Liam Calder V00963268
CSC 466
February 21 2025

Lightweight DDoS Alarm System:
Comparative Analysis of Detection Methods for Home Networks
Bi-Weekly Update 1

Planned Deliverables for Feb 21:

1. Initial System Design
    - Outline a lightweight DDoS alarm architecture.
    - Allow switching between threshold-based and anomaly-based detection methods.
2. Threshold-Based Detection Module
    - Begin coding a threshold-based approach to detect common DoS attacks (SYN Floods, UDP Floods, etc.).
    - Implement basic packet capture, logging, and alerting mechanisms.
3. Literature Review Findings
    - Investigate a recent paper or study that could improve my design, especially regarding lightweight yet effective DDoS detection.

How I Met These Goals:

Initial System Design

- Modular Python Script: I wrote a Python script using Scapy (for packet sniffing) and psutil (for interface management). It supports threshold-based detection while setting the groundwork for anomaly detection.
- Plug-and-Play Detection: I developed classes for ThresholdDetector and AnomalyDetector so I or anyone in the future could switch or expand detection methods simply by updating the class in use.
- Lightweight Footprint: My focus was on minimal overhead by capturing straightforward, packet-level features (e.g., SYN/UDP counts, source IPs) that can run comfortably on home-network hardware.

Threshold-Based Detection Module:

- Packet Counts & Time Windows: I used counters for TCP SYN packets and UDP packets, and monitored over a configurable time window (e.g., 10 seconds). When counts exceed my set thresholds (THRESHOLD_SYN, THRESHOLD_UDP), it flags a potential DoS attack. For now, I have the window set to 10 seconds to avoid making the system too sensitive to short-term spikes but shortening the window could increase detection time and accuracy. I will further experiment with this.
- Alerts & Logging:

    - Console Alerts: The system reports when an attack starts and ends, including which source IPs might be involved.
    - Detailed Metrics: After each time window, I append data to performance_metrics.txt (true positives, detection time, etc.) for later analysis. These metrics still need work as their accuracy right now is low.
- Testing with Synthetic DoS Traffic: I simulated large bursts of SYN and UDP traffic, verifying that the threshold-based module raises appropriate alerts.

```
=== ALERT ===
DoS Alert: UDP Flood Detected
Attack started at 12:17:38.
Source(s): redacted
==============


=== ALERT ===
DoS Alert: UDP Flood Ended
Attack ended at 12:18:28.
Duration: 49.46 seconds.
Source(s): redacted
==============
```

```
Welcome to the DoS Alarm System!

Available Network Interfaces:
0: \Device\NPF_{AA08BA13-6669-4385-8F9A-B0576A55A7AA} (Ethernet)
1: \Device\NPF_{CF83EE84-0E1F-4AA2-8784-5D7B5E2996FA} (VirtualBox Host-Only Network)
2: \Device\NPF_{4F610AB2-3F8C-4E26-B0B4-A3E3E3841B5A} (Local Area Connection* 3)
3: \Device\NPF_{0C1DD229-33E0-4416-B8F2-5CCF1579E368} (Local Area Connection* 4)
4: \Device\NPF_{6563B7AA-E715-4B74-87AD-DDB4D577F47E} (Wi-Fi)
5: \Device\NPF_{0CDFF42D-B437-4379-B15B-542C026B396E} (Bluetooth Network Connection)
6: \Device\NPF_{B459DF4A-512D-4D05-AD11-9986D13DDB68} (Loopback Pseudo-Interface 1)
7: \Device\NPF_{536EF720-157C-42E9-A12C-0AEA26037F58} (Unknown)
8: \Device\NPF_{BB917FD2-5B1A-41D0-8376-F6A3DED3D1B1} (Unknown)
9: \Device\NPF_Loopback (Unknown)
```

Current ThresholdDetector class:

```python
class ThresholdDetector:
    """Handles threshold-based detection by counting packets."""
    def __init__(self, syn_threshold, udp_threshold):
        self.syn_threshold = syn_threshold
        self.udp_threshold = udp_threshold
        self.packet_counts = defaultdict(int)
        self.source_ips = set()  # Collect source IP addresses

    def update(self, packet):
        if packet.haslayer(IP):
            self.source_ips.add(packet[IP].src)
        if packet.haslayer(TCP):
            tcp_layer = packet.getlayer(TCP)
            if tcp_layer.flags == 'S':  # SYN flag
                self.packet_counts['SYN'] += 1
        if packet.haslayer(UDP):
            self.packet_counts['UDP'] += 1

    def detect(self):
        alerts = []
        if self.packet_counts['SYN'] > self.syn_threshold:
            alerts.append(f"High SYN packet rate: {self.packet_counts['SYN']} packets in {WINDOW_SIZE} seconds")
        if self.packet_counts['UDP'] > self.udp_threshold:
            alerts.append(f"High UDP packet rate: {self.packet_counts['UDP']} packets in {WINDOW_SIZE} seconds")
        return alerts

    def reset(self):
        self.packet_counts.clear()
        self.source_ips.clear()
```

Literature Review: Insights from “LUCID: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection”:

- Lightweight Architecture
    - LUCID’s CNN (Convolutional Neural Networks) model shows it’s possible to achieve accurate DDoS detection in resource-constrained environments by reducing the number of network parameters. This resonates with my design, where I want to keep overhead low for real-time monitoring.
- Minimal Feature Set
    - The authors concentrate on packet-level features (e.g., packet length, IP flags), avoiding the complexity of detailed flow-level or payload-based attributes. This approach mirrors my approach of keeping detection nimble with basic counters and traffic metadata.
- High Accuracy With Simple Preprocessing
    - Their work uses modest window sizes and transforms packets into 2D arrays, maintaining high detection accuracy without consuming a lot of resources. This affirms that lightweight real-time detection is not only viable but also effective.
- Suitable for Edge & Home Systems
    - Tested on NVIDIA Jetson TX2 boards, LUCID demonstrates a CNN’s feasibility in smaller devices. Since my target is a home network environment, it’s reassuring to see successful results on lower-power hardware.
- Hybrid Strategy
    - While LUCID relies on a CNN, it hints at the value of “fast trigger” rules (like thresholds) working alongside more complex anomaly models. My plan is indeed to marry quick, threshold-based alerts with deeper ML-based or autoencoder-based checks for subtler threats.

From these insights, I’ve gained confidence that my two-pronged approach of threshold-based detection plus a forthcoming anomaly-based module will meet the demands of a home network’s limited resources without sacrificing detection accuracy.

Next Steps:

1. Complete the Anomaly Detection Module
    - Expand the AnomalyDetector to possibly include a CNN or autoencoder, following LUCID’s emphasis on minimal overhead.
2. Refine Modular Switching
    - Continue streamlining the command-line or config-based selection for threshold vs. anomaly detection.
    - Compare false positives, detection times, etc., under real or simulated attack traffic.

3. Advanced Alerts & Logging
    - Add email or SMS alerts.
    - Implement rate-limiting to avoid repetitive alerts if an attack persists.
    - Track CPU/memory usage to ensure the system stays “lightweight.”
4. DoS Simulation & Testing
    - Develop or refine scripts (SYN Flood, UDP Flood) to rigorously test and collect performance data.
    - Automate experiments so I can compare threshold-based and anomaly-based detection over repeatable test runs.

References:

Doriguzzi-Corin, R., Millar, S., Scott-Hayward, S., Martinez-del-Rincon, J., & Siracusa, D. (2020). LUCID: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection. *IEEE Transactions on Network and Service Management*. Advance online publication. https://doi.org/10.1109/TNSM.2020.2971776