Day 34

Exploitation Analyst

User Management and PAM:

Enforce strong password:

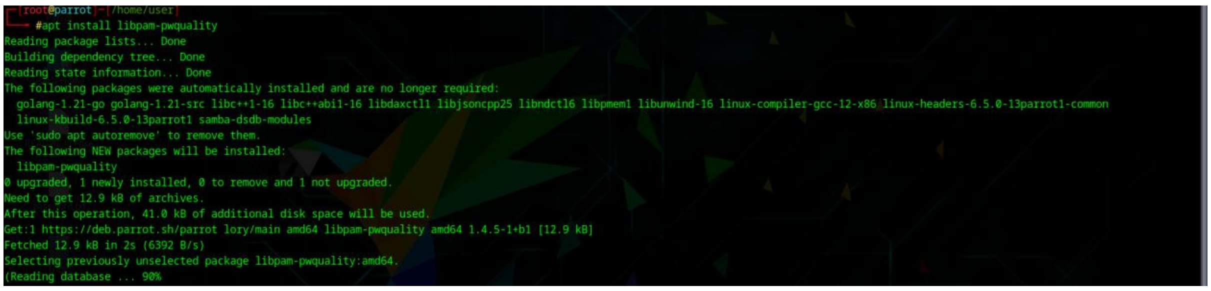
Why Enforcing strong password is important?

Enforcing strong passwords is important because it significantly reduces the risk of unauthorized access to systems and sensitive data. Weak passwords can be easily guessed, cracked through brute force, or obtained via phishing, giving attackers direct entry into accounts. Strong passwords—long, complex, and unique—make such attacks far more difficult and time-consuming, increasing overall security, protecting user privacy, and helping organizations comply with security regulations and best practices.

Working to enforce the strong password:

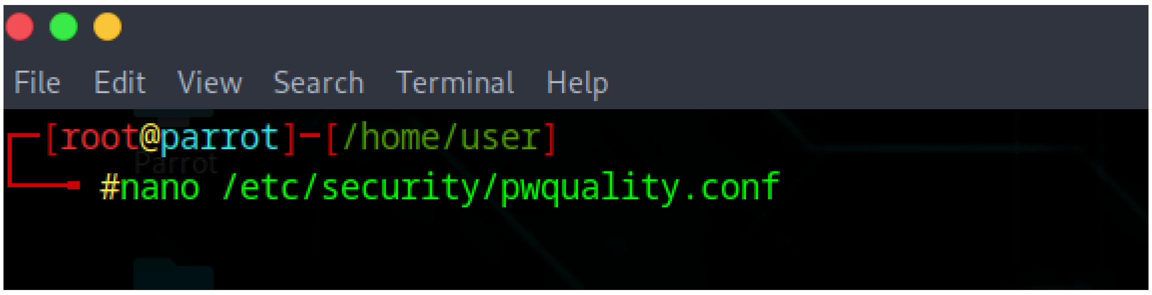
Steps:

Install this library: libpam-pwqualty



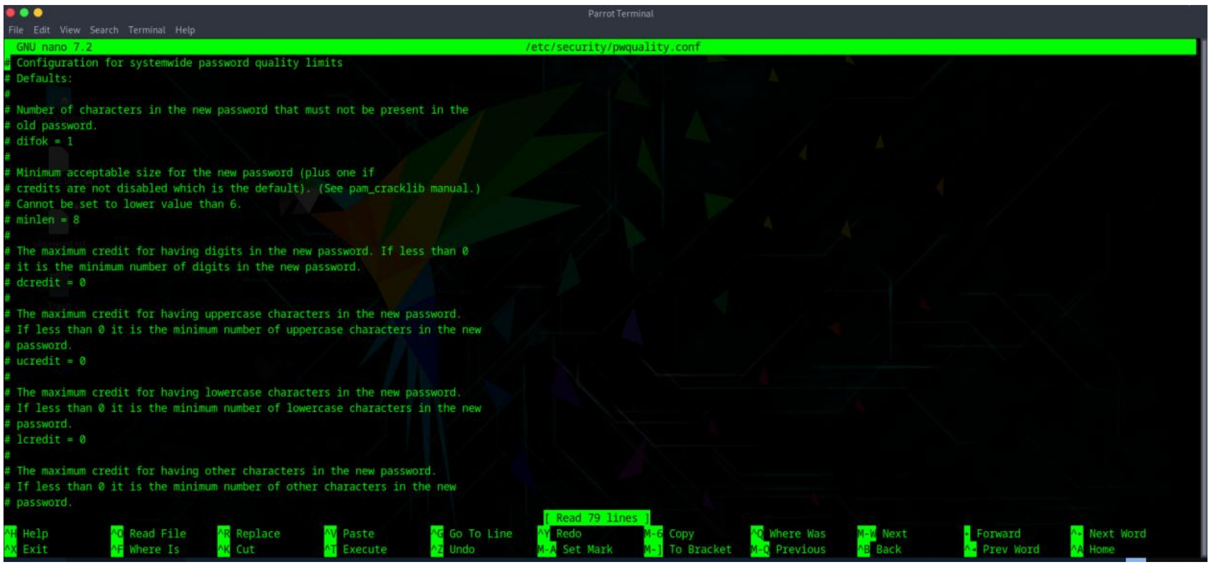
```
┌─[root@parrot]─[/home/user]
└──╼ #apt install libpam-pwquality
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  golang-1.21-go golang-1.21-src libc++1-16 libc++abi1-16 libdaxctl1 libjsoncpp25 libndctl16 libpmem1 libunwind-16 linux-compiler-gcc-12-x86 linux-headers-6.5.0-13parrot1-common
  linux-kbuild-6.5.0-13parrot1 samba-dsdb-modules
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  libpam-pwquality
0 upgraded, 1 newly installed, 0 to remove and 1 not upgraded.
Need to get 12.9 kB of archives.
After this operation, 41.0 kB of additional disk space will be used.
Get:1 https://deb.parrot.sh/parrot lory/main amd64 libpam-pwquality amd64 1.4.5-1+b1 [12.9 kB]
Fetched 12.9 kB in 2s (6392 B/s)
Selecting previously unselected package libpam-pwquality:amd64.
(Reading database ... 90%
```

Once it get installed open the /etc/security/pwquality.conf using the nano editor:



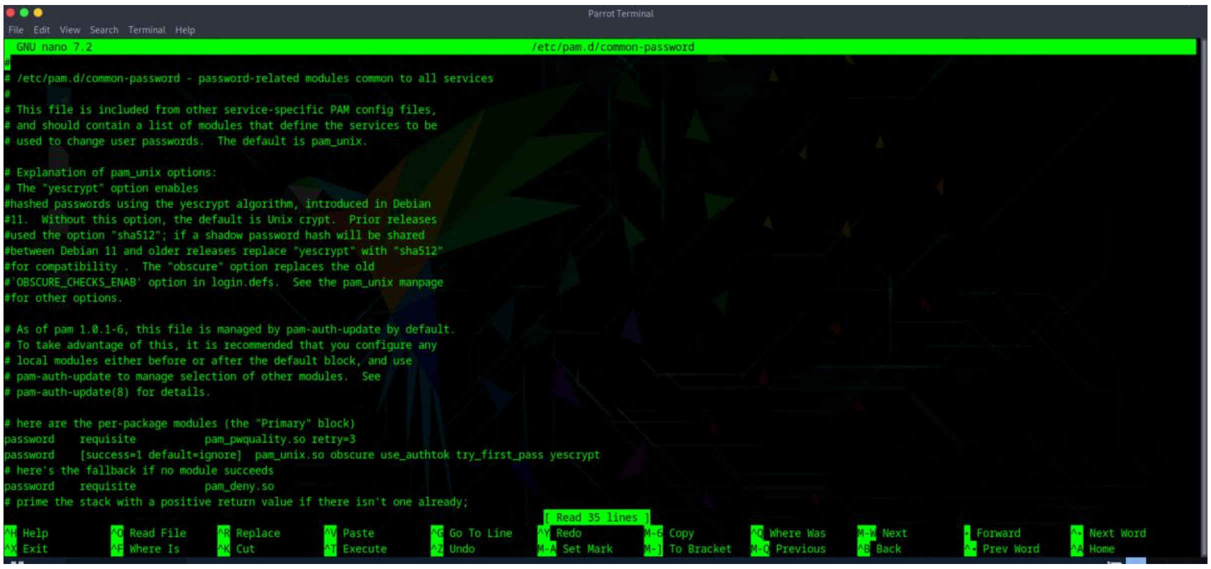
```
┌─[root@parrot]─[/home/user]
└──╼ #nano /etc/security/pwquality.conf
```

Following screen will appear: Edit as per the policy.



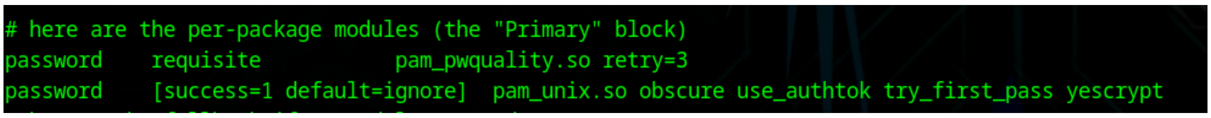
```
GNU nano 7.2                                    /etc/security/pwquality.conf
# Configuration for systemwide password quality limits
# Defaults:
#
# Number of characters in the new password that must not be present in the
# old password.
# difok = 1
#
# Minimum acceptable size for the new password (plus one if
# credits are not disabled which is the default). (See pam_cracklib manual.)
# Cannot be set to lower value than 6.
# minlen = 8
#
# The maximum credit for having digits in the new password. If less than 0
# it is the minimum number of digits in the new password.
# dcredit = 0
#
# The maximum credit for having uppercase characters in the new password.
# If less than 0 it is the minimum number of uppercase characters in the new
# password.
# ucredit = 0
#
# The maximum credit for having lowercase characters in the new password.
# If less than 0 it is the minimum number of lowercase characters in the new
# password.
# lcredit = 0
#
# The maximum credit for having other characters in the new password.
# If less than 0 it is the minimum number of other characters in the new
# password.
                                            [ Read 79 lines ]
```

Then open the /etc/pam.d/common-password using the nano text editor:



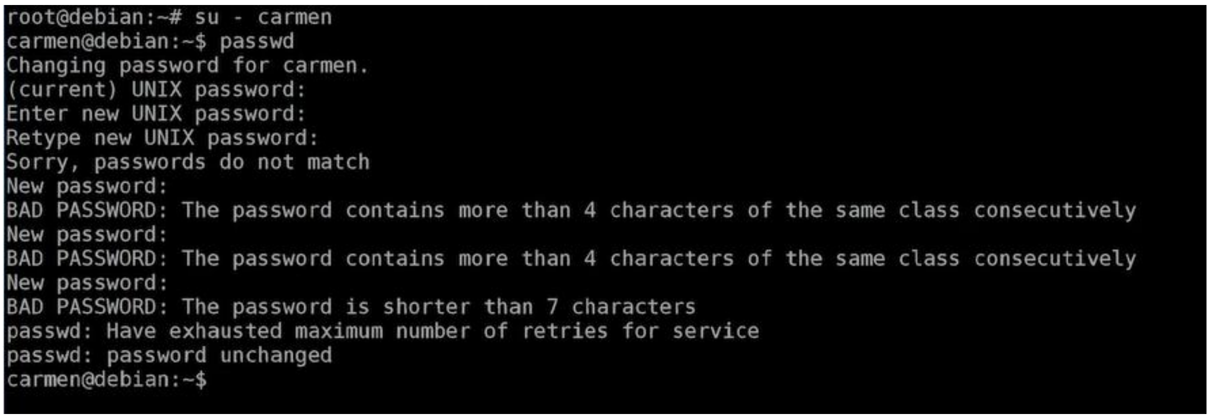
```
GNU nano 7.2                                    /etc/pam.d/common-password
#
# /etc/pam.d/common-password - password-related modules common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of modules that define the services to be
# used to change user passwords.  The default is pam_unix.

# Explanation of pam_unix options:
# The "yescrypt" option enables
#hashed passwords using the yescrypt algorithm, introduced in Debian
#11.  Without this option, the default is Unix crypt.  Prior releases
#used the option "sha512"; if a shadow password hash will be shared
#between Debian 11 and older releases replace "yescrypt" with "sha512"
#for compatibility .  The "obscure" option replaces the old
#`OBSCURE_CHECKS_ENAB' option in login.defs.  See the pam_unix manpage
#for other options.

# As of pam 1.0.1-6, this file is managed by pam-auth-update by default.
# To take advantage of this, it is recommended that you configure any
# local modules either before or after the default block, and use
# pam-auth-update to manage selection of other modules.  See
# pam-auth-update(8) for details.

# here are the per-package modules (the "Primary" block)
password    requisite           pam_pwquality.so retry=3
password    [success=1 default=ignore]  pam_unix.so obscure use_authtok try_first_pass yescrypt
# here's the fallback if no module succeeds
password    requisite           pam_deny.so
# prime the stack with a positive return value if there isn't one already;
                                            [ Read 35 lines ]
```

You can see this, line there telling that to change password maximum number of tries allowed is 3 only.



```
# here are the per-package modules (the "Primary" block)
password    requisite           pam_pwquality.so retry=3
password    [success=1 default=ignore]  pam_unix.so obscure use_authtok try_first_pass yescrypt
```

Then, just try to change the a user password and test:



```
root@debian:~# su - carmen
carmen@debian:~$ passwd
Changing password for carmen.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
Sorry, passwords do not match
New password:
BAD PASSWORD: The password contains more than 4 characters of the same class consecutively
New password:
BAD PASSWORD: The password contains more than 4 characters of the same class consecutively
New password:
BAD PASSWORD: The password is shorter than 7 characters
passwd: Have exhausted maximum number of retries for service
passwd: password unchanged
carmen@debian:~$
```

But what actually happens in the background?

When you edit **/etc/security/pwquality.conf** and **/etc/pam.d/common-password**, you are configuring how PAM (Pluggable Authentication Modules) enforces password strength during creation or change. When a user sets a password, PAM runs the common-password stack, which calls the pam_pwquality.so module. This module reads global rules from pwquality.conf and any inline settings in common-password, then checks the password for length, character variety, blacklist words, similarity to the username, and patterns like sequences or repetitions. If the password meets all criteria, it's passed to the hashing module (pam_unix.so), salted, hashed, and stored in /etc/shadow; if not, PAM rejects it and prompts the user to retry. Without common-password referencing pam_pwquality.so, the rules in pwquality.conf are ignored.

--The End--