Network Intrusion Detection System (NIDS) Project Report

Sugnana Murthy GM

June 15, 2025

1 Introduction

The Network Intrusion Detection System (NIDS) is designed to monitor network traffic for suspicious activities, such as port scans and SYN floods, to enhance network security. Developed as a Python-based application, the system captures TCP packets using the Scapy library, analyzes them for potential threats, and logs alerts with detailed information. The project aims to provide real-time detection of malicious activities, including access to vulnerable ports and spam-like behavior, with a customizable logging mechanism that clears previous logs on each run to focus on current session data.

2 System Design

The NIDS is composed of four main modules, each handling a specific functionality:

- **main.py**: Initializes the system, sets up logging, and starts packet sniffing using Scapys `sniff` function. It runs synchronously to ensure reliable packet capture.
- **packet_analyzer.py**: Analyzes captured packets for threats, including port scans (3+ unique ports), SYN floods (5+ SYN packets in 10 seconds), vulnerable port access (e.g., ports 445, 3389), and spam-like behavior (5+ packets to a port in 5 seconds).
- **alert_logger.py**: Manages logging to both console and `alerts.log`, with a custom format including alert ID, timestamp, severity, and packet details (e.g., source/destination IPs, ports, counts). The log file is cleared on each run.
- **config.py**: Stores configuration parameters, such as thresholds (`SYN_FLOOD_THRESHOLD`, `PORT_SCAN_THRESHOLD`) and network interface (`Wi-Fi`).

The system architecture ensures modularity, allowing easy updates to detection logic or logging formats.

3 Implementation

The NIDS implements the following features:

1. **Packet Capture**: Uses Scapy to capture TCP packets on the specified interface (e.g., `Wi-Fi`).

2. **Threat Detection**:
   - **Port Scan**: Detects when a source IP targets 3 or more unique ports, logging details like ports scanned and count.
   - **SYN Flood**: Identifies 5 or more SYN packets from a source IP within 10 seconds, indicating a potential denial-of-service attack.
   - **Vulnerable Ports**: Monitors access to ports 445 (SMB), 3389 (RDP), 1433 (MSSQL), 23 (Telnet), and 21 (FTP), logging each first access with a count.
   - **Spam-like Behavior**: Detects 5 or more TCP packets to a single port within 5 seconds, suggesting brute-force or spam attempts.
3. **Logging**: Generates detailed logs with:
   - Unique alert ID (UUID).
   - Timestamp, severity (INFO, WARNING, CRITICAL).
   - Packet details (source/destination IPs, ports, protocol, counts).

   The `alerts.log` file is cleared on each run to store only current session data.
4. **Console Output**: Displays colored alerts (green for INFO, yellow for WARNING, red for CRITICAL) using the Colorama library.

Example log entry in `alerts.log`:

```
2025-06-15 23:17:05 - WARNING - AlertID: <uuid> - Port Scan detected from
    192.168.1.100: 3 unique ports targeted (src_ip: 192.168.1.100, dst_ip:
    192.168.1.9, ports: 80, 445, 3389, protocol: TCP, port_count: 3)
```

4 Testing

To test the NIDS, follow these steps:

1. **Setup**:
   - Move the project to `C:\NIDS_project` to avoid OneDrive issues.
   - Install dependencies: `pip install scapy colorama uuid`.
   - Verify network interface in `config.py` using:

   ```
   from scapy.all import get_if_list
   print(get_if_list())
   ```
2. **Run NIDS**:

   ```
   cd C:\NIDS_project
   python main.py
   ```

   Expected console output:

   ```
   [GREEN][INFO] AlertID: <uuid> - Testing logging functionality
   Starting NIDS... Press Ctrl+C to stop.
   ```

3. **Generate Traffic**:

   - Install nmap from https://nmap.org/download.html.
   - Run a port scan targeting vulnerable ports:

   ```
   nmap -sS -p 80,443,445,3389 192.168.1.9
   ```

   - Simulate spam with hping3 (on Linux/WSL):

   ```
   sudo apt install hping3
   hping3 -S -p 80 -i u10000 192.168.1.9
   ```

4. **Verify Output**: Check alerts.log for entries like:

```
2025-06-15 23:17:05 - WARNING - AlertID: <uuid> - Vulnerable port
    access detected from 192.168.1.100 to port 445 (src_ip:
    192.168.1.100, dst_ip: 192.168.1.9, dst_port: 445, protocol: TCP,
    vuln_count: 1)
2025-06-15 23:17:05 - WARNING - AlertID: <uuid> - Port Scan detected
    from 192.168.1.100: 3 unique ports targeted (src_ip: 192.168.1.100,
     dst_ip: 192.168.1.9, ports: 80, 445, 3389, protocol: TCP,
    port_count: 3)
```

5 Results

As of June 15, 2025, the NIDS successfully logs the test message to alerts.log:

```
2025-06-15 17:28:46 - INFO - AlertID: <uuid> - Testing logging
    functionality
```

The system captures packets, as evidenced by console output showing port scans (3 ports by 192.168.1.9) and SYN packets (6 from 192.168.1.9). However, alerts for port scans, SYN floods, vulnerable ports, and spam are not yet logged to alerts.log, possibly due to:

- Logic errors in packet_analyzer.py.
- Packet capture issues (e.g., incorrect interface or firewall).
- File write issues, though less likely since the test log works.

Troubleshooting steps include:

- Verify console output after running nmap -sS -p 80,443,445,3389 192.168.1.9.
- Disable Windows Firewall temporarily:

```
netsh advfirewall set allprofiles state off
```

- Check dependencies: pip show scapy colorama uuid.

6 Future Work

Future enhancements include:

- **GUI**: Develop a Flask-based web interface to display real-time alerts.
- **Email Notifications**: Send alerts for critical events (e.g., SYN floods) via SMTP.
- **Visualization**: Generate charts to summarize threat frequency, such as:

```
{
  "type": "bar",
  "data": {
    "labels": ["Port Scans", "SYN Floods", "Vulnerable Ports", "Spam
       Attempts"],
    "datasets": [{
      "label": "Detected Threats",
      "data": [0, 0, 0, 0]
    }]
  }
}
```

- **Advanced Detection**: Incorporate machine learning to identify complex attack patterns.

7 Conclusion

The NIDS project successfully implements packet capture and basic logging, with a modular design that supports future enhancements. While the test log is recorded, alert logging requires further debugging. By addressing current issues and implementing proposed features, the system can become a robust tool for network security monitoring.