

Passwords and Logins

Lab

Purpose

This work makes credential handling concrete.

You will:

- Create a controlled lab account and set a password you can reproduce.
- Observe where Linux stores account identity data versus password verification data.
- Extract a password hash you created and attempt recovery in a controlled environment.
- Compare dictionary cracking to constrained brute force.
- Remove everything when finished so the system returns to a clean state.

This lab is not about “hacking.” It is about understanding how credential risk works in practice.

Rules

- Work only on the designated lab VM or lab account.
- Crack only the hashes you created in this lab.
- Do not attempt password cracking on any system you do not own or explicitly control.
- Clean up all created users and files at the end.

Tools

Identity and account inspection:

- useradd
- userdel
- passwd
- getent
- grep

Hash extraction:

- /etc/passwd
- /etc/shadow

Cracking (controlled environment):

- hashcat

Deliverables

Submit a short write-up that includes:

1. The labuser line from /etc/passwd.
2. A redacted labuser line from /etc/shadow (keep \$id\$salt\$, replace the hash tail with ...).
3. The contents of hash.txt (exactly one hash line).
4. A short paragraph on dictionary cracking results and what it demonstrates.
5. One sentence comparing the dictionary vs constrained brute force.
6. Evidence of cleanup: getent passwd labuser showing no result.

Step 1: Create a lab user and set a password

Create the user:

```
sudo useradd -m labuser
```

Set the password:

```
sudo passwd labuser
```

Choose a password you can reproduce later. You will test your own assumptions about strength.

Verify the account exists:

```
getent passwd labuser
```

Record:

- The getent output line (this will be part of your submission).

Step 2: Observe /etc/passwd and /etc/shadow

View the labuser line in /etc/passwd:

```
grep '^labuser:' /etc/passwd
```

View the labuser line in /etc/shadow (requires sudo):

```
sudo grep '^labuser:' /etc/shadow
```

Identify the hash scheme by the prefix.

- Find the \$id\$... structure and record the id (for example: \$6\$, \$y\$).

Write down:

- The /etc/passwd line (exact).
- A redacted /etc/shadow line:
 - Keep `username:idsalt$`
 - Replace everything after `username:idsalt$` with literal

Step 3: Extract the hash for cracking

Extract only the hash field for labuser and place it into `hash.txt`.

Requirements:

- `hash.txt` must contain exactly one line.
- That line must be the full `idsalt$hash` value (not the whole /etc/shadow line).

Verify:

- Show that `hash.txt` contains one line.

Deliverable:

- `hash.txt` contents (pasted into your write-up or submitted as a file).

Step 4: Dictionary cracking with Hashcat

Install `hashcat` if needed.

Use this wordlist:

https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/Pwdb_top-10000000.txt

Run `hashcat` using the mode that matches the scheme you observed.

Record all of the following:

- The hash scheme id you observed in /etc/shadow
- The hashcat mode you selected
- The full command line used
- Whether the password was recovered

Deliverable:

- A short paragraph answering:
 - What does this demonstrate about common-password risk?
 - What does it demonstrate about choosing passwords that humans can reproduce?

Step 5: Brute-force mask demo

Run a brute-force attempt that is guaranteed to finish quickly.

Constraints:

- Keep the search space small (example: digits only, length 4–6).
- This step exists to make the cost difference visible, not to “win.”

Deliverable:

- One sentence comparing dictionary cracking vs brute force.

Step 6: Cleanup

Lock the account:

```
sudo passwd -l labuser
```

Remove the account and home directory:

```
sudo userdel -r labuser
```

Verify:

- `getent passwd labuser` returns no output

Deliverable:

- Paste a screenshot of the `getent passwd labuser` result showing the account is gone.

What to notice

- Linux separates the identity registry (`/etc/passwd`) from password verification material (`/etc/shadow`).
- The stored value is not the password; it is verification material derived from it.
- If password recovery is fast with a wordlist, the weakness is not cryptography—it is predictability.
- Dictionary attacks exploit human choice. Brute force exploits a small search space.
- “Strong password rules” do not protect secrets that are copied, reused, or exposed elsewhere.