WireShark Defense!

By Stephen, Owen, Senchen and Hugo

What was our project?

- Program that monitors network activity
- Uses Wireshark to collect network information
- Monitors the network for potential threats, unusual activity, and surveys general traffic
- Sends messages/alerts when potential threats are detected
- Keeps a list of safe and blacklisted IP addresses
- Fully automated process

Motivations

Why this project?

- Interested in how automated network security works
- We like to snoop around
- Curious about network sniffing

Why Wireshark?

- We love sharks!!!
- Tells you lot about network trafficking (shows IP addresses live, packet data, domain info, etc.)
- Great for uncovering malware, unusual traffic, and all sorts of attacks
- Allows you to closely watch your network (or whoever's network…)

What is WireShark?

- Exactly what it sounds like
- A network protocol analyzer (network monitoring system)
- Analyzes traffic passing through a network
- Free to use and open source
- Widely used for network troubleshooting, security analysis and protocol development

- Helps users spot suspicious activity and detect network problems
- Works with a variety of networks (like wifi and ethernet)

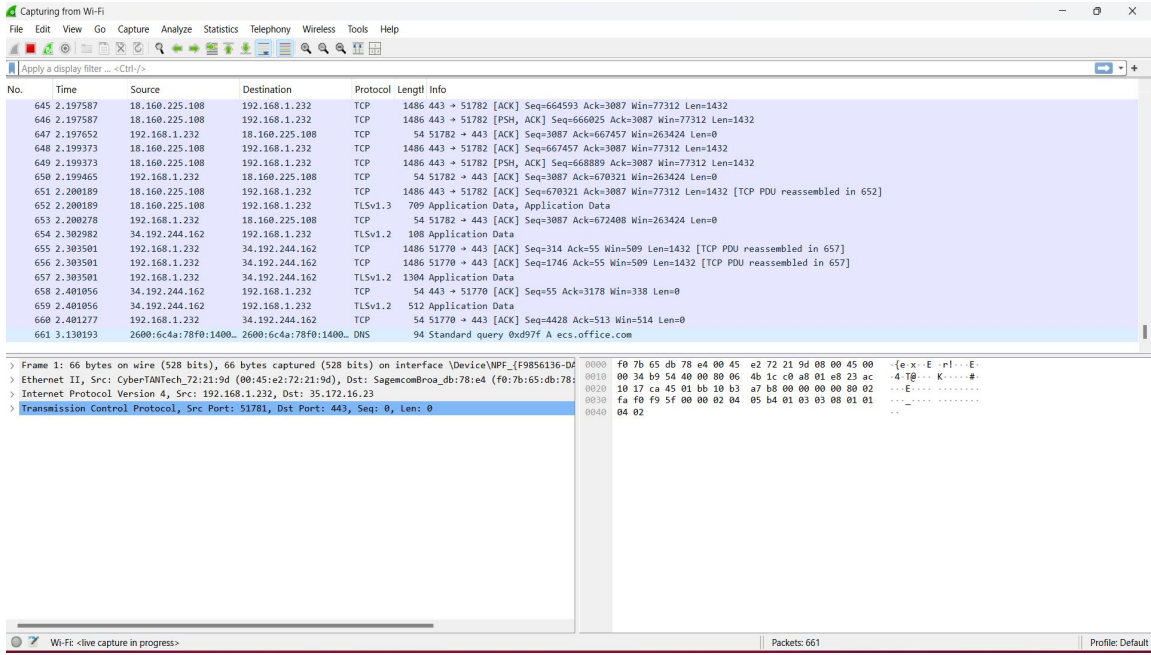
Brief Wireshark History

- Created in 1998 by Gerald Combs under the name Ethereal
- Set out to create a free, open-source alternative to the expensive network analyzers at the time
- Renamed to Wireshark in 2006 due to trademark issues
- Quickly became one of the biggest packet analyzers due to its free to use and open source nature

  - Continually developed today with many large sponsors
  - Over 2 million downloads annually
  - Supports over 20 languages

Wireshark Output

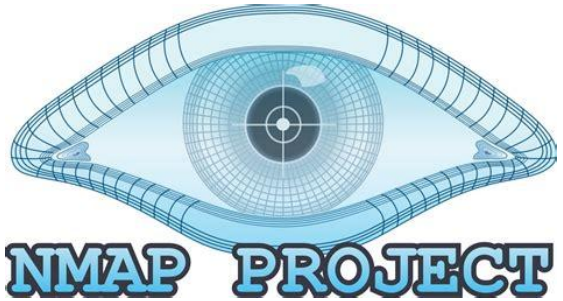
- No.: The packet number in the capture sequence
- Time: Time since the start of the capture when the packet was seen
- Source: The origin IP address or device
- Destination: The receiving IP address/device
- Protocol: The network protocol used (TCP, TLSv1.2, DNS)
- Length: Size of the packet (in bytes)
- Info: Brief summary of what's happening in the packet



Some Discussion Questions

- If you were investigating a suspected network breach, what would you look for?
- What are some creative ways attackers might try to hide their activity in network traffic, and how could you detect them?
- What challenges do organizations face when trying to monitor all network traffic, and how might they overcome them?
- How might the increasing use of IoT devices affect network security monitoring and analysis?

Explain NMap



- Powerful network discovery and security auditing tool
- Scans for live hosts, open ports, and running services
- Identifies operating systems and software versions
- Helps detect vulnerabilities and misconfigurations
- Works on most OS platforms including Linux, Windows, and macOS
- Command-line based with optional GUI (Zenmap)
- Frequently used by network admins, pen testers, and hackers

NMap Part 2

- nmap -sS: Stealth (SYN) scan for quick detection
- n- map -sV: Identify service versions (e.g., Apache 2.4.46)
- nmap -O: OS detection using TCP/IP fingerprinting
- nmap -Pn: Skip host discovery to find hidden systems
- nmap -A: Aggressive scan (OS, versions, scripts, traceroute)
- nmap -p-: Full port scan (1–65535)
- Combine options for deep enumeration (e.g., -sS -sV -O -p-)

How we combine everything

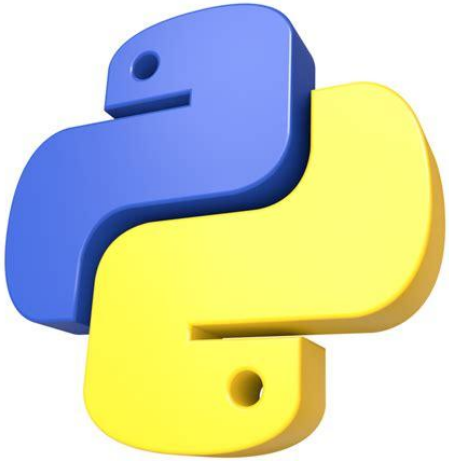
- Adds active scanning to complement packet analysis
- Scans for open ports, services, and OS fingerprints
- Targets suspicious or high-traffic IPs
- Helps validate findings from Wireshark logs
- Uses subprocess or python-nmap for automation
- Results can trigger alerts or update the local database

Application Flow

1. Capture or import .pcap network traffic
2. Run passive analysis using PyShark
3. Analyze traffic summary and device behavior
4. Detect unusual patterns and potential threats
5. Trigger email alerts or further scanning
6. Use Nmap for active verification
7. Store flagged data in local database (SQLite)

Technical Info

- Language: Python 3.11
- Packet Analysis: PyShark (wrapper around TShark)
- Active Scanning: Nmap via subprocess
- Alerting: SMTP email notifications (Gmail API)
- Storage: SQLite for storing blacklisted IPs + logs
- Modular Design: Separate modules for each task
- Extensible: Easily add more threat detection patterns



Capturing Data

- Captures traffic from a specified interface
- Stores packets to .pcap file for offline analysis
- Auto-stops after user-defined duration (e.g., 10 sec)
- Saves output to captures/output.pcap
- Ensures capture directory is created if missing

Capturing Data

```python
def start_capture(interface='Wi-Fi', output_file='captures/output.pcap', duration=10):
    # Ensure output folder exists
    os.makedirs(os.path.dirname(output_file), exist_ok=True)

    # Build the tshark command
    cmd = [
        "tshark",
        "-i", interface,
        "-a", f"duration:{duration}",  # Auto-stop after X seconds
        "-w", output_file              # Output file
    ]

    print(f"Starting capture on {interface} for {duration} seconds...")
    subprocess.run(cmd)
    print(f"Capture saved to {output_file}")

def capture_tshark_main():
    start_capture()

if __name__ == "__main__":
    # Captures tshark data, commnet out to not collect data
    capture_tshark_main()
```

```python
try:
    # Continuously sniff packets until manually interrupted
    for packet in capture.sniff_continuously():
        try:
```

Breaking down the technicalities—device activity

- Identifies active devices on the network via source/destination IPs
- Tracks total number of packets and data sent/received per device
- Flags devices with:
  - Unusually high traffic
  - Repeated connection attempts
  - Frequent communication with external IPs
- Correlates with blacklist to check for known malicious IPs
- Used to generate network activity summaries for further inspection
- Forms the basis for alert triggers and behavior profiling

IP Monitoring

```python
def check_black_listed_ips(ip_set, black_listed_ips):
    try:
        for ip in ip_set:
            if ip in black_listed_ips:
                ip_info = f"ALERT - Black listed ip is on your network!: {ip}"
                print(ip_info)
                send_gmail_email(
                    subject="Network Alert: Black Listed IP",
                    body=ip_info,
                    sender="johnsmith314350@gmail.com",
                    recipient="johnsmith314350@gmail.com",  # or any recipient
                    username="johnsmith314350@gmail.com",
                    password="ohjlpjjajarmltwi"  # <-- app password here
                )
```

- Extracts all unique IPs from .pcap file
- Checks if any match known blacklisted IPs
- Sends email alert if a match is found
- Can scan full traffic logs or live packets
- Separate function supports real-time packet monitoring
- Uses pyshark for packet parsing and SMTP for alerts

Unusual Ports and Protocol Behavior

- Flags use of low, non-standard ports (e.g., 4444, 31337)
- Checks if common services like HTTP appear on wrong ports
- Tracks incomplete TCP handshakes (potential stealth scans)
- Compares observed ports to a known-safe set
- Captures SYN-only connections without ACKs

DNS-Based Threat Monitoring

- Flags clients making excessive DNS requests
- Highlights queries to domains with suspicious TLDs (.ru, .cn, .xyz)
- Uses a domain threat feed for known bad domains
- Identifies potential command-and-control (C2) traffic
- Adds domain frequency stats to reports

Encrypted Traffic to Untrusted Hosts

- Scans TLS handshake data in captured traffic
- Flags self-signed or unusual certificate issuers
- Checks for suspicious keywords in SNI field (e.g., c2, malware)
- Detects expired or unverified certificates (optional)
- Helps expose hidden communication with malicious servers

Database

```python
class LocalDB:
    def __init__(self, db_name: str = 'traffic.db'):
        self.db_name = db_name
        self.conn = sqlite3.connect(self.db_name)
        self.cursor = self.conn.cursor()
        print(f"Connected to database: {db_name}")

    def create_table(self, table_sql: str):
        """Takes a CREATE TABLE SQL statement."""
        try:
            self.cursor.execute(table_sql)
            self.conn.commit()
            print(f"Table created or already exists. SQL: {table_sql}")
        except sqlite3.Error as e:
            print(f"Error creating table: {e}")

    def insert(self, query: str, values: Tuple[Any, ...]):
        """Insert a single row into a table."""
        try:
            self.cursor.execute(query, values)
            self.conn.commit()
        except sqlite3.Error as e:
            print(f"Insert error: {e}")

    def insert_many(self, query: str, values_list: List[Tuple[Any, ...]]):
        """Insert multiple rows using executemany."""
        try:
            self.cursor.executemany(query, values_list)
            self.conn.commit()
        except sqlite3.Error as e:
            print(f"Insert many error: {e}")
```

```python
    def insert_many(self, query: str, values_list: List[Tuple[Any, ...]]):
        """Insert multiple rows using executemany."""
        try:
            self.cursor.executemany(query, values_list)
            self.conn.commit()
        except sqlite3.Error as e:
            print(f"Insert many error: {e}")

    def query(self, query: str, params: Optional[Tuple] = None) -> List[Tuple]:
        """Run a SELECT or general query and return results."""
        try:
            self.cursor.execute(query, params or ())
            return self.cursor.fetchall()
        except sqlite3.Error as e:
            print(f"Query error: {e}")
            return []

    def delete(self, query: str, params: Tuple):
        """Delete rows matching the condition."""
        try:
            self.cursor.execute(query, params)
            self.conn.commit()
        except sqlite3.Error as e:
            print(f"Delete error: {e}")

    def close(self):
        """Close the database connection."""
        self.conn.close()
        print("Database connection closed.")
```

Traffic Summary and Alert System

- Reads and analyzes a network capture file
- Checks for large downloads by calculating how much data destinations receive
- Sends out an email alert if any device downloads more than a certain amount (100mb)
- Prints summary statistics about network traffic

```
def check_large_downloads(df, threshold_bytes, email_config):
    large_downloads = {}
    alert_body = ""

    for ip in df['Destination'].unique():
        total_received = df[df['Destination'] == ip]['Length'].sum()
        if total_received > threshold_bytes:
            large_downloads[ip] = total_received
            alert_body += f"{ip}: {total_received/1024/1024:.2f} MB\n"

    if large_downloads:
        print("\nALERT: Large downloads detected!")
        for ip, size in large_downloads.items():
            print(f"  -> {ip} received {size/1024/1024:.2f} MB")

        send_gmail_email(
            subject="Network Alert: Large Download Detected",
            body=alert_body,
            sender=email_config['sender'],
            recipient=email_config['recipient'],
            username=email_config['username'],
            password=email_config['password']
        )
```

```
def analyze(filepath):
    print("\n [Network Traffic Summary]")
    print(" Preparing to scan packets...")

    try:
        cap = pyshark.FileCapture(
            filepath,
            only_summaries=True,
            display_filter="dns or http"
        )
    except Exception as e:
        print(f" Could not open the file: {e}")
        return

    start_time = time.time()

    try:
        print(" Preloading packets into memory...")
        cap.load_packets()
        print(f" Loaded {len(cap)} packets in {round(time.time() - start_time, 2)} seconds.")
    except Exception as e:
        print(f" Failed to load packets: {e}")
        return
```

```
packets = []
print(" Starting analysis...")

for i, pkt in enumerate(cap):
    try:
        packets.append({
            'No': int(pkt.no),
            'Time': float(pkt.time),
            'Source': pkt.source,
            'Destination': pkt.destination,
            'Protocol': pkt.protocol,
            'Length': int(pkt.length),
        })
        if i % 100 == 0:
            print(f"   -> Processed {i} packets...", end='\r')
    except Exception:
        continue

cap.close()
print(f"\n Processed {len(packets)} packets in total.")

df = pd.DataFrame(packets)

check_large_downloads(
```

```
check_large_downloads(
    df=df,
    threshold_bytes=100 * 1024 * 1024,  # 100 MB
    email_config={
        'sender': "johnsmith314350@gmail.com",
        'recipient': "johnsmith314350@gmail.com",
        'username': "johnsmith314350@gmail.com",
        'password': "ohjlpjjajarmltwi"
    }
)

print(f"\n Total messages observed: {len(df)}")
print(f" Total data transferred: {df['Length'].sum()} bytes")

print("\n Most Common Communication Types (Protocols):")
for protocol, count in df['Protocol'].value_counts().head(10).items():
    print(f"   -> {protocol}: {count} messages")

print("\n Devices that sent the most messages (Source IPs):")
for ip, count in df['Source'].value_counts().head(5).items():
    print(f"   -> {ip}: {count} messages sent")

print("\n Most contacted devices or websites (Destination IPs):")
for ip, count in df['Destination'].value_counts().head(5).items():
    print(f"   -> {ip}: {count} messages received")
```

In Class Demo

Which display of network activity looks better (safer network)?

What possible threats can you notice?

No.	Time	Source IP	Destination IP	Protocol	Length	Info
1	0.000000	192.168.1.15	10.0.0.5	TCP	66	[SYN] Seq=0 Win=64240 Len=0
2	0.000100	10.0.0.5	192.168.1.15	TCP	66	[SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
3	0.000200	192.168.1.15	10.0.0.5	TCP	54	[ACK] Seq=1 Ack=1 Win=64240 Len=0
4	0.001000	192.168.1.15	10.0.0.5	TCP	1500	[PSH, ACK] Encrypted Data (TLS)
5	0.002000	192.168.1.15	10.0.0.5	TCP	1500	[PSH, ACK] Encrypted Data (TLS)
...	...	...	...	...	...	...
100	0.100000	192.168.1.15	10.0.0.5	TCP	1500	[PSH, ACK] Encrypted Data (TLS)
...	...	...	...	...	...	...
200	0.200000	192.168.1.15	10.0.0.5	TCP	1500	[PSH, ACK] Encrypted Data (TLS)

No.	Time	Source IP	Destination IP	Protocol	Length	Info
1	0.000000	192.168.1.10	8.8.8.8	DNS	74	Standard query A www.google.com
2	0.001200	8.8.8.8	192.168.1.10	DNS	90	Standard query response A 142.250.190.4
3	0.003400	192.168.1.10	142.250.190.4	TCP	66	[SYN] Seq=0 Win=64240 Len=0 MSS=1460
4	0.003800	142.250.190.4	192.168.1.10	TCP	66	[SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
5	0.004100	192.168.1.10	142.250.190.4	TCP	54	[ACK] Seq=1 Ack=1 Win=64240 Len=0
6	0.005200	192.168.1.10	142.250.190.4	HTTP	518	GET /search?q=wireshark HTTP/1.1
7	0.006500	142.250.190.4	192.168.1.10	HTTP	1500	HTTP/1.1 200 OK (text/html)

Applications

What could this be used for?

- Campus network monitoring for suspicious activity
- Small business cybersecurity with minimal cost
- Detecting early signs of malware or ransomware behavior
- Monitoring employee or device network habits
- Teaching tool for cybersecurity classesDIY home network intrusion detection system

Real-world counterparts

Similar software already out there:

- Snort – Real-time packet-based IDS
- Suricata – High-performance IDS/IPS with rule-based filtering
- Zeek (Bro) – Network analysis framework for behavioral monitoring
- Security Onion – Full-stack Linux distro for monitoring
- Splunk – Enterprise-level log and traffic analyzer
- Darktrace – AI-based anomaly detection

Possible additions for the future

What we might add down the line (but we ain't actually doing this)Web-based dashboard for traffic visualization:

- Machine learning-based anomaly detection
- Integration with threat intelligence feeds (e.g., AbuseIPDB)
- Real-time packet classification
- GeoIP mapping of traffic sources
- Auto-quarantine or firewall rule updates
- Custom rules engine for defining threat conditions

Our Takeaways from the project — All if us

- What did we all learn from this?
- What were some unexpected struggles?
- Our main takeaways

Sources?

https://www.wireshark.org/

https://www.webasha.com/blog/wireshark-the-ultimate-network-protocol-analyzer-for-beginners

Thank You!