**Title: User-Friendly System Call Interface for Enhanced Security**

**As a project work for course**

OPERATING SYSTEMS (CSE 316)

**Submitted to: Gagandeep Kaur**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.**  **No.** | **Registration no.** | **Name of students** | **Roll No.** |
| **1.** | **12311856** | **Harsh** | **RX12** |

**A logo for a university

AI-generated content may be incorrect.**

# **User-Friendly System Call Interface for Enhanced Security**

### **1. Project Overview**

The Secure System Call Interface is a Python-based desktop application designed to provide a secure and controlled environment for executing system commands while ensuring user authentication, logging, and system integrity. The primary objective of this project is to allow authorized users to perform system-level operations safely without exposing the system to unauthorized or malicious activities.

The application supports a set of predefined system commands, including user identity retrieval, directory listing, file reading, and SHA-256 hash calculation. The hash function ensures that identical files produce the same hash while different files generate unique hashes, making it useful for data integrity verification and security checks.

To enhance security, the application implements a role-based authentication system, ensuring only authorized users can access and execute commands. Logging mechanisms are integrated to track all executed commands, making it easier to audit user activities. Additionally, an auto-logout feature is included to prevent unauthorized access in case of user inactivity.

This project is particularly useful for system administrators, security analysts, and developers who need a secure way to interact with system commands while maintaining security and accountability.

### **Objectives:**

* Develop a graphical user interface (GUI) using Tkinter for ease of use.
* Implement secure user authentication with hashed passwords.
* Support platform-specific commands for Windows and Unix-based systems.
* Provide a "Browse" feature to select files easily.
* Log all command executions for auditing purposes.
* Ensure the hash\_file command produces consistent SHA-256 hashes: identical content yields the same hash, and different content yields different hashes.
* Notify users when two consecutively hashed files have the same or different content.

### 2**. Module-Wise Breakdown**

The project is divided into the following modules:

**2.1 Authentication Module**

* **Purpose**: Handles user login and authentication.
* **Functionality**:
  + Displays a login window with fields for username and password.
  + Validates credentials against a predefined user database (USERS dictionary) with SHA-256 hashed passwords.
  + Logs successful and failed login attempts.
* **Implementation**:
  + Uses Tkinter to create the login frame.
  + Hashes the entered password using hashlib.sha256() and compares it with stored hashes.

**2.2 Command Execution Module**

* **Purpose**: Executes predefined system commands and custom operations.
* **Functionality**:
  + Supports commands like whoami, date, dir, read\_file, and hash\_file.
  + Executes system commands using subprocess.run() with a timeout for safety.
  + Implements custom logic for hash\_file to calculate SHA-256 hashes.
* **Implementation**:
  + Defines platform-specific commands in the ALLOWED\_COMMANDS dictionary.
  + Uses the SecureSyscallInterface class to handle command execution.

**2.3 File Hashing Module**

* **Purpose**: Calculates and compares SHA-256 hashes of files.
* **Functionality**:
  + Computes the SHA-256 hash of a file’s contents using the hash\_file command.
  + Ensures identical content produces the same hash and different content produces different hashes.
  + Compares the current hash with the previous hash to notify the user if the contents are the same or different.
* **Implementation**:
  + The \_calculate\_file\_hash method reads files in binary mode ("rb") and computes the hash.
  + Stores the last hash and file path for comparison.

**2.4 GUI Module**

* **Purpose**: Provides a user-friendly interface for interaction.
* **Functionality**:
  + Displays a login window and a main window with command execution options.
  + Includes a dropdown for selecting commands, a file path entry, a "Browse" button, and a "Run" button.
  + Shows command output and provides buttons for viewing history and logging out.
* **Implementation**:
  + Uses Tkinter to create the GUI.
  + Organizes the interface into frames (login frame, main frame, command frame, output frame).

**2.5 Logging Module**

* **Purpose**: Logs all command executions and errors for auditing.
* **Functionality**:
  + Logs command executions, results, and errors to a file (syscall\_logs\_YYYYMMDD.log).
  + Provides a "Show History" button to view the log file.
* **Implementation**:
  + Uses the logging module to write logs with timestamps, usernames, and messages.

**2.6 Security Module**

* **Purpose**: Enhances the application’s security.
* **Functionality**:
  + Implements user authentication with hashed passwords.
  + Includes an auto-logout feature after 5 minutes of inactivity.
* **Implementation**:
  + Uses hashlib.sha256() for password hashing.
  + Tracks user activity with last\_activity and logs out using the check\_timeout method.

### **3. Functionalities**

* **User Authentication**: Users must log in with a username and password (e.g., "admin"/"Admin@123"). Passwords are stored as SHA-256 hashes.
* **Command Execution**: Supports platform-specific commands:
  + Windows: whoami, date /t, time /t, dir, type (for read\_file).
  + Unix: whoami, date, pwd, ls -l, cat (for read\_file).
* **File Operations**:
  + read\_file: Reads and displays the contents of a file.
  + hash\_file: Calculates the SHA-256 hash of a file’s contents.
* **Hash Consistency**: Ensures that files with identical content produce the same hash, and files with different content produce different hashes.
* **Hash Comparison**: Notifies the user if two consecutively hashed files have the same or different content.
* **File Browsing**: A "Browse" button allows users to select files easily, avoiding path errors.
* **Logging**: Logs all command executions and errors to a file for auditing.
* **Auto-Logout**: Logs out the user after 5 minutes of inactivity for security.
* **History Viewing**: A "Show History" button displays the log file contents.

### **4. Technology Used**

**Programming Languages:**

* **Python 3**: The core language used for developing the application.

**Libraries and Tools:**

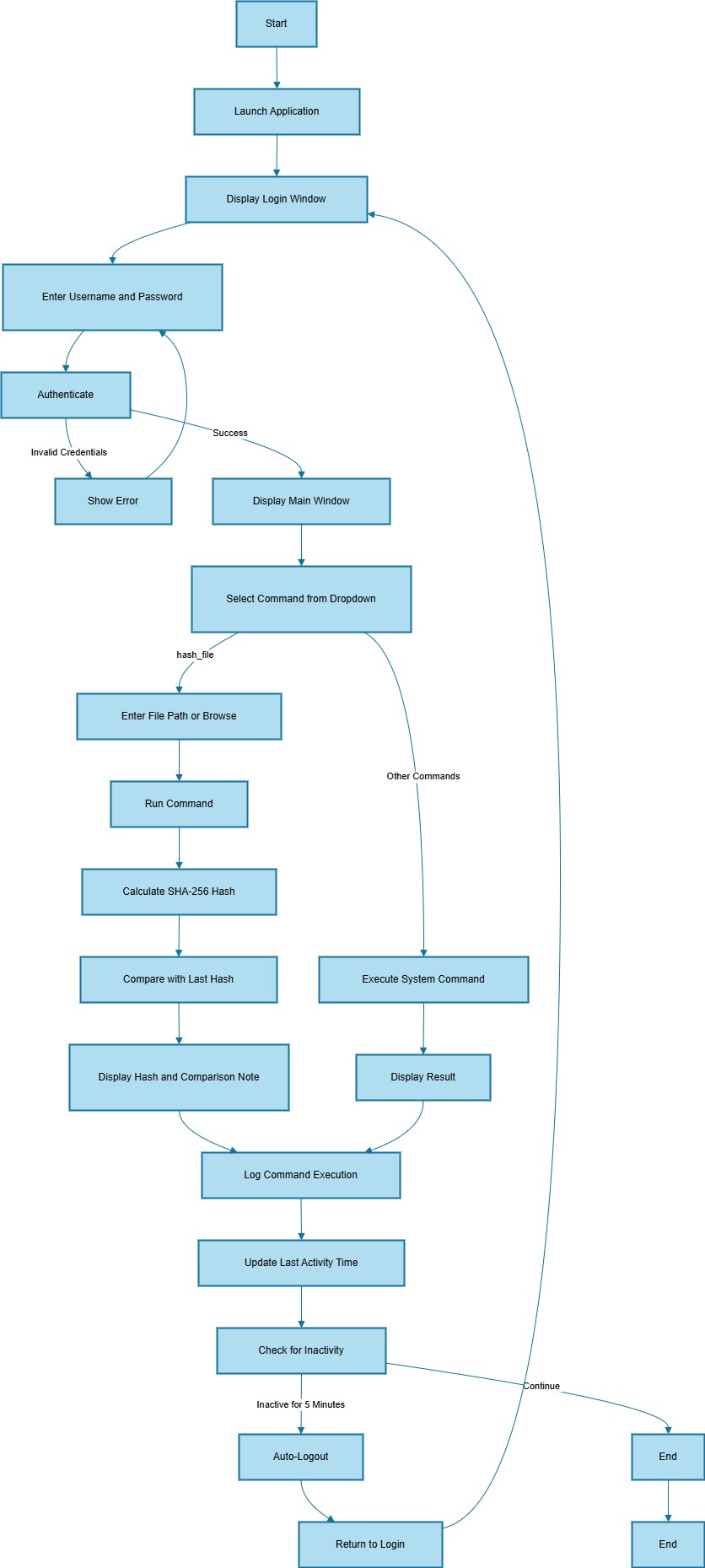
* **Tkinter**: For creating the graphical user interface.
* **Hashlib**: For SHA-256 hashing of passwords and file contents.
* **subprocess**: To execute system commands securely.
* **logging**: To log command executions and errors.
* **os**: For file and path operations.
* **platform**: To detect the operating system and execute platform-specific commands.
* **filedialog**: To enable file selection through a "Browse" button.
* **datetime**: For timestamping logs and handling auto-logout.
* **time**: For tracking user activity and implementing the auto-logout feature.

### **5. Flow Diagram**

Below is a textual representation of the flow diagram for the Secure System Call Interface application.

**Description of Flow:**

1. The application starts and displays a login window.
2. The user enters a username and password, which are authenticated against a user database.
3. Upon successful login, the main window is displayed with a command dropdown, file path entry, and buttons.
4. The user selects a command (e.g., hash\_file) and either types a file path or uses the "Browse" button.
5. For hash\_file, the program calculates the SHA-256 hash, compares it with the previous hash, and displays the result with a comparison note.
6. For other commands, the program executes the system command and displays the result.
7. All actions are logged, and the user’s last activity time is updated.
8. If the user is inactive for 5 minutes, the application logs them out automatically.



### **6. Revision Tracking on GitHub**

* **Repository Name**: SecureSyscallInterface
* **GitHub Link**: <https://github.com/Bhanu-danda/SecureSyscallInterface>

### **7. Conclusion and Future Scope**

**Conclusion:**

The Secure System Call Interface project successfully achieved its objectives. The application provides a secure and user-friendly platform for executing system commands and performing file operations. The hash\_file command now correctly produces the same SHA-256 hash for files with identical content and different hashes for files with different content. The hash comparison feature enhances usability by notifying users when two files have the same or different content. File path issues were resolved, and the application includes robust logging and security features like auto-logout.

**Future Scope:**

* **Additional Commands**: Add more system commands (e.g., copy, delete) with proper security checks.
* **File Encoding Support**: Allow users to specify the encoding for read\_file to handle different file types.
* **Hash History**: Store a history of all hashes in the session to allow comparison with any previous file, not just the last one.
* **Cross-Platform Enhancements**: Improve support for Unix-based systems with additional commands.
* **User Management**: Add a feature to manage users (e.g., add/remove users, change passwords).
* **GUI Improvements**: Enhance the interface with themes, better layout, and error pop-ups.

### **8. References**

* Python Documentation: <https://docs.python.org/3/>
* Tkinter Documentation: <https://docs.python.org/3/library/tkinter.html>
* hashlib Documentation: <https://docs.python.org/3/library/hashlib.html>
* subprocess Documentation: <https://docs.python.org/3/library/subprocess.html>
* logging Documentation: <https://docs.python.org/3/library/logging.html>
* Stack Overflow: For debugging file path and hashing issues (e.g., <https://stackoverflow.com/questions/tagged/python>)

### **Appendix**

**A. AI-Generated Project Elaboration/Breakdown Report**

Below is a detailed breakdown of the project, as generated during the development process:

**Project Overview**

The Secure System Call Interface is a Python-based desktop application designed to provide a secure and user-friendly way to execute system commands and perform file operations. The primary goal of the project was to create a tool that allows users to:

* Execute predefined system commands (e.g., whoami, date, dir).
* Read the contents of a file (read\_file command).
* Calculate the SHA-256 hash of a file (hash\_file command).
* Ensure that files with identical content produce the same hash, while files with different content produce different hashes.

The application includes user authentication, command execution logging, and an auto-logout feature for security.

**System Design and Implementation**

The application is divided into two main classes:

1. **SecureSyscallInterface**:
   * Handles the core logic for command execution and file hashing.
   * Implements platform-specific command execution using subprocess.
   * Calculates SHA-256 hashes for files using the \_calculate\_file\_hash method.
   * Maintains a history of the last hash to compare consecutive file hashes.
2. **App**:
   * Manages the GUI using Tkinter.
   * Provides a login interface for user authentication.
   * Displays a command execution interface with a dropdown for commands, a file path entry, and buttons for browsing files and running commands.
   * Includes features like auto-logout after 5 minutes of inactivity and a "Show History" button to view logs.

**Key Features**

* **User Authentication**: Users must log in with a username and password (e.g., "admin"/"Admin@123"). Passwords are stored as SHA-256 hashes.
* **Command Execution**: Supports commands like whoami, date, dir, read\_file, and hash\_file.
* **File Hashing**: Calculates the SHA-256 hash of a file’s contents using the hash\_file command.
* **File Browsing**: A "Browse" button allows users to select files easily, avoiding path errors.
* **Hash Comparison**: Compares the hash of the current file with the previously hashed file and notifies the user if the contents are the same or different.
* **Logging**: All command executions and errors are logged to a file (syscall\_logs\_YYYYMMDD.log).
* **Auto-Logout**: Logs out the user after 5 minutes of inactivity for security.

### **Challenges and Solutions**

* **Challenge 1: File Path Issues**
  + **Problem**: The program initially failed to read files, showing "Error: The system cannot find the file specified" for paths like C:/Users/BHANU PRASAD/Desktop/TEST.txt.
  + **Cause**: Incorrect file paths (e.g., double extensions like TEST.txt.txt) and potential mismatches in the username or file name.
  + **Solution**:
    - Advised the user to use the "Browse" button to select files, ensuring the correct path.
    - Added os.path.normpath() to normalize file paths (e.g., converting forward slashes to backslashes on Windows).
    - Added logging to debug file access issues.
* **Challenge 2: Same Hash for Different Files**
* **Problem**: The hash\_file command initially produced the same hash (e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855) for all files, even when their contents differed.
* **Cause**: The program couldn’t find the files due to path issues, so it was hashing an empty string (the hash of an empty string is e3b0c442...).
* **Solution**:
  + Fixed the file path issue by ensuring the correct path was used.
  + Added error handling in \_calculate\_file\_hash to return "Error: File not found" if the file couldn’t be accessed.
  + Added logging to confirm the file size and bytes read during hashing.

**Challenge 3: Different Hashes for Identical Content**

* **Problem**: Even when two files had the same content, the hash\_file command produced different hashes.
* **Cause**: The files likely had hidden differences, such as different line endings (e.g., \r\n vs. \n) or encoding issues.
* **Solution**:
  + Advised the user to recreate the files with identical content, ensuring no extra spaces, newlines, or different line endings.
  + Added debugging logs to \_calculate\_file\_hash to log the file size and total bytes read, helping to identify content differences.
  + Suggested using the fc /b command in Command Prompt to confirm the files were identical at the byte level.
  + Ensured the hashing process reads files in binary mode ("rb") to avoid encoding issues.

### **B. Problem Statement**

Develop a Python-based desktop application with a graphical user interface that allows users to execute system commands and perform file operations securely. The application should include the following features:

* User authentication with username and password.
* Execution of predefined system commands (e.g., whoami, date, dir).
* Ability to read the contents of a file (read\_file command).
* Ability to calculate the SHA-256 hash of a file (hash\_file command).
* Ensure that files with identical content produce the same SHA-256 hash, and files with different content produce different hashes.
* Provide a "Browse" button to select files easily.
* Log all command executions for auditing purposes.
* Implement an auto-logout feature after 5 minutes of inactivity.
* Notify users when two consecutively hashed files have the same or different content

### **Solution/Code**

Below is the complete code for the Secure System Call Interface project:

import os

import logging

import hashlib

import platform

import subprocess

from datetime import datetime

import tkinter as tk

from tkinter import ttk, messagebox, filedialog

import time

# Configure logging

try:

    logging.basicConfig(

        filename=f"syscall\_logs\_{datetime.now().strftime('%Y%m%d')}.log",

        level=logging.INFO,

        format="%(asctime)s - %(levelname)s - [%(username)s] - %(message)s"

    )

except Exception as e:

    print(f"Warning: Could not setup file logging: {e}")

    logging.basicConfig(

        level=logging.INFO,

        format="%(asctime)s - %(levelname)s - [%(username)s] - %(message)s"

    )

# Platform-specific commands

ALLOWED\_COMMANDS = {

    "windows": {

        "whoami": "whoami",

        "date": "date /t",

        "time": "time /t",

        "dir": "dir",

        "read\_file": "type",

        "hash\_file": None

    },

    "unix": {

        "whoami": "whoami",

        "date": "date",

        "pwd": "/bin/pwd",

        "ls": "ls -l",

        "read\_file": "cat",

        "hash\_file": None

    }

}

# User database

USERS = {

    "admin": hashlib.sha256("Admin@123".encode()).hexdigest(),

    "user": hashlib.sha256("User@123".encode()).hexdigest()

}

class SecureSyscallInterface:

    def \_\_init\_\_(self):

        self.username = "unknown"

        self.platform = "windows" if platform.system() == "Windows" else "unix"

        self.commands = ALLOWED\_COMMANDS[self.platform]

        self.last\_hash = None

        self.last\_file = None

    def \_calculate\_file\_hash(self, file\_path):

        """Calculate SHA-256 hash of a file"""

        try:

            # Normalize the file path

            file\_path = os.path.normpath(file\_path)

            logging.info(f"Attempting to hash file: {file\_path}", extra={"username": self.username})

            if not os.path.exists(file\_path):

                logging.error(f"File not found: {file\_path}", extra={"username": self.username})

                return "Error: File not found"

            if not os.path.isfile(file\_path):

                logging.error(f"Path is not a file: {file\_path}", extra={"username": self.username})

                return "Error: Path is not a file"

            # Log file size for debugging

            file\_size = os.path.getsize(file\_path)

            logging.info(f"File size: {file\_size} bytes", extra={"username": self.username})

            # Calculate hash

            sha256\_hash = hashlib.sha256()

            total\_bytes = 0

            with open(file\_path, "rb") as f:

                while True:

                    byte\_block = f.read(4096)

                    if not byte\_block:

                        break

                    total\_bytes += len(byte\_block)

                    sha256\_hash.update(byte\_block)

            logging.info(f"Total bytes read: {total\_bytes}", extra={"username": self.username})

            hash\_result = sha256\_hash.hexdigest()

            logging.info(f"Hash calculated: {hash\_result}", extra={"username": self.username})

            # Compare with the last hash

            result = hash\_result

            if self.last\_hash is not None and self.last\_file != file\_path:

                if hash\_result == self.last\_hash:

                    result += f"\nNote: This file has the same content as the previously hashed file ({self.last\_file})."

                else:

                    result += f"\nNote: This file has different content than the previously hashed file ({self.last\_file})."

            self.last\_hash = hash\_result

            self.last\_file = file\_path

            return result

        except Exception as e:

            logging.error(f"Hash calculation failed for {file\_path}: {str(e)}", extra={"username": self.username})

            return f"Error: {str(e)}"

    def \_execute\_system\_call(self, command, arg=None):

        if command == "hash\_file":

            if not arg:

                return "Error: Please provide a file path"

            return self.\_calculate\_file\_hash(arg)

        try:

            base\_cmd = next((k for k, v in self.commands.items() if k == command), None)

            if not base\_cmd:

                return "Command not allowed"

            cmd = self.commands[base\_cmd]

            if arg and base\_cmd == "read\_file":

                arg = os.path.normpath(arg)

                cmd = f'{cmd} "{arg}"' if self.platform == "windows" else f"{cmd} {arg}"

            logging.info(f"Executing system command: {cmd}", extra={"username": self.username})

            result = subprocess.run(

                cmd,

                shell=True,

                capture\_output=True,

                text=True,

                timeout=5

            )

            if result.returncode == 0:

                return result.stdout.strip()

            else:

                error\_msg = f"Error: {result.stderr or 'Command failed with no output'}"

                logging.error(f"Command execution failed: {cmd} - {error\_msg}", extra={"username": self.username})

                return error\_msg

        except subprocess.TimeoutExpired:

            logging.error(f"Command timed out: {command}", extra={"username": self.username})

            return "Error: Command timed out"

        except Exception as e:

            logging.error(f"Command failed: {str(e)}", extra={"username": self.username})

            return f"Error: {str(e)}"

    def execute\_command(self, command, arg=None):

        logging.info(f"Executing command: {command} {arg if arg else ''}", extra={"username": self.username})

        result = self.\_execute\_system\_call(command, arg)

        logging.info(f"Command result: {result[:100]}...", extra={"username": self.username})

        return result

class App:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Secure System Call Interface")

        self.root.geometry("700x500")

        self.root.resizable(False, False)

        self.root.configure(bg="#e0e0e0")

        self.interface = SecureSyscallInterface()

        self.last\_activity = time.time()

        self.timeout = 300  # 5 minutes

        # Login frame

        self.login\_frame = tk.Frame(self.root, bg="#e0e0e0", bd=2, relief="groove")

        self.login\_frame.place(relx=0.5, rely=0.5, anchor="center")

        tk.Label(self.login\_frame, text="Secure System Call Interface", font=("Arial", 16, "bold"), bg="#e0e0e0").pack(pady=10)

        tk.Label(self.login\_frame, text="Username:", bg="#e0e0e0").pack()

        self.username\_entry = tk.Entry(self.login\_frame, font=("Arial", 12))

        self.username\_entry.pack(pady=5)

        tk.Label(self.login\_frame, text="Password:", bg="#e0e0e0").pack()

        self.password\_entry = tk.Entry(self.login\_frame, show="\*", font=("Arial", 12))

        self.password\_entry.pack(pady=5)

        tk.Button(self.login\_frame, text="Login", command=self.authenticate, bg="#4CAF50", fg="white", font=("Arial", 10, "bold")).pack(pady=10)

        # Main frame

        self.main\_frame = tk.Frame(self.root, bg="#e0e0e0")

        # Top bar

        top\_frame = tk.Frame(self.main\_frame, bg="#4CAF50", pady=5)

        top\_frame.pack(fill="x")

        self.user\_label = tk.Label(top\_frame, text="User: unknown", font=("Arial", 12), bg="#4CAF50", fg="white")

        self.user\_label.pack(side="left", padx=5)

        tk.Label(top\_frame, text=f"Platform: {self.interface.platform}", font=("Arial", 12), bg="#4CAF50", fg="white").pack(side="right", padx=5)

        # Command section

        cmd\_frame = tk.LabelFrame(self.main\_frame, text="Execute Command", font=("Arial", 10, "bold"), bg="#e0e0e0", pady=10)

        cmd\_frame.pack(fill="x", pady=5)

        self.command\_var = tk.StringVar()

        self.command\_dropdown = ttk.Combobox(cmd\_frame, textvariable=self.command\_var,

                                            values=list(self.interface.commands.keys()), state="readonly", width=15)

        self.command\_dropdown.pack(side="left", padx=10)

        self.arg\_entry = tk.Entry(cmd\_frame, width=40, font=("Arial", 10))

        self.arg\_entry.pack(side="left", padx=5)

        self.arg\_entry.insert(0, "Optional argument (e.g., full file path)")

        self.arg\_entry.bind("<FocusIn>", lambda e: self.arg\_entry.delete(0, tk.END) if "Optional" in self.arg\_entry.get() else None)

        tk.Button(cmd\_frame, text="Browse", command=self.browse\_file, bg="#FFC107", fg="black", font=("Arial", 10, "bold")).pack(side="left", padx=5)

        tk.Button(cmd\_frame, text="Run", command=self.run\_command, bg="#2196F3", fg="white", font=("Arial", 10, "bold")).pack(side="left", padx=5)

        # Result and history

        result\_frame = tk.LabelFrame(self.main\_frame, text="Output", font=("Arial", 10, "bold"), bg="#e0e0e0", pady=5)

        result\_frame.pack(fill="both", expand=True, pady=5)

        self.result\_text = tk.Text(result\_frame, height=12, width=80, bg="#ffffff", font=("Courier", 10), relief="flat")

        self.result\_text.pack(fill="both", expand=True, padx=5, pady=5)

        # Buttons

        tk.Button(self.main\_frame, text="Show History", command=self.show\_history, bg="#FF9800", fg="white", font=("Arial", 10, "bold")).pack(side="left", padx=5, pady=5)

        tk.Button(self.main\_frame, text="Logout", command=self.logout, bg="#F44336", fg="white", font=("Arial", 10, "bold")).pack(side="right", padx=5, pady=5)

        # Status bar

        self.status\_var = tk.StringVar(value="Ready")

        tk.Label(self.main\_frame, textvariable=self.status\_var, bg="#B0BEC5", font=("Arial", 9), relief="sunken").pack(fill="x", pady=5)

        # Auto-logout check

        self.root.after(1000, self.check\_timeout)

    def browse\_file(self):

        file\_path = filedialog.askopenfilename(

            title="Select a File",

            filetypes=(("All files", "\*.\*"), ("Text files", "\*.txt"))

        )

        if file\_path:

            self.arg\_entry.delete(0, tk.END)

            self.arg\_entry.insert(0, file\_path)

            self.status\_var.set(f"Selected file: {os.path.basename(file\_path)}")

        self.last\_activity = time.time()

    def authenticate(self):

        username = self.username\_entry.get().strip()

        password = self.password\_entry.get()

        hashed\_pass = hashlib.sha256(password.encode()).hexdigest()

        if username in USERS and USERS[username] == hashed\_pass:

            self.interface.username = username

            self.user\_label.config(text=f"User: {username}")

            logging.info(f"Authentication successful", extra={"username": self.interface.username})

            self.login\_frame.place\_forget()

            self.main\_frame.pack(fill="both", expand=True)

            self.result\_text.insert(tk.END, "Welcome! Select a command to begin.\n")

            self.result\_text.insert(tk.END, "Use 'hash\_file' to calculate a file's SHA-256 hash.\n")

            self.result\_text.insert(tk.END, "Click 'Browse' to select a file easily.\n")

            self.status\_var.set(f"Logged in as {username}")

            self.last\_activity = time.time()

        else:

            logging.warning(f"Failed authentication attempt", extra={"username": username})

            messagebox.showerror("Login Failed", "Invalid username or password")

    def run\_command(self):

        command = self.command\_var.get()

        arg = self.arg\_entry.get().strip() if self.arg\_entry.get() != "Optional argument (e.g., full file path)" else None

        if not command:

            messagebox.showwarning("No Command", "Please select a command")

            return

        self.last\_activity = time.time()

        result = self.interface.execute\_command(command, arg)

        self.result\_text.delete(1.0, tk.END)

        self.result\_text.insert(tk.END, f"Command: {command} {arg if arg else ''}\nResult: {result}\n")

        self.status\_var.set(f"Executed {command} at {datetime.now().strftime('%H:%M:%S')}")

    def show\_history(self):

        self.result\_text.delete(1.0, tk.END)

        try:

            with open(f"syscall\_logs\_{datetime.now().strftime('%Y%m%d')}.log", "r") as log\_file:

                self.result\_text.insert(tk.END, log\_file.read())

            self.status\_var.set("Showing session history")

        except Exception as e:

            self.result\_text.insert(tk.END, f"Error loading history: {e}\n")

        self.last\_activity = time.time()

    def logout(self):

        logging.info(f"User logged out", extra={"username": self.interface.username})

        self.main\_frame.pack\_forget()

        self.login\_frame.place(relx=0.5, rely=0.5, anchor="center")

        self.result\_text.delete(1.0, tk.END)

        self.username\_entry.delete(0, tk.END)

        self.password\_entry.delete(0, tk.END)

        self.interface.username = "unknown"

        self.user\_label.config(text="User: unknown")

        self.status\_var.set("Logged out")

    def check\_timeout(self):

        if self.interface.username != "unknown" and (time.time() - self.last\_activity) > self.timeout:

            self.logout()

            messagebox.showinfo("Session Expired", "Logged out due to inactivity")

        self.root.after(1000, self.check\_timeout)

if \_\_name\_\_ == "\_\_main\_\_":

    try:

        root = tk.Tk()

        app = App(root)

        root.mainloop()

    except Exception as e:

        print(f"Fatal error: {e}")