Network Protocol Defense System

Anthonios Deeb (ID: 324933993) Wasim Shebalny (ID: 323944280)

September 30, 2024

Contents

1 Introduction

The purpose of this project is to develop a network traffic analysis and intrusion detection system. The system is designed to receive and process network traffic in real-time, classify it into various flows, and detect any potential security threats based on the network data. Using traffic replay techniques, the system simulates real-world network environments and uses a 5-tuple flow mechanism to analyze the flow of information across various protocols. The goal is to provide an efficient and scalable solution for detecting malicious activity in network traffic.

The project consists of multiple components, including flow management, intrusion detection, system logging, and parsing capabilities, all integrated with a traffic simulation environment using `tcpreplay`. This system also features a user interface that visualizes the network traffic data and highlights potential attacks detected by the intrusion detection system (IDS).

2 Flow Management

The flow management system is responsible for tracking and analyzing the flow of data between different network entities. A flow is defined by a 5-tuple structure that consists of the following components:

- Source IP Address
- Destination IP Address
- Source Port
- Destination Port
- Protocol (TCP/UDP)

The system captures each packet of network traffic and categorizes it into one of these flows. Metadata is then extracted for each flow, including the total amount of data transferred, the number of packets, and the duration of the flow. This data is crucial for identifying anomalies or potential attacks within the network traffic.

In this project, we implement a real-time flow management module that captures live network traffic and processes each packet. The 5-tuple mechanism is used to group packets into distinct flows, allowing for precise tracking of network activity. The metadata for each flow is then logged and visualized in the system's web-based dashboard, providing a clear view of network usage and potential security risks.

3 Intrusion Detection System

The intrusion detection system (IDS) is designed to identify potential security threats within the network traffic. By analyzing the captured flows, the IDS is able to detect common attack patterns, such as port scans, SYN floods, and Slowloris attacks.

The IDS uses a combination of signature-based and anomaly-based detection techniques. Signature-based detection involves matching known attack signatures with the incoming traffic, while anomaly-based detection looks for deviations from normal traffic

patterns. Any detected threats are logged and visualized on the dashboard for further investigation.

4 Packet Processing

Packet processing ensures that all incoming network packets are properly parsed and analyzed before passing through the detection system.

- **Packet Parsing**: Extracts key information (source/destination IP, protocol, etc.) from raw packets.
- **Flow Aggregation**: Groups related packets into flows for analysis.
- **Attack Detection Integration**: Passes parsed packets to the detection module for threat evaluation.
- **Error Handling**: Manages malformed or corrupted packets to avoid system crashes.

5 Packet Validation

Before processing packets, the system ensures that they are valid and follow protocol standards. Invalid packets are discarded.

- **Packet Integrity Checks**: Verifies packet structure and format adherence to networking protocols.
- **Validation of Protocols**: Ensures that packets conform to expected protocols and flags any discrepancies.
- **Anomaly Detection**: Detects abnormal packet behaviors such as large payloads or mismatched headers.
- **Rejection of Malformed Packets**: Discards corrupted packets to prevent disruption of network analysis.

5.1 Features of Detection Functions

Each detection function is tailored to identify a specific type of attack or security breach. The following are key features of the detection capabilities:

5.1.1 Slowloris Attack Detection

- **Purpose**: Detects Slowloris attacks, where a server is flooded with incomplete HTTP requests to exhaust its resources.
- **Feature**: The function monitors open connections and identifies suspiciously slow or incomplete connections from the same IP address over time.

5.1.2 SYN Flood Attack Detection

- **Purpose**: Identifies SYN Flood attacks, a type of Denial of Service (DoS) attack that disrupts TCP connections by overloading the target with half-open connections.
- **Feature**: The detection function tracks the number of SYN packets that have not completed the TCP handshake. If a threshold is crossed, the activity is flagged as a potential attack.

5.1.3 Port Scanning Detection

- **Purpose**: Detects port scanning activities, where attackers scan for open ports to exploit vulnerabilities.
- **Feature**: The system tracks connection attempts to multiple ports from a single source. A pattern of rapid port scans is flagged as malicious behavior.

5.1.4 Data Exfiltration Detection

- **Purpose**: Detects unauthorized data transfers from the internal network to an external source, potentially indicating data exfiltration.
- **Feature**: The function monitors outbound traffic and flags unusual data transfer volumes or connections to suspicious destinations. Alerts are triggered when thresholds are breached.

5.1.5 ARP Spoofing Detection

- **Purpose**: Detects ARP spoofing attacks, where an attacker sends falsified ARP (Address Resolution Protocol) messages to link their MAC address to another legitimate IP address.
- **Feature**: The detection function monitors ARP requests and responses for inconsistencies, triggering an alert if a spoofed packet is detected.

5.1.6 Brute Force Attack Detection

- **Purpose**: Detects brute force attempts targeting authentication mechanisms, such as FTP or SSH logins.
- **Feature**: Tracks failed login attempts and flags repeated unsuccessful access attempts from the same IP address as a potential brute force attack.

5.1.7 URL Filtering and SQL Injection Detection

- **Purpose**: Detects potential SQL injection attempts and identifies access to known malicious URLs.
- **Feature**: The detection system inspects HTTP request payloads for suspicious patterns and matches URLs against a blacklist of known malicious domains.

5.1.8 DNS Tunneling Detection

- **Purpose**: Identifies DNS tunneling, where data is exfiltrated through DNS queries.
- **Feature**: The function analyzes the frequency and content of DNS queries, flagging suspicious patterns that may indicate tunneling activity.

5.1.9 ICMP Flood Detection

- **Purpose**: Detects ICMP (Internet Control Message Protocol) flood attacks, where a network is overwhelmed with ICMP Echo requests.
- **Feature**: The function tracks the rate of ICMP requests and flags abnormal spikes as potential flood attacks.

5.1.10 DNS Spoofing Detection

- **Purpose**: Detects DNS spoofing, where an attacker sends fake DNS responses to divert traffic.
- **Feature**: Monitors DNS responses and flags inconsistencies between queries and their respective responses.

6 Database Utilities and Packet Storage

The system relies on a well-structured SQLite database to store and organize traffic data efficiently. This component plays a critical role in ensuring that all network packets, flows, and detected attacks are correctly logged and can be accessed for further analysis.

6.1 Key Features:

- **Database Initialization**: At the start of the system, the database is initialized to ensure that essential tables, such as `traffic_data`, `flow_data`, and `detected_attacks`, are ready to store incoming traffic and attack details.
- **Efficient Data Handling**: To prevent performance issues, the system groups incoming packets into batches before storing them. This allows the system to process large volumes of data quickly without overloading the database.
- **Flow Tracking**: The system organizes packets into network flows based on key parameters like source/destination IP addresses, ports, and protocol. This structure provides insights into the behavior of network traffic over time.
- **Asynchronous Attack Logging**: Attack logs are processed asynchronously, meaning that detection events are recorded without interrupting real-time traffic analysis. This allows the system to remain highly responsive even during heavy traffic or attack detection events.
- **Concurrency and Error Handling**: The system is designed to handle multiple database operations simultaneously through threading, ensuring smooth and reliable performance even when the database is under heavy load.

7 Logging System

The logging system is responsible for managing log entries generated by the IDS. Logs are crucial for monitoring and investigating detected attacks and other network events.

7.1 Logging Utilities

- **Thread-Safe Logging**: Manages logs asynchronously using a queue system, ensuring that logging does not interfere with real-time traffic processing.
- **Log Attack to Database**: Handles database logging for detected attacks, ensuring that entries are properly stored and indexed.
- **Log Flush**: Ensures no log data is lost, even in high-traffic scenarios.

8 Signal Handling and System Resilience

The system includes mechanisms for robust signal handling to ensure resilience during operation. This component provides graceful shutdown and resource cleanup capabilities, particularly useful in long-running network monitoring processes.

8.1 Key Features:

- **Graceful Shutdown**: The system listens for termination signals such as `SIGINT` (interrupt from keyboard) and `SIGTERM` (termination signal). When these signals are received, the system ensures that all ongoing processes are properly terminated, and any remaining data is flushed to the database before shutting down.
- **Resource Management**: Signal handling ensures that resources like database connections, file handles, and threads are safely closed when the system is stopped. This prevents data corruption and memory leaks.
- **Asynchronous Handling**: By incorporating threading and asynchronous operations, the system ensures that critical tasks, such as attack logging or flow analysis, can complete without interruption, even when a shutdown signal is received.
- **System Recovery**: The system can be restarted cleanly after termination, with no loss of previously processed data. Any temporary states, like unfinished batch commits, are safely handled on restart.

9 Shared Variables and Configuration

The `shared.py` file plays a crucial role in the system by providing globally accessible variables and configurations used across different modules. It is primarily responsible for sharing flow tracking data and the directory path for the SQLite database.

9.1 Key Features:

- **Flow Tracking**: The `flows` variable is a shared dictionary, implemented using Python's `defaultdict`, to store and manage network flows throughout the system. This allows multiple components, such as packet processors and flow analyzers, to update and query the status of various network flows concurrently.
- **Database Directory Path**: The `DIR` variable specifies the directory path where the SQLite database is stored. It is configured to be flexible, with one path for Docker-based execution (`/app/db/`) and another for local execution during development (`Database/`).
- **Centralized Configuration**: By keeping shared variables like `flows` and `DIR` in a dedicated file, the system achieves better modularity and reduces redundancy. Any updates or changes to these variables are easily propagated across all modules that rely on them.

10 User Interface and Visualization

The user interface is managed by the `appy.py` file, which handles web server functionality using Flask. This file is responsible for rendering the web pages, connecting to the database, and displaying real-time data about network traffic and detected attacks.

The interface provides the following features:

10.1 Real-time Network Data

The dashboard displays key metrics such as:

- Total number of packets processed
- Amount of data transferred (in bytes)
- Current network flows and their details (source, destination, protocol, etc.)
- Detected attacks with timestamps and descriptions

10.2 Traffic Visualization

The web server periodically queries the SQLite database for new traffic data, which is visualized in a user-friendly format. The interface includes:

- A dynamic line chart showing data transferred over time
- A table listing the real-time network flows, including source IP, destination IP, and port numbers
- A section highlighting detected attacks, with details such as attack type and description

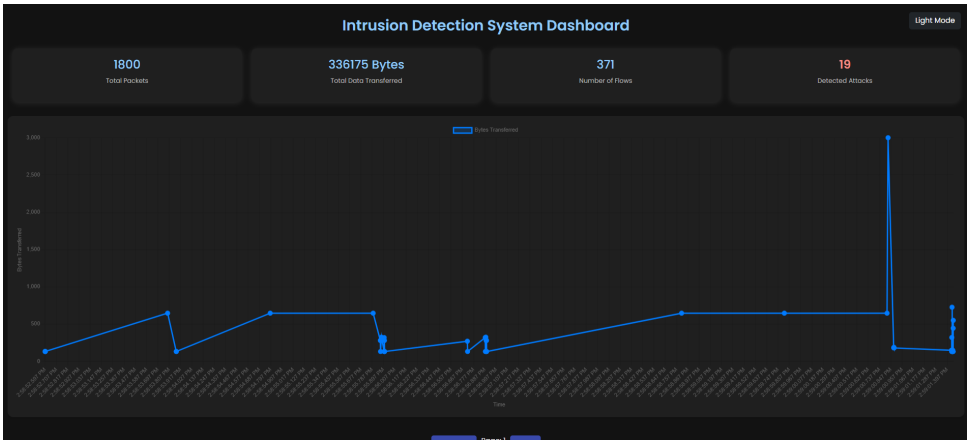

Figure 1: Real-time network traffic dashboard.

10.3 Attack Monitoring

Any detected attacks are logged in the SQLite database and displayed in a dedicated section of the dashboard. This allows the user to:

- View the attack type (e.g., SYN Flood, Slowloris, etc.)
- See the description and timestamp of each attack
- Monitor the frequency and severity of attacks in real time

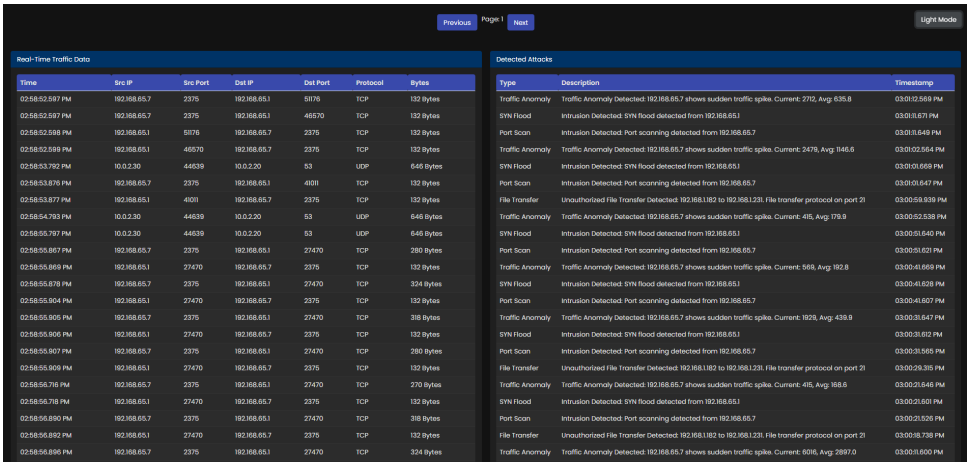

Figure 2: Detected attacks and traffic data visualization.

10.4 Database Interaction

The `appy.py` file connects to the SQLite database, retrieving stored network traffic and attack data. Key functionalities include:

- Retrieving flow data to populate the dashboard
- Fetching detected attacks for real-time monitoring

- Initiating and closing database connections securely to avoid corruption

The web interface updates periodically, ensuring that the user has access to the most recent network traffic information and attack logs.

11 Unit Testing

To ensure the system operates correctly and efficiently, we implemented a series of unit tests using the Python `unittest` framework. These tests focus on verifying the key components of the Intrusion Detection System (IDS), ensuring that every module functions as expected and handling edge cases effectively.

11.1 Testing Strategy

The following areas of the system are covered by unit tests:

- **Flow Management**: Verifies that network packets are correctly grouped into flows based on the 5-tuple model (source IP, destination IP, source port, destination port, protocol). Tests include adding packets to existing flows, creating new flows, and ensuring accurate metadata collection.
- **Intrusion Detection**: Tests check whether the IDS correctly identifies various attack types such as SYN Flood, Slowloris, ARP Spoofing, and others. The unit tests simulate attack traffic and evaluate whether the detection algorithms flag the attacks as expected.
- **Database Operations**: These tests verify that the system can correctly insert and retrieve data from the SQLite database. The focus is on ensuring that packet information, flows, and attack logs are stored and queried correctly without data loss or corruption.
- **Logging and Error Handling**: Unit tests ensure that all detected attacks are logged asynchronously without disrupting real-time traffic processing. The system also handles any unexpected errors gracefully, and these behaviors are validated through tests.
- **Packet Validation and Processing**: Tests in this area ensure that the system correctly parses network packets, checks for protocol compliance, and appropriately handles malformed packets. Valid packets are passed through the system for further analysis, while invalid ones are discarded.
- **Signal Handling**: These tests verify that the system responds appropriately to termination signals (such as `SIGINT` or `SIGTERM`), ensuring that ongoing processes are completed before the system shuts down. This prevents data loss and ensures the graceful termination of all components.

11.2 Execution of Unit Tests

The unit tests in the system are written using Python's `unittest` framework, which allows for both individual and comprehensive test execution. Each test module focuses on a particular aspect of the system (e.g., flow management, database operations, or intrusion detection). By running the tests, we can ensure that all parts of the system interact correctly.

The test suite is executed with the following command:

```
python -m unittest test_ids.py -v
```

This command runs all test files in the designated directory, ensuring that each function, module, and integration point is tested comprehensively.

11.3 Importance of Unit Testing

Unit tests serve as a safety net during development, allowing us to:

- Detect and fix bugs early in the development cycle.
- Ensure that changes to one part of the system do not break other components.
- Validate that the system correctly identifies and logs different attack types.
- Provide confidence that the system performs well under different traffic loads and conditions.

By covering all the core functions of the IDS, the unit tests help to guarantee system reliability, accuracy, and resilience during real-time network monitoring.

12 Docker Setup

The project is designed to run inside Docker containers to ensure a consistent and isolated environment for the Intrusion Detection System (IDS), the web server, and the traffic simulation. Docker provides an easy way to package, deploy, and manage these services, making it easier to test and run the system.

12.1 Docker Architecture

The project is divided into three main services, each running in its own container:

- **IDS System:** Responsible for capturing network traffic and detecting potential intrusions.
- **Web Server:** Hosts the Flask-based web interface to visualize network traffic and detected attacks.
- **Traffic Simulator:** Replays pre-recorded network traffic using `tcpreplay` to simulate real network conditions.

These services communicate with each other and share the same database using Docker's volume sharing mechanism.

12.2 Docker Compose Configuration

Docker Compose is used to manage the services and simplify the process of building and running the containers. Below is an explanation of the `docker-compose.yaml` file.

Listing 1: docker-compose.yaml

```
version: '3'

services:
  ids-system:
    build:
      context: .
      dockerfile: Dockerfile.ids
    stdin_open: true
    tty: true
    network_mode: "host"
    volumes:
      - shared-db:/app/db

  web-server:
    build:
      context: .
      dockerfile: Dockerfile.web
    ports:
      - "5001:5000"
    volumes:
      - shared-db:/app/db
    command: python3 appy.py

  traffic-simulator:
    build:
      context: .
      dockerfile: Dockerfile.tcpreplay
    depends_on:
      - ids-system
    network_mode: "host"
    volumes:
      - shared-db:/app/db

volumes:
  shared-db:
    driver: local
```

12.2.1 IDS System

The `ids-system` service is built from the `Dockerfile.ids`. It is responsible for processing incoming network traffic and detecting intrusions. The `stdin_open` and `tty` options are enabled to allow interactive terminal access to the container. The system operates on the host network and uses a shared database volume.

12.2.2 Web Server

The `web-server` service is built from the `Dockerfile.web` and serves the Flask-based dashboard. The service exposes port 5001 (which maps to port 5000 inside the container) and uses the shared database volume to access traffic data and detected attacks. The command `python3 appy.py` is executed to start the Flask web application.

12.2.3 Traffic Simulator

The `traffic-simulator` service uses `Dockerfile.tcpreplay` to replay pre-recorded network traffic files. It depends on the IDS system to ensure that the IDS container is running before it begins traffic simulation. The simulator also uses the host network and shares the database volume.

12.3 Dockerfiles

The Dockerfiles for each service define how the containers are built. Here's a breakdown of each Dockerfile.

12.3.1 Dockerfile for IDS System

Listing 2: Dockerfile.ids

```
FROM python:3.9-slim

# Install necessary dependencies
RUN apt-get update && \
    apt-get install -y tcpdump tcpreplay sqlite3 && \
    apt-get clean

# Set the working directory
WORKDIR /app

# Install Python dependencies
RUN pip install --no-cache-dir scapy chardet

# Copy the source code
COPY src/ /app/

# Clear the database when the container starts
RUN rm -f /app/db/traffic.db && mkdir -p /app/db && chmod -R 777 /app/db

# Start the IDS system
CMD ["python3", "IDS.py", "--interface", "eth0"]
```

This Dockerfile builds the IDS system container. It installs necessary tools such as `tcpdump`, `tcpreplay`, and `sqlite3`. The Python dependencies, including `scapy` and `chardet`, are also installed. It ensures that the database is cleared each time the container starts and runs the IDS on the `eth0` interface.

12.3.2 Dockerfile for Web Server

Listing 3: Dockerfile.web

```
FROM python:3.9-slim

WORKDIR /app

# Install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt && mkdir -p /app/db && c

# Copy the application files
COPY appy.py /app/
COPY templates/ /app/templates/

# Expose the port for the Flask web app
EXPOSE 5000

# Run the Flask app
CMD ["python3", "appy.py"]
```

The web server container is built using `python:3.9-slim`. It installs the required Python dependencies listed in `requirements.txt`, copies the web application files, and exposes port 5000 to serve the dashboard.

12.3.3 Dockerfile for Traffic Simulator

Listing 4: Dockerfile.tcpreplay

```
FROM debian:bullseye-slim

# Install tcpreplay
RUN apt-get update && \
    apt-get install -y tcpreplay && \
    apt-get clean

# Set the working directory
WORKDIR /app

# Copy the PCAP files
COPY pcap/ /app/pcap/

# Copy the replay script
COPY replay_pcap.sh /app/

# Make the script executable
RUN chmod +x /app/replay_pcap.sh

# Run the traffic replay script
```

```
CMD ["/app/replay_pcap.sh"]
```

This Dockerfile builds the traffic simulation container. It installs `tcpreplay`, sets up the working directory, and copies the PCAP files and replay script into the container. The replay script is made executable and run when the container starts.

12.3.4 Traffic Replay Script

Listing 5: replay_pcap.sh

```
#!/bin/bash

PCAP_DIR="/app/pcap"

# Loop through all PCAP files in the directory
for pcap_file in $PCAP_DIR/*.pcap; do
  echo "Replaying $pcap_file"

  # Replay the traffic from the PCAP file
  tcpreplay --intf1=eth0 $pcap_file

  # Add a sleep interval between replays
  sleep 2
done

# Keep the container running
exec "$@"
```

This Bash script loops through all the PCAP files in the `pcap/` directory and replays the traffic through the `eth0` interface using `tcpreplay`. After replaying each file, the script pauses for 2 seconds before moving to the next file. It keeps the container running after all files are replayed.

13 Traffic Replay and Simulation

The system uses `tcpreplay` to simulate real network traffic. This allows us to test the IDS under different conditions by replaying pre-recorded PCAP files. The `tcpreplay` utility sends network packets at various speeds, simulating live network traffic for the IDS to analyze.

13.1 Running `tcpreplay`

The traffic simulator container replays pre-captured PCAP files, which simulate real network traffic for the IDS to monitor and analyze. The PCAP files contain a mix of normal and malicious traffic.

14 Running the Intrusion Detection System

This section provides a comprehensive guide on setting up and running the Intrusion Detection System (IDS). Follow these steps to ensure the IDS operates correctly within your network environment.

14.1 System Requirements

Before initiating the setup, ensure your system meets the following requirements:

- A Unix-like operating system (e.g., Linux, macOS)
- Python 3.8 or higher
- Required Python packages are listed in `requirements.txt`. Note that additional dependencies may be required depending on your specific setup and additional features you may wish to use.
- Network access with permissions to capture traffic

14.2 Installation Steps

1. Ensure you have the source code on your local machine. If it has been transferred from another system, ensure it's placed in an appropriate directory.
2. Navigate to the IDS directory:

   ```
   cd IDS
   ```

3. Install the required dependencies. Be aware that the `requirements.txt` file might not cover all dependencies needed for specific configurations or extensions:

   ```
   pip install -r requirements.txt
   ```

4. If any dependencies are missing, you may need to install them separately based on error messages encountered during testing or execution.

14.3 Configuration

Configure the system parameters according to your network setup. Edit the configuration file located at:

`config/settings.json`

Adjust the network interface, log file settings, alert thresholds, and rule set as needed. You can specify the name of the log file in the configuration settings to save the logs with a custom filename.

14.4 Running the IDS

To start the IDS, run the following command in the terminal:

```
sudo python3 ids.py --interface eth0 --logname custom_log_name.log
```

Replace `eth0` with the appropriate network interface and `custom_log_name.log` with your desired log file name. Ensure you have the necessary permissions to capture network traffic on the interface.

You can also provide a PCAP file for scanning by specifying the file path:

```
sudo python3 ids.py --pcap /path/to/file.pcap
```

14.5 Monitoring and Logs

The IDS will log all detected events to the specified log file. Monitor the log file using:

```
tail -f /path/to/custom_log_name.log
```

Additionally, the IDS interface can be accessed through a web browser for real-time monitoring and analysis at:

```
http://localhost:5000
```

15 Conclusion

This project demonstrates an effective system for detecting network intrusions by analyzing live network traffic. The system uses real-time traffic flows and intrusion detection techniques to identify potential threats and provides a comprehensive dashboard for monitoring network activity.

The use of traffic replay allows the IDS to be tested in a variety of scenarios, ensuring that the system is robust and capable of identifying various types of attacks. Future improvements to the system could include the integration of machine learning techniques for more advanced anomaly detection and enhanced user interface features for deeper traffic analysis.