UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

PROGRAMACIÓN ORIENTADA A OBJETOS

"PacketSniff: Detección de Amenazas con Heurística y la implementación de ML"

Integrantes:

- Adauto Huaman, Matias Benjamin 20242133G
- Berrocal Barrientos, Jorge Luis 20240003I
- Racchumi Vasquez, Fernando Rafael 20244038A
- Maita De la Cruz, Luois Edgar 20220540I

Profesor:

Tello Canchapoma, Yury Oscar

2025 – I

Contenido

INTRODUCCIÓN

La creciente dependencia de los servicios en red y la exposición constante a amenazas informáticas hacen indispensable el desarrollo de herramientas que permitan supervisar, analizar y proteger el tráfico en entornos locales. En este contexto se enmarca el presente proyecto: PacketSniff, un sistema diseñado para capturar y analizar paquetes de red, con capacidad para detectar comportamientos anómalos mediante técnicas heurísticas y aprendizaje automático.

Ha sido desarrollado con un enfoque centrado en la simplicidad de uso y la eficiencia operativa. A través de una interfaz gráfica sencilla, permite al usuario seleccionar su interfaz de red, capturar tráfico en tiempo real, visualizar los paquetes, guardar las sesiones y realizar análisis automáticos. El sistema incorpora tanto reglas heurísticas como un modelo de clasificación entrenado con datos del conjunto CICIDS 2017, lo cual le permite identificar patrones asociados a ciberataques comunes. No obstante, más allá de su funcionalidad actual, su verdadero valor reside en su arquitectura modular y escalable. Esta estructura permite que el sistema sea ampliado con facilidad: nuevos algoritmos de detección, integración con bases de datos externas, capacidades remotas o automatización de respuestas son ejemplos de extensiones futuras viables.

En suma, PacketSniff es una herramienta práctica, accesible y con visión a largo plazo. Su desarrollo responde a una necesidad real en el ámbito de la ciberseguridad local, ofreciendo una base sólida para entornos educativos, proyectos de investigación o sistemas de monitoreo en redes pequeñas y medianas.

1. OBJETIVOS

Objetivo General

Desarrollar una herramienta capaz de capturar, visualizar y analizar tráfico de red en tiempo real, que combine técnicas heurísticas con algoritmos de aprendizaje automático, con el fin de detectar comportamientos potencialmente maliciosos de manera accesible, eficiente y con posibilidad de expansión futura.

Objetivos Específicos

1. Implementar un sniffer de red utilizando la librería Scapy, capaz de capturar paquetes en tiempo real desde una interfaz gráfica amigable.
2. Diseñar una interfaz visual mediante CustomTkinter que permita al usuario seleccionar interfaces, aplicar filtros, guardar capturas y observar resultados de análisis.
3. Incorporar mecanismos de análisis heurístico para identificar patrones sospechosos en los paquetes, como TTL inusuales, tráfico HTTP no cifrado, puertos destino irregulares y consultas DNS anómalas.
4. Entrenar un modelo de clasificación binaria con aprendizaje automático, utilizando datos etiquetados del conjunto CICIDS 2017, para fortalecer la detección de amenazas a nivel de flujo.
5. Integrar el modelo de Machine Learning en el sistema para analizar bloques de paquetes agrupados, generando alertas automáticas ante comportamientos identificados como maliciosos.
6. Desarrollar un simulador de tráfico malicioso para validar el funcionamiento del sistema ante ataques como SYN Flood, DNS Tunneling o tráfico HTTP manipulado.
7. Garantizar que la arquitectura del sistema sea modular y escalable, permitiendo futuras mejoras e integración con sistemas más complejos de monitoreo o respuesta en red.

2. PROYECTOS SIMILARES

Rassilion/sniffer:

Este proyecto contiene un sniffing básico de paquetes modulo socket de Python. Es un proyecto sencillo, ideal para entender los fundamentos de cómo se capturan y analizan los paquetes de red.

lassault/OSniffy:

Este repositorio presenta un sniffer de red y monitor de tráfico desarrollado en Python. Se enfoca en ser ligero y utiliza la biblioteca Scapy para sus funcionalidades. Es una herramienta útil para quienes buscan monitorear el tráfico de red de forma eficiente en sistemas operativos de tipo Unix (Linux, macOS, BSD).

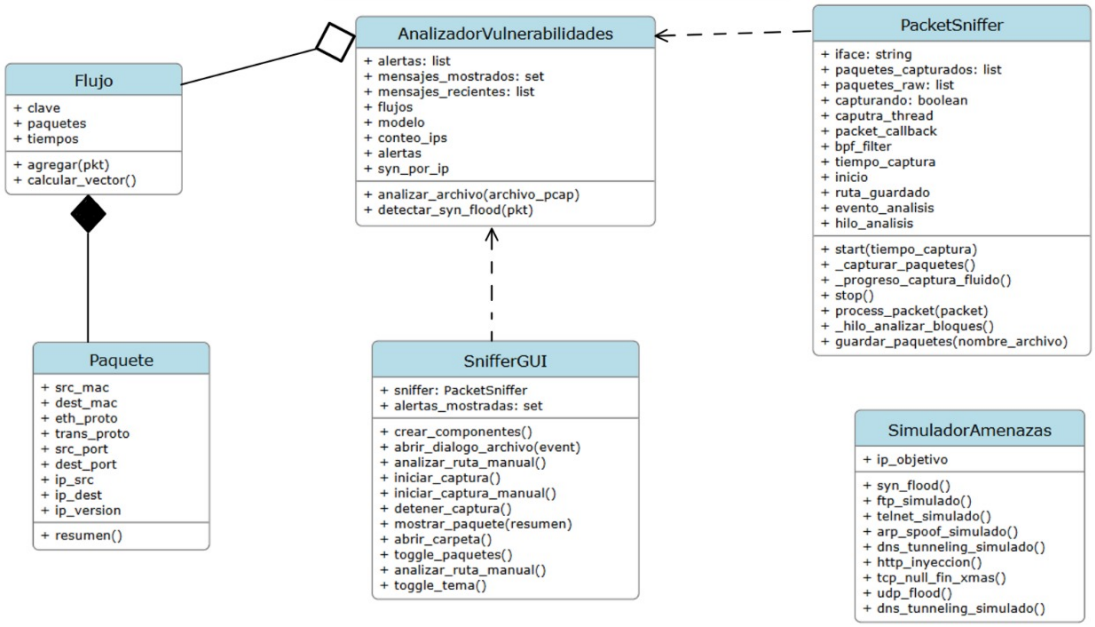
EONRaider/Packet-Sniffer:

Este repositorio ofrece un sniffer de paquetes multiplataforma implementado en Python. Destaca por su capacidad de funcionar en diferentes sistemas operativos y por incluir

características como el seguimiento de conexiones y la visualización del tráfico en tiempo real. Es una opción más completa para el análisis de red.

3. DIAGRAMAS UML

3.1 Diagrama de Clases



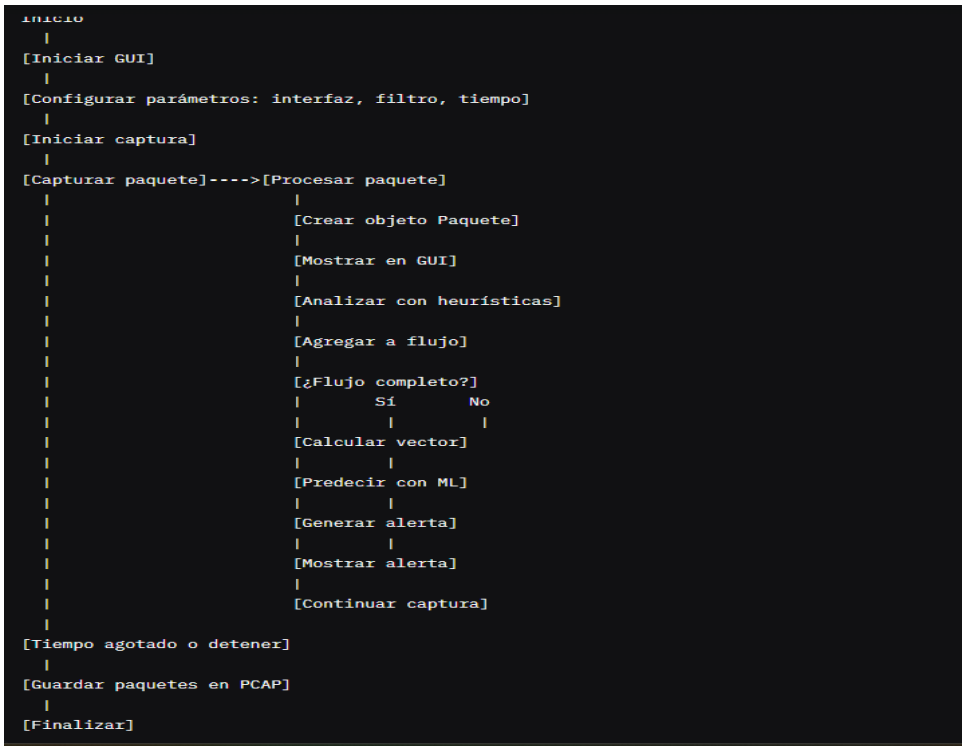
- **Relaciones**:
    - AnalizadorVulnerabilidades usa múltiples instancias de Flujo para gestionar flujos de red.
    - PacketSniffer genera objetos Paquete y los pasa a SnifferGUI y AnalizadorVulnerabilidades.
    - SnifferGUI coordina PacketSniffer y AnalizadorVulnerabilidades.

3.2 Diagrama de Secuencia

Caso de uso: Captura y análisis en tiempo real

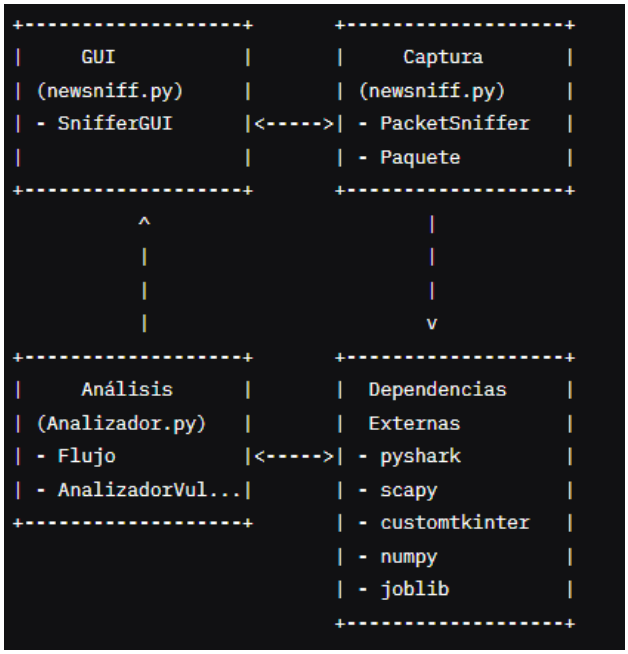
1. El usuario configura parámetros en SnifferGUI.
2. SnifferGUI inicia la captura en PacketSniffer.
3. PacketSniffer captura paquetes con scapy.sniff() y los procesa.
4. Los paquetes se envían a SnifferGUI para visualización y a AnalizadorVulnerabilidades para análisis.
5. AnalizadorVulnerabilidades agrega paquetes a un Flujo, calcula vectores y predice amenazas.
6. Las alertas se muestran en la GUI.

3.3 Diagrama de Actividades



Proceso: Captura y análisis de paquetes

3.4 Diagrama de Componentes



- **Módulo GUI**: Interfaz gráfica para interacción con el usuario.
- **Módulo Captura**: Captura y procesamiento de paquetes.
- **Módulo Análisis**: Análisis heurístico y de machine learning.
- **Dependencias Externas**: Librerías para funcionalidad específica.

4. DISEÑO DEL CÓDIGO Y CLASES

4.1 Tecnologías Utilizadas

- **Python 3.x**: Lenguaje principal para la lógica del sistema.
- **Librerías clave**:
    - pyshark: Análisis de paquetes y archivos PCAP.
    - scapy: Captura y manipulación de paquetes en tiempo real.
    - customtkinter: Creación de una interfaz gráfica moderna y personalizable.
    - numpy: Cálculos numéricos para análisis de flujos de red.
    - joblib: Carga de modelos de machine learning preentrenados.
    - threading, os, datetime, collections: Soporte para concurrencia, manejo de archivos y estructuras de datos.
- **Otros**: Dependencia de un modelo de machine learning (Random Forest) almacenado en un archivo .pkl.

Estructura POO del Sistema

El diseño orientado a objetos organiza el sistema en cuatro clases principales:

- **Flujo**: Gestiona paquetes de un flujo de red específico y calcula vectores de características para análisis.
- **AnalizadorVulnerabilidades**: Ejecuta análisis heurístico y de machine learning sobre paquetes y flujos.
- **PacketSniffer**: Captura paquetes en tiempo real y los procesa para análisis o visualización.
- **SnifferGUI**: Interfaz gráfica que coordina la captura, análisis y presentación de resultados.

Beneficios Esperados

- **Detección proactiva**: Identifica amenazas en tiempo real o en archivos PCAP.
- **Usabilidad**: Interfaz gráfica intuitiva para usuarios técnicos y no técnicos.
- **Precisión**: Combina heurísticas con machine learning para mejorar la detección.
- **Flexibilidad**: Soporta capturas en tiempo real y análisis de archivos existentes.
- **Escalabilidad**: Puede adaptarse a diferentes entornos de red mediante ajustes en los filtros BPF.

4.2 Requerimientos

Requerimientos Funcionales

- **RF1**: El sistema debe capturar paquetes de red en tiempo real desde una interfaz seleccionada por el usuario.
- **RF2**: Debe permitir la aplicación de filtros BPF para limitar la captura a protocolos o puertos específicos.
- **RF3**: Debe analizar paquetes en tiempo real utilizando heurísticas (por ejemplo, detección de TTL bajo, tráfico TELNET, Xmas Scan).
- **RF4**: Debe implementar un modelo de machine learning para clasificar flujos de red como potencialmente maliciosos.
- **RF5**: Debe analizar archivos PCAP existentes y generar reportes de alertas.
- **RF6**: Debe guardar capturas en archivos PCAP con nombres personalizables.
- **RF7**: La interfaz gráfica debe mostrar paquetes capturados, progreso de captura y alertas en tiempo real.
- **RF8**: Debe permitir cambiar entre temas claro y oscuro en la interfaz gráfica.

Requerimientos No Funcionales

- **RNF1**: El sistema debe ser compatible con sistemas operativos Windows, macOS y Linux.

- **RNF2**: La interfaz gráfica debe ser responsiva y fácil de usar, con tiempos de respuesta inferiores a 1 segundo para interacciones básicas.
- **RNF3**: El análisis en tiempo real debe procesar paquetes con una latencia mínima (< 100 ms por paquete).
- **RNF4**: El sistema debe manejar capturas prolongadas (hasta 1 hora) sin degradación significativa del rendimiento.
- **RNF5**: Debe garantizar la estabilidad durante capturas en redes de alto tráfico (hasta 10,000 paquetes por segundo).
- **RNF6**: Los archivos PCAP generados deben ser compatibles con herramientas como Wireshark.

5. DESCRIPCIÓN CÓDIGO (POO)

La funcionalidad principal del sistema PacketSniff consiste en la captura de paquetes de red desde una interfaz seleccionada por el usuario, su almacenamiento en tiempo real y su análisis automático mediante un motor heurístico y un modelo de aprendizaje automático previamente entrenado. El objetivo es identificar comportamientos anómalos o amenazas, y alertar al usuario a través de una interfaz gráfica.

5.1. Clases Principales

Archivo: Analizador.py

• AnalizadorVulnerabilidades: Encargada del análisis de archivos .pcap, usando reglas heurísticas y un modelo de Machine Learning.
• Flujo: Representa un flujo de red; extrae estadísticas para alimentar el modelo.

Archivo: newsniff.py

• PacketSniffer: Captura los paquetes, organiza bloques de tráfico y ejecuta el análisis automáticamente.
• SnifferGUI: Interfaz gráfica que permite controlar el sniffer, visualizar alertas y analizar archivos manualmente.

5.2. Métodos Clave

Flujo.calcular_vector

Este método genera el vector de características estadísticas de un flujo de paquetes, que será usado como entrada para el modelo de clasificación.

```
vector = [
    int(dst_port),
    dur,
    len(fwd_lens),
    len(bwd_lens),
    fwd_stats.max(),
    fwd_stats.min(),
    fwd_stats.mean(),
    bwd_stats.max(),
    bwd_stats.min(),
    bwd_stats.mean(),
    total_bytes / dur,
    total_pkts / dur,
    0, 0,
    total_bytes / total_pkts if total_pkts > 0 else 0,
    np.std(np.array([len(pkt.get_raw_packet()) for pkt in self.paquetes])),
    syn, fin, ack, urg
```

5.3 Análisis por Bloques

El sistema implementa un análisis por bloques: cada 10 paquetes capturados, estos se almacenan en un archivo temporal y se analiza automáticamente usando reglas heurísticas y el modelo modelo_randomforest.pkl. Esta lógica está implementada en el método _hilo_analizar_bloques dentro de PacketSniffer.

```
with tempfile.NamedTemporaryFile(delete=False, suffix=".pcap") as temp:
    wrpcap(temp.name, ultimos)

    analizador = AnalizadorVulnerabilidades()
    alertas = analizador.analizar_archivo(temp.name, incluir_resumen_ips=False)
```

5.4. Entrenamiento del Modelo.pkl con RandomForestClassifier

Creamos el archivo entrenar.modelo.py para entrenar todo el dataset que limpiamos previamente en el Google Colab descargamos todos los conjuntos de datos del CICIDS 2017, guardamos las columnas (Features) más útiles, aplicando el preprocesamiento para el entrenamiento. El modelo Random Forest construye múltiples árboles de decisión y combina sus predicciones, lo que permite obtener resultados más estables y confiables frente a variaciones en los datos. Esto es especialmente valioso en entornos de red, donde los patrones de tráfico pueden ser complejos y contener ruido. Además, esta técnica permite trabajar con conjuntos de datos desequilibrados, como es el caso del dataset CICIDS 2017, y ofrece una buena precisión sin requerir un ajuste excesivo de parámetros.

Descarga del Dataset

```
import pandas as pd

df1= pd.read_csv('Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv') # Reemplaz
df2= pd.read_csv('Monday-WorkingHours.pcap_ISCX.csv')  # Reemplaza con el nombre re
df3= pd.read_csv('Wednesday-workingHours.pcap_ISCX.csv')  # Reemplaza con el nombre
df4= pd.read_csv('Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv') # R
df5=pd.read_csv('Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv') # Reempla
```

Limpieza y Prepocesamiento

```
[ ] df_todo = df_todo[df_todo['Label'] != 'Web Attack \ufffd Brute Force']#Limpieza , estos son ataques que depende de un flujo de captura
    df_todo = df_todo[df_todo['Label'] != 'DoS slowloris']
    df_todo = df_todo[df_todo['Label'] != 'DoS Hulk']
    df_todo = df_todo[df_todo['Label'] != 'DoS GoldenEye']

[ ] df_todo["Label"] = (df_todo["Label"]=="BENIGN").astype(int)#convertir el benign en 1 y los ataques en 0

[ ] valores_unicos_Label = df_todo['Label'].unique()#guardar los valores de esa columna

[ ] print(valores_unicos_Label)

    [1 0]
```

Creación del DatasetML

```
# Paso 3: Eliminar las demás columnas
df_filtrado = df_todo[columnas_utiles]

# Paso 4: Guardar el nuevo CSV limpio
df_filtrado.to_csv('datosML.csv', index=False)

# Confirmar columnas finales
print("Columnas conservadas:", df_filtrado.columns.tolist())

Columnas conservadas: ['Destination Port', 'Fwd Packet Length Ma
```

```
Destination Port,Fwd Packet Length Max,Fwd Packet Length Min,Fwd Packet Length Mean,Bwd Packet Length M
22,456,0,64.97560976,976,0,158.0454545,111.8372093,239.6868477,7595.10464,67.12246771,1
22,456,0,64.97560976,976,0,158.0454545,111.8372093,239.6868477,7289.93681,64.42551766,1
22,0,0,0.0,0,0,0.0,0.0,0.0,0.0,12500.0,1
22,456,0,66.53658537,976,0,157.952381,111.452381,241.6427915,7182.267884,63.6753081,1
35396,0,0,0.0,0,0,0.0,0.0,0.0,0.0,38961.03896,1
22,0,0,0.0,0,0,0.0,0.0,0.0,0.0,8196.721311,1
22,456,0,66.53658537,976,0,165.85,114.1707317,243.964772,7161.659039,61.96265564,1
60058,0,0,0.0,0,0,0.0,0.0,0.0,0.0,36585.36585,1
22,0,0,0.0,0,0,0.0,0.0,0.0,0.0,11695.90643,1
22,0,0,0.0,0,0,0.0,0.0,0.0,0.0,9523.809524,1
60060,0,0,0.0,0,0,0.0,0.0,0.0,0.0,40000.0,1
35398,0,0,0.0,0,0,0.0,0.0,0.0,0.0,38961.03896,1
52320,2065,6,1035.5,0,0,0.0,1378.666667,1188.764204,1040000000.0,1000000.0,1
52320,2920,0,1231.133333,6,0,4.0,840.5,1041.088247,667521.0281,758.0953756,1
21,43,0,10.84210526,51,0,19.2,15.24444444,13.8040764,4496.974703,288.4356952,1
53235,6,6,6.0,0,0,0.0,6.0,0.0,4500000.0,750000.0,1
53234,6,6,6.0,6,6,6.0,6.0,0.0,400000.0,66666.66667,1
123,48,48,48,0,48,48,48,0,48,0,0,0,5816.066885,121.1680601,1
```

División y Entrenamiento de los Datos

```
# 2. Separar en características (X) y etiqueta (y)
X = df.drop('Label', axis=1)
y = df['Label']

# 3. Limpiar datos: reemplazar infinitos por NaN y luego rellenar con la me
X.replace([np.inf, -np.inf], np.nan, inplace=True)
X.fillna(X.mean(), inplace=True)

# 4. División en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
# 5. Crear y entrenar modelo con balanceo de clases
clf = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')
clf.fit(X_train, y_train)

# 6. Evaluar modelo
y_pred = clf.predict(X_test)
print("=== Matriz de Confusión ===")
print(confusion_matrix(y_test, y_pred))

print("\n=== Reporte de Clasificación ===")
print(classification_report(y_test, y_pred))

print("\n=== Precisión ===")
print(f"{accuracy_score(y_test, y_pred) * 100:.2f}%")
```

Resultados

Estos resultados justifican la elección de Random Forest como algoritmo de clasificación, ya que ofrece una combinación sólida entre precisión, robustez y eficiencia, claves para su uso en detección de tráfico en tiempo real.

```
=== Matriz de Confusión ===
[[ 85135    789]
 [   966 434110]]

=== Reporte de Clasificación ===
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     85924
           1       1.00      1.00      1.00    435076

    accuracy                           1.00    521000
   macro avg       0.99      0.99      0.99    521000
weighted avg       1.00      1.00      1.00    521000


=== Precisión ===
99.66%

Modelo guardado como 'modelo_randomforest.pkl'
PS C:\Users\Usuario\Documents\Phyton_programa\Proyecto_packsniffer>
```

5.5. AnalizadorVulnerabilidades.analizar_archivo

Analiza el archivo .pcap aplicando primero reglas heurísticas sobre los paquetes individuales, luego agrupa en flujos y envía al modelo si hay suficientes datos.

```
def analizar_archivo(self, archivo_pcap, incluir_resumen_ips=True):
    self.mensajes_mostrados.clear()
    self.alertas.clear()
    self.conteo_ips.clear()  # ← Reinicia el conteo al iniciar nuevo análisis
    try:
        cap = pyshark.FileCapture(archivo_pcap)
    except Exception as e:
        self.alertas.append(f"[ERROR] No se pudo abrir el archivo: {e}")
        return

    for pkt in cap:
        if 'IP' not in pkt:
            continue
        try:
            ip_src = pkt.ip.src
            self.conteo_ips[ip_src] += 1  # siempre contar IP origen
```

5.6. PacketSniffer.process_packet

Procesa cada paquete capturado, lo resume y lo almacena. Cuando se alcanza una cantidad específica, activa el análisis por bloques.

```
if len(self.paquetes_raw) >= 10 and len(self.paquetes_raw) % 10 == 0 and not self.evento_analisis.is_set():
    self.evento_analisis.set()
```

5.7. SnifferGUI.mostrar_paquete

Muestra cada paquete capturado o alerta detectada en el área visual de la GUI. También activa un mensaje emergente en caso de amenazas.

```
def mostrar_paquete(self, resumen):
    if resumen.startswith(PROGRESO_PREFIX):
        valor = int(resumen.replace(PROGRESO_PREFIX, ""))
        self.progress.set(min(valor / 100, 1.0))
        self.label_estado.configure(text=f"Captura en progreso... {valor}%")
    else:
        self.text_area.insert("end", resumen + '\n')
        if resumen.startswith("[ALERTA]"):
            messagebox.showwarning("⚠ Alerta en tiempo real", resumen)
        self.text_area.see("end")
```

5.8. SnifferGUI.analizar_ruta_manual

Permite al usuario cargar un archivo .pcap y ejecutar el análisis con visualización de alertas detectadas en una ventana emergente.

```
try:
    analizador = AnalizadorVulnerabilidades()
    analizador.analizar_archivo(ruta_pcap)

    if analizador.alertas:
        txt.insert("end", "Resultados del análisis:\n\n", "bold")
        txt.insert("end", "Advertencias detectadas:\n", "bold")
        txt.insert("end", f"\nTotal de alertas: {len(analizador.alertas)}", "bold")
        for alerta in analizador.alertas:
            txt.insert("end", alerta + "\n", "warning")
```

6. CONCLUSIONES

El desarrollo del sistema PacketSniff ha permitido demostrar la relevancia de implementar herramientas accesibles y funcionales para la detección de amenazas en redes locales. A través de una arquitectura clara y una interfaz intuitiva, se logró capturar y analizar tráfico de red en tiempo real, incorporando mecanismos de evaluación tanto heurísticos como automáticos. Esta herramienta no solo permite visualizar el comportamiento de la red, sino que también actúa como un primer paso hacia la adopción de prácticas de ciberseguridad más robustas en entornos educativos o profesionales.

Uno de los grandes aciertos del proyecto fue la elección del lenguaje Python, debido a su sintaxis clara, su comunidad activa y la gran cantidad de librerías especializadas disponibles. El uso de herramientas como Scapy, PyShark y CustomTkinter facilitó enormemente la implementación de componentes clave como el sniffer, la interfaz gráfica y el análisis profundo de paquetes. Asimismo, la integración de aprendizaje automático con el modelo Random Forest entrenado sobre el dataset CICIDS 2017 permitió añadir un nivel de inteligencia al análisis, ampliando el alcance del sistema hacia la predicción de amenazas en base a patrones reales.

Durante el proceso de desarrollo se identificaron algunas limitaciones que condicionaron el alcance del proyecto. Por un lado, el análisis de tráfico en tiempo real impone desafíos de rendimiento que deben ser afinados para entornos con alto volumen de paquetes. Además, el entrenamiento del modelo ML se realizó con un subconjunto de características limitadas por la naturaleza de los datos extraídos desde los paquetes, lo cual restringe en cierta medida la capacidad predictiva. Finalmente, el sistema fue probado en entornos controlados, por lo que una validación más extensa requeriría escenarios reales o redes complejas.

A pesar de estas limitaciones, el proyecto demuestra un alto potencial de crecimiento. Con mayor tiempo, experiencia y acceso a entornos reales, se podrían incorporar modelos más avanzados, técnicas de análisis de comportamiento, detección colaborativa y visualización más sofisticada de alertas. PacketSniff ha sido diseñado desde su estructura

para ser modular y escalable, permitiendo su evolución hacia una herramienta más completa que pueda adaptarse a distintas necesidades en el campo de la ciberseguridad.

7. REFERENCIAS

CustomTkinter. (s. f.). CustomTkinter Documentation. Recuperado el 22 de junio de 2025, de https://customtkinter.tomschimansky.com/

EONRaider. (s. f.). Packet-Sniffer [Software]. GitHub. Recuperado el 23 de junio de 2025, de https://github.com/EONRaider/Packet-Sniffer

Joblib. (s. f.). Joblib Documentation. Recuperado el 25 de junio de 2025, de https://joblib.readthedocs.io/

Kiminewt. (s. f.). Pyshark Documentation. Recuperado el 25 de junio de 2025, de https://kiminewt.github.io/pyshark/

lassault. (s. f.). OSniffy [Software]. GitHub. Recuperado el 25 de junio de 2025, de https://github.com/lassault/OSniffy

NumPy. (s. f.). NumPy Documentation. Recuperado el 24 de junio de 2025, de https://numpy.org/doc/

Python Software Foundation. (s. f.). Python Official Documentation. Recuperado el 23 de junio de 2025, de https://docs.python.org/3/

Rassilion. (s. f.). Sniffer [Software]. GitHub. Recuperado el 23 de junio de 2025, de https://github.com/Rassilion/sniffer