command-line editing modes. By default, it uses **emacs** mode, but you can switch to **vi** mode by running **set -o vi**.
- In **emacs** mode, you can use the following key combinations:
  - **Ctrl-A**: Move to the beginning of the line.
  - **Ctrl-E**: Move to the end of the line.
  - **Ctrl-U**: Delete from the cursor to the beginning of the line.
  - **Ctrl-K**: Delete from the cursor to the end of the line.
  - **Ctrl-W**: Delete from the cursor to the start of the word.
  - **Ctrl-Y**: Paste the last deleted text.
  - **Ctrl-L**: Clear the screen.
  - **Ctrl-R**: Search the history backwards.
  - **Ctrl-S**: Search the history forwards.
  - **Ctrl-C**: Cancel the command.
  - **Ctrl-D**: Log out of the current session.
  - **Ctrl-Z**: Suspend the current command.
  - **Ctrl-Alt-T**: Open a new terminal window.
  - **Ctrl-Alt-L**: Lock the screen.
  - **Ctrl-Alt-Delete**: Log out.

In **vi** mode, you can learn it by your self, not required for this course

Pipes and redirection

- Every process has three communication channels: **stdin**, **stdout**, and **stderr**. Each of these channels is represented by a file descriptor: **0** for **stdin**, **1** for **stdout**, and **2** for **stderr**.
  - **>**: Redirect **stdout** to a file, overwriting the file if it exists.
    - For example, **ls > file.txt** writes the output of **ls** to **file.txt**.
  - **>>**: Redirect **stdout** to a file, appending to the file if it exists.
    - For example, **ls >> file.txt** appends the output of **ls** to **file.txt**.
  - **<**: Redirect **stdin** from a file.
    - For example, **cat < file.txt** reads the contents of **file.txt** and writes them to **stdout**.
  - **2>**: Redirect **stderr** to a file, overwriting the file if it exists.
    - For example, **apt update 2> file.txt** writes the error output of **apt update** to **file.txt**.

Pipes and redirection

- **2>>**: Redirect **stderr** to a file, appending to the file if it exists.
  - For example, **apt update 2>> file.txt** appends the error output of **apt update** to **file.txt**.
- **&>**: Redirect **stdout** and **stderr** to a file, overwriting the file if it exists.
  - For example, **apt update &> file.txt** writes the output of **apt update** to **file.txt**.
- **&>>**: Redirect **stdout** and **stderr** to a file, appending to the file if it exists.
  - For example, **apt update &>> file.txt** appends the output of **apt update** to **file.txt**.
- **|**: Connect **stdout** of one command to **stdin** of another command.
  - For example, **ls | grep file** searches the output of **ls** for the word **file**.

- The **find** command illustrates why you might want separate handling for STDOUT and STDERR because it tends to produce output on both channels, especially when run as an unprivileged user.

```
$ find / -name my_unbelievably_important_file 2> /dev/null
# output
/home/abdou/unbelivable_directory/my_unbelievably_important_file
```

Variables and quoting

- **Variable Assignment**
  - Assign a value using the **=** operator.
  - Variable names are case-sensitive and consist only of alphanumeric characters and underscores.

```
$ echo "Today is `date`"
# output
Today is Tue  12 Mar 2024 17:03:34 CET
```

- **Using Variables**
  - Enclose variable names in curly braces to distinguish them: ***${var}***.

- **Quotes in the Shell**
  - ***Double quotes***: Enclose text to treat as a single argument.
  - ***Single quotes***: Enclose text to treat as literal characters.

```
$ echo "Hello, $USER"
# output
Hello, abcded
$ echo 'Hello, $USER'
# output
Hello, $USER
```

Environment variables

- Automatically imported into ***sh***'s variable namespace
- Set and read with standard variable syntax
- Use ***export*** to make a variable available to child processes
- **Setting Variables at Login**
  - Add commands for environment variables to ***~/.profile*** or ***~/.bash_profile***
- **cut Command**
  - Extracts sections from each line of input
  - Example: Using ***cut*** to select fields from a file or output

```
$ echo "Hello, world" | cut -d " " -f 1
# output
Hello,
```

Common commands

- **sort** is used to sort lines of text. For example:

```
$ echo -e "b\na\nc" | sort -r
# output
c
b
a
```

```
$ du -sh /usr/* | sort -rh
# output
1.5G /usr/share
1.1G /usr/lib
890M /usr/bin
...
```

Option	Description
-r	Reverse the result of comparisons.
-n	Compare according to string numerical value.
-f	Fold lower case to upper case characters.
-u	Output only the first of an equal run.
-t	Specify a field separator.
-k	Sort by a key.
-b	Ignore leading blanks.
-h	Sort human readable numbers.

SH Scripting

- **sh** is great for simple scripts that automate things you'd otherwise be typing on the command line.
- To prepare a script for execution, you need to set the execute bit with the **chmod** command. For example:

```
$ chmod +x helloworld

$ ./helloworld
# output
Hello, world!
```

Input and output

- You can use the **read** command to read a line of input from the user.

  For example:

```
#!/bin/sh

echo "What is your name?"
read name

if [ -z "$name" ]; then
  echo "You didn't tell me your name."
else
  echo "Hello, $name!"
fi
```

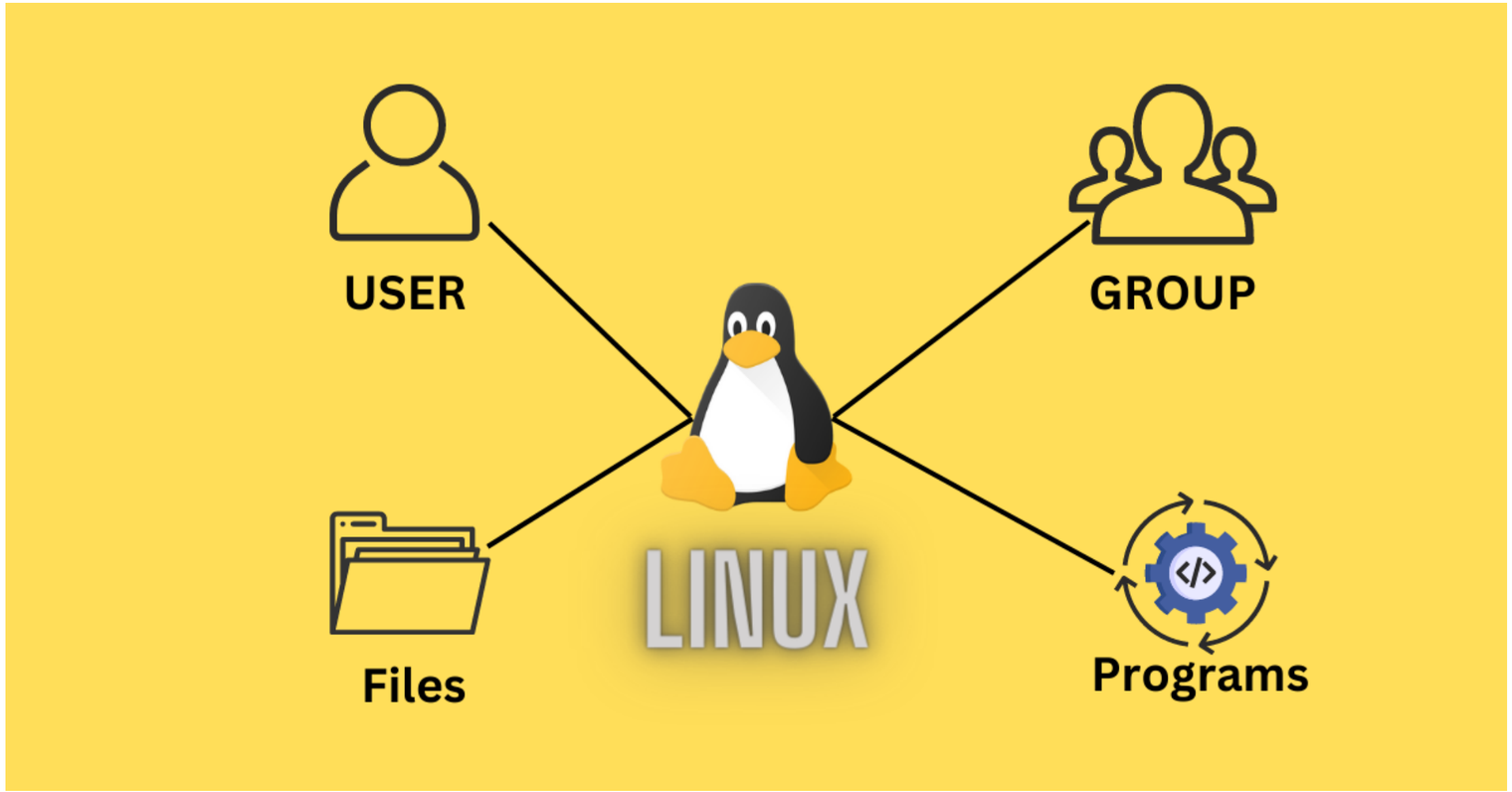
..... in the LAB

Revision control with Git

- **Revision Control Systems**
  - Help track, archive, and manage access to multiple file revisions.
  - Allow you to revert to a known-good state if changes cause problems.
- **Git: The Leading Tool**
  - The most widely used revision control system, created by ***Linus Torvalds***.
- Distributed system:
  - Each working directory is a fully functional repository.
  - Contains complete history and full version tracking.
  - Does not require network access or a central server.

User Management

User Management



Modern computing environments span physical hardware, cloud systems, and virtual hosts. Along with the flexibility of this hybrid infrastructure comes an increasing need for centralized and structured account management.

Account mechanics

- **Users and UID Numbers**
  - Each user is represented by a unique unsigned 32-bit integer known as the **user ID (UID)**.
  - Most account management revolves around this numeric UID.
- **Mapping UIDs to User Information**
  - The system provides an API to translate UIDs into user details.
  - For instance, ***getpwuid()*** takes a UID and returns a **struct passwd** with the user's name, home directory, and more.
- **Higher-Level Process Support**
  - The ***nsswitch.conf*** file enables seamless account management by defining how the system queries user data.
  - Example: Logging in as "abdou" involves a ***getpwnam()*** lookup for "abdou," followed by password validation against the returned encrypted record—regardless of the data source.

Account - The /etc/passwd file

- **/etc/passwd and User Information**
  - Lists recognized system users.
  - Historically included encrypted passwords, but these have since moved to ***shadow*** files for security.
- **Password Storage Changes**
  - ***/etc/passwd*** passwords were originally world-readable, making them vulnerable.
  - Encrypted passwords are now stored in ***shadow*** files (e.g., ***shadow*** on Linux, ***master.passwd*** on FreeBSD) with restricted access.
  - The password field in ***passwd*** contains an ***x*** (or ** on FreeBSD) to indicate that the password is located in the shadow file.

Account - The /etc/passwd file

- **Purpose and Structure of /etc/passwd**
  - Used during login to retrieve a user's UID, home directory, and shell.
  - Each line includes seven colon-separated fields:
    - **Login name**
    - **Password placeholder (x or *)**
    - **UID (User ID)**
    - **GID (Group ID)**
    - **GECOS (optional metadata like full name)**
    - **Home directory**
    - **Login shell**
- **Network Directory Services and Special Entries**
  - In systems using LDAP or other directory services, entries starting with **+** or **-** may appear in the ***passwd*** file.

Account

- **The Login Name**
  - Must be unique and adhere to any character set restrictions.
  - UNIX and Linux systems generally limit login names to ***32 characters***.
- **Password Field (Encrypted Password)**
  - Originally used ***DES*** encryption, now considered weak.
  - Transitioned to ***MD5*** and then to ***salted SHA-512*** hashes for stronger security.
  - When manually editing ***passwd***, add ***x*** (Linux) or ** (FreeBSD) to the password field as a placeholder.
  - This placeholder prevents unauthorized access until a proper password is set.

```
newuser:x:1001:1001:New User:/home/newuser:/bin/bash
```

UID (user ID) number

- **Root and Pseudo-Users**
  - ***Root*** always has UID ***0***.
  - **Pseudo-users** (e.g., ***bin***, ***daemon***) own system commands or config files.
    - These accounts are typically placed at the start of ***passwd*** with low UIDs.
    - Fake shells (e.g., ***sbin/nologin***, ***bin/false***) prevent logins.
- **Creating Users and UID Guidelines**
  - **useradd**: Automatically assigns the next available UID. Use ***-u*** to specify a custom UID. Regular users often start at UID ***1000***.
  - **Avoid recycling UIDs**: Reusing UIDs can cause security problems if old UIDs persist in files or directories.
- **Maintaining Unique UIDs**
  - **Centralized directory services** (e.g., ***LDAP***) help ensure unique UIDs across multiple systems.
  - **UID ranges**: Assigning each group its own range helps keep UIDs separate. However, this doesn’t address unique login names.

Default GID

- **GID Basics**
  - A **GID (Group ID)** is a 32-bit number.
  - **GID 0** is typically reserved for a root-level group, often called **root**, **system**, or **wheel**.
- **Historical and Modern Uses of Groups**
  - **Historical Purpose**:
    - Used for tracking and billing resources such as CPU time, login time, and disk usage.
  - **Modern Purpose**:
    - Primarily used to manage and control access to files and directories.

GECOS field

- **GECOS Field**
  - Historically associated with early UNIX systems using General Electric Comprehensive Operating Systems.
  - Contains optional user information in ***comma-separated*** entries.
- **Conventional GECOS Order**
  1. ***Full name***
  2. ***Office number and building***
  3. ***Office telephone extension***
  4. ***Home phone number***

Home directory

- **Home Directory Basics**
  - The home directory is the user's initial location upon login.
  - It's typically owned by the user and not world-writable.
  - Default storage for user files, shell customizations, SSH keys, and other account-specific settings.
- **Network-Mounted Home Directories**
  - If the home directory is mounted over a network filesystem:
    - It may be unavailable if the network or server experiences problems.
    - On login, the system may print a "no home directory" message and default to the ***root directory (/).***
- **Display Manager Logins**
  - Unlike console or terminal logins, display managers (e.g., ***xdm***, ***gdm***, ***kdm***) don't display "no home directory" messages.
  - Instead, they usually log you out immediately if they cannot access the home directory (e.g., ***~/.gnome***).

Login shell

- Typically a command interpreter, but can be any program.
- Default options:
  - ***sh*** (Bourne-shell compatible) on FreeBSD.
  - ***bash*** (GNU Bourne Again Shell) on Linux.

The /etc/shadow file

- **Shadow Password File (/etc/shadow)**
  - Readable only by the superuser.
  - Stores encrypted passwords and other security information.
  - Not a superset of ***passwd***, and ***passwd*** is not derived from it.
- **Fields in /etc/shadow**
  1. ***Login name***: Matches ***passwd*** entry for the user.
  2. ***Encrypted password***: Stored securely.
  3. ***Date of last password change***: Recorded as days since epoch.
  4. ***Minimum password age***: Number of days before password can be changed.
  5. ***Maximum password age***: Number of days until password expires.
  6. ***Days before expiration warning***: Advance warning for upcoming expiration.
  7. ***Days after expiration to disable***: Grace period after expiration.
  8. ***Account disable date***: Days since epoch when account was disabled.
  9. ***Reserved field***: Reserved for future use.

The /etc/group file

- **/etc/group File**
  - Lists UNIX group names and members.
  - Each line represents a group with four colon-separated fields:
    - ***Group name***
    - ***Password placeholder (x or *)***: Password stored in ***gshadow***.
    - ***GID*** (Group ID)
    - ***Members***: Comma-separated, no spaces.
- **/etc/gshadow File**
  - Superuser-readable only.
  - Stores encrypted group account passwords.

- **Grouping Users**
  - Old convention: Add users to general category groups (e.g., ***students***, ***finance***).
    - Risk: Users might read each other's files due to poor permission settings.
- **New convention:**
  - Each user gets a personal group (named after the user, containing only that user).
  - Easier management if personal groups' ***GIDs*** match the users' ***UIDs***.

Scrips for adding users:

Table 8.3: Commands and configuration files for user management

System	Commands	Configuration files
All Linux	useradd, usermod, userdel	/etc/login.defs /etc/default/useradd
Debian/Ubuntu[a]	adduser, deluser	/etc/adduser.conf /etc/deluser.conf
FreeBSD	adduser, rmuser	/etc/login.conf

a. This suite wraps the standard Linux version and includes a few more features.

Centralized account management

- **Centralized Account Management**
  - Essential for medium-to-large organizations.
  - Provides a single, consistent login name, ***UID***, and password across the entire site.
- **Implementation**
  - Centralized management often relies on ***LDAP (Lightweight Directory Access Protocol)***.
  - Common examples include Microsoft's ***Active Directory*** and other LDAP-based solutions.

LDAP and Active Directory

- **LDAP**
  - Generalized, database-like repository for various types of data.
  - Uses a hierarchical client/server model supporting multiple servers and clients.
  - Can enforce unique ***UIDs*** and ***GIDs*** across systems.
- **Active Directory Integration**
  - ***Active Directory*** uses ***LDAP*** and ***Kerberos*** to manage user information and other data.
  - Prefers a central role if integrated with UNIX or Linux LDAP repositories.
  - For mixed environments (Windows, UNIX, Linux), it's often simplest to let ***Active Directory*** take the lead, using UNIX LDAP databases as secondary servers.

Identity management systems

- **Identity Management (IAM)**
  - Identifies and authenticates system users.
  - Grants privileges based on verified identities.
  - Standardized by organizations such as:
    - ***World Wide Web Consortium (W3C)***
    - ***The Open Group***

Key Evaluation Areas for Identity Management Systems

- **Oversight and Management**:
  - Secure web interface for internal and external access.
  - Interface for hiring managers to request accounts based on role.
  - Integration with personnel databases to revoke access automatically when employees leave.

- **Ease of Use Features**
  - Allow users to change or reset their passwords.
    - Enforce strong password rules.
  - Provide a single operation for global password changes.

Key Evaluation Areas for Identity Management Systems

- **Account Management Features**
  - Generate globally unique user IDs.
  - Centralized account creation, modification, and deletion across various systems.
  - Workflow engine for tiered approvals before assigning privileges.
  - Easily display users with specific privileges or view privileges assigned to a particular user.
  - Implement role-based access control:
    - Provision user accounts by role.
    - Allow and approve exceptions through a defined workflow.
  - Log all administrative changes and actions.
    - Generate and customize reports (e.g., by user, by date).