**21F-9155**

**21F-9244**

**PACKET SNIFFER**

**CODE:**

import tkinter as tk

from tkinter import scrolledtext, ttk, messagebox, filedialog

import psutil

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import threading

import queue

from scapy.all import sniff, TCP, UDP, ICMP, ARP, IP, wrpcap

class PacketSnifferApp:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Enhanced Packet Sniffer")

        self.root.geometry("1200x900")

        self.root.configure(bg="#222831")

        # Display disclaimer on ethical use

        self.display\_disclaimer()

        # GUI Styling

        self.configure\_gui\_style()

        # GUI Elements

        self.create\_gui\_layout()

        # Internal attributes

        self.packet\_queue = queue.Queue()

        self.captured\_packets = []

        self.sniffing\_event = threading.Event()

        self.sniff\_thread = None

        # Alert thresholds

        self.alert\_thresholds = {

            'ICMP': 50,  # Alert if more than 50 ICMP packets are captured

            'RST': 10    # Alert if more than 10 TCP RST packets are detected

        }

        self.alert\_counts = {'ICMP': 0, 'RST': 0}

    def display\_disclaimer(self):

        """Show a popup disclaimer about ethical use."""

        messagebox.showinfo(

            "Disclaimer",

            "This tool is for educational purposes only. Ensure you have proper authorization before using this application."

        )

    def configure\_gui\_style(self):

        """Set up GUI styles for the application."""

        style = ttk.Style()

        style.theme\_use('clam')

        style.configure('Custom.TFrame', background='#393e46')

        style.configure('Custom.TLabelframe', background='#00adb5', foreground='white', font=("Helvetica", 12, "bold"))

        style.configure('Custom.TButton', background='#00adb5', foreground='#eeeeee', font=("Helvetica", 10, "bold"))

        style.configure('Custom.TCheckbutton', background='#00adb5', foreground='#eeeeee')

    def create\_gui\_layout(self):

        """Create GUI layout."""

        # Main frame

        main\_frame = ttk.Frame(self.root, padding="10", style="Custom.TFrame")

        main\_frame.pack(fill=tk.BOTH, expand=True)

        # Status Label

        self.status\_label = tk.Label(main\_frame, text="Status: Idle", bg="#222831", fg="#00adb5", font=("Helvetica", 12, "bold"))

        self.status\_label.pack(anchor="w", padx=10)

        # Network Interface selection

        self.create\_interface\_selection(main\_frame)

        # Protocol filtering checkboxes

        self.create\_protocol\_filters(main\_frame)

        # Control buttons

        self.create\_buttons(main\_frame)

        # Packet Details Output

        self.create\_packet\_output(main\_frame)

        # Packet Visualization

        self.create\_visualization(main\_frame)

    def create\_interface\_selection(self, parent):

        """Create network interface selection UI."""

        interface\_frame = ttk.LabelFrame(parent, text="Network Interface", padding="10", style="Custom.TLabelframe")

        interface\_frame.pack(fill=tk.X, pady=5)

        self.interface\_combo = ttk.Combobox(interface\_frame, values=list(psutil.net\_if\_addrs().keys()), state="readonly", font=("Helvetica", 10))

        self.interface\_combo.set("Select Interface")

        self.interface\_combo.pack(fill=tk.X, padx=5, pady=5)

        self.add\_tooltip(self.interface\_combo, "Select a network interface for sniffing packets.")

    def create\_protocol\_filters(self, parent):

        """Create protocol filter checkboxes."""

        filter\_frame = ttk.LabelFrame(parent, text="Protocol Filters", padding="10", style="Custom.TLabelframe")

        filter\_frame.pack(fill=tk.X, pady=5)

        self.tcp\_var = tk.BooleanVar()

        self.udp\_var = tk.BooleanVar()

        self.http\_var = tk.BooleanVar()

        self.https\_var = tk.BooleanVar()

        self.icmp\_var = tk.BooleanVar()

        self.arp\_var = tk.BooleanVar()

        filters = [

            ("TCP", self.tcp\_var),

            ("UDP", self.udp\_var),

            ("HTTP", self.http\_var),

            ("HTTPS", self.https\_var),

            ("ICMP", self.icmp\_var),

            ("ARP", self.arp\_var),

        ]

        for text, var in filters:

            chk = ttk.Checkbutton(filter\_frame, text=text, variable=var, style="Custom.TCheckbutton")

            chk.pack(side=tk.LEFT, padx=5)

            self.add\_tooltip(chk, f"Filter packets by {text} protocol.")

    def create\_buttons(self, parent):

        """Create control buttons."""

        button\_frame = ttk.Frame(parent, padding="10", style="Custom.TFrame")

        button\_frame.pack(fill=tk.X, pady=5)

        self.start\_button = ttk.Button(button\_frame, text="Start Sniffing", command=self.start\_sniffing\_thread, style="Custom.TButton")

        self.stop\_button = ttk.Button(button\_frame, text="Stop Sniffing", command=self.stop\_sniffing, state="disabled", style="Custom.TButton")

        self.save\_button = ttk.Button(button\_frame, text="Save Packets", command=self.save\_packets, style="Custom.TButton")

        self.clear\_button = ttk.Button(button\_frame, text="Clear", command=self.clear\_output, style="Custom.TButton")

        for btn in (self.start\_button, self.stop\_button, self.save\_button, self.clear\_button):

            btn.pack(side=tk.LEFT, padx=5)

    def create\_packet\_output(self, parent):

        """Create output area for packet details."""

        output\_frame = ttk.LabelFrame(parent, text="Packet Details", padding="10", style="Custom.TLabelframe")

        output\_frame.pack(fill=tk.BOTH, expand=True, pady=5)

        self.output\_text = scrolledtext.ScrolledText(output\_frame, width=100, height=15, font=("Courier", 10), bg="#222831", fg="#eeeeee")

        self.output\_text.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

    def create\_visualization(self, parent):

        """Create packet visualization chart."""

        self.protocol\_counts = {'TCP': 0, 'UDP': 0, 'HTTP': 0, 'HTTPS': 0, 'ICMP': 0, 'ARP': 0}

        self.fig, self.ax = plt.subplots()

        self.fig.patch.set\_facecolor('#393e46')

        self.canvas = FigureCanvasTkAgg(self.fig, master=parent)

        self.canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True, pady=5)

        self.update\_plot()

    def add\_tooltip(self, widget, text):

        """Add a tooltip to a widget."""

        def show\_tooltip(event):

            x, y, \_, \_ = widget.bbox("insert")

            x += widget.winfo\_rootx() + 25

            y += widget.winfo\_rooty() + 25

            self.tooltip = tk.Toplevel()

            self.tooltip.wm\_overrideredirect(True)

            self.tooltip.geometry(f"+{x}+{y}")

            label = tk.Label(self.tooltip, text=text, bg="yellow", relief="solid", borderwidth=1, font=("Helvetica", 10))

            label.pack()

        def hide\_tooltip(event):

            if hasattr(self, 'tooltip'):

                self.tooltip.destroy()

        widget.bind("<Enter>", show\_tooltip)

        widget.bind("<Leave>", hide\_tooltip)

    def start\_sniffing\_thread(self):

        """Start sniffing packets in a separate thread."""

        interface = self.interface\_combo.get()

        if interface == "Select Interface":

            messagebox.showerror("Error", "Please select a network interface.")

            return

        self.status\_label.config(text="Status: Sniffing")

        self.sniffing\_event.set()

        self.start\_button.config(state="disabled")

        self.stop\_button.config(state="normal")

        self.sniff\_thread = threading.Thread(target=self.start\_sniffing, args=(interface,))

        self.sniff\_thread.daemon = True

        self.sniff\_thread.start()

        self.root.after(1000, self.process\_packet\_queue)

    def start\_sniffing(self, interface):

        """Sniff packets on the specified interface."""

        try:

            sniff(

                iface=interface,

                prn=self.handle\_packet,  # Call handle\_packet for each captured packet

                store=False,

                stop\_filter=lambda \_: not self.sniffing\_event.is\_set()

            )

        except Exception as e:

            self.output\_text.insert(tk.END, f"Error: {str(e)}\n")

            self.output\_text.tag\_config("error", foreground="red")

    def handle\_packet(self, packet):

        """Add a captured packet to the queue."""

        self.packet\_queue.put(packet)

    def process\_packet\_queue(self):

        """Process packets in the queue."""

        while not self.packet\_queue.empty():

            packet = self.packet\_queue.get()

            self.captured\_packets.append(packet)

            self.process\_packet(packet)

        if self.sniffing\_event.is\_set():

            self.root.after(200, self.process\_packet\_queue)

    def process\_packet(self, packet):

        """Process a single packet."""

        try:

            protocol = "Unknown"

            if IP in packet:

                if self.tcp\_var.get() and TCP in packet:

                    self.protocol\_counts['TCP'] += 1

                    protocol = "TCP"

                elif self.udp\_var.get() and UDP in packet:

                    self.protocol\_counts['UDP'] += 1

                    protocol = "UDP"

                elif self.http\_var.get() and TCP in packet and (packet[TCP].dport == 80 or packet[TCP].sport == 80):

                    self.protocol\_counts['HTTP'] += 1

                    protocol = "HTTP"

                elif self.https\_var.get() and TCP in packet and (packet[TCP].dport == 443 or packet[TCP].sport == 443):

                    self.protocol\_counts['HTTPS'] += 1

                    protocol = "HTTPS"

                elif self.icmp\_var.get() and ICMP in packet:

                    self.protocol\_counts['ICMP'] += 1

                    protocol = "ICMP"

                elif self.arp\_var.get() and ARP in packet:

                    self.protocol\_counts['ARP'] += 1

                    protocol = "ARP"

                self.display\_packet(packet, protocol)

            elif ARP in packet and self.arp\_var.get():

                self.protocol\_counts['ARP'] += 1

                protocol = "ARP"

                self.display\_packet(packet, protocol)

            self.update\_plot()

        except Exception as e:

            self.output\_text.insert(tk.END, f"Error processing packet: {str(e)}\n", "error")

            self.output\_text.tag\_config("error", foreground="red")

    def display\_packet(self, packet, protocol):

        """Display packet details in the output text area."""

        try:

            timestamp = packet.time

            src\_ip = packet[IP].src if IP in packet else "N/A"

            dst\_ip = packet[IP].dst if IP in packet else "N/A"

            length = len(packet)

            details = (

                f"Time: {timestamp:.6f}\n"

                f"Protocol: {protocol}\n"

                f"Source: {src\_ip}\n"

                f"Destination: {dst\_ip}\n"

                f"Length: {length} bytes\n"

                f"{'-'\*50}\n"

            )

            self.output\_text.insert(tk.END, details)

            self.output\_text.see(tk.END)

        except Exception as e:

            self.output\_text.insert(tk.END, f"Error displaying packet: {str(e)}\n", "error")

            self.output\_text.tag\_config("error", foreground="red")

    def update\_plot(self):

        """Update the protocol visualization with unique colors."""

        self.ax.clear()

        protocols = list(self.protocol\_counts.keys())

        counts = list(self.protocol\_counts.values())

        # Assign unique colors for each protocol

        colors = ['#00adb5', '#ff5722', '#ffd369', '#9c27b0', '#3f51b5', '#8bc34a']

        # Create the bar chart

        bars = self.ax.bar(protocols, counts, color=colors[:len(protocols)])  # Use only the required colors

        # Set plot styling

        self.ax.set\_title("Packet Count by Protocol", color="white", fontsize=14)

        self.ax.set\_facecolor('#222831')

        self.ax.grid(color="#393e46", linestyle="--", linewidth=0.5)

        self.ax.tick\_params(colors="white")

        # Add value labels on top of each bar

        for bar in bars:

            yval = bar.get\_height()

            self.ax.text(bar.get\_x() + bar.get\_width() / 2, yval + 1, int(yval), ha='center', va='bottom', fontsize=10, color='white')

        # Redraw the canvas

        self.canvas.draw()

    def stop\_sniffing(self):

        """Stop sniffing packets."""

        self.sniffing\_event.clear()

        self.start\_button.config(state="normal")

        self.stop\_button.config(state="disabled")

        self.status\_label.config(text="Status: Idle")

    def save\_packets(self):

        """Save captured packets to a file."""

        if not self.captured\_packets:

            messagebox.showwarning("Warning", "No packets to save.")

            return

        file\_path = filedialog.asksaveasfilename(defaultextension=".pcap", filetypes=[("PCAP files", "\*.pcap"), ("All files", "\*.\*")])

        if file\_path:

            wrpcap(file\_path, self.captured\_packets)

            messagebox.showinfo("Info", f"Packets saved to {file\_path}")

    def clear\_output(self):

        """Clear the output area and reset visualization."""

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

        self.protocol\_counts = {key: 0 for key in self.protocol\_counts}

        self.captured\_packets = []

        self.update\_plot()

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

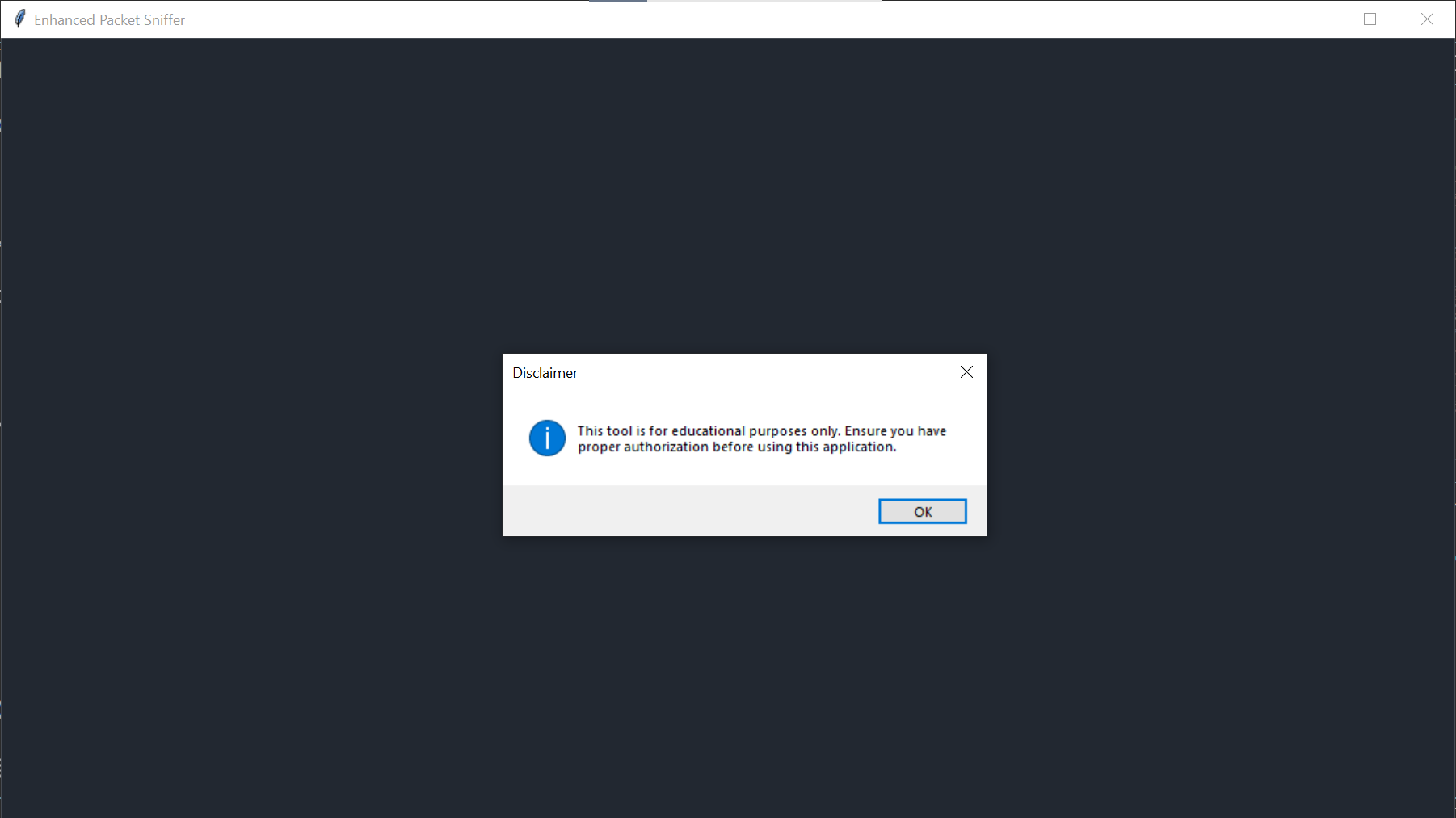
    root = tk.Tk()

    app = PacketSnifferApp(root)

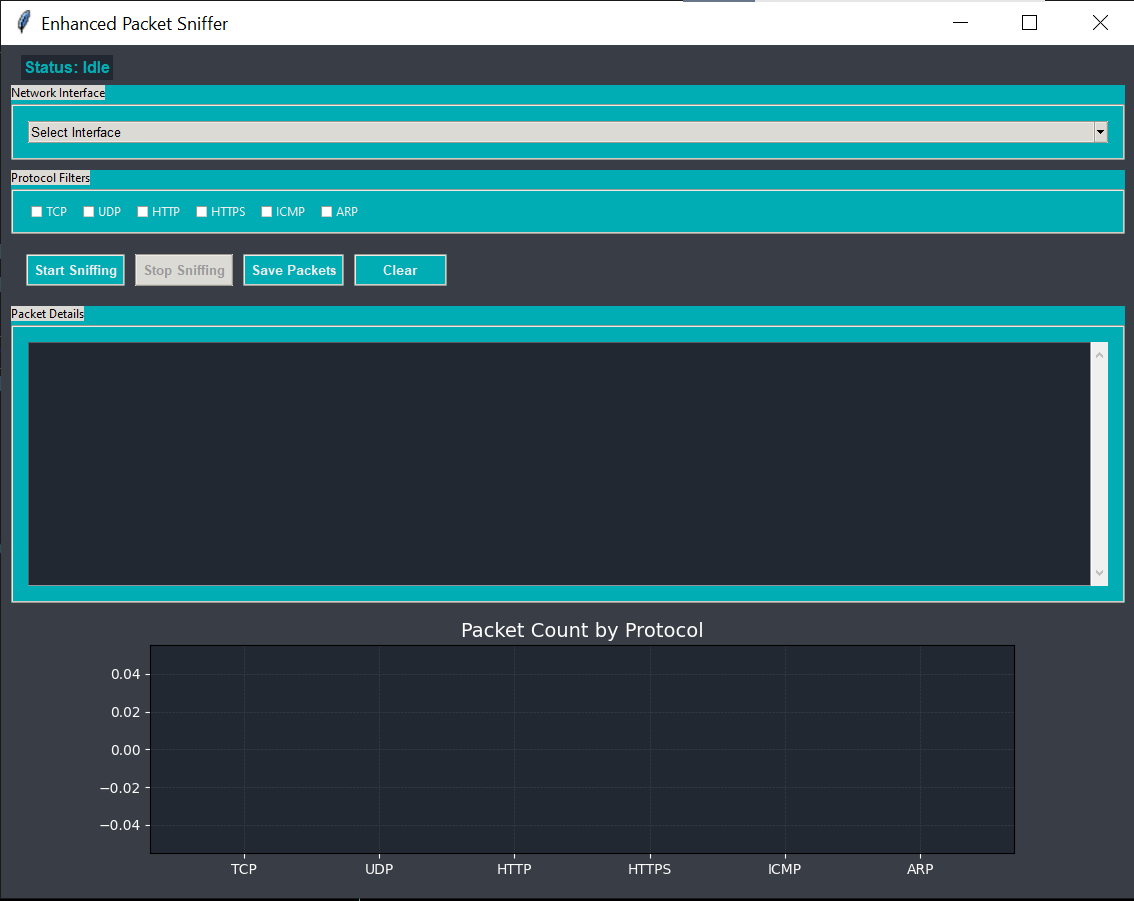
    root.mainloop()

**OUTPUTS:**

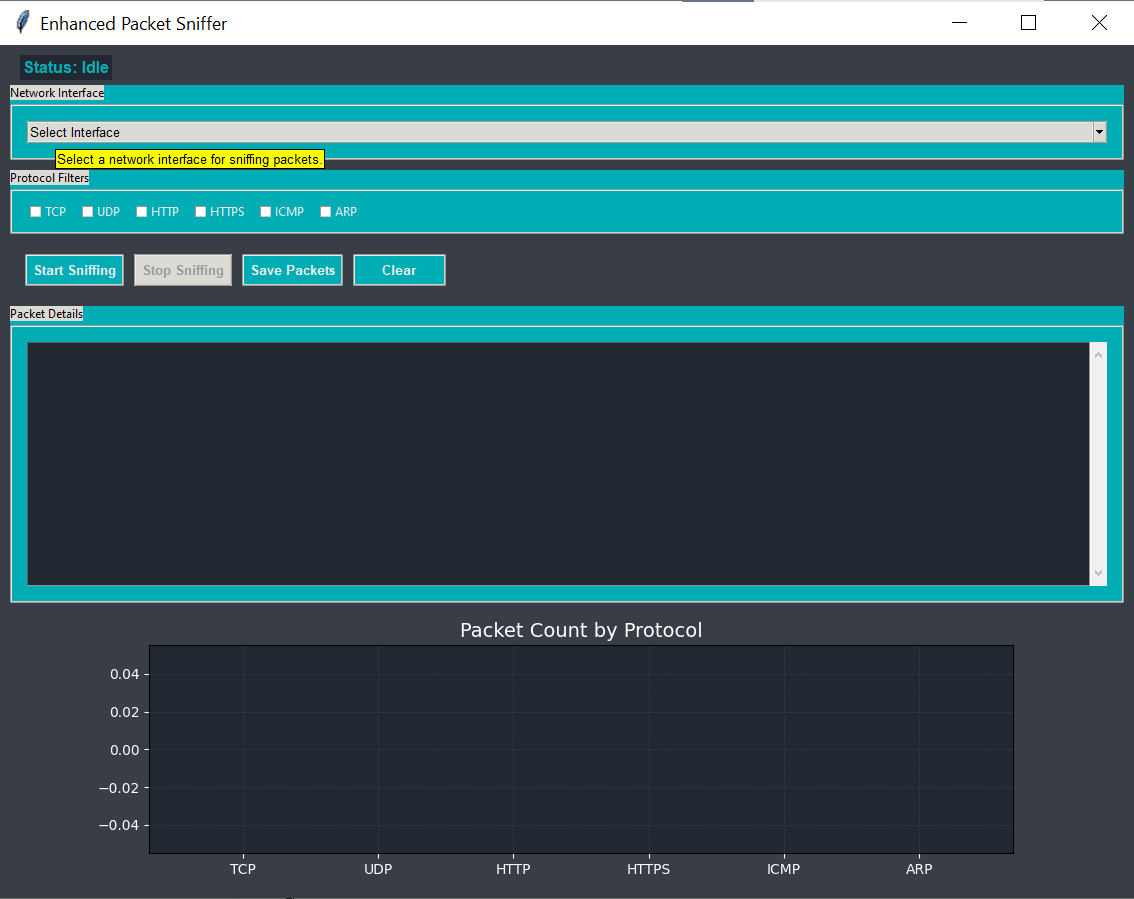
**DISCLAIMER:**

****

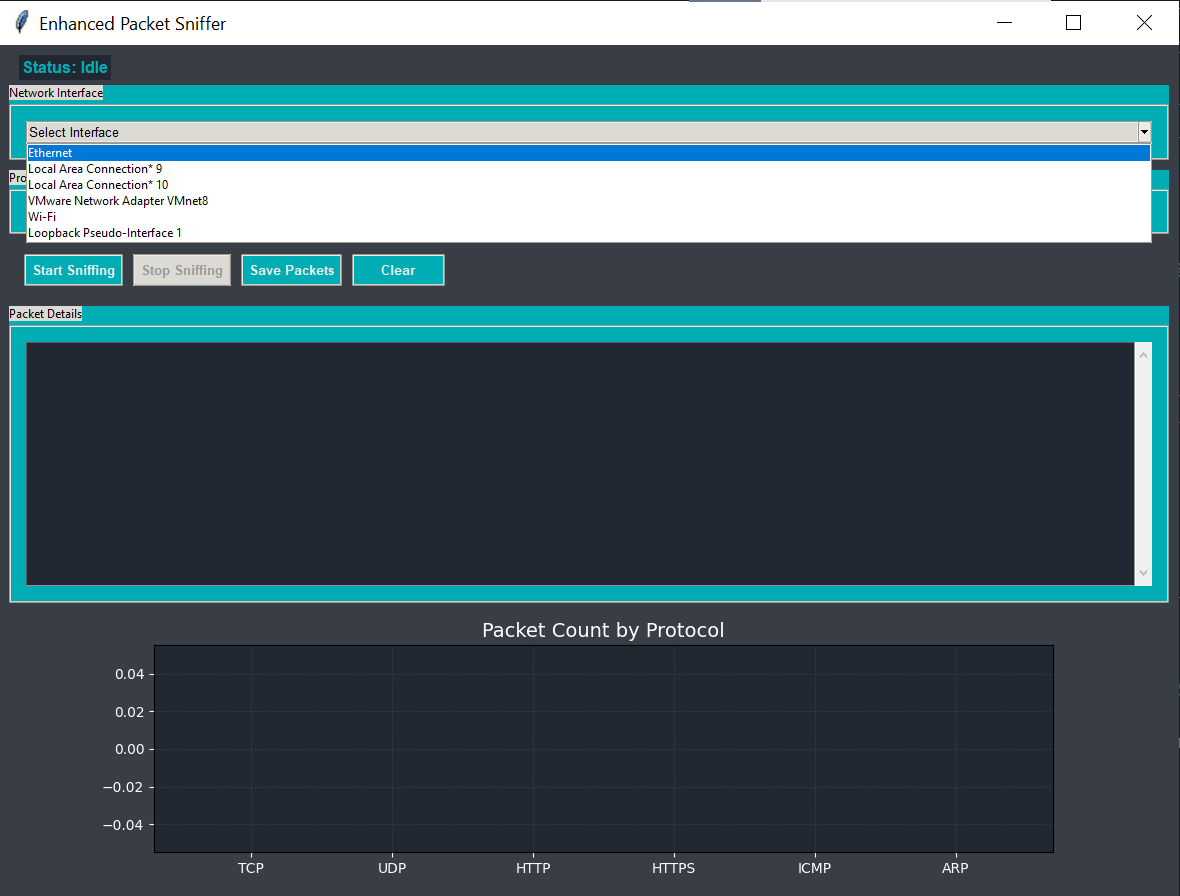
**GUI**

****

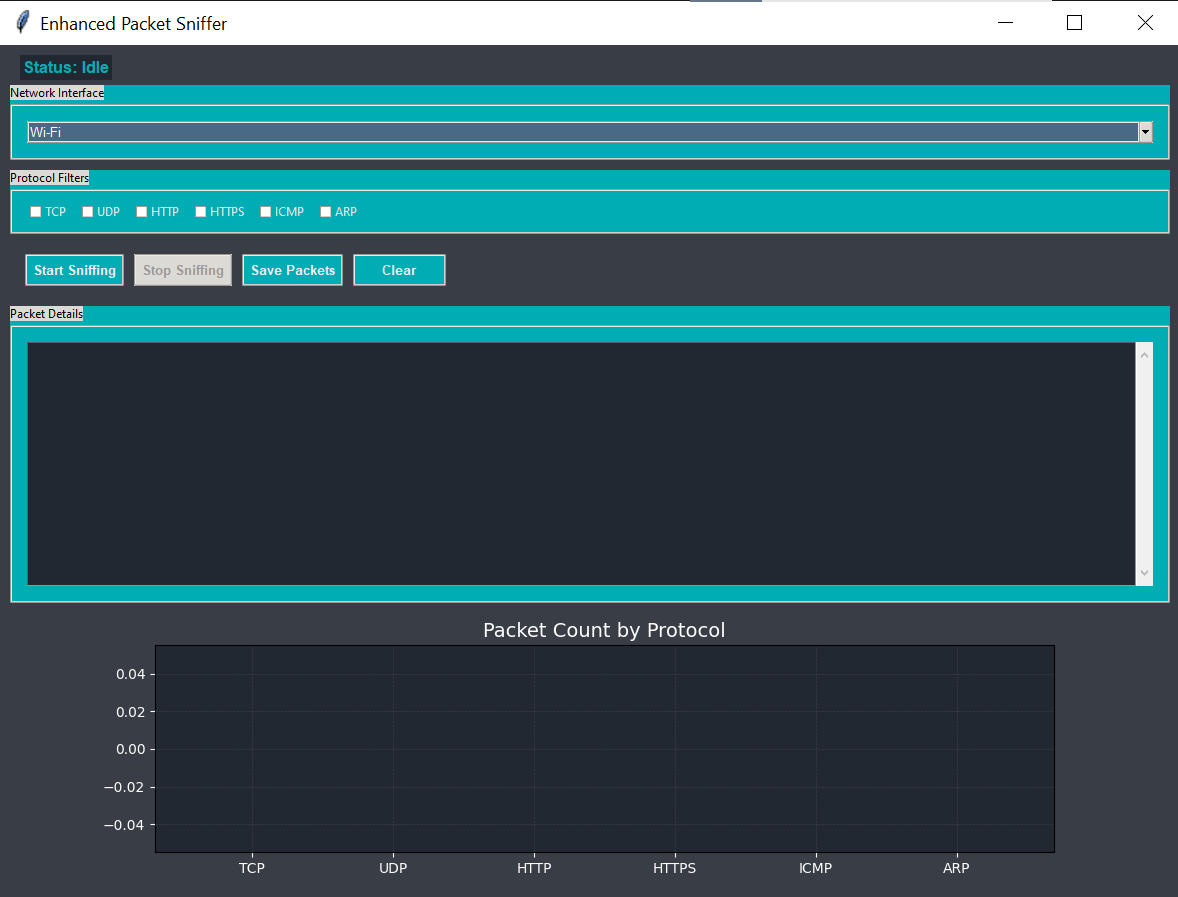
**NETWORK INTERFACE**

****

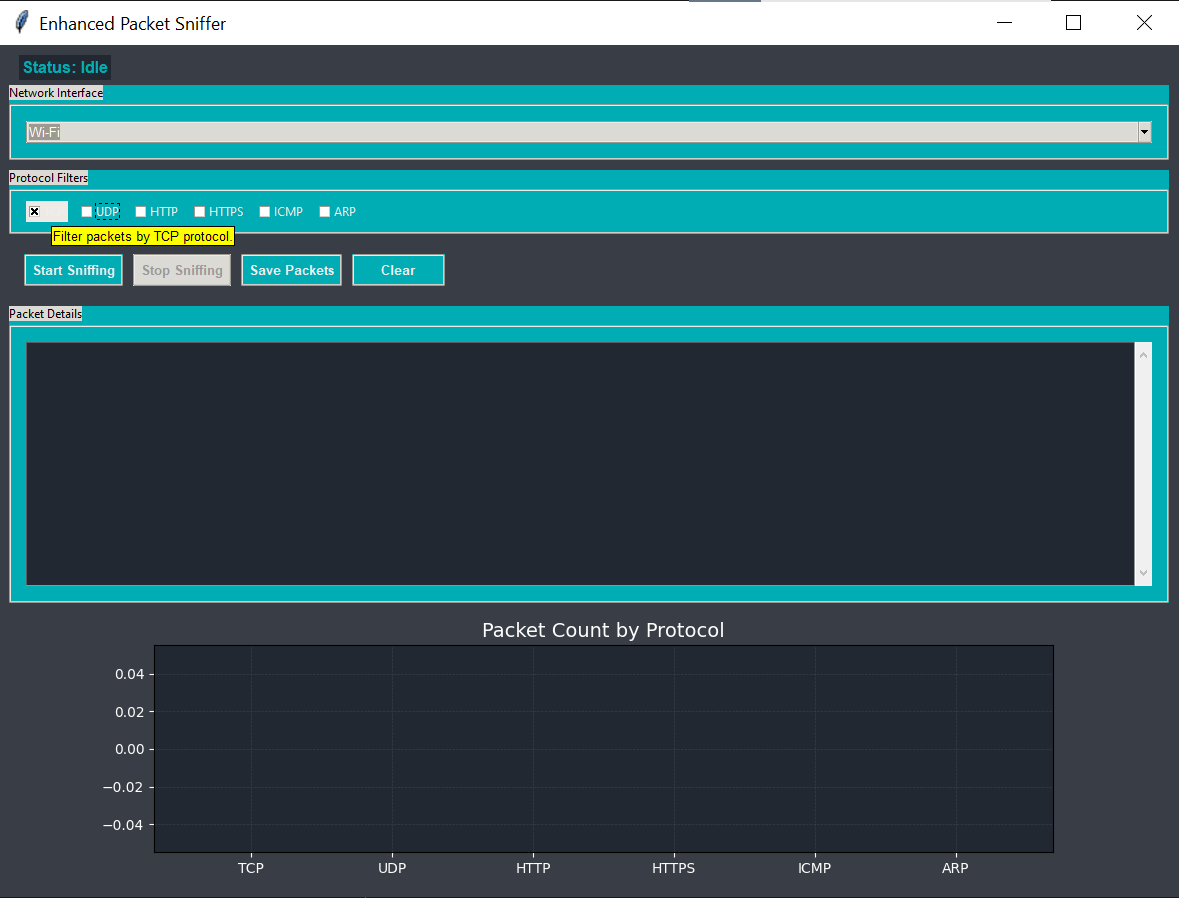
**NETWORK INTERFACE OPTIONS:**

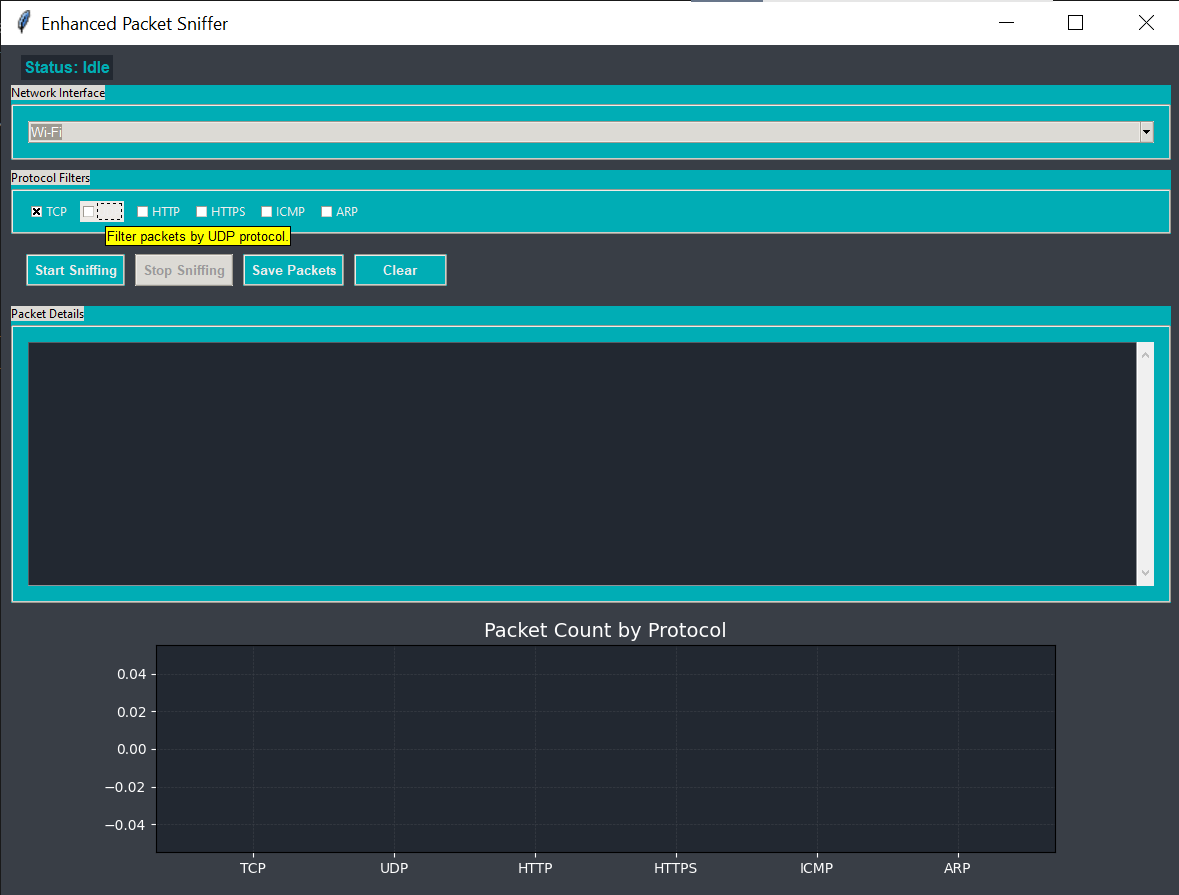
****

**WIFI IS SELECTED:**

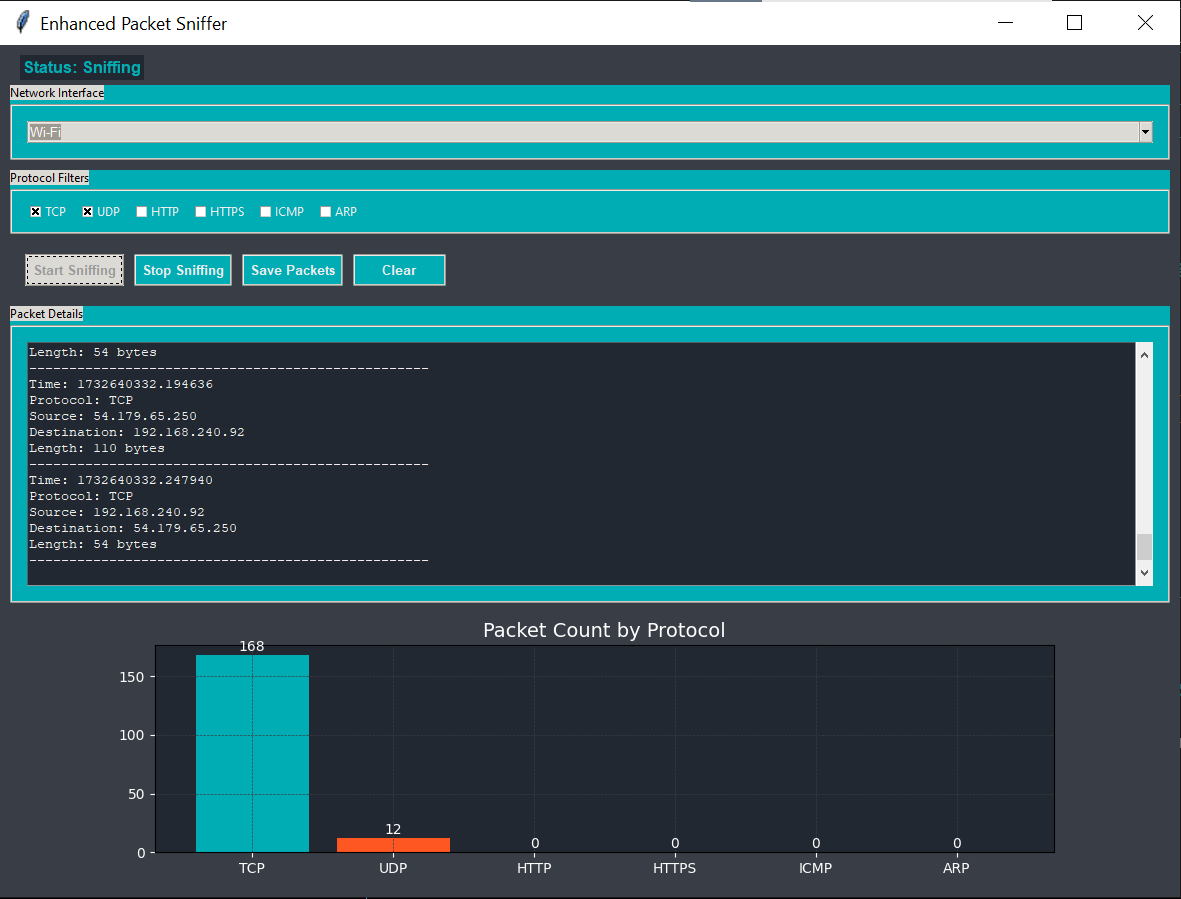
****

**PROTOCOL FILTERS CHOOSE ANY:**

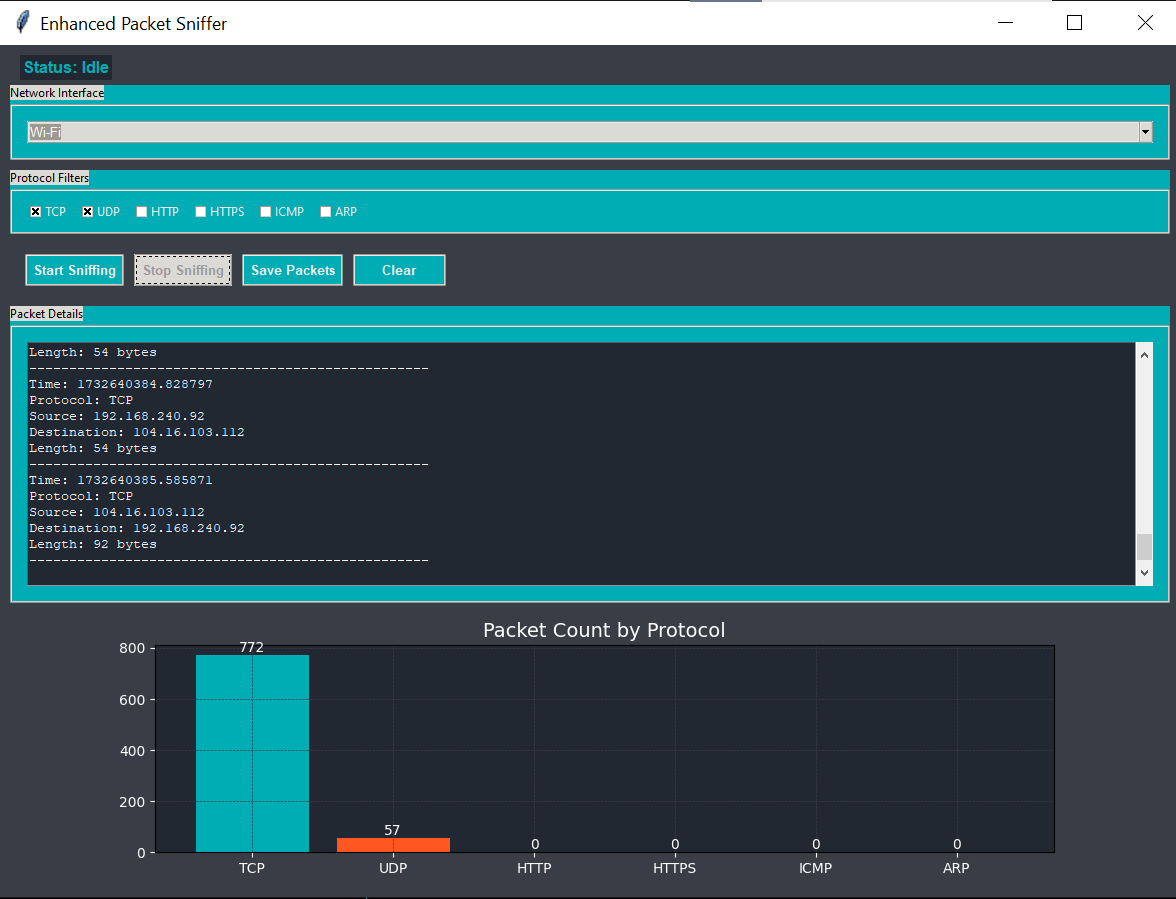
****

****

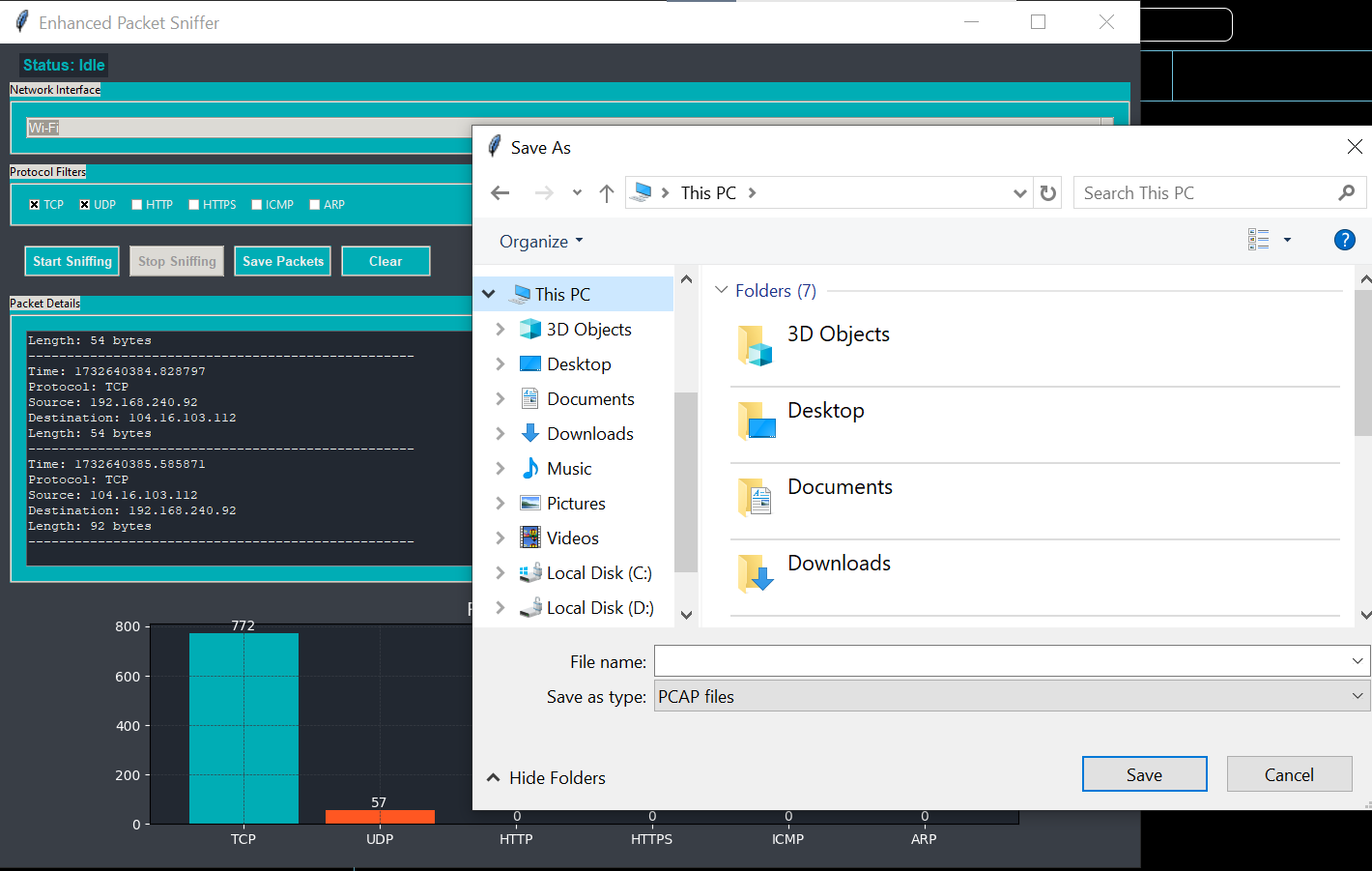
**PACKET CAPTURING:**

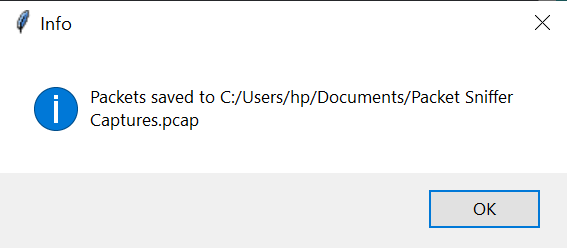
****

**STOP SNIFFING:**

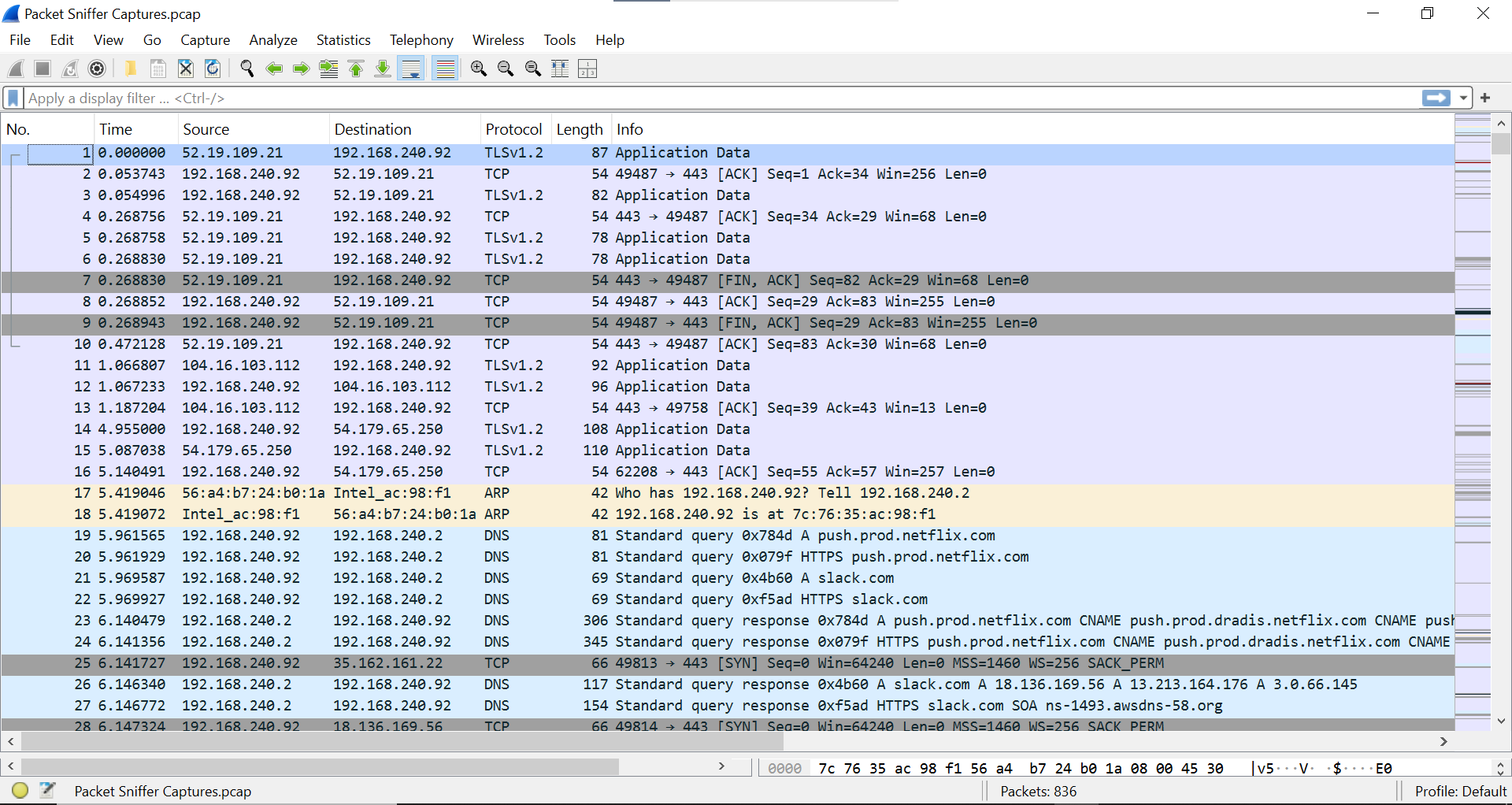
****

**SAVE PACKETS:**

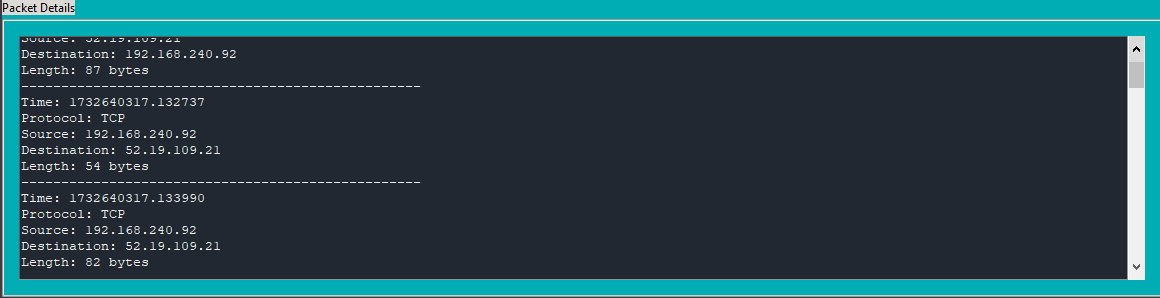
****

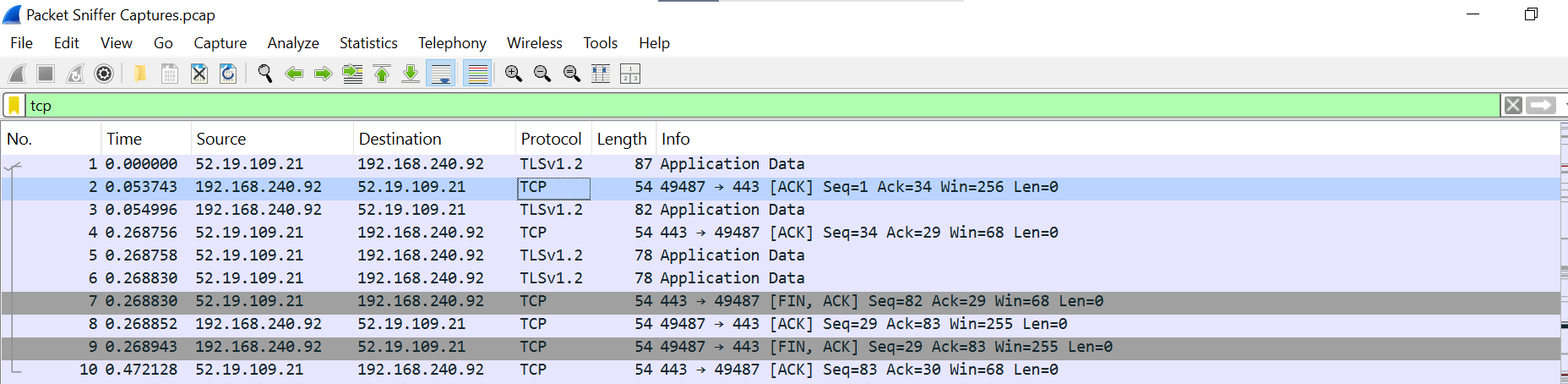
****

**WIRE SHARK FILE:**

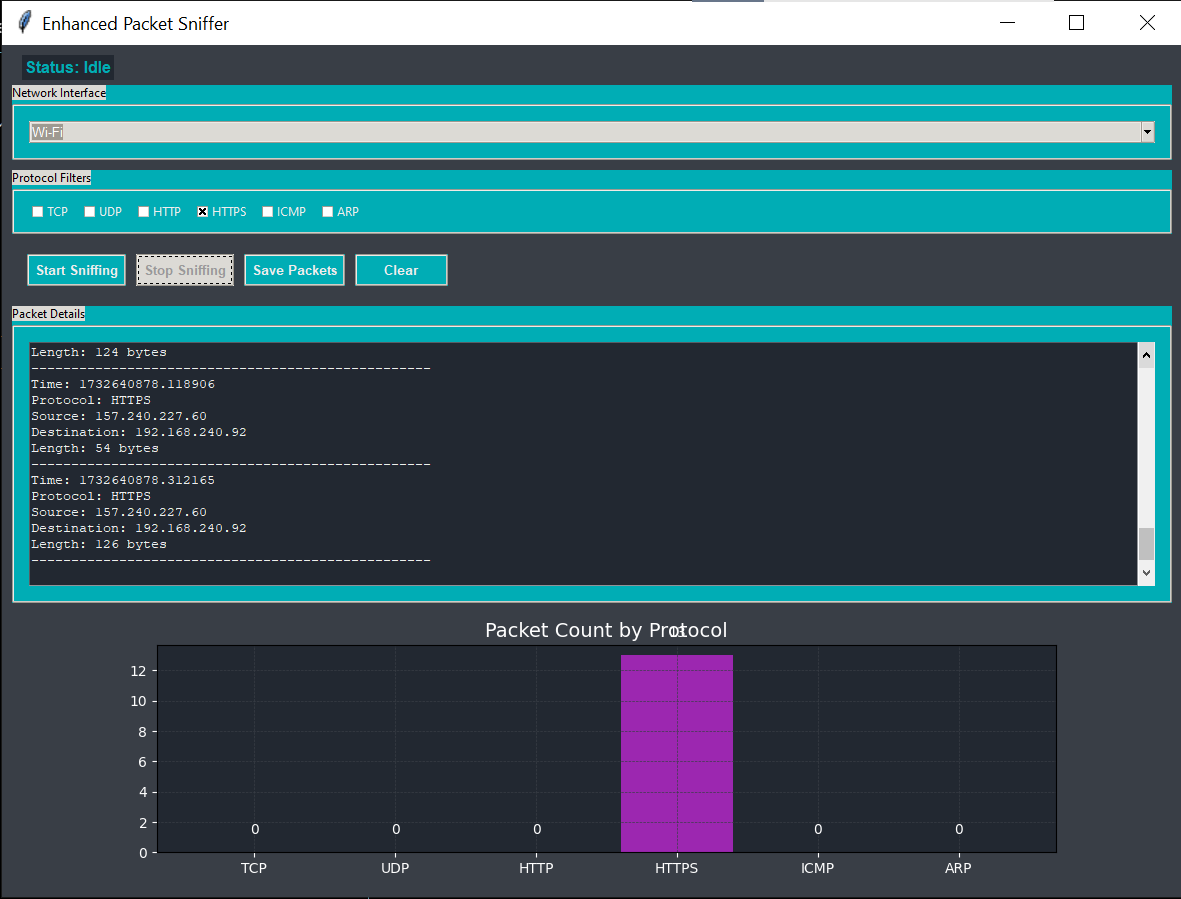
****

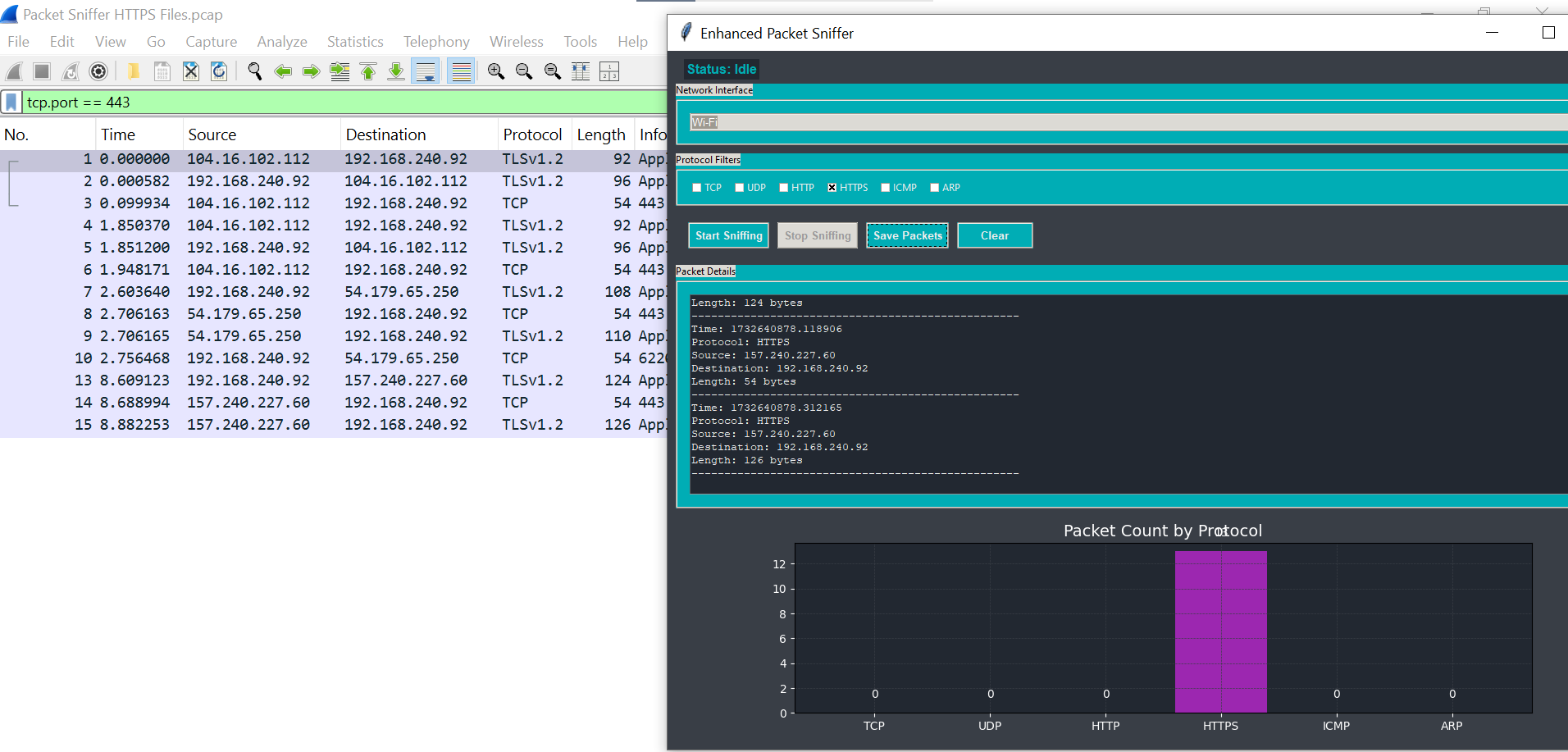
**PACKETS DETAILS:**

****

****

**CAPTURING HTTPS:**

****

****