Technological Institute of the Philippines

938 Aurora Blvd. Cubao, Quezon City

College of Computer Studies

ScanShield: MD5-Based Malware Detector

IT 022 – Threat Detection Analysis

Members:

Estrella, Matthew Joaquin G.

Gonzales, Sean

Isuan, Christian

Jimenez, Jeone

Madelo, John Ereyl

Submitted to:

Mr. Allan Burgos

Date

May 17, 2025

I. PROJECT DESCRIPTION

ScanShield is a lightweight antivirus program coded in Python that offers minimal protection by scanning only chosen files for established threats. It works through comparing the MD5 hash values of user-chosen files against an extensive database of known malicious file hashes. Once a file is chosen, ScanShield determines its MD5 hash and verifies it against the hash list. When a match is discovered, the file is declared infected to avoid any damage to the system.

This method ensures rapid identification of previously known malware using their digital fingerprint (MD5 hash) and hence ScanShield is an easy and efficient tool for scanning and cleaning files without depending on sophisticated heuristics or behavioral checking. ScanShield is a better fit for academic use or light file-checking work where simplicity and management are important.

II. BACKGROUND OF THE STUDY

The ever-evolving threat of viruses remains a major threat to computer systems, especially through infected documents that comprise data security and integrity. Although most commercial antivirus software is highly advanced, they tend to be resource-heavy and not ideal for lightweight or academic use. This research seeks to create a light but effective antivirus application, known as ScanShield, with Python. Using a massive database of well-known malicious MD5 hash values, the system can scan files chosen by the user, detect suspected threats, and remove infected files. This is a simple form of protection using hash-based detection, illustrating how simple techniques can be used to add safety to files in a convenient and user-friendly way.

III. PROJECT OBJECTIVES

The main objective of this project is to create a basic yet functional antivirus program based on Python, which performs scans on chosen files for possible threats, matches them with existing malware signatures, and gives users the choice to take necessary actions.

Specific objectives are:

- Develop a file scanning system that computes MD5 hashes and matches them against a database of known malicious signatures.
- Create scan modes (Quick Scan, Full Scan, and Custom Scan) in which users can select the extent of their scan.
- Implement a user interface to show scan results, system resource utilization, and perform actions like quarantining or deleting infected files.
- Offer explicit and actionable feedback, like marking infected files with threat status and available actions.
- Make sure that the antivirus software can monitor and display system resource utilization, including CPU, RAM, and disk usage, in real-time during scans.

IV. SCOPE AND DELIMITATION

Scope:

The system is designed with the following core functionalities:

- Performs basic malware detection and removal using an MD5 hash-based system.
- Supports manual file and folder scanning based on user selection.
- Offers multiple scan options: Quick Scan, Full Scan, and Custom Scan.
- Provides a simple and user-friendly graphical interface for ease of use.
- Displays scan results including file paths, detected threats, and system resource usage (CPU, RAM, Disk).

Delimitation:

- May not detect all forms of malware, especially new or unknown variants.
- Does not include automatic or real-time scanning features.
- Requires regular updates to maintain an up-to-date threat database.
- Limited in scanning large files or uncommon file types.
- Relies on manual scanning initiated by the user.

VI. SYSTEM FEATURES

The ScanShield Antivirus system has a unique way of detecting malware. It uses various kinds of hashing algorithms (MD5, SHA1, and SHA256) to match a file from your computer with the file in a threat database. If the hashes match it can accurately assume that it is a known malicious file. The system supports different scan modes, a Quick Scan, a Full Scan, and a Custom Scan. This allows users to choose the level of protection and what parts of the computer to scan; mitigating user concerns around performance, speed, and accuracy. ScanShield Antivirus does more than detect malware on your computer; ScanShield offers a threat management feature that allows users to delete infected files for good! The quarantine feature isolates suspicious files so nothing can happen to the rest of the OS. The quarantine area itself is not accessible from the OS either. The system has the ability to provide a full and detailed view of the system's state in real-time. CPU stats, RAM stats, and Disk Usage are all displayed in the UI. The display of real-time stats is not only educational but also promotes user awareness. Most users check their CPU, RAM, Disk Usage, and loading times when they think their computer performance is slowing. They may find out it isn't the real problem! They might delinquent system health numbers and it may be a computer infected with malware, both will behave the same way. ScanShield virus scanner helps users take a proactive approach to keeping their system healthy and working properly. In addition to these features, ScanShield includes a basic analysis feature that gives users a short summary of any detected threat. This summary includes the threat name, type, and possible origin, helping users better understand what was found on their system. It adds another layer of transparency, so users can make more informed decisions when managing detected threats.

VIII. PROJECT TIMELINE

March	April	May
Week 1	Week 5	Week 9
Project Initial Presentation	Technology & PL Evaluation	Project Finalization
Week 2	Week 6	Week 10
System Requirements Gathering	Prototype Development	**Refinement & Final Presentation**
Week 3	Week 7	—

Outline Project Goals	Core Module Development	
Week 4	Week 8	
System Architecture Design	Initial Testing	

SCREENSHOTS

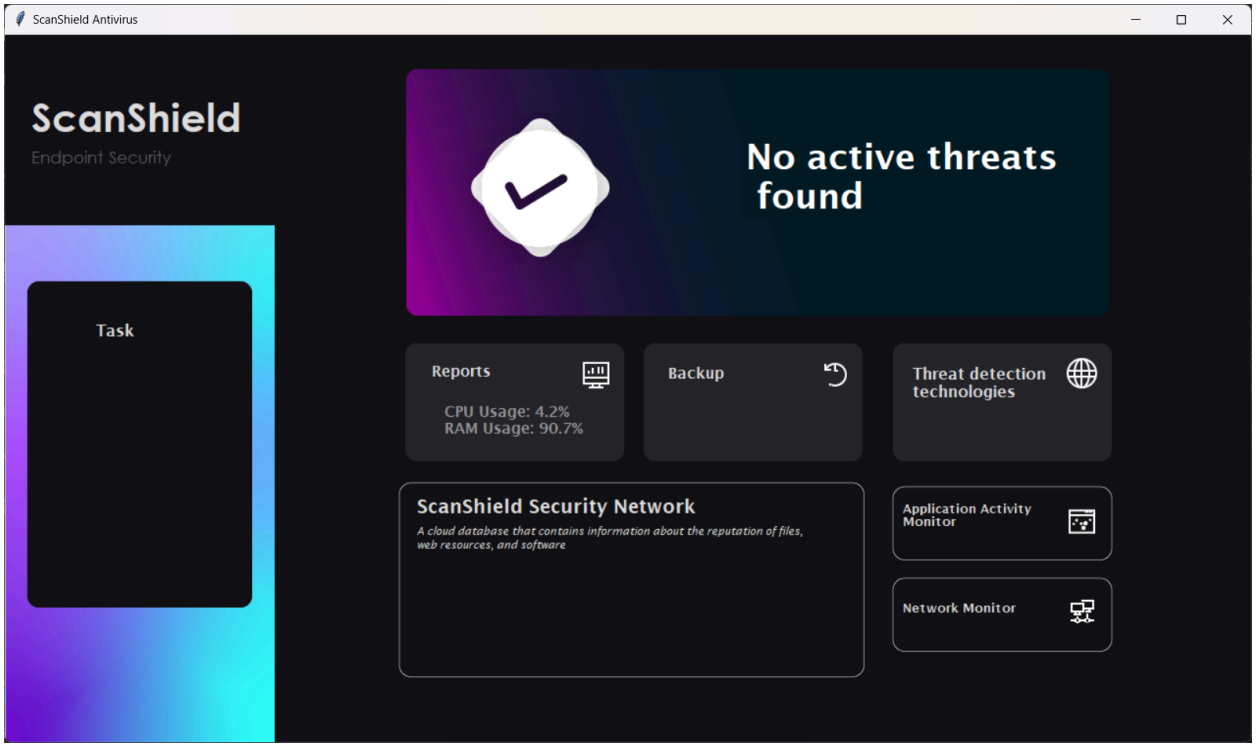

Figure 1: ScanShield Dashboard Page

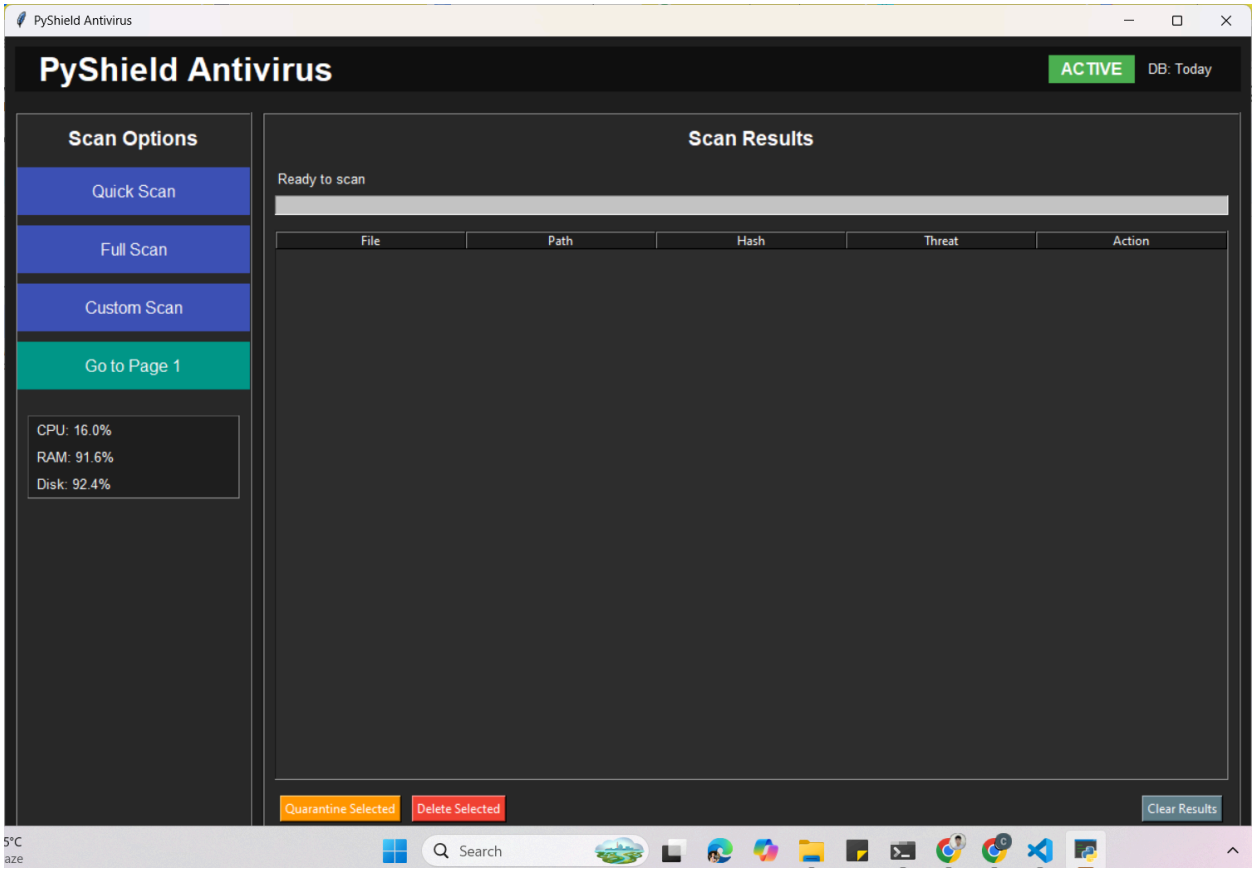

Figure 2: Main Scan Option and Result Page

III. PROGRAM

Main.dart

```
import psutil
from tkinter import *
import tkinter as tk
from tkinter import ttk, font
from PIL import ImageTk, Image
import os
import sys
import subprocess

thisPage = 1

root = Tk()
root.title("ScanShield Antivirus")
root.geometry("1270x720")
root.minsize(1270, 720)
root.maxsize(1270, 720)
root.configure(bg="#131314")

# Colors
backgroundColor = "#131314"
not_active_color = "#606060"
active_color = "#DCDCDD"
box_background = "#26262C"


def update_label(label):
    # This for initializing the CPU Percent
    cpu_usage = psutil.cpu_percent(interval=0.1)
    ram_usage = psutil.virtual_memory().percent
    # This to make it appear
    info_text = f"CPU Usage: {cpu_usage}%\nRAM
Usage: {ram_usage}%"
    label.config(text=info_text)
    root.after(300, lambda: update_label(label))


def open_page2():
    # Save the current script's directory
    current_dir =
os.path.dirname(os.path.abspath(__file__))
    # Path to page2.py in the same directory
    page2_path = os.path.join(current_dir, "page2.py")
```

```
    # Close the current window
    root.destroy()

    # Open page2.py using the same Python interpreter
    subprocess.Popen([sys.executable, page2_path])


def hoverMenuButtons(event, i):
    global buttonsMain
    if (i == 0):
        buttonsMain[i].config(font=("Lucida Sans", 16,
"bold"), bg=backgroundColor, fg="#DEDEE0",
                              activeforeground="#5A5A5B")
        buttonsMain[i].place(relx=0.05, rely=0.4)
    elif (i == 1):
        buttonsMain[i].config(font=("Lucida Sans", 16,
"bold"), bg=backgroundColor, fg="#DEDEE0",
                              activeforeground="#5A5A5B")
        buttonsMain[i].place(relx=0.064, rely=0.44)
    elif (i == 2):
        buttonsMain[i].config(font=("Lucida Sans", 16,
"bold"), bg=backgroundColor, fg="#DEDEE0",
                              activeforeground="#5A5A5B")
        buttonsMain[i].place(relx=0.065, rely=0.49)
    elif (i == 3):
        buttonsMain[i].config(font=("Lucida Sans", 16,
"bold"), bg=backgroundColor, fg="#DEDEE0",
                              activeforeground="#5A5A5B")
        buttonsMain[i].place(relx=0.069, rely=0.54)
    elif (i == 4):
        buttonsMain[i].config(font=("Lucida Sans", 16,
"bold"), bg=backgroundColor, fg="#DEDEE0",
                              activeforeground="#5A5A5B")
        buttonsMain[i].place(relx=0.063, rely=0.58)


def leaveMenuButtons(event, i):
    global buttonsMain
    if (i == 0):
        root.after(50)
        buttonsMain[i].config(font=("Lucida Sans", 12,
"bold"), fg=active_color, bg=backgroundColor)
        buttonsMain[i].place(relx=0.07, rely=0.4)
    elif (i == 1):
        root.after(50)
        buttonsMain[i].config(font=("Lucida Sans", 12),
fg=not_active_color, bg=backgroundColor)
        buttonsMain[i].place(relx=0.075, rely=0.45)
    elif (i == 2):
        root.after(50)
        buttonsMain[i].config(font=("Lucida Sans", 12),
fg=not_active_color, bg=backgroundColor)
        buttonsMain[i].place(relx=0.075, rely=0.50)
    elif (i == 3):
        root.after(50)
        buttonsMain[i].config(font=("Lucida Sans", 12),
fg=not_active_color, bg=backgroundColor)
        buttonsMain[i].place(relx=0.078, rely=0.55)
    elif (i == 4):
        root.after(50)
        buttonsMain[i].config(font=("Lucida Sans", 12),
fg=not_active_color, bg=backgroundColor)
        buttonsMain[i].place(relx=0.074, rely=0.595)


# UI Elements (your original code)
image_frame =
ImageTk.PhotoImage(Image.open("1x/Panel.png"))
main_frame = tk.Frame(root, bg="black")
main_frame.pack(side=tk.LEFT, fill=tk.Y)
main_frame.pack_propagate(FALSE)
main_frame.configure(width=275, height=720)

label = Label(main_frame, image=image_frame,
borderwidth=0)
label.pack()

imgAnti =
ImageTk.PhotoImage(Image.open("1x/AntiPanel.png"))
canvas = Canvas(root, bg="#131314",
highlightthickness=0)
labelAnti = Label(root, fg="white", image=imgAnti,
borderwidth=0)
labelAnti.pack(pady=20)

textAnit = Label(root, text="No active threats\n found",
font=('Lucida Sans', 27, 'bold'), bg="#001C24",
fg="white",
              justify="left")
textAnit.place(relx=0.72, rely=0.2, anchor="center")

C = Canvas(root, bg="#131314", width=992,
height=512.5, highlightthickness=0)
C.place(rely=0.77, relx=0.61, anchor="center")

boxImage =
ImageTk.PhotoImage(Image.open("1x/box.png"))
labelBox = Label(root, bg="#131314", image=boxImage,
borderwidth=0)
labelBox.place(rely=0.52, relx=0.409, anchor="center")
labelBox2 = Label(root, bg="#131314",
image=boxImage, borderwidth=0)
labelBox2.place(rely=0.52, relx=0.6, anchor="center")
labelBox3 = Label(root, bg="#131314",
image=boxImage, borderwidth=0)
labelBox3.place(rely=0.52, relx=0.8, anchor="center")
```

```
bigBoxImage =
ImageTk.PhotoImage(Image.open("1x/bigBox.png"))
labelBig = Label(root, bg="#131314",
image=bigBoxImage, borderwidth=0)
labelBig.place(rely=0.77, relx=0.502, anchor="center")

text_big = Label(root, text="ScanShield Security
Network", bg=backgroundColor, fg=active_color,
borderwidth=0,
            font=("Lucida Sans", 14, "bold"))
text_big.place(relx=.33, rely=0.65)

desc_big = Label(root,
            text="A cloud database that contains
information about the reputation of files,\nweb
resources, and software",
            bg=backgroundColor, fg=active_color,
borderwidth=0, font=("Lucida Sans", 8), justify=LEFT)
desc_big.place(relx=.33, rely=0.69)

tinyImage =
ImageTk.PhotoImage(Image.open("1x/boxTiny.png"))
labelTiny = Label(root, bg="#131314", image=tinyImage,
borderwidth=0)
labelTiny.place(rely=0.69, relx=0.8, anchor="center")

text_tiny = Label(root, text="Application
Activity\nMonitor", bg="#131314", fg=active_color,
borderwidth=0,
             justify=LEFT, font=("Lucida Sans", 9, "bold"))
text_tiny.place(relx=0.72, rely=0.66)

app_img =
ImageTk.PhotoImage(Image.open("1x/icon4.png"))
Label(root, image=app_img,
bg=backgroundColor).place(relx=0.848, rely=0.658)

labelTiny2 = Label(root, bg="#131314",
image=tinyImage, borderwidth=0)
labelTiny2.place(rely=0.82, relx=0.8, anchor="center")

text_tiny2 = Label(root, text="Network Monitor",
bg="#131314", fg=active_color, borderwidth=0,
justify=LEFT,
             font=("Lucida Sans", 9, "bold"))
text_tiny2.place(relx=0.72, rely=0.8)

net_img =
ImageTk.PhotoImage(Image.open("1x/icon5.png"))
Label(root, image=net_img,
bg=backgroundColor).place(relx=0.848, rely=0.787)

text_Info = Label(root, text="Reports",
bg=box_background, fg=active_color, font=("Lucida
Sans", 11, "bold"))
text_Info.place(relx=0.34, rely=0.46)

report_img =
ImageTk.PhotoImage(Image.open("1x/icon1.png"))
Label(root, image=report_img,
bg=box_background).place(relx=0.458, rely=0.45)

info_label = Label(root, bg=box_background,
fg="#989899", font=("Lucida Sans", 11, "bold"),
justify=LEFT)
info_label.place(relx=0.35, rely=0.517)
update_label(info_label)

button_Info2 = Button(root, bg=box_background,
fg=active_color, font=("Lucida Sans", 11, "bold"),
            activebackground=box_background,
borderwidth=0, activeforeground=active_color, padx=97,
pady=45)
button_Info2.place(relx=0.52, rely=0.443)

backup_img =
ImageTk.PhotoImage(Image.open("1x/icon2.png"))
Label(root, image=backup_img,
bg=box_background).place(relx=0.65, rely=0.45)

text_Info2 = Label(root, text="Backup", fg=active_color,
font=("Lucida Sans", 11, "bold"), bg=box_background)
text_Info2.place(relx=0.53, rely=0.463)

button_Info3 = Button(root, bg=box_background,
borderwidth=0, fg=active_color, font=("Lucida Sans", 11,
"bold"),
             activebackground=box_background,
activeforeground=active_color, padx=97, pady=45)
button_Info3.place(relx=0.72, rely=0.443)

text_Info3 = Label(root, text="Threat
detection\ntechnologies", fg=active_color, font=("Lucida
Sans", 12, "bold"),
            bg=box_background, justify=LEFT)
text_Info3.place(relx=0.726, rely=0.463)

threat_img =
ImageTk.PhotoImage(Image.open("1x/icon3.png"))
Label(root, image=threat_img,
bg=box_background).place(relx=0.848, rely=0.45)

nameAnti = Label(root, text="ScanShield",
font=('Century Gothic', 30, "bold"), bg=backgroundColor,
fg=active_color, pady=0,
```

```
            padx=0)
nameAnti.place(relx=0.02, rely=0.08)

desAnti = Label(root, text="Endpoint Security",
font=('Century Gothic', 13), bg=backgroundColor,
fg="#5A5A5B", pady=0,
            padx=0)
desAnti.place(relx=0.02, rely=0.155)

# Nav Buttons
buttonsMain = ["button_monitoring"]
buttonsMain[0] = Button(root, text="Task", font=("Lucida
Sans", 12, "bold"), fg=active_color,
bg=backgroundColor,
                    activebackground=backgroundColor,
highlightthickness=0, borderwidth=0,
command=open_page2)
buttonsMain[0].place(relx=0.07, rely=0.4)
buttonsMain[0].bind("<Enter>", lambda event, i=0:
hoverMenuButtons(event, i))
buttonsMain[0].bind("<Leave>", lambda event, i=0:
leaveMenuButtons(event, i))

root.mainloop()
```

libservices

```
# ... Keep imports unchanged
import os
import hashlib
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import json
import threading
import time
from datetime import datetime
import shutil
import psutil
import subprocess

class AntivirusApp:
    def __init__(self, root):
        self.root = root
        self.root.title("ScanShield Antivirus")
        self.root.geometry("1200x800")
        self.root.configure(bg="#1e1e1e")

        self.malware_hashes =
self.load_malware_database()
        self.quarantine_dir =
os.path.join(os.path.expanduser("~"),
"ScanShield_Quarantine")
        os.makedirs(self.quarantine_dir, exist_ok=True)
        self.setup_ui()
        self.update_system_stats()

    def load_malware_database(self):
        try:
            with open("malware_hashes.json", "r") as f:
                return json.load(f)
        except:
            return {
                "md5": {
                    "d41d8cd98f00b204e9800998ecf8427e":
"Empty file test",
                    "44d88612fea8a8f36de82e1278abb02f":
"EICAR test virus",
                    "098f6bcd4621d373cade4e832627b4f6":
"Test MD5 hash"
                },
                "sha1": {},
                "sha256": {}
            }

    def setup_ui(self):
        style = ttk.Style()
        style.theme_use("default")
        style.configure("Treeview", background="#2e2e2e",
foreground="white", fieldbackground="#2e2e2e")
        style.configure("Treeview.Heading",
background="#1e1e1e", foreground="white")

        header_frame = tk.Frame(self.root, bg="#121212")
        header_frame.pack(fill="x", padx=10, pady=10)

        tk.Label(header_frame, text="ScanShield
Antivirus", font=("Helvetica", 24, "bold"),
                 bg="#121212", fg="white").pack(side="left",
padx=20)

        status_frame = tk.Frame(header_frame,
bg="#121212")
        status_frame.pack(side="right", padx=20)

        self.protection_status = tk.Label(status_frame,
text="ACTIVE", font=("Helvetica", 12, "bold"),
                                  bg="#4CAF50", fg="white",
padx=10, pady=2)
        self.protection_status.pack(side="left", padx=5)

        self.last_update_label = tk.Label(status_frame,
text="DB: Today", font=("Helvetica", 10),
                                  bg="#121212", fg="white")
        self.last_update_label.pack(side="left", padx=5)
```

```
        main_frame = tk.Frame(self.root, bg="#1e1e1e")
        main_frame.pack(fill="both", expand=True,
padx=10, pady=10)

        left_panel = tk.Frame(main_frame, bg="#2b2b2b",
bd=2, relief="groove")
        left_panel.pack(side="left", fill="y", padx=(0, 10))

        tk.Label(left_panel, text="Scan Options",
font=("Helvetica", 14, "bold"),
                 bg="#2b2b2b", fg="white",
pady=10).pack(fill="x")

        btn_style = {"font": ("Helvetica", 12), "bd": 0, "padx":
20, "pady": 10, "width": 20}

        tk.Button(left_panel, text="Quick Scan",
bg="#3f51b5", fg="white",
                  command=lambda: self.start_scan("quick"),
**btn_style).pack(pady=5)

        tk.Button(left_panel, text="Full Scan",
bg="#3f51b5", fg="white",
                  command=lambda: self.start_scan("full"),
**btn_style).pack(pady=5)

        tk.Button(left_panel, text="Custom Scan",
bg="#3f51b5", fg="white",
                  command=self.custom_scan,
**btn_style).pack(pady=5)

        tk.Button(left_panel, text="Go to Page 1",
bg="#009688", fg="white",
                  command=self.open_page1,
**btn_style).pack(pady=5)

        sysinfo_frame = tk.Frame(left_panel,
bg="#1e1e1e", bd=1, relief="sunken")
        sysinfo_frame.pack(fill="x", pady=20, padx=10)

        self.cpu_label = tk.Label(sysinfo_frame, text="CPU:
0%", font=("Helvetica", 10),
                                  bg="#1e1e1e", fg="white",
anchor="w")
        self.cpu_label.pack(fill="x", padx=5, pady=2)

        self.mem_label = tk.Label(sysinfo_frame,
text="RAM: 0%", font=("Helvetica", 10),
                                  bg="#1e1e1e", fg="white",
anchor="w")
        self.mem_label.pack(fill="x", padx=5, pady=2)

        self.disk_label = tk.Label(sysinfo_frame, text="Disk:
0%", font=("Helvetica", 10),
                                   bg="#1e1e1e", fg="white",
anchor="w")
        self.disk_label.pack(fill="x", padx=5, pady=2)

        right_panel = tk.Frame(main_frame, bg="#2b2b2b",
bd=2, relief="groove")
        right_panel.pack(side="right", fill="both",
expand=True)

        tk.Label(right_panel, text="Scan Results",
font=("Helvetica", 14, "bold"),
                 bg="#2b2b2b", fg="white",
pady=10).pack(fill="x")

        self.progress_frame = tk.Frame(right_panel,
bg="#2b2b2b")
        self.progress_frame.pack(fill="x", padx=10,
pady=5)

        self.progress_label = tk.Label(self.progress_frame,
text="Ready to scan",
                                       font=("Helvetica", 10),
bg="#2b2b2b", fg="white", anchor="w")
        self.progress_label.pack(fill="x")

        self.progress_bar =
ttk.Progressbar(self.progress_frame, orient="horizontal",
mode="determinate")
        self.progress_bar.pack(fill="x", pady=5)

        columns = ("file", "path", "hash", "threat", "action")
        self.results_tree = ttk.Treeview(right_panel,
columns=columns, show="headings", height=15)

        for col in columns:
            self.results_tree.heading(col,
text=col.capitalize())
            self.results_tree.column(col, width=150)

        self.results_tree.pack(fill="both", expand=True,
padx=10, pady=5)

        action_frame = tk.Frame(right_panel,
bg="#2b2b2b")
        action_frame.pack(fill="x", padx=10, pady=10)

        tk.Button(action_frame, text="Quarantine
Selected", bg="#FF9800", fg="white",

command=self.quarantine_selected).pack(side="left",
padx=5)
```

```
        tk.Button(action_frame, text="Delete Selected",
bg="#F44336", fg="white",

command=self.delete_selected).pack(side="left",
padx=5)

        tk.Button(action_frame, text="Clear Results",
bg="#607d8b", fg="white",

command=self.clear_results).pack(side="right", padx=5)

        footer_frame = tk.Frame(self.root, bg="#121212",
height=30)
        footer_frame.pack(fill="x", side="bottom")

        tk.Label(footer_frame, text="ScanShield Antivirus
v1.0 | © 2023",
                 bg="#121212", fg="white").pack(side="right",
padx=20)

    def update_system_stats(self):
        self.cpu_label.config(text=f"CPU:
{psutil.cpu_percent()}%")
        self.mem_label.config(text=f"RAM:
{psutil.virtual_memory().percent}%")
        self.disk_label.config(text=f"Disk:
{psutil.disk_usage('/').percent}%")
        self.root.after(2000, self.update_system_stats)

    def start_scan(self, scan_type):
        scan_paths = {"quick": [os.path.expanduser("~")],
"full": ["/"]}
        if scan_type not in scan_paths:
            return
        self.clear_results()
        self.progress_label.config(text=f"Starting
{scan_type} scan...")
        self.progress_bar["value"] = 0
        threading.Thread(target=self.perform_scan,
args=(scan_paths[scan_type],), daemon=True).start()

    def custom_scan(self):
        directory = filedialog.askdirectory()
        if directory:
            self.clear_results()
            self.progress_label.config(text=f"Starting custom
scan of {directory}...")
            self.progress_bar["value"] = 0
            threading.Thread(target=self.perform_scan,
args=([directory],), daemon=True).start()

    def perform_scan(self, directories):
        start_time = time.time()
        scanned_files = 0
        infected_files = 0
        total_files = sum(len(files) for d in directories for _,
_, files in os.walk(d))

        for dir_path in directories:
            for root, _, files in os.walk(dir_path):
                for file in files:
                    file_path = os.path.join(root, file)
                    scanned_files += 1
                    progress = (scanned_files / total_files) * 100
if total_files > 0 else 0
                    self.root.after(0, self.update_progress,
progress, file_path)
                    try:
                        file_hash =
self.calculate_file_hash(file_path)
                        threat_name =
self.check_malware(file_hash)
                        if threat_name:
                            infected_files += 1
                            self.root.after(0, self.add_infected_file,
file_path, file_hash, threat_name)
                    except:
                        continue

        scan_time = time.time() - start_time
        self.root.after(0, self.scan_completed,
scanned_files, infected_files, scan_time)

    def calculate_file_hash(self, file_path):
        hash_md5 = hashlib.md5()
        hash_sha1 = hashlib.sha1()
        hash_sha256 = hashlib.sha256()
        with open(file_path, "rb") as f:
            for chunk in iter(lambda: f.read(4096), b""):
                hash_md5.update(chunk)
                hash_sha1.update(chunk)
                hash_sha256.update(chunk)
        return {
            "md5": hash_md5.hexdigest(),
            "sha1": hash_sha1.hexdigest(),
            "sha256": hash_sha256.hexdigest()
        }

    def check_malware(self, file_hash):
        for hash_type in ["md5", "sha1", "sha256"]:
            if file_hash[hash_type] in
self.malware_hashes.get(hash_type, {}):
                return
self.malware_hashes[hash_type][file_hash[hash_type]]
        return None
```

```
    def update_progress(self, progress, current_file):
        self.progress_bar["value"] = progress
        self.progress_label.config(text=f"Scanning:
{current_file}")

    def add_infected_file(self, file_path, file_hash,
threat_name):
        filename = os.path.basename(file_path)
        dirname = os.path.dirname(file_path)
        display_hash = file_hash["md5"][:8] + "..." if
file_hash["md5"] else "N/A"
        self.results_tree.insert("", "end", values=(
            filename, dirname, display_hash, threat_name,
"Pending"
        ))

    def scan_completed(self, scanned_files,
infected_files, scan_time):
        self.progress_label.config(
            text=f"Scan completed: {scanned_files} files
scanned, {infected_files} threats found in {scan_time:.2f}
seconds"
        )
        if infected_files > 0:
            self.protection_status.config(text="THREATS
FOUND", bg="#F44336")
            messagebox.showwarning("Scan Completed",
                                   f"Scan found {infected_files}
potential threats!\n\nReview the results and take
appropriate action.")
        else:
            messagebox.showinfo("Scan Completed",
                                f"Scan completed successfully. No
threats found in {scanned_files} files.")

    def quarantine_selected(self):
        selected_items = self.results_tree.selection()
        if not selected_items:
            messagebox.showwarning("No Selection",
"Please select files to quarantine")
            return
        for item in selected_items:
            values = self.results_tree.item(item, "values")
            file_path = os.path.join(values[1], values[0])
            try:
                quarantine_path =
os.path.join(self.quarantine_dir,

os.path.basename(file_path) + "_" +
datetime.now().strftime("%Y%m%d_%H%M%S"))
                shutil.move(file_path, quarantine_path)
                self.results_tree.set(item, "action",
"Quarantined")
            except Exception as e:
                self.results_tree.set(item, "action", f"Error:
{str(e)}")
        messagebox.showinfo("Quarantine",
f"{len(selected_items)} files moved to quarantine")

    def delete_selected(self):
        selected_items = self.results_tree.selection()
        if not selected_items:
            messagebox.showwarning("No Selection",
"Please select files to delete")
            return
        if not messagebox.askyesno("Confirm Deletion",
                                   f"Are you sure you want to
permanently delete {len(selected_items)} file(s)?"):
            return
        for item in selected_items:
            values = self.results_tree.item(item, "values")
            file_path = os.path.join(values[1], values[0])
            try:
                os.remove(file_path)
                self.results_tree.set(item, "action", "Deleted")
            except Exception as e:
                self.results_tree.set(item, "action", f"Error:
{str(e)}")
        messagebox.showinfo("Deletion",
f"{len(selected_items)} files deleted permanently")

    def clear_results(self):
        for item in self.results_tree.get_children():
            self.results_tree.delete(item)
        self.progress_bar["value"] = 0
        self.progress_label.config(text="Ready to scan")
        self.protection_status.config(text="ACTIVE",
bg="#4CAF50")

    def open_page1(self):
        try:
            self.root.destroy()  # Close the current
ScanShield window
            subprocess.Popen(["python", "page1.py"])  #
Open page1.py
        except Exception as e:
            messagebox.showerror("Error", f"Failed to open
Page 1:\n{str(e)}")

if __name__ == "__main__":
    root = tk.Tk()
    app = AntivirusApp(root)
    root.mainloop()
```

III. Members Contribution

Name	Contributions
Estrella, Matthew	● Project Documentation
Gonzales, Sean	● System Developer
Isuan, Christian	● Project Documentation
Jimenez, Jeone	● Project Documentation
Madelo, John Ereyl	● System Developer