Day 31

Exploitation Analyst

User Management and PAM:

Task/Topic	Description	Command/Config File
1. Check if a Service Uses PAM	Confirm if a service (like SSH, sudo, su) uses PAM for authentication.	Check /etc/pam.d/ for service-specific files.
2. Common-auth File Explained	Shared PAM config used by many services for basic auth steps like password validation.	/etc/pam.d/common-auth → used by login, sudo...
3. Restrict Old Password Reuse	Prevent users from reusing previous passwords.	Use pam_unix.so remember=5 in /etc/pam.d/common-password
4. Set Password Expiration	Force users to change passwords periodically.	Use chage command or configure /etc/login.defs
5. Enforce Strong Passwords	Enforce complexity: length, character types, no dictionary words.	Use pam_pwquality.so or libpam-cracklib.so
6. Sudo Access (Restrict & Secure)	Limit sudo usage to trusted users/groups. Log sudo usage and alert on abnormal access.	Configure /etc/sudoers or use visudo
7. Disable Root Login	Prevent direct root login (especially via SSH) to minimize attack surface.	Set PermitRootLogin no in /etc/ssh/sshd_config
8. /etc/securetty File	Controls **where root can log in** (e.g., only physical terminals).	Remove all entries to disable remote root access
9. Limit cron/at Job Access	Restrict who can schedule jobs using cron or at.	Create /etc/cron.allow and /etc/at.allow
10. Lock User After Failed Attempts	Temporarily lock accounts after failed login attempts to prevent brute force.	Use pam_faillock.so or pam_tally2.so
11. Enable 2FA Authentication	Add multi-factor authentication for SSH or local login.	Use pam_google_authenticator.so
12. Log and Monitor Authentication	Monitor login attempts, failures, PAM module behavior, and sudo access.	Check /var/log/auth.log or journalctl (journalctl -xe)

How to Check If a Service Uses PAM

Why checking this is important?

Checking if a service uses PAM helps you identify which services are under centralized authentication control, so you can apply, audit, and enforce critical security po licies and detect any tampering.

1. Centralized Authentication Control
   PAM controls how authentication works for key services like ssh, sudo, login, su. If a service uses PAM, you can apply consistent policies (like 2FA, lockouts, password rules) through PAM config files.
2. Prevent Privilege Escalation
   Misconfigured PAM modules can let attackers bypass authentication, reuse old passwords, or even gain root access. By knowing which services use PAM, you can audit and harden them.
3. Detect Malicious Backdoors
   An attacker with root access could insert a malicious .so module into PAM config to create a hidden user or allow silent logins. Knowing which services use PAM helps you monitor and validate those files.
4. Enforce Organization-Wide Policies
   Security teams often enforce policies like:
   - No root login via SSH
   - Lock account after 5 failed attempts
   - Password change every 60 days

   These only work if you know which services are PAM-dependent, so you can configure the correct files.
5. Hardening Attack Surface
   If you find a service using PAM that shouldn't allow external access (e.g., rsh, rlogin), you can disable or restrict it before it becomes an entry point.
6. Log Monitoring
   PAM-enabled services log authentication attempts via /var/log/auth.log (or journalctl), which is essential for intrusion detection and alerting.

Steps:

List PAM-configured services: use command ls /etc/pam.d/

Each file shown above represents a service that uses PAM for authentication.

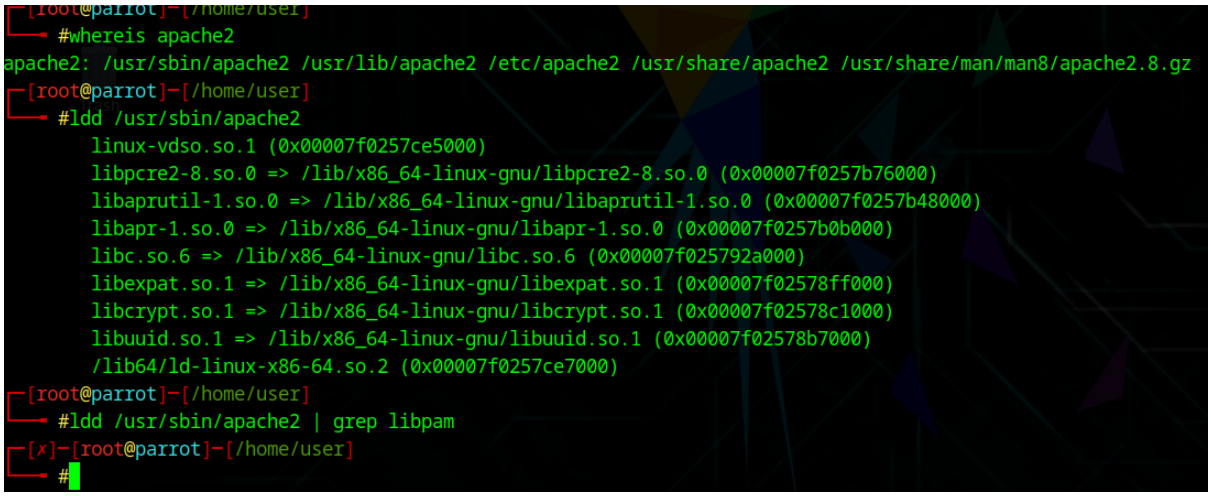
Look inside a PAM file: use command cat /etc/pam.d/sshd

```
┌─[root@parrot]─[/home/user]
└──╼ #cat /etc/pam.d/sshd
# PAM configuration for the Secure Shell service

# Standard Un*x authentication.
@include common-auth

# Disallow non-root logins when /etc/nologin exists.
account    required     pam_nologin.so

# Uncomment and edit /etc/security/access.conf if you need to set complex
# access limits that are hard to express in sshd_config.
# account  required     pam_access.so

# Standard Un*x authorization.
@include common-account

# SELinux needs to be the first session rule.  This ensures that any
# lingering context has been cleared.  Without this it is possible that a
# module could execute code in the wrong domain.
session [success=ok ignore=ignore module_unknown=ignore default=bad]        pam_selinux.so close

# Set the loginuid process attribute.
session    required     pam_loginuid.so

# Create a new session keyring.
session    optional     pam_keyinit.so force revoke

# Standard Un*x session setup and teardown.
@include common-session

# Print the message of the day upon successful login.
# This includes a dynamically generated part from /run/motd.dynamic
```

Now, check the libraries on which login depends on:

```
┌─[root@parrot]─[/home/user]
└──╼ #ldd /bin/login
        linux-vdso.so.1 (0x00007f7168709000)
        libpam.so.0 => /lib/x86_64-linux-gnu/libpam.so.0 (0x00007f71686cb000)
        libpam_misc.so.0 => /lib/x86_64-linux-gnu/libpam_misc.so.0 (0x00007f71686c6000)
        libaudit.so.1 => /lib/x86_64-linux-gnu/libaudit.so.1 (0x00007f7168695000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f71684b4000)
        libcap-ng.so.0 => /lib/x86_64-linux-gnu/libcap-ng.so.0 (0x00007f71684ac000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f716870b000)
┌─[root@parrot]─[/home/user]
└──╼ #ldd /bin/login | grep libpam
        libpam.so.0 => /lib/x86_64-linux-gnu/libpam.so.0 (0x00007f909bb00000)
        libpam_misc.so.0 => /lib/x86_64-linux-gnu/libpam_misc.so.0 (0x00007f909bafb000)
┌─[root@parrot]─[/home/user]
└──╼ #
```

Now, let’s check if apache2 depends on the libpam for authentication:



Clearly as per the output, it shows that apache2 doesn’t depends on the PAM.

Common-auth file explained:

Why You Should Know About /etc/pam.d/common-auth

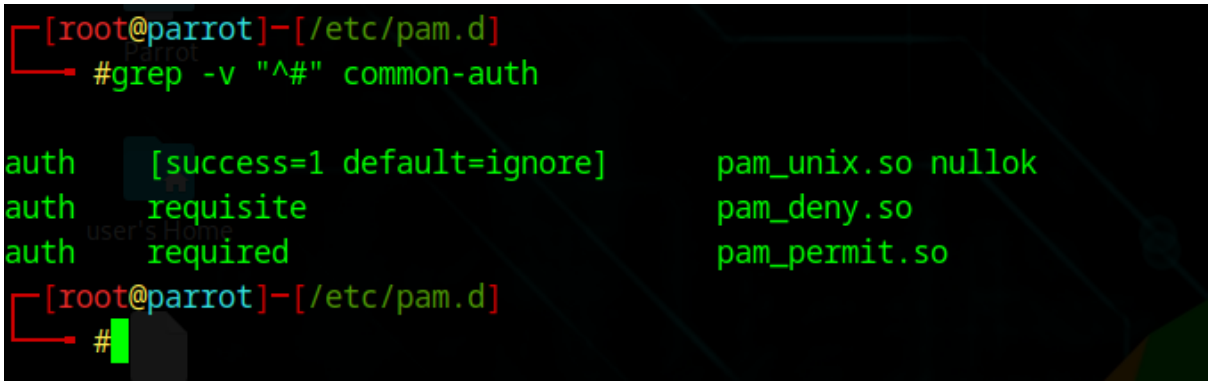
1. It controls how authentication works for most services.
   /etc/pam.d/common-auth defines the rules for verifying users during login, sudo, SSH, su, and more. If it's misconfigured, users can’t log in — including root.
2. It applies system-wide policies from one place.
   Instead of editing each service config (sudo, login, etc.), they use @include common-auth, which ensures uniform password policies (like 2FA, lockouts, etc.).
3. Attackers may target this file.
   If someone replaces it with pam_permit.so, they could allow anyone to log in without a password. If they add a backdoored .so, they could silently authenticate.
   You need to monitor and protect it (e.g., with chattr +i, aide, auditd).

What If /etc/pam.d/common-auth Gets Deleted or Corrupted?

- You’ll lose all standard password authentication.
- sudo, login, ssh, etc., may deny access, give “PAM error”, or lock you out entirely.
- Recovery becomes harder, especially on headless or remote systems.

Understanding the Common-auth file:

It removes all the comments (#) and shows only the real PAM rules:



This PAM configuration uses a smart flow to manage authentication securely. The first line uses pam_unix.so with a control flag [success=1 default=ignore], which means: if the password check succeeds, it will **skip the next line** (the deny rule). If it fails, it will go to the next line. That next line is pam_deny.so with requisite, which always fails and **immediately denies** access. So, if the password is wrong, login is blocked right away. If the password is correct, the deny rule is skipped, and the last line, pam_permit.so, runs. This module always allows success but is **harmless here** because it only runs if the password check passed. Overall, this structure ensures that login only succeeds if the password is valid, and any failure triggers an immediate denial.

What's the Overall Logic?

1. Try authenticating with pam_unix.so.
2. If it **succeeds** → skip the deny rule and go straight to permit = Login.
3. If it **fails** → hit pam_deny.so = Fail immediately.

So it works securely, despite the strange look.

--The End--