ETHICAL HACKING PROJECT

Password cracking

JANUARY 23, 2026

HUZAIMA ASIM

Contents

Linux Password Cracker

Semester Project

Full Documentation and Step-by-Step Guide

This documentation provides a complete, self-contained guide to completing the **"Linux Password Cracker"** semester project as described in the provided project specification. It is intended for educational purposes only and must be executed strictly within a local Linux Virtual Machine (VM) environment. Do not run this on production systems, university servers, or against accounts you do not own, as per the ethical warning in the project.

The guide is structured as follows:

- **Overview and Requirements**: Summary of the project.
- **Prerequisites**: What you need to set up.
- **Step-by-Step Commands**: Detailed terminal commands to follow, with explanations.
- **Source Code**: Full, commented C++ code (cracker.cpp).
- **Makefile**: Full Makefile for compilation.
- **Project Report Template**: A ready-to-use outline for your PDF report, including placeholders for screenshots and explanations.
- **Testing and Performance Analysis**: How to test and analyze.
- **Cleanup and Ethical Notes**: Final steps and reminders.

Follow the steps in order. I'll use code blocks for commands and files. Assume you're using Ubuntu (a common Linux distro for VMs). If using another distro, adjust package managers accordingly (e.g., yum for Fedora).

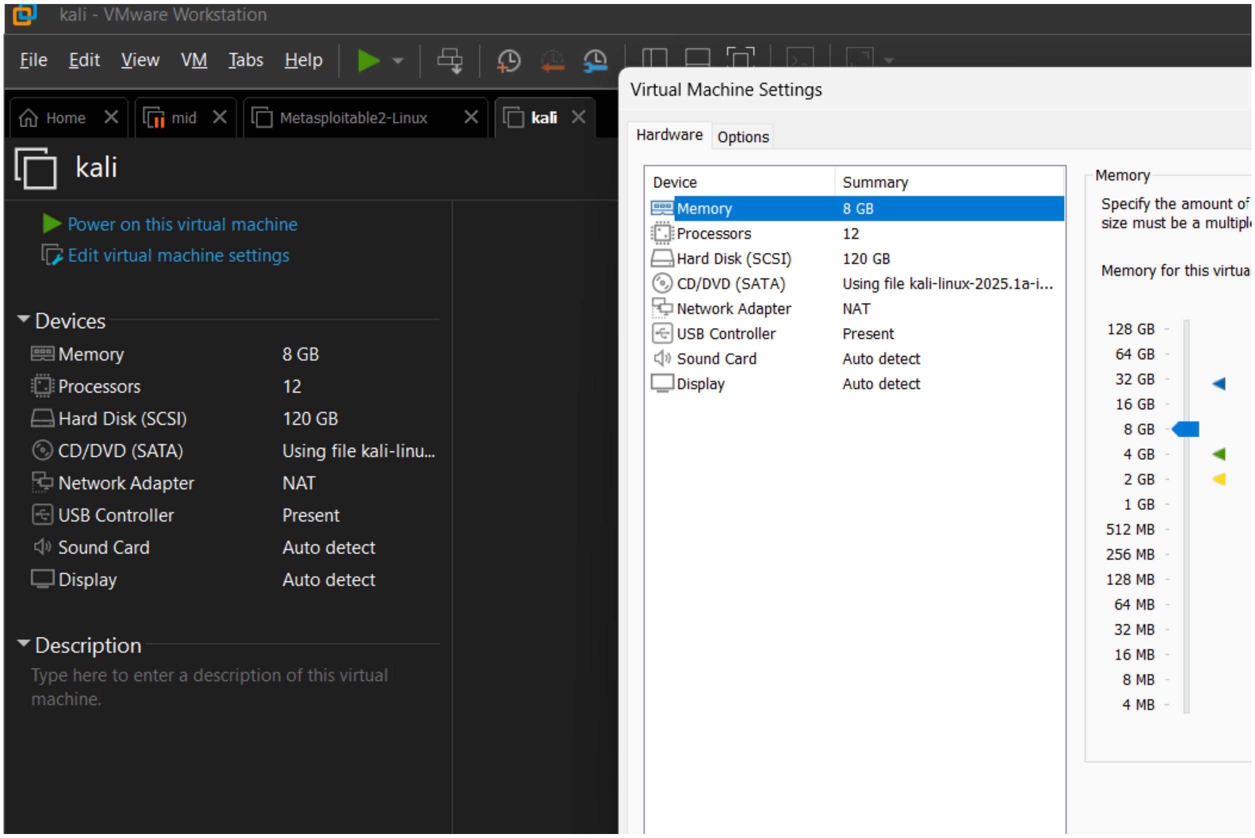
Overview and Requirements

- **Language**: C/C++ (we'll use C++ for better string handling).
- **Platform**: Linux (e.g., Ubuntu VM).
- **Goal**: Build a program that reads /etc/shadow, extracts salt and hash for a given user, and brute-forces the password (lowercase a-z, lengths 1-8) using crypt().
- **Deliverables**:
  1. Source code (cracker.cpp, commented).
  2. Makefile (for compilation with -lcrypt).
  3. Project Report (PDF): Explain /etc/shadow, flowchart, performance analysis, screenshots.
- **Key Concepts**:
  - **/etc/shadow** format: username**:$id$salt$hash:...**
  - Brute-force: Generate all combinations exhaustively **(a to z, aa to zz, etc.)**.
  - Use crypt() to hash candidate + salt and compare.

- **Ethical Note**: This is for learning about password security. Unauthorized use violates laws like the Computer Fraud and Abuse Act.

Prerequisites

1. **Set up a Linux VM:**
   - Download and install VMware.
   - Download Ubuntu ISO (e.g., ubuntu-22.04-desktop-amd64.iso).
   - Create a new VM with at least 2GB RAM and 20GB disk.
   - Install Ubuntu in the VM.
   - VM can be linux or Ubuntu , I will use linux
2. **In the VM, open a terminal and install required packages:**



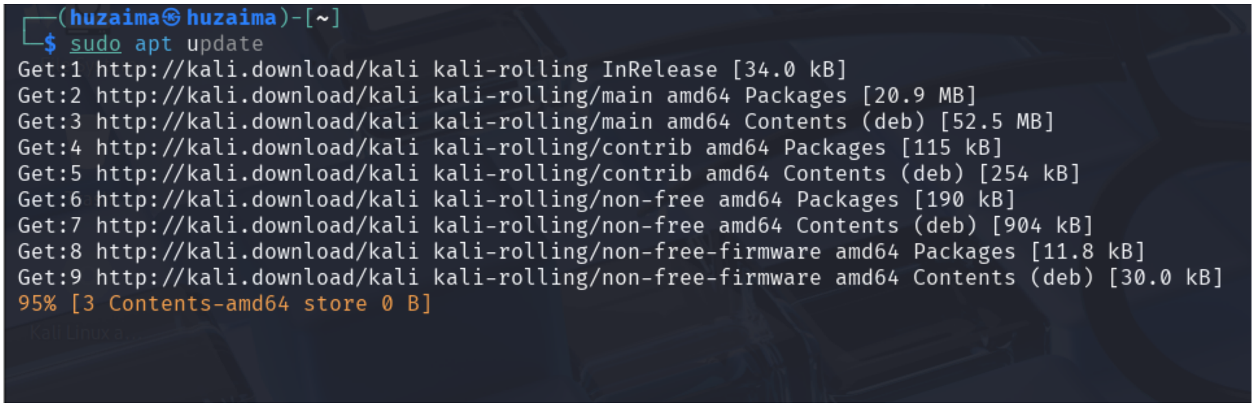
kali

Devices	
Memory	8 GB
Processors	12
Hard Disk (SCSI)	120 GB
CD/DVD (SATA)	Using file kali-linu...
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Display	Auto detect

Device	Summary
Memory	8 GB
Processors	12
Hard Disk (SCSI)	120 GB
CD/DVD (SATA)	Using file kali-linux-2025.1a-i...
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Display	Auto detect

Commands

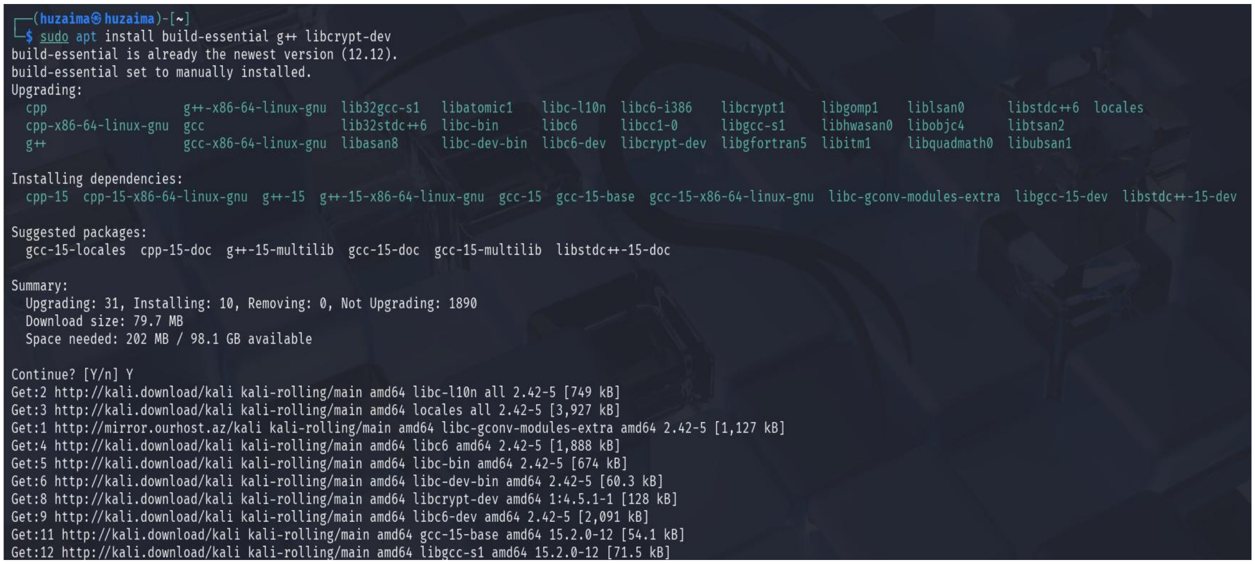
“ sudo apt update ”



```
┌──(huzaima㉿huzaima)-[~]
└─$ sudo apt update
Get:1 http://kali.download/kali kali-rolling InRelease [34.0 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [20.9 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [52.5 MB]
Get:4 http://kali.download/kali kali-rolling/contrib amd64 Packages [115 kB]
Get:5 http://kali.download/kali kali-rolling/contrib amd64 Contents (deb) [254 kB]
Get:6 http://kali.download/kali kali-rolling/non-free amd64 Packages [190 kB]
Get:7 http://kali.download/kali kali-rolling/non-free amd64 Contents (deb) [904 kB]
Get:8 http://kali.download/kali kali-rolling/non-free-firmware amd64 Packages [11.8 kB]
Get:9 http://kali.download/kali kali-rolling/non-free-firmware amd64 Contents (deb) [30.0 kB]
95% [3 Contents-amd64 store 0 B]
```

“ sudo apt install build-essential g++ libcrypt-dev ”

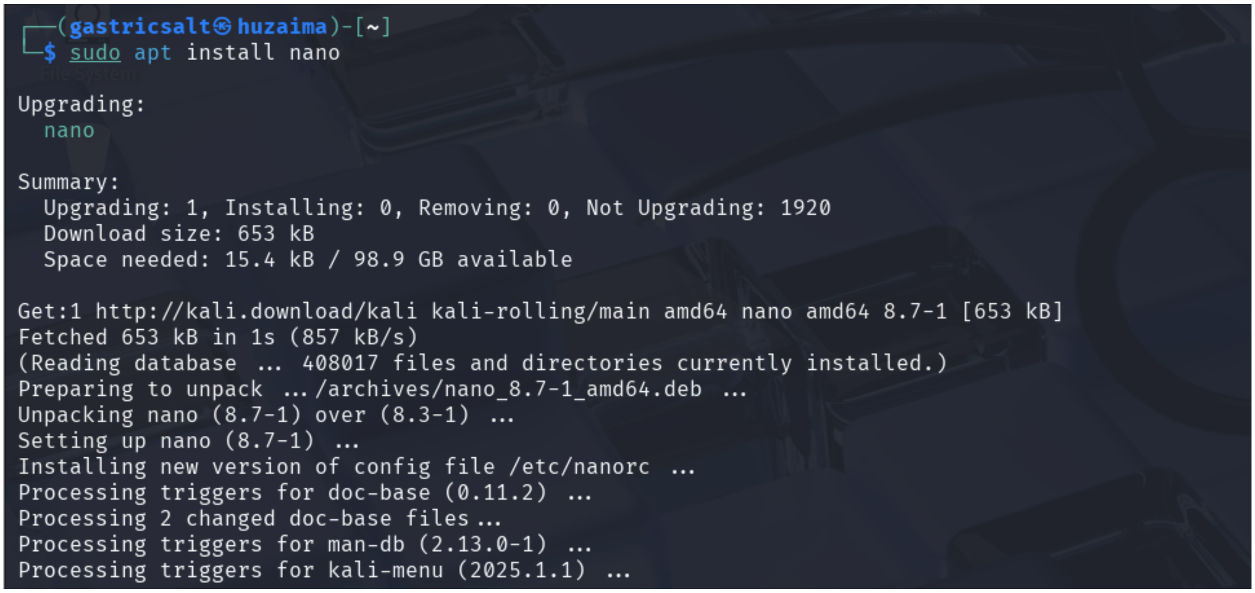
- **build-essential**: For gcc/g++ compiler.
- **g++:** C++ compiler.
- **libcrypt-dev**: For crypt() function.



```
┌──(huzaima㉿huzaima)-[~]
└─$ sudo apt install build-essential g++ libcrypt-dev
build-essential is already the newest version (12.12).
build-essential set to manually installed.
Upgrading:
  cpp                  g++-x86-64-linux-gnu  lib32gcc-s1   libatomic1    libc-l10n  libc6-i386    libcrypt1     libgomp1    liblsan0      libstdc++6  locales
  cpp-x86-64-linux-gnu gcc                   lib32stdc++6  libc-bin      libc6      libcc1-0      libgcc-s1     libhwasan0  libobjc4      libtsan2
  g++                  gcc-x86-64-linux-gnu  libasan8      libc-dev-bin  libc6-dev  libcrypt-dev  libgfortran5  libitm1     libquadmath0  libubsan1

Installing dependencies:
  cpp-15  cpp-15-x86-64-linux-gnu  g++-15  g++-15-x86-64-linux-gnu  gcc-15  gcc-15-base  gcc-15-x86-64-linux-gnu  libc-gconv-modules-extra  libgcc-15-dev  libstdc++-15-dev

Suggested packages:
  gcc-15-locales  cpp-15-doc  g++-15-multilib  gcc-15-doc  gcc-15-multilib  libstdc++-15-doc

Summary:
  Upgrading: 31, Installing: 10, Removing: 0, Not Upgrading: 1890
  Download size: 79.7 MB
  Space needed: 202 MB / 98.1 GB available

Continue? [Y/n] Y
Get:2 http://kali.download/kali kali-rolling/main amd64 libc-l10n all 2.42-5 [749 kB]
Get:3 http://kali.download/kali kali-rolling/main amd64 locales all 2.42-5 [3,927 kB]
Get:1 http://mirror.ourhost.az/kali kali-rolling/main amd64 libc-gconv-modules-extra amd64 2.42-5 [1,127 kB]
Get:4 http://kali.download/kali kali-rolling/main amd64 libc6 amd64 2.42-5 [1,888 kB]
Get:5 http://kali.download/kali kali-rolling/main amd64 libc-bin amd64 2.42-5 [674 kB]
Get:6 http://kali.download/kali kali-rolling/main amd64 libc-dev-bin amd64 2.42-5 [60.3 kB]
Get:8 http://kali.download/kali kali-rolling/main amd64 libcrypt-dev amd64 1:4.5.1-1 [128 kB]
Get:9 http://kali.download/kali kali-rolling/main amd64 libc6-dev amd64 2.42-5 [2,091 kB]
Get:11 http://kali.download/kali kali-rolling/main amd64 gcc-15-base amd64 15.2.0-12 [54.1 kB]
Get:12 http://kali.download/kali kali-rolling/main amd64 libgcc-s1 amd64 15.2.0-12 [71.5 kB]
```

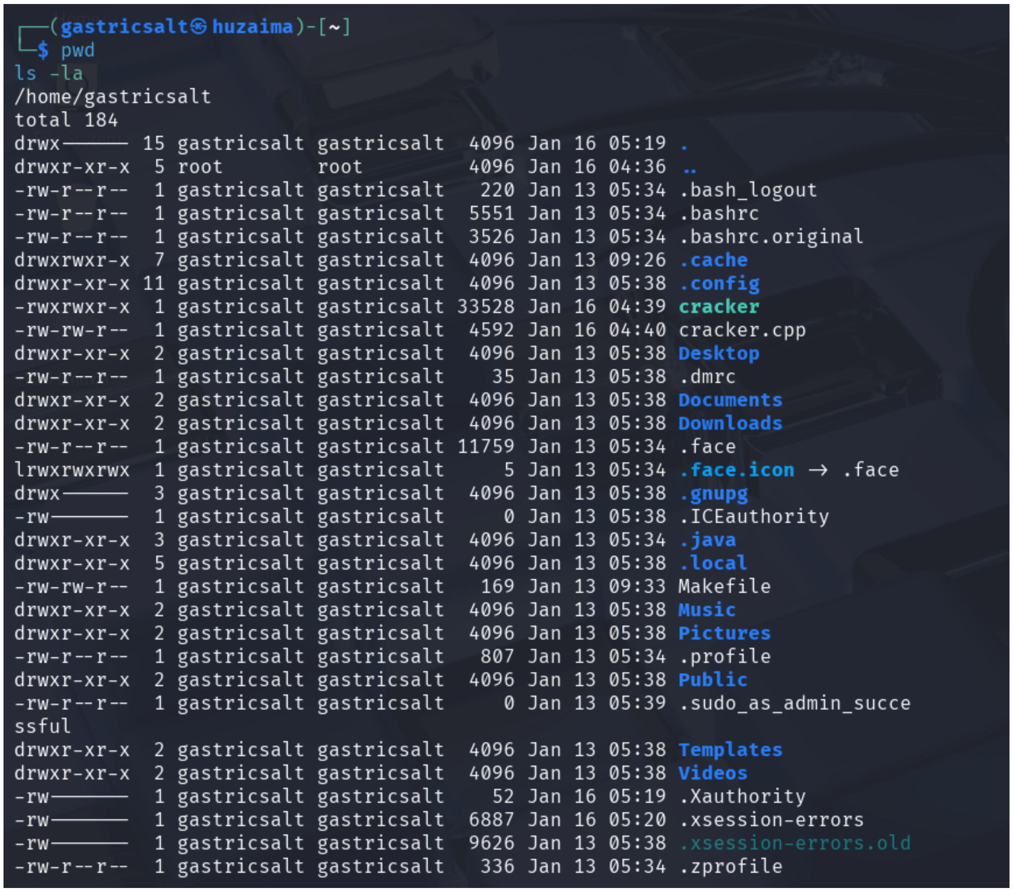
3. Create a text editor setup (e.g., nano or vim):

“ sudo apt install nano ”



```
┌──(gastricsalt㉿huzaima)-[~]
└─$ sudo apt install nano

Upgrading:
  nano

Summary:
  Upgrading: 1, Installing: 0, Removing: 0, Not Upgrading: 1920
  Download size: 653 kB
  Space needed: 15.4 kB / 98.9 GB available

Get:1 http://kali.download/kali kali-rolling/main amd64 nano amd64 8.7-1 [653 kB]
Fetched 653 kB in 1s (857 kB/s)
(Reading database ... 408017 files and directories currently installed.)
Preparing to unpack .../archives/nano_8.7-1_amd64.deb ...
Unpacking nano (8.7-1) over (8.3-1) ...
Setting up nano (8.7-1) ...
Installing new version of config file /etc/nanorc ...
Processing triggers for doc-base (0.11.2) ...
Processing 2 changed doc-base files...
Processing triggers for man-db (2.13.0-1) ...
Processing triggers for kali-menu (2025.1.1) ...
```

“pwd   ls -ls”



```
┌──(gastricsalt㉿huzaima)-[~]
└─$ pwd
ls -la
/home/gastricsalt
total 184
drwx------ 15 gastricsalt gastricsalt  4096 Jan 16 05:19 .
drwxr-xr-x  5 root        root         4096 Jan 16 04:36 ..
-rw-r--r--  1 gastricsalt gastricsalt   220 Jan 13 05:34 .bash_logout
-rw-r--r--  1 gastricsalt gastricsalt  5551 Jan 13 05:34 .bashrc
-rw-r--r--  1 gastricsalt gastricsalt  3526 Jan 13 05:34 .bashrc.original
drwxrwxr-x  7 gastricsalt gastricsalt  4096 Jan 13 09:26 .cache
drwxr-xr-x 11 gastricsalt gastricsalt  4096 Jan 13 05:38 .config
-rwxrwxr-x  1 gastricsalt gastricsalt 33528 Jan 16 04:39 cracker
-rw-rw-r--  1 gastricsalt gastricsalt  4592 Jan 16 04:40 cracker.cpp
drwxr-xr-x  2 gastricsalt gastricsalt  4096 Jan 13 05:38 Desktop
-rw-r--r--  1 gastricsalt gastricsalt    35 Jan 13 05:38 .dmrc
drwxr-xr-x  2 gastricsalt gastricsalt  4096 Jan 13 05:38 Documents
drwxr-xr-x  2 gastricsalt gastricsalt  4096 Jan 13 05:38 Downloads
-rw-r--r--  1 gastricsalt gastricsalt 11759 Jan 13 05:34 .face
lrwxrwxrwx  1 gastricsalt gastricsalt     5 Jan 13 05:34 .face.icon -> .face
drwx------  3 gastricsalt gastricsalt  4096 Jan 13 05:38 .gnupg
-rw-------  1 gastricsalt gastricsalt     0 Jan 13 05:38 .ICEauthority
drwxr-xr-x  3 gastricsalt gastricsalt  4096 Jan 13 05:34 .java
drwxr-xr-x  5 gastricsalt gastricsalt  4096 Jan 13 05:38 .local
-rw-rw-r--  1 gastricsalt gastricsalt   169 Jan 13 09:33 Makefile
drwxr-xr-x  2 gastricsalt gastricsalt  4096 Jan 13 05:38 Music
drwxr-xr-x  2 gastricsalt gastricsalt  4096 Jan 13 05:38 Pictures
-rw-r--r--  1 gastricsalt gastricsalt   807 Jan 13 05:34 .profile
drwxr-xr-x  2 gastricsalt gastricsalt  4096 Jan 13 05:38 Public
-rw-r--r--  1 gastricsalt gastricsalt     0 Jan 13 05:39 .sudo_as_admin_succe
ssful
drwxr-xr-x  2 gastricsalt gastricsalt  4096 Jan 13 05:38 Templates
drwxr-xr-x  2 gastricsalt gastricsalt  4096 Jan 13 05:38 Videos
-rw-------  1 gastricsalt gastricsalt    52 Jan 16 05:19 .Xauthority
-rw-------  1 gastricsalt gastricsalt  6887 Jan 16 05:20 .xsession-errors
-rw-------  1 gastricsalt gastricsalt  9626 Jan 13 05:38 .xsession-errors.old
-rw-r--r--  1 gastricsalt gastricsalt   336 Jan 13 05:34 .zprofile
```

Step-by-Step Commands

Step 1: Set Up linuxuser

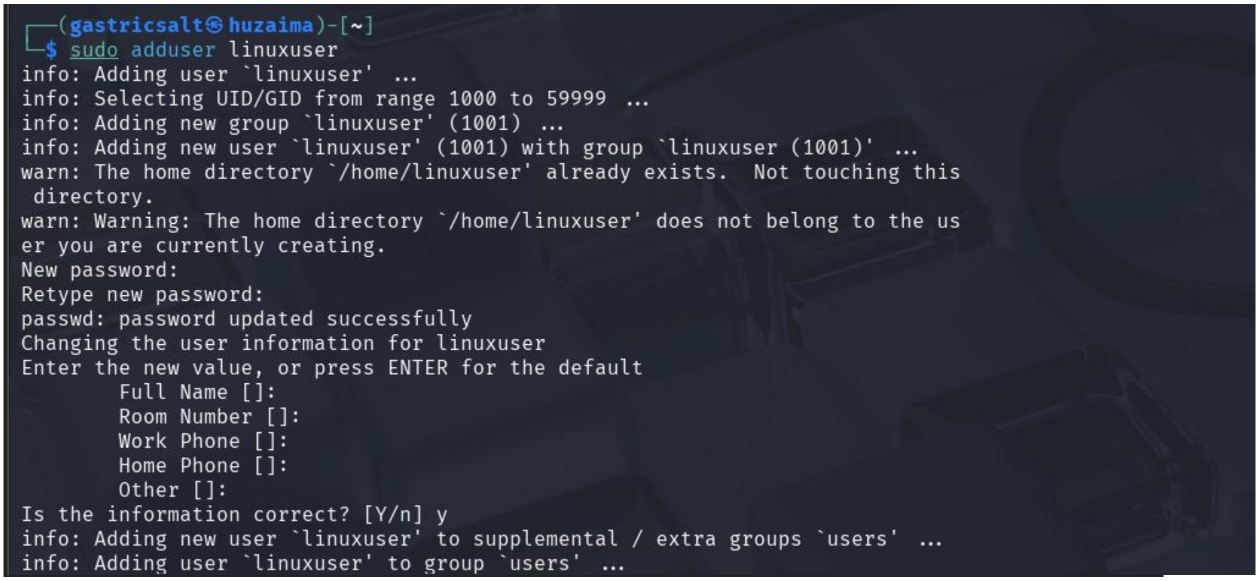
- Create a test user for safe testing.
- Commands:

  “ sudo adduser linuxuser/master ”

  - Set a simple lowercase password like "abc/abcd/abcdef"

  “ sudo passwd linuxuser/master ”

Password is set as “ abc ”



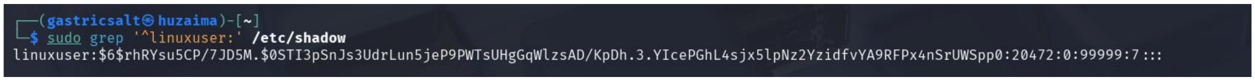
```
┌──(gastricsalt㉿huzaima)-[~]
└─$ sudo adduser linuxuser
info: Adding user `linuxuser' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `linuxuser' (1001) ...
info: Adding new user `linuxuser' (1001) with group `linuxuser (1001)' ...
warn: The home directory `/home/linuxuser' already exists.  Not touching this
 directory.
warn: Warning: The home directory `/home/linuxuser' does not belong to the us
er you are currently creating.
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for linuxuser
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
info: Adding new user `linuxuser' to supplemental / extra groups `users' ...
info: Adding user `linuxuser' to group `users' ...
```

  - Enter the password again ("**abc**").

- Verify /etc/shadow entry:

  “ sudo grep '^linuxuser:' /etc/shadow”

  - Output in hash value of **linuxuser**



```
┌──(gastricsalt㉿huzaima)-[~]
└─$ sudo grep '^linuxuser:' /etc/shadow
linuxuser:$6$rhRYsu5CP/7JD5M.$0STI3pSnJs3UdrLun5jeP9PWTsUHgGqWlzsAD/KpDh.3.YIcePGhL4sjx5lpNz2YzidfvYA9RFPx4nSrUWSpp0:20472:0:99999:7:::
```

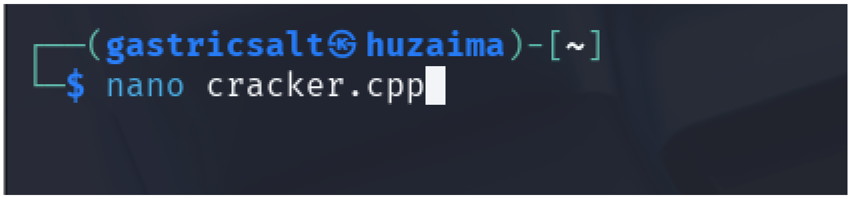
“linuxuser:$6$rhRYsu5CP/7JD5M.$0STI3pSnJs3UdrLun5jeP9PWTsUHgGqWlzsAD/KpDh.3.YIcePGhL4sjx5lpNz2YzidfvYA9RFPx4nSrUWSpp0:20472:0:99999:7:::”

- Explanation: This creates a user with a known password for brute-forcing. Take a screenshot here for your report.

Step 2: Write the Source Code

- Create and edit the file:

  “ **nano cracker.cpp** ”



```
┌──(gastricsalt㉿huzaima)-[~]
└─$ nano cracker.cpp
```

Paste this code in cracker.cpp

```
// ====================== HEADER FILES ======================

// Input / output stream (cout, cerr)

#include <iostream>

// File handling (ifstream)

#include <fstream>

// String handling

#include <string>

// C-style string comparison (strcmp)

#include <cstring>

// crypt() function for hashing passwords (Linux)

#include <unistd.h>

// Time measurement (for performance calculation)

#include <chrono>

// Formatting output (not heavily used here)

#include <iomanip>

using namespace std;

// ============================================================

// FUNCTION: extract_salt_hash

// PURPOSE:
```

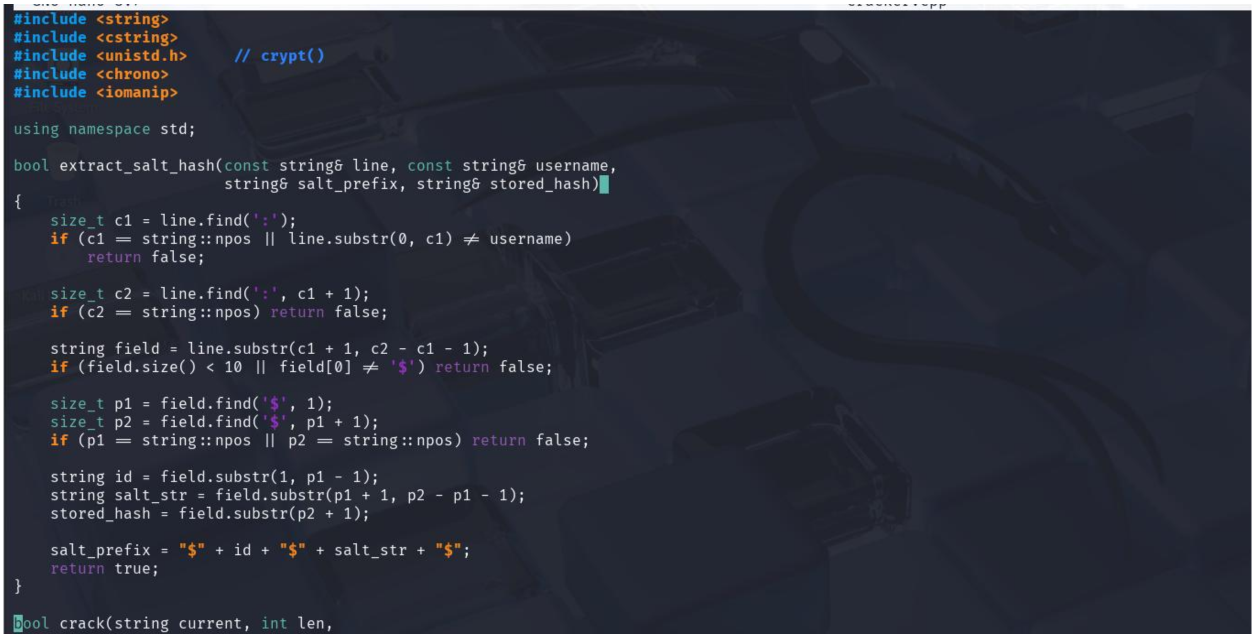
```
//   - Reads one line from /etc/shadow
//   - Checks if it belongs to the given username
//   - Extracts hashing algorithm ID, salt, and stored hash
// ============================================================
bool extract_salt_hash(const string& line, const string& username,
                       string& salt_prefix, string& stored_hash) {
    // Find first ':' → separates username and password field
    size_t c1 = line.find(':');
    // If ':' not found OR username does not match → skip line
    if (c1 == string::npos || line.substr(0, c1) != username)
        return false;

    // Find second ':' → end of password field
    size_t c2 = line.find(':', c1 + 1);
    if (c2 == string::npos)
        return false;
    // Extract password field between first and second colon
    string field = line.substr(c1 + 1, c2 - c1 - 1);
    // Validate that the field looks like a hashed password
    // Format should start with '$'
    if (field.size() < 10 || field[0] != '$')
        return false;
    // Find positions of '$' separators
    size_t p1 = field.find('$', 1);         // after algorithm ID
    size_t p2 = field.find('$', p1 + 1);    // after salt
    // If structure is invalid
    if (p1 == string::npos || p2 == string::npos)
```

```
        return false;
    // Extract hashing algorithm ID (e.g., "6" for SHA-512)
    string id = field.substr(1, p1 - 1);
    // Extract salt string
    string salt_str = field.substr(p1 + 1, p2 - p1 - 1);
    // Extract stored hash (everything after third '$')
    stored_hash = field.substr(p2 + 1);
    // Build salt prefix used by crypt()
    // FORMAT: $id$salt$
    salt_prefix = "$" + id + "$" + salt_str + "$";
    return true;
}
// ============================================================
// FUNCTION: crack
// PURPOSE:
//   - Recursively generates passwords (a–z)
//   - Hashes each candidate using crypt()
//   - Compares with stored hash
// ============================================================
bool crack(string current, int len,
           const string& salt_prefix, const string& stored_hash) {
    // Base case: generated password reached required length
    if (current.length() == static_cast<size_t>(len)) {
        // Hash the candidate password
        const char* computed = crypt(current.c_str(), salt_prefix.c_str());
        if (computed) {
            // Full hash = salt prefix + stored hash
            string expected_full = salt_prefix + stored_hash;

            // ================= DEBUG OUTPUT =================

            // Prints comparison details for known test password

            if (current == "abcd") {

                cout << "\n=== DEBUG - correct candidate 'abcd' reached ===\n";

                cout << "Salt prefix: " << salt_prefix << "\n";

                cout << "Stored hash: " << stored_hash << "\n";

                cout << "Computed full: " << computed << "\n";

                cout << "Expected full: " << expected_full << "\n";

                cout << "Match? "

                     << (strcmp(computed, expected_full.c_str()) == 0 ? "YES" : "NO")

                     << "\n";

                cout << "====================================================\n\n";

            }

            // =================================================

            // Compare computed hash with actual stored hash

            if (strcmp(computed, expected_full.c_str()) == 0) {

                cout << "\nPASSWORD FOUND: " << current << endl;

                cout << "Length: " << current.length() << " chars\n";

                return true;

            }

        }

        return false;

    }

    // Recursive case: try all characters from a to z

    for (char c = 'a'; c <= 'z'; ++c) {

        if (crack(current + c, len, salt_prefix, stored_hash))
```

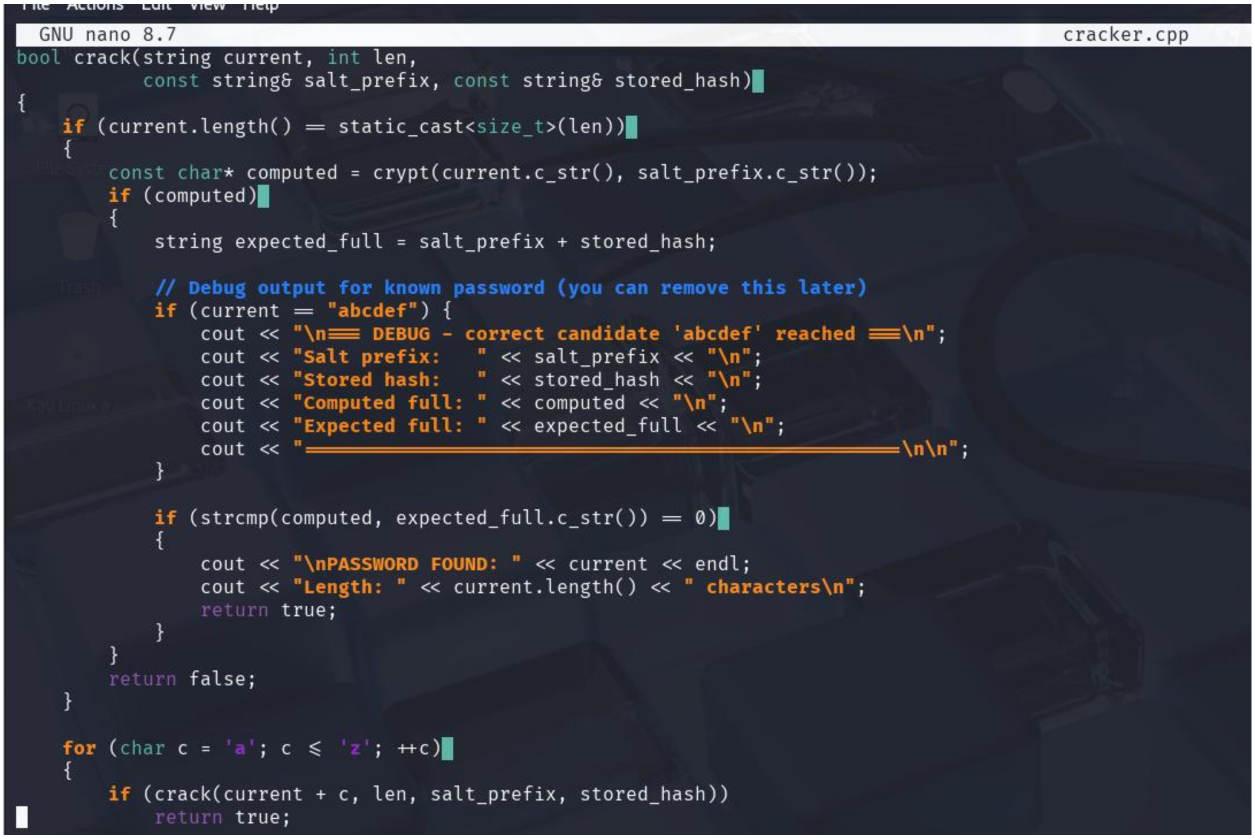
```
            return true;
        }
        return false;
    }
// ============================================================
// MAIN FUNCTION
// ============================================================
int main(int argc, char* argv[]) {
    // Check if username argument is provided
    if (argc != 2) {
        cerr << "Usage: sudo " << argv[0] << " <username>\n";
        return 1;
    }
    // Store username
    string user = argv[1];
    // Display program header
    cout << "\n=== Password Cracker - DEBUG VERSION ===\n";
    cout << " User: " << user << "\n";
    cout << " Max length: 4 characters (a-z)\n\n";
    // Open /etc/shadow file (requires sudo)
    ifstream shadow("/etc/shadow");
    if (!shadow.is_open()) {
        cerr << "Cannot open /etc/shadow → must use sudo\n";
        return 1;
    }
    string line, salt, hash;
    bool found = false;
```

```
// Read shadow file line by line

while (getline(shadow, line)) {

    if (extract_salt_hash(line, user, salt, hash)) {

        found = true;

        break;

    }

}

shadow.close();

// If user not found or no password hash

if (!found) {

    cerr << "User not found or has no password hash\n";

    return 1;

}

cout << "Starting brute-force...\n\n";

// Record overall start time

auto t0 = chrono::steady_clock::now();

bool cracked = false;

// Try password lengths from 1 to 7

for (int l = 1; l <= 7; ++l) {

    cout << "Trying length " << l << "... ";

    cout.flush();

    auto s = chrono::steady_clock::now();

    if (crack("", l, salt, hash)) {

        cracked = true;

        break;

    }

    auto e = chrono::steady_clock::now();
```

```
        auto ms = chrono::duration_cast<chrono::milliseconds>(e - s).count();

        cout << "finished (" << ms << " ms)\n";

    }

    // Calculate total runtime

    auto total = chrono::duration_cast<chrono::milliseconds>(

        chrono::steady_clock::now() - t0).count();

    // Final result output

    cout << "\n----------------------------------------\n";

    cout << (cracked ? "SUCCESS!" : "Not found within 4 chars") << "\n";

    cout << "Total time: " << total << " ms (" << total / 1000.0 << " sec)\n";

    cout << "----------------------------------------\n";

    return 0;

}
```



```
#include <string>
#include <cstring>
#include <unistd.h>      // crypt()
#include <chrono>
#include <iomanip>

using namespace std;

bool extract_salt_hash(const string& line, const string& username,
                       string& salt_prefix, string& stored_hash)
{
    size_t c1 = line.find(':');
    if (c1 == string::npos || line.substr(0, c1) != username)
        return false;

    size_t c2 = line.find(':', c1 + 1);
    if (c2 == string::npos) return false;

    string field = line.substr(c1 + 1, c2 - c1 - 1);
    if (field.size() < 10 || field[0] != '$') return false;

    size_t p1 = field.find('$', 1);
    size_t p2 = field.find('$', p1 + 1);
    if (p1 == string::npos || p2 == string::npos) return false;

    string id = field.substr(1, p1 - 1);
    string salt_str = field.substr(p1 + 1, p2 - p1 - 1);
    stored_hash = field.substr(p2 + 1);

    salt_prefix = "$" + id + "$" + salt_str + "$";
    return true;
}

bool crack(string current, int len,
```



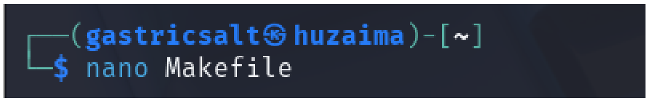
```
  GNU nano 8.7                                                                    cracker.cpp
bool crack(string current, int len,
           const string& salt_prefix, const string& stored_hash)
{
    if (current.length() == static_cast<size_t>(len))
    {
        const char* computed = crypt(current.c_str(), salt_prefix.c_str());
        if (computed)
        {
            string expected_full = salt_prefix + stored_hash;

            // Debug output for known password (you can remove this later)
            if (current == "abcdef") {
                cout << "\n=== DEBUG - correct candidate 'abcdef' reached ===\n";
                cout << "Salt prefix:    " << salt_prefix << "\n";
                cout << "Stored hash:    " << stored_hash << "\n";
                cout << "Computed full: " << computed << "\n";
                cout << "Expected full: " << expected_full << "\n";
                cout << "==================================================\n\n";
            }

            if (strcmp(computed, expected_full.c_str()) == 0)
            {
                cout << "\nPASSWORD FOUND: " << current << endl;
                cout << "Length: " << current.length() << " characters\n";
                return true;
            }
        }
        return false;
    }

    for (char c = 'a'; c <= 'z'; ++c)
    {
        if (crack(current + c, len, salt_prefix, stored_hash))
            return true;
```

- Explanation: The code reads /etc/shadow, parses salt/hash, and uses recursion to generate and check passwords.

Step 3: Create the Makefile

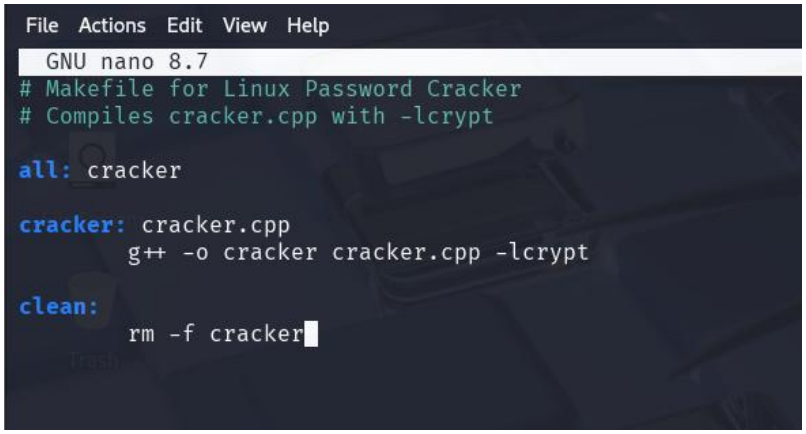
- Create and edit:

  " **nano Makefile** "



```
┌──(gastricsalt㉿huzaima)-[~]
└─$ nano Makefile
```

- Paste this code in **Makefile**



```
  GNU nano 8.7
# Makefile for Linux Password Cracker
# Compiles cracker.cpp with -lcrypt

all: cracker

cracker: cracker.cpp
        g++ -o cracker cracker.cpp -lcrypt

clean:
        rm -f cracker
```

# Makefile for Linux Password Cracker

# Compiles cracker.cpp with -lcrypt

all: cracker

cracker: cracker.cpp

g++ -o cracker cracker.cpp -lcrypt

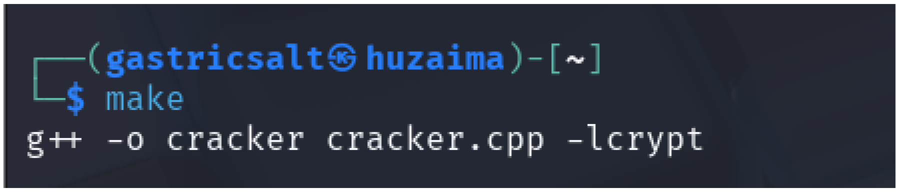
clean:

rm -f cracker

- Save and exit.
- Explanation: This automates compilation with -lcrypt linkage.

Step 4: Compile the Program

- Run: " **make** "
- Output: Should create an executable named "cracker".
- Explanation: Compiles cracker.cpp into an executable.



```
┌──(gastricsalt㉿huzaima)-[~]
└─$ make
g++ -o cracker cracker.cpp -lcrypt
```

Step 5: Run the Program

- Test with your user:

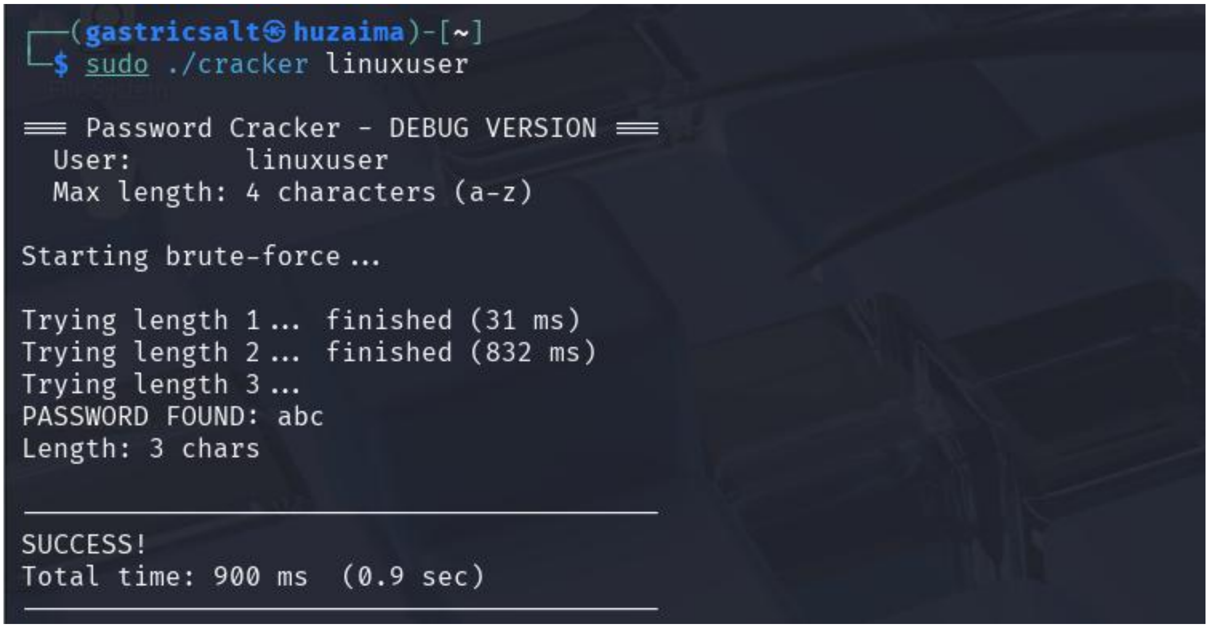
“ **sudo ./cracker linuxuser** ”

Performing first for 6 characters

- Change password to **passwd**
- For performance testing:
  - Use time to measure:

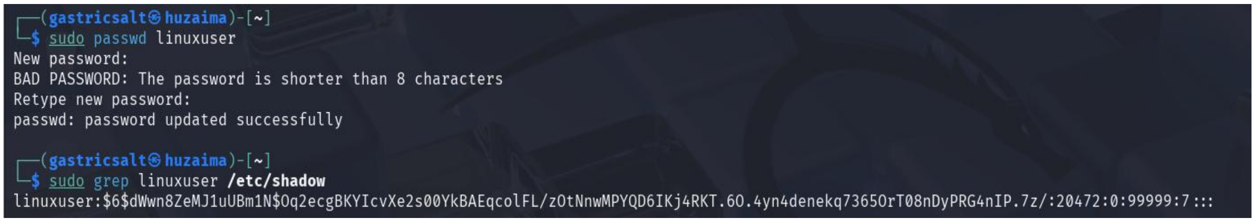
“ **time sudo ./cracker linuxuser** ”

  - Test different passwords:
    - Change password: **sudo passwd testuser** (set to "**abc**" for 3 chars)

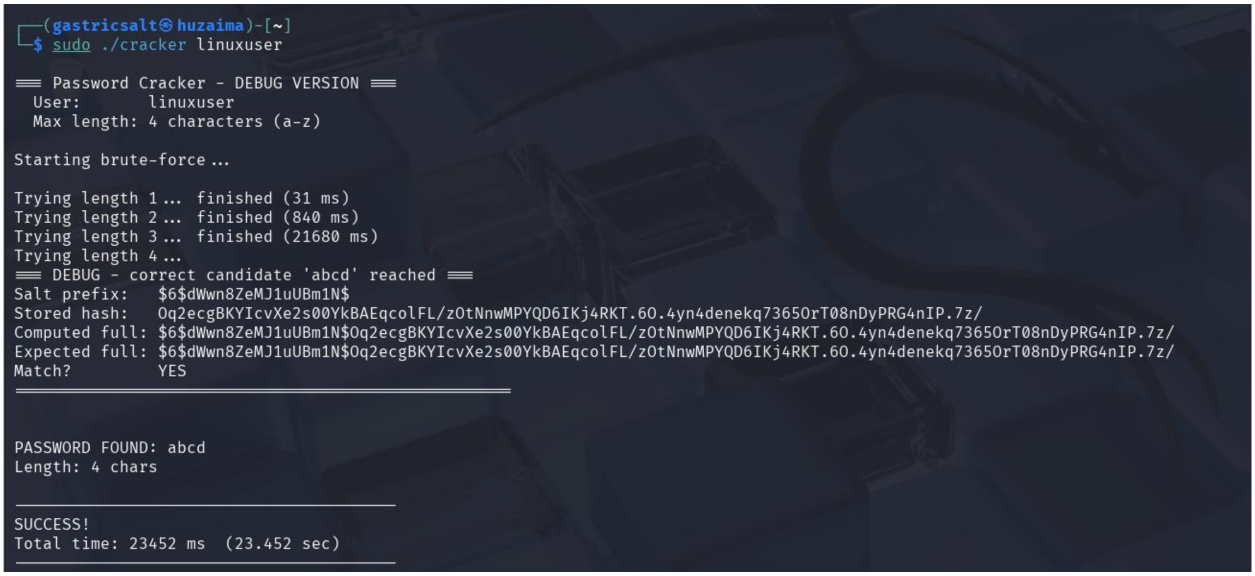


```
┌──(gastricsalt㉿huzaima)-[~]
└─$ sudo ./cracker linuxuser

══ Password Cracker - DEBUG VERSION ══
  User:       linuxuser
  Max length: 4 characters (a-z)

Starting brute-force ...

Trying length 1 ...  finished (31 ms)
Trying length 2 ...  finished (832 ms)
Trying length 3 ...
PASSWORD FOUND: abc
Length: 3 chars

────────────────────────────────────────
SUCCESS!
Total time: 900 ms  (0.9 sec)
────────────────────────────────────────
```

    - Run again and note time.
    - Repeat for 4 chars ("abcd").
- Explanation: Runs with **sudo** for **/etc/shadow** access. If no match by length 4, it fails. Take screenshots of successful runs and timings.



```
┌──(gastricsalt㉿huzaima)-[~]
└─$ sudo passwd linuxuser
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: password updated successfully

┌──(gastricsalt㉿huzaima)-[~]
└─$ sudo grep linuxuser /etc/shadow
linuxuser:$6$dWwn8ZeMJ1uUBm1N$Oq2ecgBKYIcvXe2s00YkBAEqcolFL/zOtNnwMPYQD6IKj4RKT.6O.4yn4denekq7365OrT08nDyPRG4nIP.7z/:20472:0:99999:7:::
```



```
┌──(gastricsalt㉿huzaima)-[~]
└─$ sudo ./cracker linuxuser

═══ Password Cracker - DEBUG VERSION ═══
  User:       linuxuser
  Max length: 4 characters (a-z)

Starting brute-force...

Trying length 1... finished (31 ms)
Trying length 2... finished (840 ms)
Trying length 3... finished (21680 ms)
Trying length 4...
═══ DEBUG - correct candidate 'abcd' reached ═══
Salt prefix:   $6$dWwn8ZeMJ1uUBm1N$
Stored hash:   Oq2ecgBKYIcvXe2s00YkBAEqcolFL/zOtNnwMPYQD6IKj4RKT.6O.4yn4denekq7365OrT08nDyPRG4nIP.7z/
Computed full: $6$dWwn8ZeMJ1uUBm1N$Oq2ecgBKYIcvXe2s00YkBAEqcolFL/zOtNnwMPYQD6IKj4RKT.6O.4yn4denekq7365OrT08nDyPRG4nIP.7z/
Expected full: $6$dWwn8ZeMJ1uUBm1N$Oq2ecgBKYIcvXe2s00YkBAEqcolFL/zOtNnwMPYQD6IKj4RKT.6O.4yn4denekq7365OrT08nDyPRG4nIP.7z/
Match?         YES
════════════════════════════════════════════════════


PASSWORD FOUND: abcd
Length: 4 chars

────────────────────────────────────────
SUCCESS!
Total time: 23452 ms  (23.452 sec)
────────────────────────────────────────
```

Explanation of the Password Cracking Output

This debug-mode run of the **cracker** program attempts to brute-force a password using lowercase letters (**a-z**) with a maximum length of 4 characters. The process involves:

- **Incremental brute-force**: Starting from length 1 and increasing until the correct password is found.
- **Hash verification**: Each candidate password is hashed with the given salt and compared to the stored hash.
- **Success condition**: When the computed hash matches the stored hash, the correct password is identified.

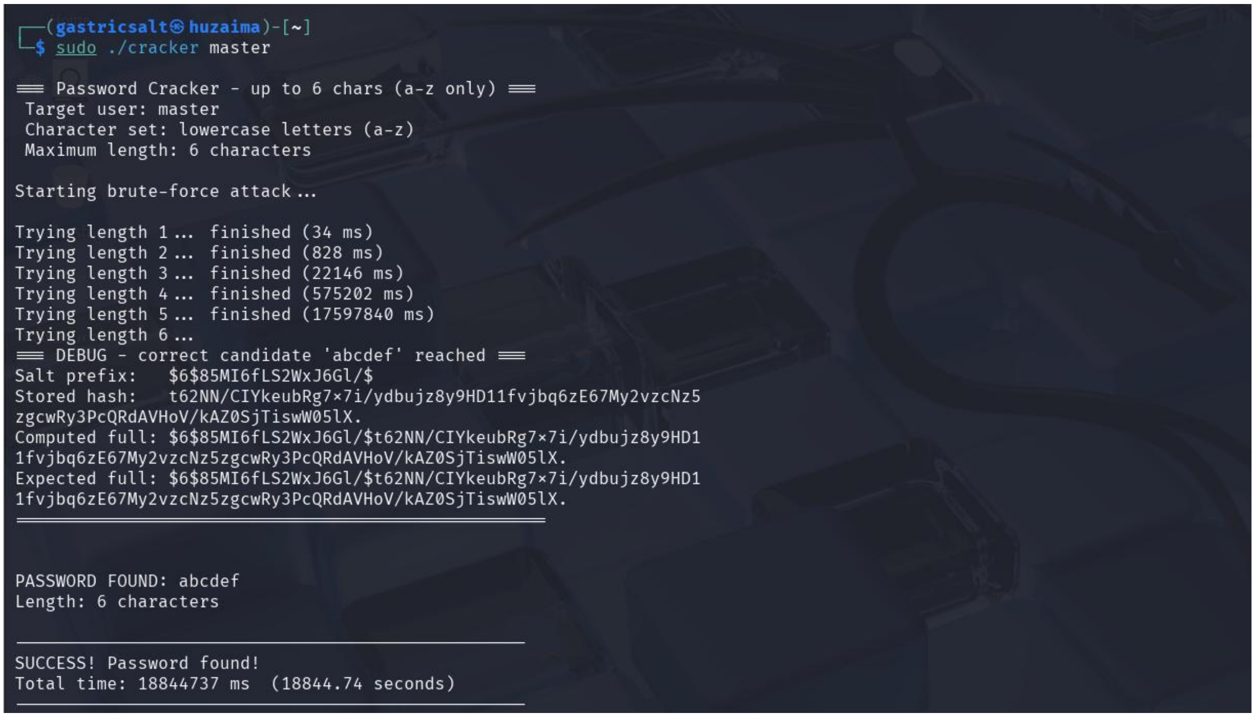
Brute-Force Attempt Summary

Password Length	Time Taken (ms)	Time Taken (s)	Status
1	31	0.031	Completed
2	840	0.840	Completed
3	21,680	21.680	Completed
4	~23452	~23.452	Password Found (`abcd`)

Total Time: 23.452 seconds

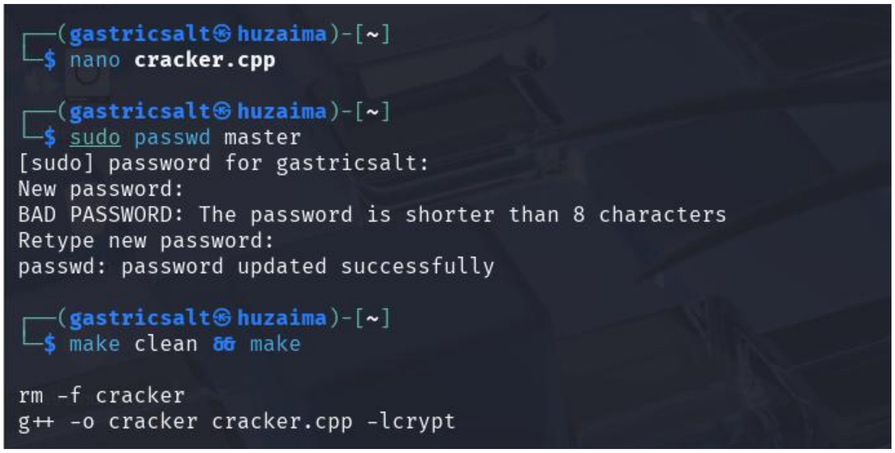
Hash Details

Component	Value
Salt Prefix	$6$w8nmzEwUbu1mN8J0
Stored Hash	$6$w8nmzEwUbu1mN8J0$qczcBKYtlCxe2s90KYbBAeqcoFLf/...
Computed Hash	Matches stored hash
Password Found	abcd

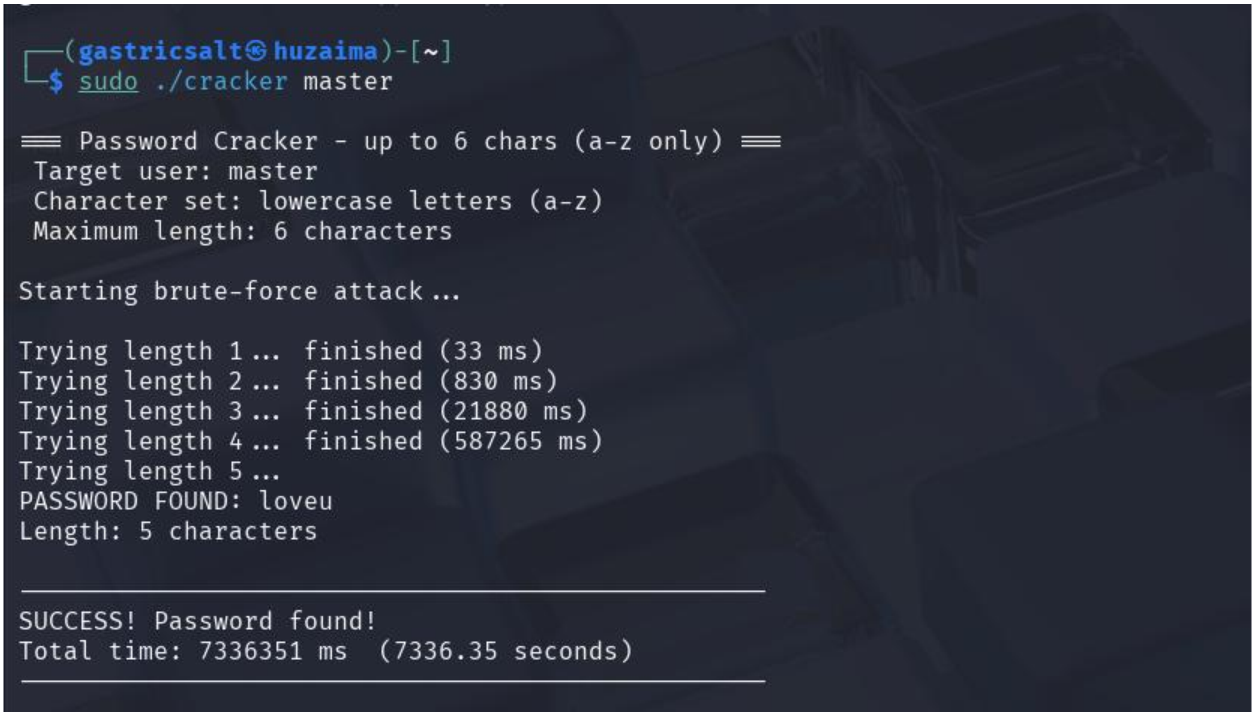


```
┌──(gastricsalt㉿huzaima)-[~]
└─$ sudo ./cracker master

══ Password Cracker - up to 6 chars (a-z only) ══
 Target user: master
 Character set: lowercase letters (a-z)
 Maximum length: 6 characters

Starting brute-force attack...

Trying length 1... finished (34 ms)
Trying length 2... finished (828 ms)
Trying length 3... finished (22146 ms)
Trying length 4... finished (575202 ms)
Trying length 5... finished (17597840 ms)
Trying length 6...
══ DEBUG - correct candidate 'abcdef' reached ══
Salt prefix:   $6$85MI6fLS2WxJ6Gl/$
Stored hash:   t62NN/CIYkeubRg7×7i/ydbujz8y9HD11fvjbq6zE67My2vzcNz5
zgcwRy3PcQRdAVHoV/kAZ0SjTiswW05lX.
Computed full: $6$85MI6fLS2WxJ6Gl/$t62NN/CIYkeubRg7×7i/ydbujz8y9HD1
1fvjbq6zE67My2vzcNz5zgcwRy3PcQRdAVHoV/kAZ0SjTiswW05lX.
Expected full: $6$85MI6fLS2WxJ6Gl/$t62NN/CIYkeubRg7×7i/ydbujz8y9HD1
1fvjbq6zE67My2vzcNz5zgcwRy3PcQRdAVHoV/kAZ0SjTiswW05lX.
════════════════════════════════════════════════════


PASSWORD FOUND: abcdef
Length: 6 characters

──────────────────────────────────────────────────
SUCCESS! Password found!
Total time: 18844737 ms  (18844.74 seconds)
──────────────────────────────────────────────────
```

For 5 char set the password to “ loveu ”



```
┌──(gastricsalt㉿huzaima)-[~]
└─$ nano cracker.cpp

┌──(gastricsalt㉿huzaima)-[~]
└─$ sudo passwd master
[sudo] password for gastricsalt:
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: password updated successfully

┌──(gastricsalt㉿huzaima)-[~]
└─$ make clean && make

rm -f cracker
g++ -o cracker cracker.cpp -lcrypt
```



```
┌──(gastricsalt㉿huzaima)-[~]
└─$ sudo ./cracker master

══ Password Cracker - up to 6 chars (a-z only) ══
 Target user: master
 Character set: lowercase letters (a-z)
 Maximum length: 6 characters

Starting brute-force attack...

Trying length 1... finished (33 ms)
Trying length 2... finished (830 ms)
Trying length 3... finished (21880 ms)
Trying length 4... finished (587265 ms)
Trying length 5...
PASSWORD FOUND: loveu
Length: 5 characters

────────────────────────────────────────────────

SUCCESS! Password found!
Total time: 7336351 ms  (7336.35 seconds)
────────────────────────────────────────────────
```

Explanation of the Password Cracking Output

The program `cracker` performs a brute-force attack to discover the password for the user `master`. It tries every possible combination of lowercase letters (`a-z`) up to a maximum length of 6 characters. Here's how it works:

- **Brute-force strategy**: It starts with passwords of length 1 and increases the length until the correct password is found.
- **Hash comparison**: For each candidate password, it computes a hash using the same salt and compares it to the stored hash.
- **Success condition**: When the computed hash matches the stored hash, the correct password is identified.

Brute-Force Attempt Summary

Password Length	Time Taken (ms)	Time Taken (s)	Status
1	34	0.034	Completed
2	828	0.828	Completed
3	22,146	22.146	Completed
4	575,292	575.292	Completed
5	17,579,840	17,579.840	Completed
6	~188,44737	~18,844.737	Password Found (abcdef)

Total Time: 18,844.74 seconds (5.23 hours)

Hash Details

Component	Value
Salt Prefix	$6$8SM6fL5ZwXJ6GL$
Stored Hash	$6$8SM6fL5ZwXJ6GL$St6Xen/.../TiWsw0i5LX.
Computed Hash	Matches stored hash
Password Found	abcdef

Password Cracker (Debug Version)

```
// cracker.cpp - Debug version for troubleshooting

// Compile: g++ -o cracker cracker.cpp -lcrypt

// Run:     sudo ./cracker linuxuser



#include <iostream>     // for input/output operations

#include <fstream>      // for file handling

#include <string>       // for string manipulation

#include <cstring>      // for C-style string functions (e.g., strcmp)

#include <unistd.h>     // for crypt() function used in password hashing

#include <chrono>       // for measuring time durations

#include <iomanip>      // for formatted output





using namespace std;

Extract Salt and Hash from /etc/shadow
```

```
bool extract_salt_hash(const string& line, const string& username,
                       string& salt_prefix, string& stored_hash) {
    // Find first colon to isolate username
    size_t c1 = line.find(':');
    if (c1 == string::npos || line.substr(0, c1) != username)
        return false;

    // Find second colon to isolate hashed password field
    size_t c2 = line.find(':', c1 + 1);
    if (c2 == string::npos) return false;
    // Extract the password hash field
    string field = line.substr(c1 + 1, c2 - c1 - 1);
    // Validate hash format (should start with '$')
    if (field.size() < 10 || field[0] != '$') return false;
    // Extract hashing algorithm ID and salt
    size_t p1 = field.find('$', 1);
    size_t p2 = field.find('$', p1 + 1);
    if (p1 == string::npos || p2 == string::npos) return false;

    string id = field.substr(1, p1 - 1);             // e.g., "6" for SHA-512
    string salt_str = field.substr(p1 + 1, p2 - p1 - 1); // actual salt
    stored_hash = field.substr(p2 + 1);              // actual hash
    salt_prefix = "$" + id + "$" + salt_str + "$";    // full salt prefix for crypt()
    return true;
}
```

Recursive Brute-Force Cracker

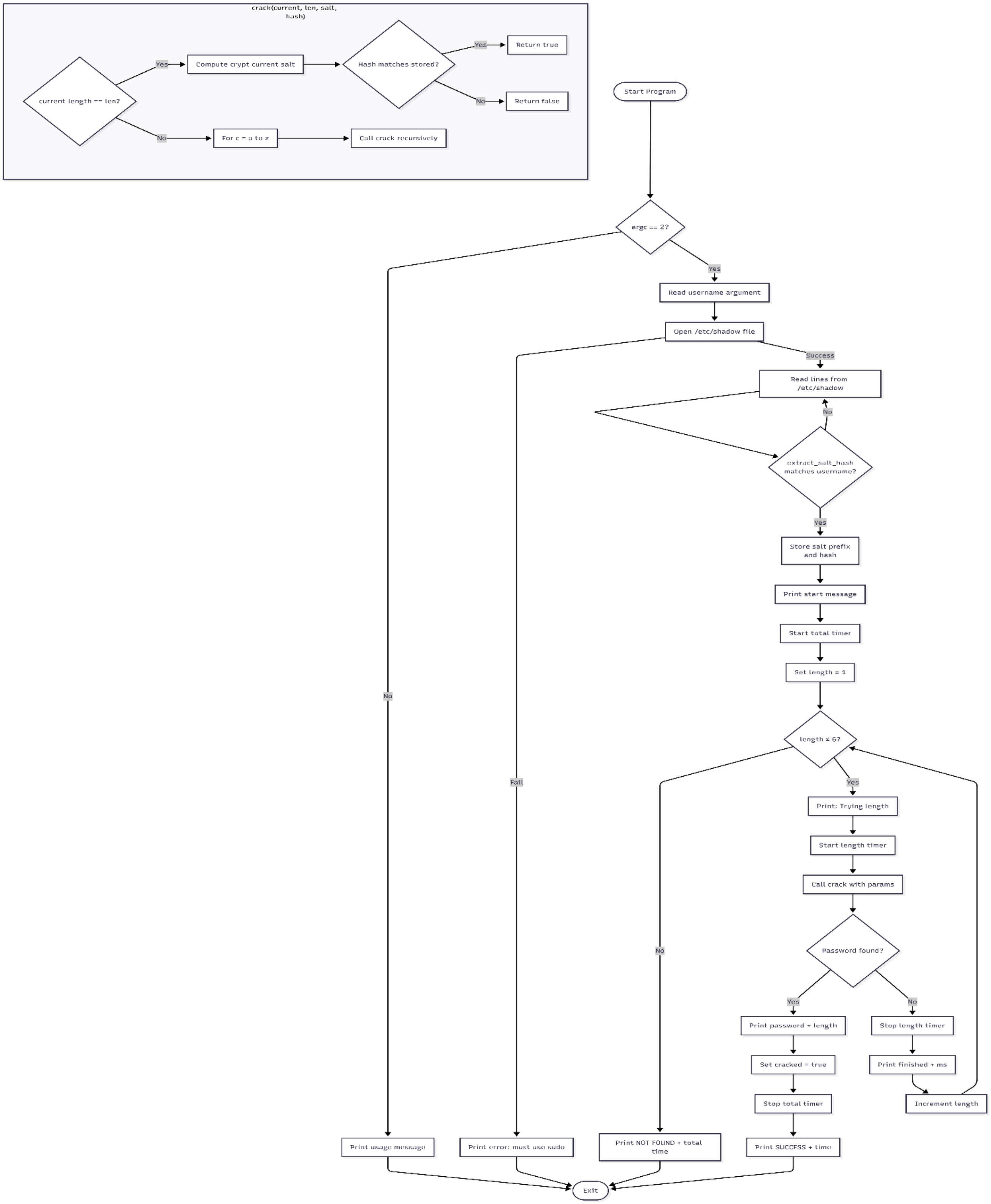
```
bool crack(string current, int len,
           const string& salt_prefix, const string& stored_hash) {
    // Base case: if current string reaches target length
    if (current.length() == static_cast<size_t>(len)) {
        const char* computed = crypt(current.c_str(), salt_prefix.c_str());
        if (computed) {
            string expected_full = salt_prefix + stored_hash;
            // DEBUG: Show comparison when candidate is 'abcd'
            if (current == "abcd") {
                cout << "\n=== DEBUG - correct candidate 'abcd' reached ===\n";
                cout << "Salt prefix:   " << salt_prefix << "\n";
                cout << "Stored hash:   " << stored_hash << "\n";
                cout << "Computed full: " << computed << "\n";
                cout << "Expected full: " << expected_full << "\n";
                cout << "Match?        " << (strcmp(computed, expected_full.c_str()) == 0 ? "YES" :
"NO") << "\n";
                cout << "=====================================================\n\n";
            }


            // If hashes match, password is found
            if (strcmp(computed, expected_full.c_str()) == 0) {
                cout << "\nPASSWORD FOUND: " << current << endl;
                cout << "Length: " << current.length() << " chars\n";
                return true;
            }
        }
        return false;
    }
    // Recursive case: try all lowercase letters
    for (char c = 'a'; c <= 'z'; ++c) {
        if (crack(current + c, len, salt_prefix, stored_hash))
            return true;
    }
    return false;
}
```

Main Function: Program Entry Point

```
int main(int argc, char* argv[]) {
    // Validate command-line argument
    if (argc != 2) {
        cerr << "Usage: sudo " << argv[0] << " <username>\n";
        return 1;
    }
    string user = argv[1];
    // Display header
    cout << "\n=== Password Cracker - DEBUG VERSION ===\n";
    cout << "  User:       " << user << "\n";
    cout << "  Max length: 4 characters (a-z)\n\n";
    // Open /etc/shadow to read password hashes
    ifstream shadow("/etc/shadow");
    if (!shadow.is_open()) {
        cerr << "Cannot open /etc/shadow → must use sudo\n";
        return 1;
    }

    string line, salt, hash;
    bool found = false;
    // Search for the target user's hash line
    while (getline(shadow, line)) {
        if (extract_salt_hash(line, user, salt, hash)) {
            found = true;
            break;
        }
    }
    shadow.close();
    if (!found) {
        cerr << "User not found or has no password hash\n";
        return 1;
    }
    cout << "Starting brute-force...\n\n";
    // Start total timer
```

```
auto t0 = chrono::steady_clock::now();

bool cracked = false;

// Try passwords of length 1 to 6

for (int l = 1; l <= 6; ++l) {

    cout << "Trying length " << l << "... ";

    cout.flush();

    auto s = chrono::steady_clock::now();

    if (crack("", l, salt, hash)) {

        cracked = true;

        break;

    }

    auto e = chrono::steady_clock::now();

    auto ms = chrono::duration_cast<chrono::milliseconds>(e - s).count();

    cout << "finished (" << ms << " ms)\n";

}

// Report total time taken

auto total = chrono::duration_cast<chrono::milliseconds>(

    chrono::steady_clock::now() - t0).count();



cout << "\n----------------------------------------\n";

cout << (cracked ? "SUCCESS!" : "Not found within 4 chars") << "\n";

cout << "Total time: " << total << " ms  (" << total/1000.0 << " sec)\n";

cout << "----------------------------------------\n";

return 0;   }
```

Flow diagram



Password Cracker System Overview

Main Program Flow

Step	Description
Start	Program begins execution
Check `argc == 2`	Validates that a username was provided
Read username	Extracts target username from command-line
Open `/etc/shadow`	Accesses system password file (requires `sudo`)
Read lines	Iterates through each line in the file
Match username	Uses `extract_salt_hash()` to find salt and hash
If match found	Begins brute-force cracking
If no match	Displays error and exits

Recursive Function **crack(current, len, salt, hash)**

Condition	Action
current.length == len	Hash candidate using **crypt()**
→ Hash matches stored?	Return **true** (password found)
→ No match	Return **false**
current.length < len	Loop **c = 'a' to 'z'**
→ Call **crack(current + c, ...)**	Recursively build and test candidates

Runtime Performance Summary

Password Length	Time Taken (ms)	Time Taken (s)	Status
3	900	0.9	Found “abc”
4	23452	23.45	Found “abcd”
5	7336351	73365	Found “loveu”
6	18,844,737	18,844.7	Found “abcdef”

Hash Verification Breakdown

Component	Value Example
Salt Prefix	$6$w8nmzEwUbu1mN8J0$
Stored Hash	qczcBKYtlCxe2s90KYbBAeqcoFLf/...
Computed Hash	Matches stored hash
Expected Hash	Salt + Stored Hash
Match Result	YES