**PROJECT**

**ON**

**“CCH FORENSICS TOOLKIT:**

**UIUX**

**& FRONTEND”**

**SUBMITTED TO**

**CYBER CRIME HELPLINE LLP**

**DATE OF SUBMISSION:14/11/2024 FULL NAME: ANAND SACHIN GAIKWAD**

**MOBILE NUMBER:** **9370150563**

**AADHAR CARD NO.** **7832 6086 0521**

A. Aim

The aim of this project is to develop a user-friendly, highly functional UI/UX for a digital forensics application, named CCH Forensics Toolkit. This toolkit will assist Cyber Crime Helpline LLP in conducting various forensic tasks, including file hashing, metadata analysis, system auditing, and log extraction. By integrating these processes within a cohesive and visually effective interface, the toolkit aims to streamline cyber investigations, data gathering, and evidence preservation in digital forensics.

B. Introduction

The CCH Forensics Toolkit is an integrated digital forensics application created for Cyber Crime Helpline LLP. It automates essential cyber forensic processes such as file hashing, metadata extraction, log file analysis, and data extraction. Built with Python’s Tkinter library for its graphical user interface, this toolkit provides investigators with the necessary tools to efficiently analyze and gather digital evidence. The toolkit leverages standalone forensic executable files for fast, reliable analysis while ensuring the data’s integrity is preserved and stored in a structured format for legal and investigative use.

This project focuses on delivering a user-friendly, efficient, and tailored toolkit that aids in handling critical forensic tasks seamlessly. The design prioritizes accessibility, operational efficiency, and data reliability, particularly beneficial for field-based cyber investigations.

C. Requirements

Hardware Requirements:

- Processor: Intel Core i5 or above

- RAM: Minimum 4 GB

- Storage: At least 100 MB of free space

- Operating System: Windows 10 or later

Software Requirements:

- Programming Language: Python 3.x

- Libraries: Tkinter, Pillow (PIL) for image handling, subprocess, os

- Forensic Tools: Executable files for forensic tasks (e.g., `Hash.exe`, `MetaData.exe`)

- Image Editing Software: (Optional) Photoshop, Paint, or similar software for logo customization and editing

**Required Installations**

* Pillow (mandatory for image handling)
* Pandas (optional, depending on your data manipulation needs)
* NumPy (optional, useful for numerical computations)
* Requests (optional, for API interactions)
* Matplotlib (optional, for future data visualization needs)

D. Methodology

1. Designing the User Interface (UI/UX):

- The interface was crafted using Python’s Tkinter library, offering a structured layout with sections dedicated to each forensic task (hashing, metadata extraction, auditing, etc.). The toolkit’s layout was optimized for ease of use and visibility, featuring grid-based button placement.

- Customization Features:

- Buttons: Styled with custom gradient effects for better interactivity and a polished appearance.

- Logos: Company branding integrated using Pillow, maintaining visual consistency.

- Scrolling Marquee: A marquee bar at the bottom of the UI continuously displays important messages to users.

2. Functional Button Implementation:

- Each button links to a specific .exe file representing a forensic task. Upon button click, the respective executable is triggered using Python’s `subprocess` module, and output is stored in a designated results folder.

- Folder Selection: The "Data Folder" button allows users to select a directory, making it simple to designate input files for forensic processing.

3. Handling and Organizing Results:

- The output generated from each forensic tool (e.g., hashing, metadata extraction) is saved in a dynamically created “Results” folder, named based on the computer’s identification. This folder can be accessed via the "Results Folder" button, enhancing the toolkit’s ease of organization and evidence storage.

- Dynamic Folder Creation: Through Python’s `os` module, the results folder is automatically created if absent, minimizing setup effort for users.

4. Ensuring Smooth Execution with Error Handling:

- The toolkit includes error management to alert users if any task encounters an issue, preventing the application from crashing and ensuring users are informed of the status of each forensic process.

5. Visual Customization for Professional Appearance:

- For a professional look, fonts, colors, and logos were meticulously selected, yielding a sleek and cohesive UI. The design prioritizes clarity, making it accessible and intuitive for users conducting time-sensitive forensic work.

E. Source Code

```python

import sys

import os

import subprocess

import tkinter as tk

from tkinter import messagebox, filedialog

from PIL import Image, ImageTk  # Make sure PIL is installed: pip install Pillow

import json

# Load configuration from a JSON file

def load\_config():

try:

with open('config.json', 'r') as f:

return json.load(f)

except FileNotFoundError:

messagebox.showerror("Error", "Configuration file not found.")

sys.exit()

except json.JSONDecodeError:

messagebox.showerror("Error", "Error reading configuration file.")

sys.exit()

# Function to find a file in multiple potential paths

def find\_file(filename):

current\_directory = os.path.dirname(os.path.abspath(\_\_file\_\_))

potential\_paths = [

os.path.join(current\_directory, 'src', filename),

os.path.join(current\_directory, '', filename),

os.path.join(current\_directory, filename),

filename,

os.path.join(os.getenv('USERPROFILE'), filename),

os.path.join('E:\\CYBERCRIMEHELPLINE', filename)

]

for path in potential\_paths:

if os.path.isfile(path):

return path

raise FileNotFoundError(f"{filename} not found in any of the specified paths.")

def get\_script\_dir():

if getattr(sys, 'frozen', False):  # Check if running as an EXE

return sys.\_MEIPASS  # PyInstaller uses this for the temporary folder path

else:

return os.path.dirname(os.path.abspath(\_\_file\_\_))

def find\_and\_execute\_exe(exe\_name, task\_name):

# Adjust the directory to account for EXE or script path

base\_dir = get\_script\_dir()

AllExeFiles = os.path.join(base\_dir, 'AllExeFiles')

exe\_path = os.path.join(AllExeFiles, exe\_name)

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

messagebox.showerror("Error", f"{exe\_name} not found at {exe\_path}.")

return

# Ensure RESULTS\_folder is based in the script’s directory

computer\_name = os.getenv("COMPUTERNAME")

pendrive\_path = 'E:\\COMPUTERNAME'

RESULTS\_folder = os.path.join(pendrive\_path, computer\_name)

os.makedirs(RESULTS\_folder, exist\_ok=True)

# Create the RESULTS file path

RESULTS\_file = os.path.join(RESULTS\_folder, f"{task\_name}\_results.txt")

# Update command to redirect output to RESULTS\_file

command = f'"{exe\_path}" > "{RESULTS\_file}" 2>&1'

# Debugging: Print command for verification

print(f"Executing command: {command}")

try:

# Start the process

subprocess.Popen(command, shell=True, cwd=AllExeFiles)

messagebox.showinfo("Success", f"{task\_name} process started. RESULTS saved to: {RESULTS\_file}")

except Exception as e:

messagebox.showerror("Error", f"Error executing {task\_name} process: {str(e)}")

# Function to open the RESULTS folder

def open\_RESULTS\_folder():

computer\_name = os.getenv("COMPUTERNAME")

pendrive\_path = 'E:\\COMPUTERNAME'

RESULTS\_folder = os.path.join(pendrive\_path, computer\_name)

if not os.path.exists(RESULTS\_folder):

try:

os.makedirs(RESULTS\_folder)

messagebox.showinfo("Info", f"RESULTS folder created: {RESULTS\_folder}")

except Exception as e:

messagebox.showerror("Error", f"Failed to create RESULTS folder: {str(e)}")

return

subprocess.Popen(f'explorer "{RESULTS\_folder}"')

# Custom GradientButton class for styled buttons

class GradientButton(tk.Frame):

def \_\_init\_\_(self, parent, text, command=None, width=20):

super().\_\_init\_\_(parent, bg="#000000")

self.button = tk.Button(self, text=text, command=command,

font=("Comic Sans MS", 12, "bold"),

bg="#FFEB3B", fg="black",

borderwidth=0, padx=12, pady=12,

activebackground="#FFD54F",

highlightthickness=0,

relief="flat",

width=width,

anchor='center')

self.button.pack(expand=True, fill='both')

self.button.bind("<Enter>", self.on\_hover)

self.button.bind("<Leave>", self.on\_leave)

self.button.bind("<ButtonPress>", self.on\_click)

self.button.bind("<ButtonRelease>", self.on\_release)

def on\_hover(self, event):

self.button.config(bg="#FFD54F")

def on\_leave(self, event):

self.button.config(bg="#FFEB3B")

def on\_click(self, event):

self.button.config(bg="#FFC107")

def on\_release(self, event):

self.button.config(bg="#FFD54F")

# Function to browse folders

def browse\_folder(folder\_type):

folder\_path = filedialog.askdirectory()

if folder\_path:

messagebox.showinfo(f"{folder\_type} Folder Selected", f"You selected: {folder\_path}")

# Function to confirm exit

def confirm\_exit():

if messagebox.askokcancel("Quit", "Do you really want to quit?"):

root.destroy()

# Updated function for DATA FOLDER to fetch selected file for Hash and Meta Data

def data\_folder\_function():

# Open a file dialog to select a file

selected\_file = filedialog.askopenfilename()

if selected\_file:

# Call the hash and meta data functions for the selected file

find\_and\_execute\_exe("HASH VALUE.exe", "Hash Value")

find\_and\_execute\_exe("META DATA.exe", "Meta Data")

# Create the main window

root = tk.Tk()

root.title("Cyber Crime Helpline LLP - Digital Forensics Tool")

# Disable window resizing

root.resizable(False, False)

# Adjusting window size

top\_margin = 100

bottom\_margin = 100

left\_margin = 150

right\_margin = 150

root.geometry(f"{1400 - left\_margin - right\_margin}x{900 - top\_margin - bottom\_margin}")

root.configure(bg="#000000")

# Configure grid layout

for i in range(3):

root.grid\_columnconfigure(i, weight=1)

# Title and Subtitle Labels

title\_label = tk.Label(root, text="CYBER CRIME HELPLINE LLP", font=("Comic Sans MS", 32, "bold"), fg="red", bg="#000000")

title\_label.grid(row=0, column=0, columnspan=3, pady=(22, 5), sticky="n")

subtitle\_label = tk.Label(root, text="DIGITAL FORENSICS TOOLKIT", font=("Comic Sans MS", 22, "bold"), fg="#F7F7F7", bg="#000000")

subtitle\_label.grid(row=1, column=0, columnspan=3, pady=(0, 5), sticky="n")

# Load and display logo

try:

left\_image\_path = find\_file('cchlogo.jpg')

left\_image = Image.open(left\_image\_path)

except FileNotFoundError as e:

messagebox.showerror("Error", str(e))

root.quit()

left\_logo = ImageTk.PhotoImage(left\_image)

left\_logo\_label = tk.Label(root, image=left\_logo, bg="#000000")

left\_logo\_label.grid(row=0, column=0, sticky="nw", padx=(30, 0), pady=(15, 0))

# Dashed line above folder buttons

top\_dashed\_line = tk.Label(root, text="===================================================================",

font=("Comic Sans MS", 12, "bold"), fg="white", bg="#000000")

top\_dashed\_line.grid(row=5, column=0, columnspan=3, pady=(20, 5), sticky="nsew")

# Button Data

button\_data = [

("HASH VALUE", "Hash Value.exe"),

("META DATA", "Meta Data.exe"),

("WINDOWS AUDIT", "Windows Audit.exe"),

("ACTIVITY DETAILS", "Activity Details.exe"),

("WIRELESS DETAILS", "Wireless Details.exe"),

("SERVER AUDIT", "Server Audit.exe"),

("BROWSER FORENSICS", "Browser Forensics.exe"),

("FILES EXTRACTION", "Files Extraction.exe"),

("COMPLETE FORENSICS", "Complete Forensics.exe")

]

# Set width for all buttons in button\_data

button\_width = 15

# Create buttons in a grid

for index, (text, exe\_name) in enumerate(button\_data):

row = index // 3

col = index % 3

button\_frame = tk.Frame(root, bg="black")

button\_frame.grid(row=row + 2, column=col, padx=10, pady=10, sticky="nsew")

button = GradientButton(button\_frame, text,

command=lambda exe\_name=exe\_name, text=text: find\_and\_execute\_exe(exe\_name, text),

width=button\_width)

button.button.config(font=("Comic Sans MS", 12, "bold"), padx=8, pady=8)

button.pack(expand=True, fill='both')

# Frame to hold the DATA FOLDER and RESULTS FOLDER buttons side by side

folders\_frame = tk.Frame(root, bg="#000000")

folders\_frame.grid(row=6, column=0, columnspan=3, pady=(30, 10))

# Updated DATA FOLDER button with new function

data\_folder\_button = GradientButton(folders\_frame, "DATA FOLDER", command=data\_folder\_function, width=20)

data\_folder\_button.pack(side="left", padx=10, fill='both')

# RESULTS FOLDER button

results\_folder\_button = GradientButton(folders\_frame, "RESULTS FOLDER", command=open\_RESULTS\_folder, width=20)

results\_folder\_button.pack(side="left", padx=10, fill='both')

# Bottom dashed line

bottom\_dashed\_line = tk.Label(root, text="===================================================================",

font=("Comic Sans MS", 12, "bold"), fg="white", bg="#000000")

bottom\_dashed\_line.grid(row=7, column=0, columnspan=3, pady=(20, 5), sticky="nsew")

# Smooth marquee scroll function

def smooth\_marquee\_scroll():

current\_x = marquee\_label.winfo\_x()

# If the text has fully moved off the left side, reset it to the right side

if current\_x < -marquee\_label.winfo\_width():

marquee\_label.place(x=root.winfo\_width())

else:

marquee\_label.place(x=current\_x - 2)  # Move 2 pixels to the left for smooth scrolling

root.after(50, smooth\_marquee\_scroll)  # Adjust timing for smoothness (20ms interval)

# Updated marquee configuration using pixel-based scrolling

marquee\_text = "THIS IS A PROPRIETARY TOOL OF CYBER CRIME HELPLINE LLP. FOR COURSES AND INVESTIGATIONS, WHATSAPP ON +91 9595427200.THIS IS A PROPRIETARY TOOL OF CYBER CRIME HELPLINE LLP. FOR COURSES AND INVESTIGATIONS, WHATSAPP ON +91 9595427200.  " \* 3

marquee\_label = tk.Label(root, text=marquee\_text, font=("Comic Sans MS", 18, "bold"), fg="white", bg="#000000", anchor='w')

# Set the initial position of the marquee label outside the visible area on the right

marquee\_label.place(x=root.winfo\_width(), y=650)  # Adjust y as per your grid or layout

smooth\_marquee\_scroll()

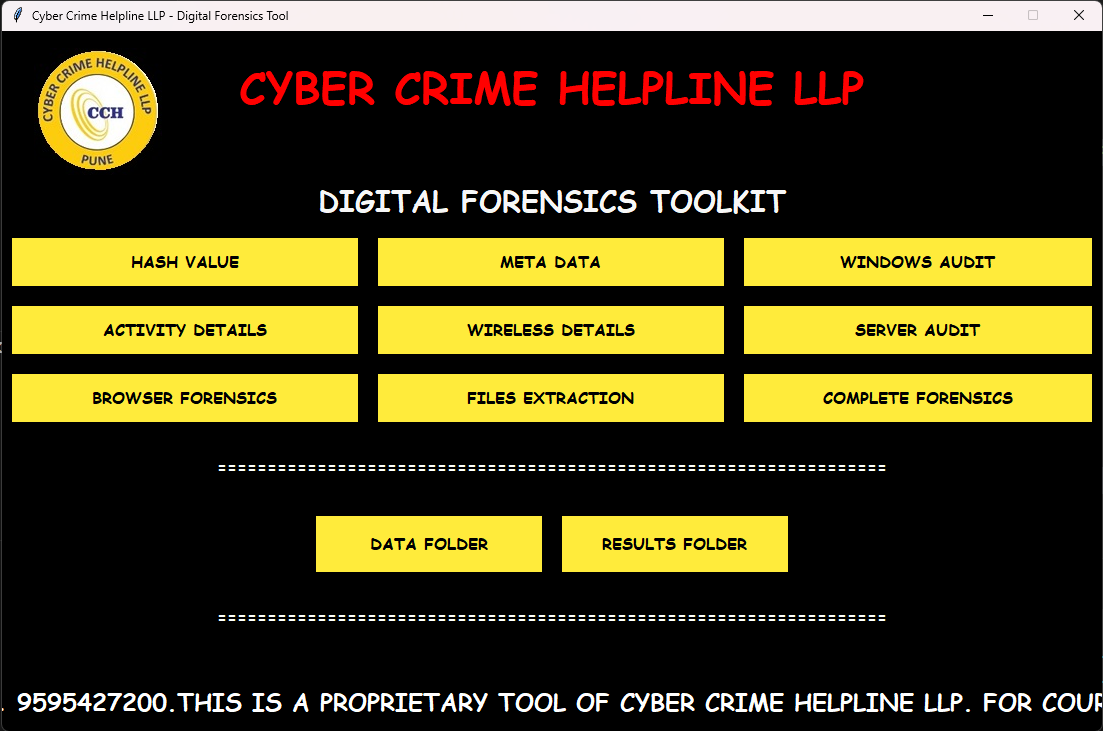
# Start the Tkinter main loop

root.mainloop()

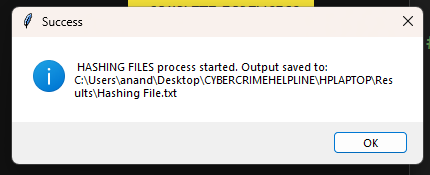
```

**F. EXAMPLE PICTURES**

1. User Interface (UI)  
   ![UI](path/to/screenshot)  
   Figure : The User Interface showing forensic task buttons.



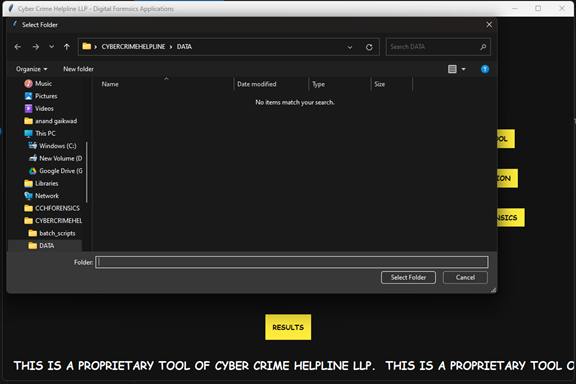
2. Hashing Output  
   ![Hashing](path/to/screenshot)  
   Figure : Example of the hashing process output.



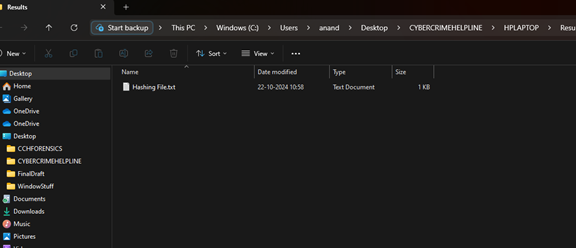
3. Data Folder

![Data Folder](path/to/screenshot)

Figure : The **Data Folder button** is intended for users to select a directory where the input data for the forensic tools is located.



3. Results Folder  
   ![Results Folder](path/to/screenshot)  
   Figure 3: Automatically generated results folder based on the computer name.



The process of creating an executable using PyInstaller involves several steps, which are detailed below:

# Step 1: Install Python

If Python is not already installed on your system, download and install it from the [official Python website](https://www.python.org/downloads/).

# Step 2: Install PyInstaller

Open your command line interface (Command Prompt on Windows, Terminal on Mac/Linux) and execute the following command:

``cmd

pip install pyinstaller

```

# Step 3: Prepare Your Python Script

Ensure your Python script is functioning correctly. For this report, we will refer to the script as CCH.py.

# Step 4: Using PyInstaller

To create an executable, navigate to the directory containing your script using the command line. You can use the following commands based on your requirements:

Command 1: Basic Executable Creation

```cmd

pyinstaller --onefile --console --add-data " C:\Users\anand\Desktop\CYBERCRIMEHELPLINE;." --add-data " C:\Users\anand\Desktop\CYBERCRIMEHELPLINE;." CCH.py

```

This command creates a single executable file. The `--add-data` option allows you to include additional files necessary for your script. Each additional file is specified with its path followed by a semicolon (`;`) and a dot (`.`) which indicates the current directory.

Command 2: Executable with an Icon

```cmd

pyinstaller --onefile --console --add-data " C:\Users\anand\Desktop\CYBERCRIMEHELPLINE;." --icon=" "C:\Users\anand\Downloads\intelligence.ico" "CCH.py

```

This command similarly creates a standalone executable, but also specifies an icon for the application using the `--icon` option.

G. Summary of Project in Brief

This project involved developing the CCH Forensics Toolkit, a Python-based digital forensics application designed for Cyber Crime Helpline LLP. Using Tkinter for the UI and Python’s subprocess module, the toolkit integrates standalone forensic executables for efficient evidence processing. The interface allows users to initiate forensic tasks, with results stored in a centralized folder. Custom button designs, integrated logos, and error handling enhance usability, while a scrolling marquee provides continuous updates. The toolkit thus supports cybercrime investigations with an effective and well-structured digital forensic tool.