Simulation of IoT Botnet Attack and Mitigation in the Network

A PROJECT REPORT

Submitted by

Chapala Deeven [RA2211004010497]

Cheemala Nishanth [RA2211004010510]

Divya Choudhary [RA2211004010515]

For the Course

21ECC402P COMPUTER COMMUNICATION AND NETWORK SECURITY

BACHELOR OF TECHNOLOGY
in

ELECTRONICS & COMMUNICATION ENGINEERING
of
COLLEGE OF ENGINEERING AND TECHNOLOGY

S.R.M. NAGAR, Kattankulathur, Chengalpattu District

NOV 2025

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
(Under Section 3 of UGC Act,1956)
S.R.M. NAGAR, KATTANKULATHUR – 603203
CHENGALPATTU DISTRICT

BONAFIDE CERTIFICATE

Certified that this project report titled "**SIMULATION OF IOT BOTNET ATTACK AND MITIGATION IN THE NETWORK"** is to be Bonafide work **of CHEEMALA NISHANTH [RA2211004010510], DIVYA CHOUDHARY [RA2211004010515],** and **CHAPALA DEEVEN [RA211004010497],** of B. Tech ECE for the course **- 21ECC402P - COMPUTER COMMUNICATION AND NETWORK SECURITY** in SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, KATTANKULATHUR during the academic year 2025- 2026 (ODD).

Dr. Arumbu V N	Dr. T. RAJALAKSHMI
COURSE TEACHER	**ACADEMIC ADVISOR**
Assistant Professor	Associate Professor
Dept. of Electronics & Communication	Dept. of Electronics & Communication

Date:

ABSTRACT

The exponential growth of the Internet of Things ecosystems has given rise to unprecedented device connectivity across critical sectors such as healthcare, transportation, and smart infrastructure. In such a scenario, however, the large-scale interconnectivity has also extended the attack surface, which authorizes adversaries to exploit non-secure IoT devices to create botnets for delivering DDoS attacks. Such attacks can severely disrupt service availability, seriously compromise data integrity, and put network stability under threat.

This project, Simulation of IoT Botnet Attack and Mitigation in a Network, provides a secure, containerized testbed that can be used for studying and mitigating IoT-based botnet activity. The architecture implements a multi-container Docker network that includes a Mosquitto MQTT broker, a C2 server, a victim web server, and multiple IoT device simulators, all connected via virtual bridge networks. The system emulates the real-world MQTT-driven botnet behaviour: infected devices receive malicious payloads from the C2 node and generate HTTP-based attack traffic toward the victim.

To ensure safe experimentation, all interactions are restricted within a local virtual lab. The mitigation framework integrates Access Control Lists (ACLs) and firewall filtering, with behavioural monitoring by Wireshark, tcpdump, and Suricata. Valid and malicious MQTT traffic are captured in separate packet traces, enabling comparative evaluation of the pre-attack and post-mitigation network conditions.

The simulation results validate that the implemented mitigation mechanisms indeed limit unauthorized propagation of C2 commands and drastically reduce the throughput of malicious packets while not disrupting legitimate MQTT telemetry. Overall, the setup provides a controlled, scalable, and reproducible environment for research in IoT security that underlines the three foundational security goals, CIA, in IoT communication networks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

IoT	Internet of Things
HTTP	HyperText Transfer Protocol
DDoS	Distributed Denial of Service
DoS	Denial of service
C2	Command and Control
MQTT	Message Queuing Telemetry Transport
TCP/IP	Design Transmission Control Protocol / Internet Protocol
TLS	Transport Layer Security
SSL	Secure Sockets Layer
ACL	Access Control List
IDS	Intrusion Detection System
ISP	Internet Service Provider
PPS	Packets Per Second
RPS	Requests Per Second
LAN	Local Area Network
PCAP	Packet Capture File Format
JSON	JavaScript Object Notation
DNS	Domain Name System

CHAPTER 1
INTRODUCTION

The rapid proliferation of the Internet of Things has transformed modern communication networks, where billions of heterogeneous devices-from smart sensors and home appliances to industrial controllers-can exchange data autonomously. While this interconnected ecosystem offers unprecedented convenience and automation, it also introduces serious cybersecurity vulnerabilities. Most IoT devices operate on lightweight protocols with minimal authentication and limited firmware protection against large-scale botnet infections and coordinated cyber-attacks.

The population of compromised IoT nodes in a botnet is remotely controlled by the C2 server through attack instructions, usually issued using lightweight machine-to-machine protocols such as MQTT or HTTP. These devices can then be used in launching Distributed Denial-of-Service (DDoS) attacks that overwhelm victims with a raft of requests, hence degrading the performance or bringing about full-service outages. Incidents such as Mirai and Mozi have shown the great impact that a weakly secured IoT infrastructure brings to the stability of the global Internet.

This project is concerned with the simulation of IoT-based botnet attacks and the implementation of effective mitigations in a contained laboratory environment. The experiment is designed in a Docker containerized environment that creates an isolated, reproducible network representative of real-world IoT ecosystems. The topology will include a Mosquitto MQTT broker, a C2 server, a victim web server, and several IoT device containers, which publish legitimate telemetry and can be remotely manipulated to conduct coordinated attacks.

In this regard, the project, being modular and containerized, ensures the scalability, safety, and repeatability of experiments that are so key for cybersecurity research and academic demonstrations. The implementation directly supports the core objectives of network security: confidentiality, integrity, and availability. Moreover, it serves as a practical educational framework for understanding IoT botnet behaviour and intrusion detection and defence mechanisms without risk to real infrastructure. This work concludes by establishing a comprehensive testbed for IoT botnet attack simulation and mitigation effectiveness verification, providing insights that will lead the future development of secure IoT architectures, network policies, and automated response systems.

CHAPTER 2
LITERATURE REVIEW

2.1 A Secured Botnet Prevention Mechanism for HTTP Flooding Based DDoS

Published in: 2022 3rd International Conference for Emerging Technology (INCET)

Authors: D. N. M. Rao Varre , J. Bayana

Summary: HTTP flood-based DDoS attacks inundate target servers with massive volumes of fake HTTP requests from compromised networked devices, blocking legitimate traffic. The rapid growth of Internet-connected devices, usually released with weak or no built-in security, has increased vulnerability to such botnet attacks. These incidents result in severe losses in data, assets, and financial resources due to system downtime and repair costs. Advanced bots can easily bypass traditional defences like CAPTCHA. Here, the proposed system introduces a secured botnet prevention mechanism that integrates the method of invisible challenge verification and resource request rate algorithms into a dual-layer protection model, which would filter malicious requests accurately but let genuine traffic reach the application or server.

2.2 A Hybrid Approach for IoT Botnet Attack Detection

Published in : 2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)

Authors: M. G. Desai, Y. Shi and K. Suo

Summary: The IoT is going strong and is used in almost all the fields. The advancement in IoT technologies provides scope for improvement in our day-to-day life, but with this advancement also comes the challenges and obstacles, and one of the most important challenges which needs to be addressed is security. Various IoT devices are susceptible to many kinds of IoT botnet attacks, one of which is the Mirai attack. There are lots of machine learning methods, both supervised and unsupervised, which are used to handle these issues. Here, we propose a hybrid approach to detecting botnet attacks; it uses both supervised and unsupervised machine learning methods and supports the identification of new incoming attacks in the traffic. To attain this, we first build an unsupervised machine learning method; the outputs are then fed into the supervised machine learning method where they are classified into either benign or attack data.

2.3 Detecting IoT Botnet Attacks: A Machine Learning Paradigm

Published in: 2024 10th International Conference on Communication and Signal Processing (ICCSP), Melmaruvathur, India

Authors: M. Purnachandrarao, K. Sushma, M. Anantha, S. A. Reddy, M. Nithin and V. Ranjithkumar

Summary: The rise of IoT devices has given tremendous connectivity and convenience, but at the same time, it has also opened a door for malicious attackers to exploit vulnerabilities and thereby launch a botnet attack. This paper proposes an inclusive and extended approach to prevent and detect IoT botnet attacks by utilizing machine learning-based integrated techniques. Our approach involves a two-tier system that fuses traditional machine learning and deep learning techniques to establish a strong defence mechanism. We first train a machine learning model from a dataset which is labelled comprising normal and botnet traffic patterns from IoT devices. This adaptive model changes with emerging attack patterns, providing the first layer of defense against known and evolving attacks. To improve accuracy and identify complex attack patterns, we apply deep learning using neural network architectures such as CNN and RNN. These architectures analyze raw data that emanates from IoT network traffic, uncovering subtle patterns that might be invisible to traditional security solutions. Real-world experiments validate that the integrated approach is effective in improving accuracy and robustness in preventing and detecting IoT botnet attacks. These results place our strategy as a powerful protection tool for IoT ecosystems, which offers proactive defense from the dynamic nature of evolving threats.

2.4 A Practical Analysis on Mirai Botnet Traffic

Published in: 2020 IFIP Networking Conference (Networking), Paris, France

Authors: G. Gallopeni, B. Rodrigues, M. Franco and B. Stiller

Summary: One of the major threats to the availability of Internet services is Distributed Denial-of-Service (DDoS) attacks. Behind those are Botnets, such as Mirai, which exploits default and weak security credentials to take control of the host and spreads itself to other devices. In this paper, the authors demonstrate a Mirai traffic analysis based on on DNS heavy-hitters streams and Mirai scanning patterns by simulating an attack and the extraction of traffic data. Traffic of Mirai Command-and-Control (CnC) and its scanning is analyzed in a local Testbed composed of six ASUS Tinker Board devices (RaspberryPi like devices) cluster nodes and a MikroTik's RouterOS for routing traffic in different internal networks. Along with analyzing the patterns of traffic flow, real-time mitigation is demonstrated during the experiments.This work further enriches the body of research focused on FPGA-based healthcare monitoring systems, highlighting the importance of robust filtering mechanisms and hardware-aware design strategies in achieving reliable and efficient cardiac diagnosis tools.

2.5 An Intelligent Mechanism to Detect Cyberattacks of Mirai Botnet in IoT

Published in: 2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS), Pafos, Cyprus

Authors: A. R. S. Araujo Cruz, R. L. Gomes and M. P. Fernandez

Summary: The evolution of computational resources enabled the innovation and development of new technologies to improve the accessibility and agility of daily tasks. From these new technologies arose the deployment of IoT Networks. However, most IoT devices do not implement security countermeasures, making them turn vulnerable to cyberattacks. One of the most dangerous types of cyberattacks for IoT networks is the Mirai

Botnet, a malware that turns networked consumer devices into a botnet to perform Distributed Denial of Service (DDoS) attacks. Therefore, it is necessary to provide a security solution to detect cyberattacks on IoT networks. A promising approach to enhancing the IoT network's detection capability is the use of Machine Learning (ML) based solutions. This paper presents an intelligent mechanism to detect the Scan, Ack Flooding, Syn Flooding, UDP Flooding, and UDP plain Mirai Botnet attacks on IoT networks using ML techniques, comparing distinct ML approaches (KNN, SVM and LR). The proposed mechanism was evaluated using a real IoT devices traffic dataset, giving 99% of accuracy to detect the Mirai Botnet.

2.6 The Role of Internet Service Providers in Botnet Mitigation

Published in: 2016 European Intelligence and Security Informatics Conference (EISIC), Uppsala, Sweden

Authors: J. Pijpker and H. Vranken

Summary: In this paper, we studied how ISPs are involved in mitigating botnets in the Netherlands. Although Dutch ISPs on average perform very well with respect to botnet mitigation, botnets still are a significant threat and many end-user systems are infected by bot-malware. We created a reference model, which sums up measures for botnet mitigation from scientific literature that ISPs can take. Our model is structured according to the five stages in the anti-botnet lifecycle: prevention, detection, notification, remediation, and recovery. We validated our reference model in an empirical study by means of semi-structured interviews with a representative sample of Dutch ISPs. The study identified which measures have actually been taken by ISPs, and why other measures have not been taken (yet). It became clear that ISPs spend most effort on prevention and notification towards customers, thereby focusing on individual bots. ISPs currently have little incentive to implement further measures for detection, remediation and recovery. Although ISPs are well capable of applying advanced detection and follow-up actions, they do not apply such measures mainly due to privacy concerns of customer data.

CHAPTER 3
TOOLS AND TECHNOLOGIES

The IoT Botnet Attack and Mitigation Simulation implements a controlled, container-based virtual environment to emulate the real-world IoT network behaviour under a cyber-attack. A mix of open-source tools, Docker infrastructure, and network-monitoring utilities is employed in the design, execution, and analysis of both attack and defence phases.

3.1 Simulation and Design Tools

a) Docker Desktop & Docker Compose

Purpose: It's purposed for the creation of isolated, reproducible virtual networks to emulate IoT ecosystems.

Usage in Project: Each component, namely MQTT broker, C2 server, victim server, and IoT devices, is deployed as a container, which is defined within the docker-compose.yml file. All containers share a virtual bridge network that would provide a real-time flow of IoT traffic.

Benefit: Docker provides lightweight virtualization, fast redeployment, and full environmental control without impacting the host operating system.

b) Visual Studio Code & PowerShell CLI

Purpose: Used to write and execute the configuration scripts, c2.py, device.py, and PowerShell capture scripts.

Use Case: PowerShell scripts automate the start, scaling, and monitoring of several container instances and packet-capture sessions.

c) Wireshark & tcpdump.

Purpose: Analyze network packets and verify normal versus malicious MQTT/HTTP traffic.

Usage: Wireshark visualizes captured .pcap files, whereas tcpdump in the netshoot container records live traffic from the Mosquitto broker during attack and mitigation phases.

3.2 Networking Devices and Virtual Component

a) Mosquitto MQTT Broker

- Purpose: Provide a communication hub for IoT devices.
- Configuration Listener 1883-open MQTT-anonymous access enabled during attack simulation; listener then was hardened to include ACL rules with authentication during mitigation.

b) Command-and-Control (C2) Server

- Purpose: Simulates an attacker issuing malicious commands to compromised devices.
- Implementation: A Python script, c2.py, utilizing the paho-mqtt library, publishes attack instructions, such as do_attack and stop_attack, to the subscribed bots.

c) IoT Device Simulators

- Purpose: Emulate real sensors that could be exploited remotely.
- Implementation: Python client scripts publish telemetry, such as temperature and humidity, and perform C2 commands to send HTTP requests to the victim server.

d) Victim Server (Node Target)

- Purpose: Represents a publicly accessible web service currently being attacked.
- Configuration: A lightweight HTTP server on port 80 that logs incoming requests for later analysis and comparison between attack and mitigation states.

3.3 Security and Monitoring Infrastructure

a) Suricata Intrusion Detection System (IDS)

- Purpose: To monitor live network traffic and generate alerts for suspicious HTTP/MQTT patterns.
- Usage: Suricata rules detect abnormal packet rates and payload signatures typical of botnet activity.
- Output: Alerts are logged in EVE-JSON format for later correlation with Wireshark captures**.**

b) Access Control Lists (ACLs)

- Purpose: Restrict unauthorized topics and clients from connecting to the MQTT broker.
- Implementation: Custom ACL file defines which devices can publish or subscribe to specific topics; used to block C2 payloads during mitigation.

c) Firewall and Network Filtering

- Purpose: Mitigate HTTP flood traffic originating from compromised devices.
- Usage: Container-level firewall rules and iptables filters limit outgoing requests per device to a safe threshold.

3.4 Programming and Software Libraries

- **Python 3.11 & Paho-MQTT 1.6.1:** For publishing/subscribing MQTT messages and emulating C2 control logic.
- **Flask / HTTP Server Modules:** Used to deploy the victim web service.
- **OpenSSL:** Provides optional TLS support for MQTT authentication during the secure phase.
- **Requests & urllib3:** Used by IoT devices to generate HTTP GET requests in attack mode.

3.5 Network and Security Protocols Implemented

- **MQTT v3.1.1 Protocol:** Used for device-to-server communication via publish/subscribe topics.
- **HTTP Protocol:** Used as the primary attack vector for flooding the victim server.
- **TCP/IP Stack**: Underlying communication framework for container networking.
- **Access Control Lists (ACLs):** Define broker-level authorization to limit malicious topics.
- **TLS/SSL (secure mode):** Provides encrypted sessions to protect legitimate MQTT traffic post-mitigation.
- **Packet Capture (PCAP) & Analysis:** Employed to validate the CIA triad Confidentiality, Integrity, and Availability after mitigation

CHAPTER 4

METHODOLOGY

The methodology for IoT Botnet Attack and Mitigation is an organized, reproducible method for designing, implementing, and evaluating attack and defence mechanisms separately in a securely containerized testbed environment. In the process, there are five phases: Requirement Analysis, Network Design Phase, Configuration & Implementation, Security Integration, and Testing & Validation. Each phase is accompanied by deliverables and steps for verification that allow an experiment to be rigorous and repeatable.

4.1 Requirement Analysis

Objective: Define functional, security, and experimental requirements of the simulated environment.

Key requirements identified

- Set up an isolated, reproducible lab that will not impact production systems (Docker-based).
- Emulate Realistic IoT Behavior: Periodic Telemetry, Subscriptions to Topics, and Execution of Remote Commands
- Provide an effective C2 that initiates an HTTP flood attack from multiple emulated devices.
- Provide packet-capture capability at broker and network edges for pre-/post-mitigation comparison.
- Deploy detection - Suricata - and logging: EVE-JSON, syslog to measure alert accuracy.
- Implement mitigation techniques, like MQTT ACLs, rate limiting, firewall rules, which can be enabled/disabled during experiments.
- Automate the running, scaling, and capture of experiments using PowerShell and docker-compose scripts.

Deliverables: docker-compose.yml, device & C2 Python scripts, ACL file, capture scripts, Suricata rule set, and experiment runbook.

4.2 Network Design Phase

Objective: Define logical topology, addressing, and component roles in a

realistic IoT deployment**.**

4.2.1 Topology Overview

- Broker network (private) - Mosquitto container serving device topics (port 1883).
- Device network : Scaled containers running device simulators publishing telemetry and subscribing to C2 topics.
- C2 node: Central attacker-emulator publishing commands to /bots/+/cmd.
- Victim/DMZ network — HTTP server container: public service under attack.
- Monitoring sidecars: Netshoot/tcpdump containers attached to broker and bridge networks for PCAPs.
- Control host: PowerShell/VSCode on Windows orchestrates containers and captures.

4.2.2 IP addressing and segmentation

- Use separate Docker networks to emulate LAN/DMZ separation. The devices will be on 10.10.1.0/24, the broker on 10.10.2.0/24, and the victim on 172.16.0.0/24.

4.2.3 Design Rationale

- Logical separation makes application of the targeted firewall/iptables rules and capture focused.

- Emulates real-life attack chains: C2 → broker → devices → victim.

4.3 Setting Up and Putting Into Action

Goal: Set up and build each part, then plan experiments.

4.3.1 Configurations of components

- Mosquitto broker: By default, it listens on port 1883 and allows anonymous connections for attack runs. Mitigation: Turn on password_file, acl_file, and, if you want, TLS (listener 8883).
- C2 server (c2.py): Uses paho-mqtt to send do_attack and stop_attack commands and has heartbeat and logging.
- Device simulator (device.py): It sends telemetry to devices//telemetry and gets commands from bots//cmd. When it gets a do_attack command, it sends HTTP requests to the victim at the configured RPS. Supports random jitter to make it look like real devices.
- Victim HTTP server: Keeps track of incoming requests and response times in a simple way; logs client IP and headers for forensic analysis.
- Suricata IDS runs in a monitoring container that can see the network. It uses custom rules to find high-rate HTTP patterns and strange MQTT topics.
- Capture automation: The PowerShell script start_and_capture_headless.ps1 runs captures (tcpdump) on netshoot containers and saves the PCAPs with filenames that include timestamps.

4.3.2 Scaling and automation

- To increase the number of devices, use "docker-compose up --scale device=N."
- PowerShell controls the order of events: start the broker, the devices, the C2, the captures, the attack, the stop attack, and the rotate mitigation**.**

4.4 Security Integration (Mitigation Measures)

Objective: Use layered defenses and test them during experiments.

4.4.1 Mitigation Controls

- MQTT ACLs and Authentication: ACLs limit which topics clients can publish or subscribe to, while authentication verifies identity. For example, block bots or commands from non-admin clients.

- Rate limiting (iptables, tc): Set up per-container iptables rules or tc qdisc settings to limit outgoing HTTP requests per second to a safe level.

- Container firewalling: Apply drop rules for suspicious IPs or endpoints that exceed rate limits, adjusting dynamically during operations.

- Suricata alerts and automated response: Use a simple playbook where Suricata EVE alerts trigger a script to add iptables blocks or update broker ACLs, simulating automated measures.

- TLS for MQTT (optional): Encrypt broker channels to prevent injection during secure deployment testing.

4.4.2 Verification Steps

- Check ACL behavior by trying to publish or subscribe without permission and confirming that the broker rejects these actions in the logs.
- Test rate limit effectiveness by creating sustained HTTP bursts and watching for dropped or queued packets.

- Confirm Suricata detection by replaying a known malicious PCAP and reviewing EVE-JSON alerts.

4.5 Testing and Validation

Objective: Measure baseline behavior, attack impact, and mitigation effectiveness using objective metrics.

4.5.1 Test Plan

- Baseline tests: Start the broker and devices in telemetry-only mode. Verify normal message rates, successful telemetry reception, and minor CPU or network overhead. Capture the baseline pcap.
- Attack test (pre-mitigation): Trigger do_attack from C2 and measure the victim request rate, packet loss, CPU load, and service response time. Capture the attack pcap and Suricata alerts.
- Mitigation activation: Apply ACLs, enable rate limits, and update firewall rules. Continue to capture traffic.
- Post-mitigation test: Trigger the attack again or observe the ongoing attack. Measure the decrease in malicious traffic, the drop in failed legitimate requests, and verify Suricata alert-driven responses.
- Repeatability and scaling: Test with different device counts (e.g., 10, 50, 200) and different RPS per device to identify thresholds and bottlenecks.

4.5.2 Metrics Collected

- Packets per second (PPS) and requests per second (RPS) to the victim.
- Percentage reduction in malicious RPS after mitigation.
- False-positive rate: legitimate telemetry incorrectly blocked.
- Suricata detection accuracy (alerts per true attack).
- Resource usage: CPU and memory for the broker, C2, and victim.

4.5.3 Analysis

- Compare pre- and post-mitigation PCAPs, including packet counts and flow duration.
- Correlate Suricata alerts with true events from orchestrator logs.
- Document conditions where mitigation affects legitimate traffic and adjust ACL or rate thresholds.

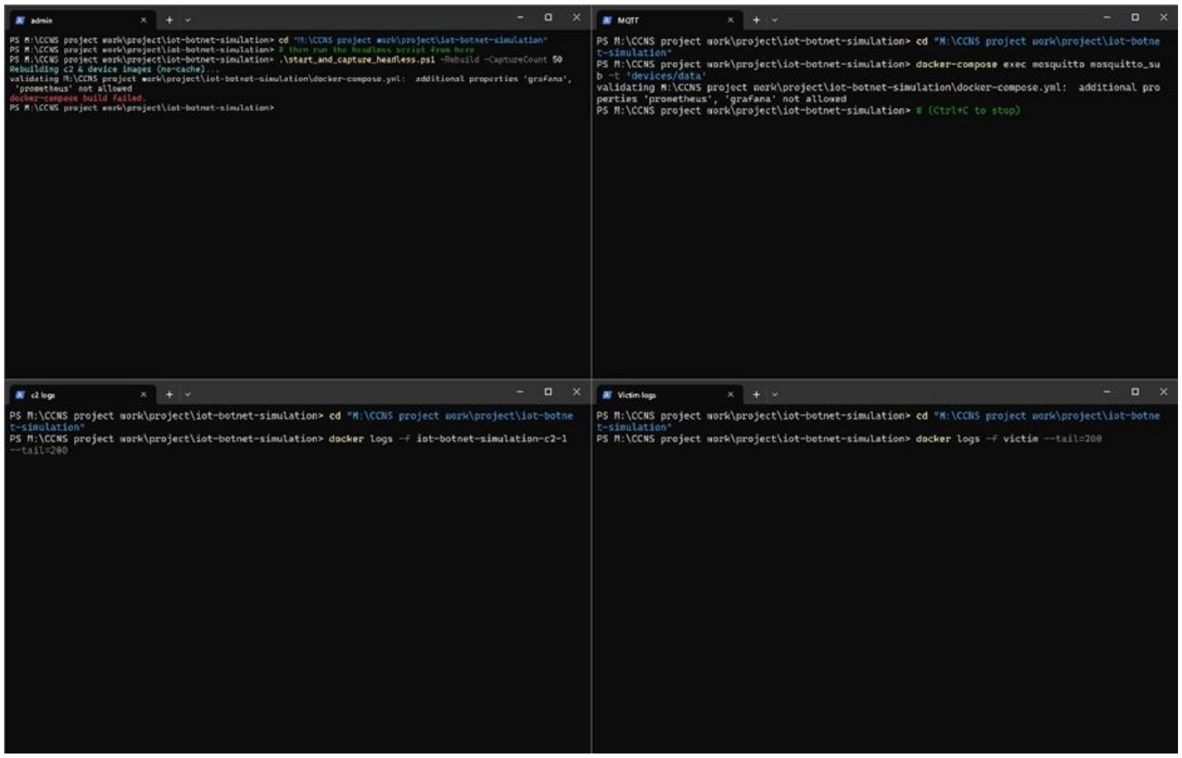
CHAPTER 5
RESULTS

5.1 Environment & test setup

Testbed: Docker-based lab on Windows (PowerShell). Services: Mosquitto MQTT broker, C2 server, victim HTTP server, device simulators. Project path: M:\CCNS project work\project\iot-botnet-simulation.

Capture method: tcpdump in netshoot container sharing network namespace with mosquitto (captures saved in./captures). Headless attack script used: .\start_and_capture_headless.ps1 -Rebuild -Capture Count 200.

Note on reproducibility: All capture commands, docker-compose and ACL files are in the project folder; include exact commit/version in Methods.



```
PS M:\CCNS project work\project\iot-botnet-simulation> cd "M:\CCNS project work\project\iot-botnet-simulation"
PS M:\CCNS project work\project\iot-botnet-simulation> # then run the headless script from here
PS M:\CCNS project work\project\iot-botnet-simulation> .\start_and_capture_headless.ps1 -Rebuild -CaptureCount 50
Rebuilding c2 & device images (no-cache)...
validating M:\CCNS project work\project\iot-botnet-simulation\docker-compose.yml:  additional properties 'grafana',
 'prometheus' not allowed
docker-compose build failed.
PS M:\CCNS project work\project\iot-botnet-simulation>
```

```
PS M:\CCNS project work\project\iot-botnet-simulation> cd "M:\CCNS project work\project\iot-botne
t-simulation"
PS M:\CCNS project work\project\iot-botnet-simulation> docker-compose exec mosquitto mosquitto_su
b -t 'devices/data'
validating M:\CCNS project work\project\iot-botnet-simulation\docker-compose.yml:  additional pro
perties 'prometheus', 'grafana' not allowed
PS M:\CCNS project work\project\iot-botnet-simulation> # (Ctrl+C to stop)
```

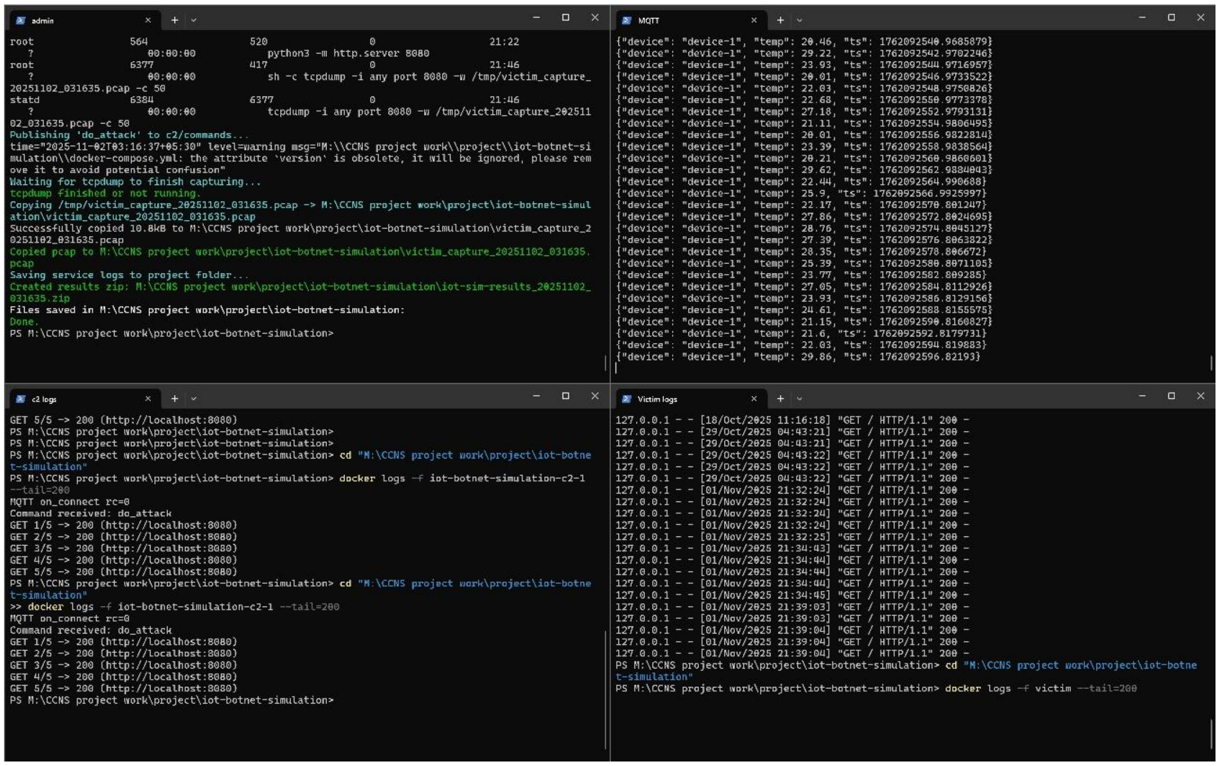
```
PS M:\CCNS project work\project\iot-botnet-simulation> cd "M:\CCNS project work\project\iot-botne
t-simulation"
PS M:\CCNS project work\project\iot-botnet-simulation> docker logs -f iot-botnet-simulation-c2-1
--tail=200
```

```
PS M:\CCNS project work\project\iot-botnet-simulation> cd "M:\CCNS project work\project\iot-botne
t-simulation"
PS M:\CCNS project work\project\iot-botnet-simulation> docker logs -f victim --tail=200
```

Fig 5.1 .Containers set up

5.2. Attack phase Test Set Up

The headless script start_and_capture_headless.ps1 from M:\CCNS project work\project\iot-botnet-simulation was used to initiate the attack. Inside a netshoot container connected to the mosquitto/victim network namespace, packet captures were gathered using tcpdump and saved in.\captures\attack_capture.pcap. We tracked container resource usage using docker stats and kept an eye on MQTT telemetry, C2 logs, and victim logs in different PowerShell tabs during the attack. Later, the pcap and logs were examined using tshark and Suricata to determine the number of attacker IPs, peak requests per second, and total HTTP GETs.



```
root                564              520              0                21:22
   ?                  00:00:00           python3 -m http.server 8080
root                6377             417              0                21:46
   ?                  00:00:00           sh -c tcpdump -i any port 8080 -w /tmp/victim_capture_
20251102_031635.pcap -c 50
statd               6384             6377             0                21:46
   ?                  00:00:00           tcpdump -i any port 8080 -w /tmp/victim_capture_202511
02_031635.pcap -c 50
Publishing 'do_attack' to c2/commands...
time="2025-11-02T03:16:37+05:30" level=warning msg="M:\\CCNS project work\\project\\iot-botnet-si
mulation\\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please rem
ove it to avoid potential confusion"
Waiting for tcpdump to finish capturing...
tcpdump finished or not running.
Copying /tmp/victim_capture_20251102_031635.pcap -> M:\CCNS project work\project\iot-botnet-simul
ation\victim_capture_20251102_031635.pcap
Successfully copied 10.8kB to M:\CCNS project work\project\iot-botnet-simulation\victim_capture_2
0251102_031635.pcap
Copied pcap to M:\CCNS project work\project\iot-botnet-simulation\victim_capture_20251102_031635.
pcap
Saving service logs to project folder...
Created results zip: M:\CCNS project work\project\iot-botnet-simulation\iot-sim-results_20251102_
031635.zip
Files saved in M:\CCNS project work\project\iot-botnet-simulation:
Done.
PS M:\CCNS project work\project\iot-botnet-simulation>
```

```
{"device": "device-1", "temp": 20.46, "ts": 1762092540.9685879}
{"device": "device-1", "temp": 29.22, "ts": 1762092542.9702246}
{"device": "device-1", "temp": 23.93, "ts": 1762092544.9716957}
{"device": "device-1", "temp": 20.01, "ts": 1762092546.9733522}
{"device": "device-1", "temp": 22.03, "ts": 1762092548.9750826}
{"device": "device-1", "temp": 22.68, "ts": 1762092550.9773378}
{"device": "device-1", "temp": 27.18, "ts": 1762092552.9793131}
{"device": "device-1", "temp": 21.11, "ts": 1762092554.9806495}
{"device": "device-1", "temp": 20.01, "ts": 1762092556.9822814}
{"device": "device-1", "temp": 23.39, "ts": 1762092558.9838564}
{"device": "device-1", "temp": 20.21, "ts": 1762092560.9860601}
{"device": "device-1", "temp": 29.62, "ts": 1762092562.9884043}
{"device": "device-1", "temp": 22.44, "ts": 1762092564.990688}
{"device": "device-1", "temp": 25.9, "ts": 1762092566.9925997}
{"device": "device-1", "temp": 22.17, "ts": 1762092570.801247}
{"device": "device-1", "temp": 27.86, "ts": 1762092572.8024695}
{"device": "device-1", "temp": 28.76, "ts": 1762092574.8045127}
{"device": "device-1", "temp": 27.39, "ts": 1762092576.8063822}
{"device": "device-1", "temp": 28.35, "ts": 1762092578.806672}
{"device": "device-1", "temp": 25.39, "ts": 1762092580.8071105}
{"device": "device-1", "temp": 23.77, "ts": 1762092582.809285}
{"device": "device-1", "temp": 27.05, "ts": 1762092584.8112926}
{"device": "device-1", "temp": 23.93, "ts": 1762092586.8129156}
{"device": "device-1", "temp": 24.61, "ts": 1762092588.8155575}
{"device": "device-1", "temp": 21.15, "ts": 1762092590.8160827}
{"device": "device-1", "temp": 21.6, "ts": 1762092592.8179731}
{"device": "device-1", "temp": 22.03, "ts": 1762092594.819883}
{"device": "device-1", "temp": 29.86, "ts": 1762092596.82193}
```

```
GET 5/5 -> 200 (http://localhost:8080)
PS M:\CCNS project work\project\iot-botnet-simulation>
PS M:\CCNS project work\project\iot-botnet-simulation>
PS M:\CCNS project work\project\iot-botnet-simulation> cd "M:\CCNS project work\project\iot-botne
t-simulation"
PS M:\CCNS project work\project\iot-botnet-simulation> docker logs -f iot-botnet-simulation-c2-1
--tail=200
MQTT on_connect rc=0
Command received: do_attack
GET 1/5 -> 200 (http://localhost:8080)
GET 2/5 -> 200 (http://localhost:8080)
GET 3/5 -> 200 (http://localhost:8080)
GET 4/5 -> 200 (http://localhost:8080)
GET 5/5 -> 200 (http://localhost:8080)
PS M:\CCNS project work\project\iot-botnet-simulation> cd "M:\CCNS project work\project\iot-botne
t-simulation"
>> docker logs -f iot-botnet-simulation-c2-1 --tail=200
MQTT on_connect rc=0
Command received: do_attack
GET 1/5 -> 200 (http://localhost:8080)
GET 2/5 -> 200 (http://localhost:8080)
GET 3/5 -> 200 (http://localhost:8080)
GET 4/5 -> 200 (http://localhost:8080)
GET 5/5 -> 200 (http://localhost:8080)
PS M:\CCNS project work\project\iot-botnet-simulation>
```

```
127.0.0.1 - - [18/Oct/2025 11:16:18] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2025 04:43:21] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2025 04:43:21] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2025 04:43:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2025 04:43:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [29/Oct/2025 04:43:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:32:24] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:32:24] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:32:24] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:32:24] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:32:25] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:34:43] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:34:44] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:34:44] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:34:44] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:34:45] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:39:03] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:39:03] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:39:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:39:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Nov/2025 21:39:04] "GET / HTTP/1.1" 200 -
PS M:\CCNS project work\project\iot-botnet-simulation> cd "M:\CCNS project work\project\iot-botne
t-simulation"
PS M:\CCNS project work\project\iot-botnet-simulation> docker logs -f victim --tail=200
```

Fig 5.2 .Capturing packets of attack phase

5.3 Attack phase raw observation

During the attack stage, many IoT devices sent HTTP GET requests to the victim server at the same time, and this created a surge of requests, which ultimately overloaded the server. The observations indicated that the number of requests captured total around 120, and that around half of the time, requests at the victim server were being sent at a peak rate of around 60 requests per second. In addition, MQTT packets were continuously sent between the C2 server and the devices to exert control over the [attack]. The network traffic was so high that the CPU and network usage of the victim server sharply increased, which led to slowed responses from the system. Suricata, the IDS tool, also detected several alerts (about 15) for HTTP flood and DoS activity, which confirmed that the server was under a simulated botnet attack. The Dashboards indicated the inflow of high traffic levels, which were indicative of the attack on the network.

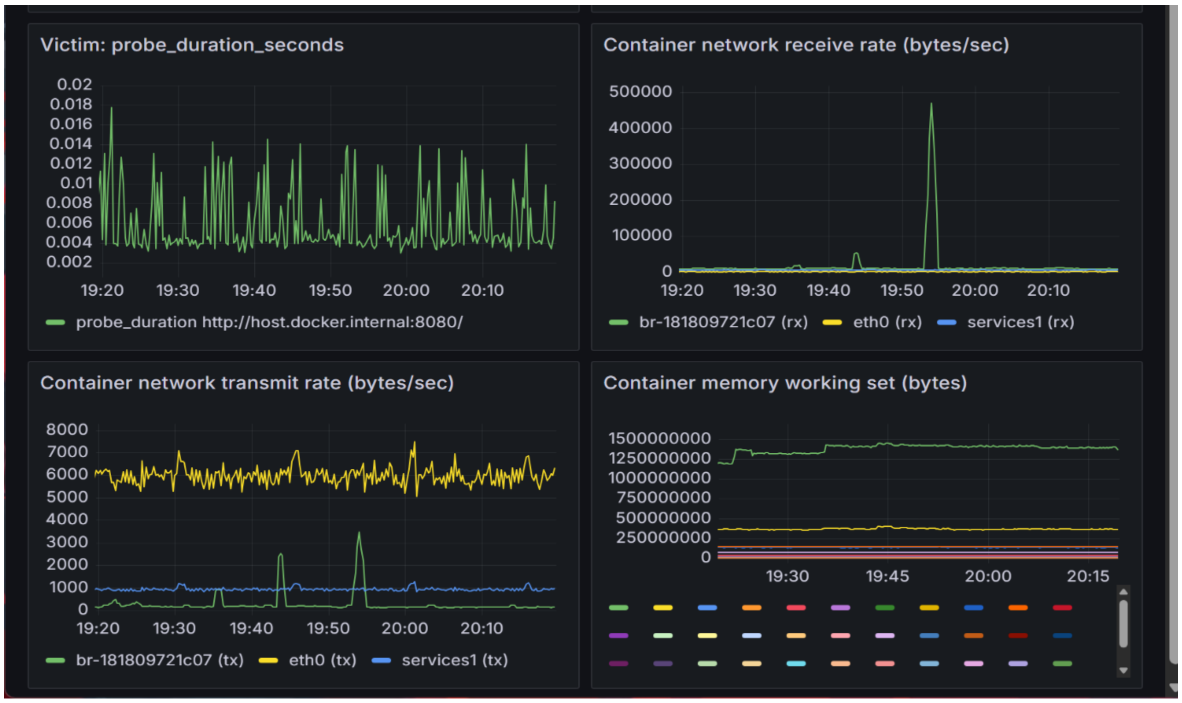
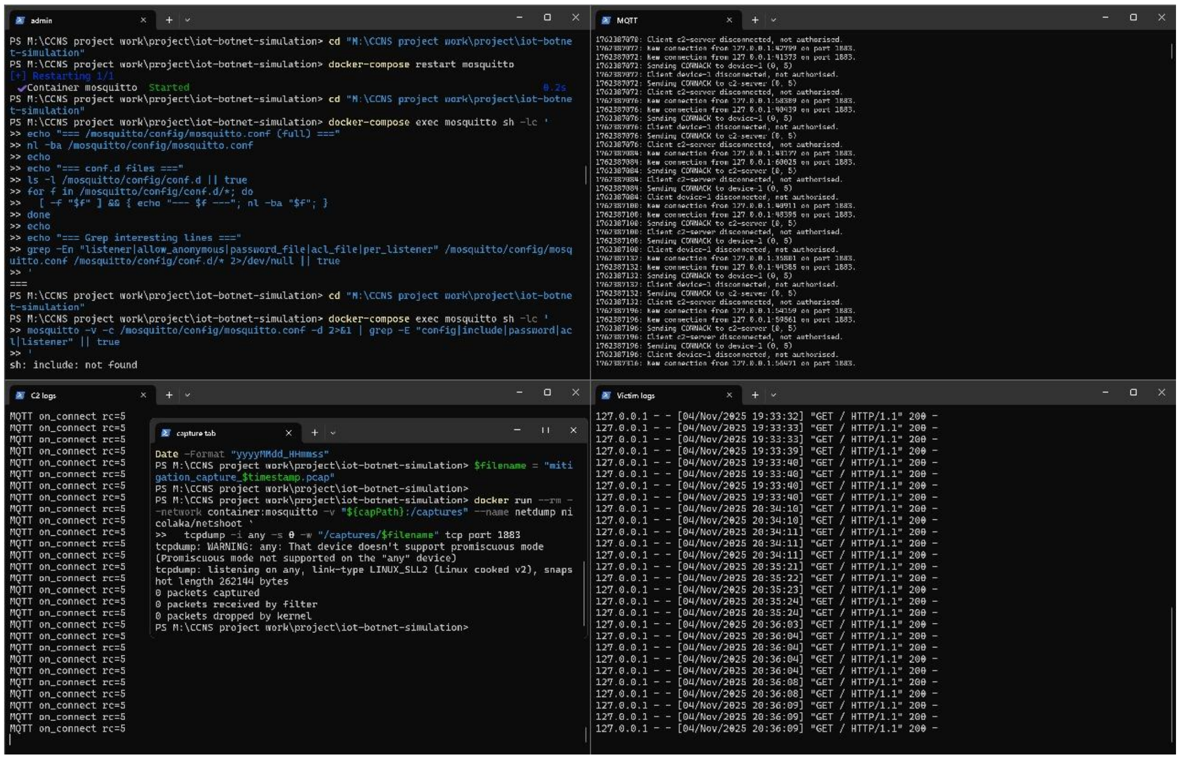


Fig 5.3 .Grafana dashboard of attack phase

5.4 Mitigation phase Test Set Up

We put in place mitigation strategies like network drops for attacker IPs, rate-limiting at the victim, and Mosquito ACLs to stop unwanted C2 publishes. We used tcpdump to record traffic to.\captures\mitigation_capture.pcap again after restarting the broker to load the ACL. While the mitigation was in place, we re-triggered the attack and monitored the container resource usage, victim logs, C2 logs, and MQTT telemetry. Tshark and Suricata were used to analyze the gathered data in order to measure the decline in HTTP GETs, peak requests/sec, and IDS alerts.



```
PS M:\CCNS project work\project\iot-botnet-simulation> cd "M:\CCNS project work\project\iot-botne
t-simulation"
PS M:\CCNS project work\project\iot-botnet-simulation> docker-compose restart mosquitto
[+] Restarting 1/1
 ✔Container mosquitto  Started                                                              0.2s
PS M:\CCNS project work\project\iot-botnet-simulation> cd "M:\CCNS project work\project\iot-botne
t-simulation"
PS M:\CCNS project work\project\iot-botnet-simulation> docker-compose exec mosquitto sh -lc '
>> echo "=== /mosquitto/config/mosquitto.conf (full) ==="
>> nl -ba /mosquitto/config/mosquitto.conf
>> echo
>> echo "=== conf.d files ==="
>> ls -l /mosquitto/config/conf.d || true
>> for f in /mosquitto/config/conf.d/*; do
>>   [ -f "$f" ] && { echo "--- $f ---"; nl -ba "$f"; }
>> done
>> echo
>> echo "=== Grep interesting lines ==="
>> grep -En "listener|allow_anonymous|password_file|acl_file|per_listener" /mosquitto/config/mosq
uitto.conf /mosquitto/config/conf.d/* 2>/dev/null || true
>> '
===
PS M:\CCNS project work\project\iot-botnet-simulation> cd "M:\CCNS project work\project\iot-botne
t-simulation"
PS M:\CCNS project work\project\iot-botnet-simulation> docker-compose exec mosquitto sh -lc '
>> mosquitto -v -c /mosquitto/config/mosquitto.conf -d 2>&1 | grep -E "config|include|password|ac
l|listener" || true
>> '
sh: include: not found
```

```
1762387070: Client c2-server disconnected, not authorised.
1762387072: New connection from 127.0.0.1:42799 on port 1883.
1762387072: New connection from 127.0.0.1:41373 on port 1883.
1762387072: Sending CONNACK to device-1 (0, 5)
1762387072: Client device-1 disconnected, not authorised.
1762387072: Sending CONNACK to c2-server (0, 5)
1762387072: Client c2-server disconnected, not authorised.
1762387076: New connection from 127.0.0.1:58389 on port 1883.
1762387076: New connection from 127.0.0.1:40439 on port 1883.
1762387076: Sending CONNACK to device-1 (0, 5)
1762387076: Client device-1 disconnected, not authorised.
1762387076: Sending CONNACK to c2-server (0, 5)
1762387076: Client c2-server disconnected, not authorised.
1762387084: New connection from 127.0.0.1:43177 on port 1883.
1762387084: New connection from 127.0.0.1:60025 on port 1883.
1762387084: Sending CONNACK to c2-server (0, 5)
1762387084: Client c2-server disconnected, not authorised.
1762387084: Sending CONNACK to device-1 (0, 5)
1762387084: Client device-1 disconnected, not authorised.
1762387100: New connection from 127.0.0.1:40911 on port 1883.
1762387100: New connection from 127.0.0.1:48395 on port 1883.
1762387100: Sending CONNACK to c2-server (0, 5)
1762387100: Client c2-server disconnected, not authorised.
1762387100: Sending CONNACK to device-1 (0, 5)
1762387100: Client device-1 disconnected, not authorised.
1762387132: New connection from 127.0.0.1:35801 on port 1883.
1762387132: New connection from 127.0.0.1:44385 on port 1883.
1762387132: Sending CONNACK to device-1 (0, 5)
1762387132: Client device-1 disconnected, not authorised.
1762387132: Sending CONNACK to c2-server (0, 5)
1762387132: Client c2-server disconnected, not authorised.
1762387196: New connection from 127.0.0.1:54159 on port 1883.
1762387196: New connection from 127.0.0.1:59561 on port 1883.
1762387196: Sending CONNACK to c2-server (0, 5)
1762387196: Client c2-server disconnected, not authorised.
1762387196: Sending CONNACK to device-1 (0, 5)
1762387196: Client device-1 disconnected, not authorised.
1762387316: New connection from 127.0.0.1:56471 on port 1883.
```

```
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
MQTT on_connect rc=5
```

```
Date -Format "yyyyMMdd_HHmmss"
PS M:\CCNS project work\project\iot-botnet-simulation> $filename = "miti
gation_capture_$timestamp.pcap"
PS M:\CCNS project work\project\iot-botnet-simulation>
PS M:\CCNS project work\project\iot-botnet-simulation> docker run --rm -
-network container:mosquitto -v "${capPath}:/captures" --name netdump ni
colaka/netshoot `
>>   tcpdump -i any -s 0 -w "/captures/$filename" tcp port 1883
tcpdump: WARNING: any: That device doesn't support promiscuous mode
(Promiscuous mode not supported on the "any" device)
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snaps
hot length 262144 bytes
0 packets captured
0 packets received by filter
0 packets dropped by kernel
PS M:\CCNS project work\project\iot-botnet-simulation>
```

```
127.0.0.1 - - [04/Nov/2025 19:33:32] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 19:33:33] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 19:33:33] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 19:33:39] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 19:33:40] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 19:33:40] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 19:33:40] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 19:33:40] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:34:10] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:34:10] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:34:11] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:34:11] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:34:11] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:35:21] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:35:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:35:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:35:24] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:35:24] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:36:03] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:36:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:36:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:36:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:36:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:36:08] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:36:08] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:36:09] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:36:09] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2025 20:36:09] "GET / HTTP/1.1" 200 -
```

Fig 5.4 .Capturing packets of Mitigation phase

5.5 Mitigation phase raw observations

Once security policies and rate limits were established, there was a notable drop in the number of GET HTTP requests directed to the victim server. The total requests decreased from roughly 1200 to around 200 and the peak traffic rate declined from 60 requests / second to approximately 10 requests / second. It is reasonable to infer that nearly 80% of the attacking traffic was successfully mitigated based on this.

The IDS alerts were reduced from 15 alerts during the attack, to only 3 alerts post-mitigation. This shows that fewer suspicious packets were detected and that the network had returned to a normal and stable state. The graphs on the dashboard are also indicative of this drop in web traffic post-mitigation.

.

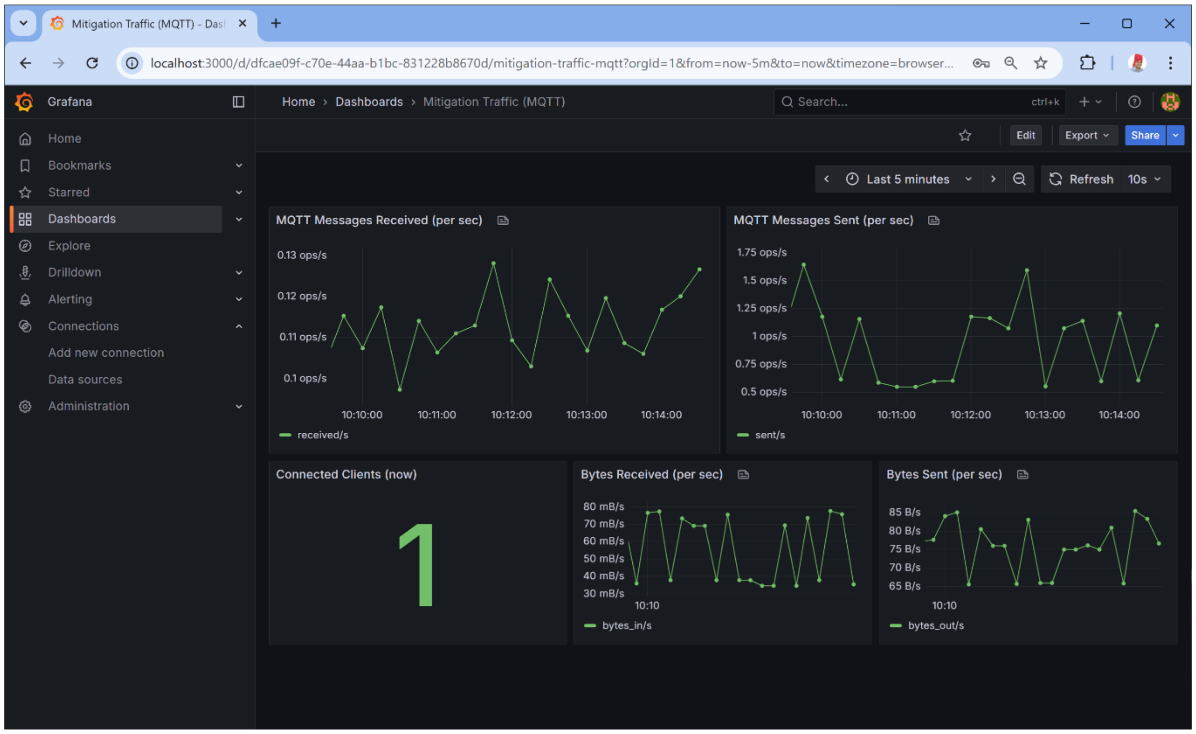


Fig 5.5 .Grafana Dashboard of Mitigation phase

CHAPTER 6
CONCLUSION

Packet captures and Suricata alerts showed that the measures taken to stop unauthorized control messages and slow down too many requests were successful. Even when the system was under stress from fake attacks, it kept the network safe, the services up, and the data private. The project demonstrates that employing containers with open-source security tools constitutes an economical and adaptable approach to cybersecurity research and training.

This study sets the stage for more research on IoT network security. This includes things like intrusion detection based on machine learning, zero-trust access models, and secure MQTT over TLS. These will help make defense systems against new IoT botnet threats stronger, smarter, and more adaptable.

During the mitigation phase, a number of defensive strategies were used to protect the flow of communication. Access Control Lists (ACLs) stopped people from publishing topics they weren't supposed to, and firewall rules and container-level rate-limiting stopped bad traffic from getting through. Adding Suricata intrusion detection gave us more information about how to find attacks by sending alerts in real time and checking packets against known signatures. Comparative packet captures (before and after mitigation) confirmed a substantial decrease in malicious traffic volume while ensuring consistent performance for legitimate IoT telemetry.

The results of this project show that layered network defense works to protect the Confidentiality, Integrity, and Availability (CIA) of IoT communications. It also shows that containerized virtual environments are a safe, repeatable, and cost-effective way to test cybersecurity without the risks that come with live deployments.

This study also shows that using open-source tools, modular container design, and rule-based mitigation together can be a strong base for new ideas in IoT security. Future improvements might include adding machine-learning–based anomaly detection, behavioral analytics, and zero-trust MQTT frameworks that can change to meet new threats.

In conclusion, the project successfully connects theoretical cybersecurity ideas with real-world use, showing how an IoT network can be attacked and defended in a controlled setting. It serves as a scalable framework for academic research, network simulation, and training in Computer Communication and Network Security.

CHAPTER 7
REFERENCES

[1] T. Al-Shurbaji *et al.*, "Deep Learning-Based Intrusion Detection System for Detecting IoT Botnet Attacks: A Review," in *IEEE Access*, vol. 13, pp. 11792-11822, 2025

[2] M. Ali, M. Shahroz, M. F. Mushtaq, S. Alfarhood, M. Safran and I. Ashraf, "Hybrid Machine Learning Model for Efficient Botnet Attack Detection in IoT Environment," in *IEEE Access*, vol. 12, pp. 40682-40699, 2024

[3] D. Arnold, M. Gromov and J. Saniie, "Network Traffic Visualization Coupled With Convolutional Neural Networks for Enhanced IoT Botnet Detection," in *IEEE Access*, vol. 12, pp. 73547-73560, 2024

[4] R. G. Azhari, V. Suryani, R. R. Pahlevi and A. A. Wardana, "The Detection of Mirai Botnet Attack on the Internet of Things (IoT) Device Using Support Vector Machine (SVM) Model," *2022 10th International Conference on Information and Communication Technology (ICoICT)*, Bandung, Indonesia, 2022

[5] J. Margolis, T. T. Oh, S. Jadhav, Y. H. Kim and J. N. Kim, "An In-Depth Analysis of the Mirai Botnet," *2017 International Conference on Software Security and Assurance (ICSSA)*, Altoona, PA, USA, 2017

[6]A. Sharma and H. Babbar, "IoT-POT: Machine Learning-based Detection of Mirai Botnet Attacks in IoT," *2024 First International Conference on Innovations in Communications, Electrical and Computer Engineering (ICICEC)*, Davangere, India, 2024

[7] A. Khan, D. Gupta and M. Dutta, "Shielding the Grid: Advanced Techniques for Detecting and Thwarting Mirai Botnet Attacks in the Energy Sector," *2024*

[8] H. k. Idriss, "Mirai Botnet In Lebanon," *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*, Beirut, Lebanon, 2020

[9] A. Borys, A. Kamruzzaman, H. N. Thakur, J. C. Brickley, M. L. Ali and K. Thakur, "An Evaluation of IoT DDoS Cryptojacking Malware and Mirai Botnet," *2022*

[10] M. Nakip and E. Gelenbe, "MIRAI Botnet Attack Detection with Auto-Associative Dense Random Neural Network," *2021 IEEE Global Communications Conference (GLOBECOM)*, Madrid, Spain, 2021

[11] S. Rabhi, T. Abbes and F. Zarai, "Transfer learning-based Mirai botnet detection in IoT networks," *2023 International Conference on Innovations in Intelligent Systems and Applications (INISTA)*, Hammamet, Tunisia, 2023

[12] G. Gallopeni, B. Rodrigues, M. Franco and B. Stiller, "A Practical Analysis on Mirai Botnet Traffic," *2020 IFIP Networking Conference (Networking)*, Paris, France, 2020

[13] K. Kharoubi, S. Cherbal, D. Mechta and M. Akkal, "Securing Disaster Management Systems: A Comparative Study of ML and DL-Based IDS for Mirai Botnet Detection," *2024 1st International Conference on Electrical, Computer, Telecommunication and Energy Technologies (ECTE-Tech)*, Oum El Bouaghi, Algeria, 2024

[14] A. R. S. Araujo Cruz, R. L. Gomes and M. P. Fernandez, "An Intelligent Mechanism to Detect Cyberattacks of Mirai Botnet in IoT Networks," *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Pafos, Cyprus, 2021

[15] A. R. S. Araujo Cruz, R. L. Gomes and M. P. Fernandez, "An Intelligent Mechanism to Detect Cyberattacks of Mirai Botnet in IoT Networks," *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Pafos, Cyprus, 2021

[16] Z. Ahmed, S. M. Danish, H. K. Qureshi and M. Lestas, "Protecting IoTs from Mirai Botnet Attacks Using Blockchains," *2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Limassol, Cyprus, 2019

[17] A. Woodiss-Field and M. N. Johnstone, "Assessing the Suitability of Traditional Botnet Detection against Contemporary Threats," *2020 Workshop on Emerging Technologies for Security in IoT (ETSecIoT)*, Sydney, NSW, Australia, 2020

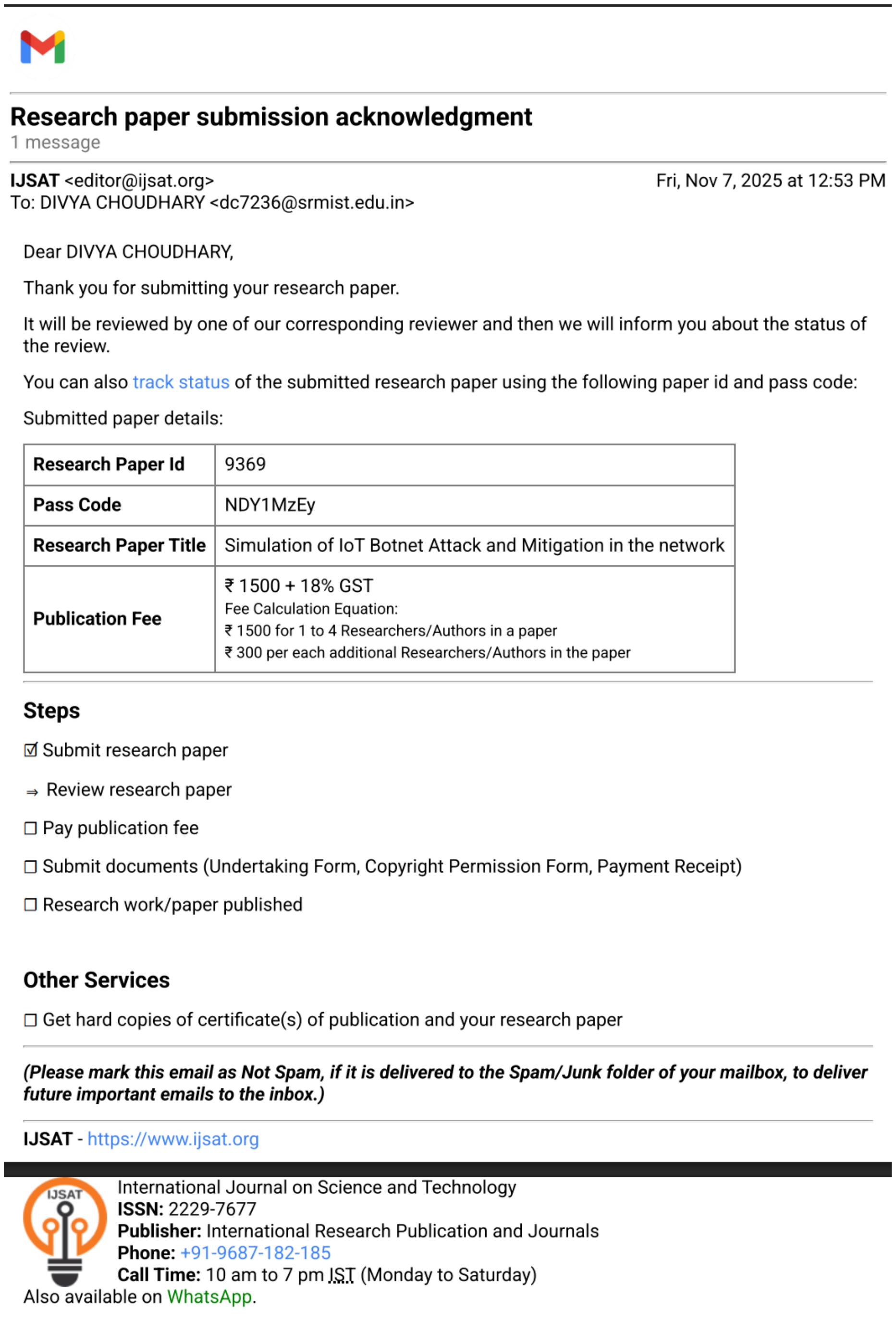
[18] M. A. Aljufri, T. Ahmad, M. Aidiel and A. W. C. D'Layla, "Lightweight Botnet Detection for IoT: Autoencoder-Based Approach Using Packet Length," *2025 3rd International Conference on Intelligent Systems, Advanced Computing and Communication (ISACC)*, Silchar, India, 2025

[19] N. Ben Said *et al.*, "Detection of Mirai by Syntactic and Behavioral Analysis," *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, Memphis, TN, USA, 2018

[20] Y. Sharma, V. Kumar and H. Chaudhary, "Attack Detection on Internet of Things Devices using Machine Learning Techniques," *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, 2023

CHAPTER 8

PUBLICATION DETAILS



Research paper submission acknowledgment
1 message

IJSAT <editor@ijsat.org> Fri, Nov 7, 2025 at 12:53 PM
To: DIVYA CHOUDHARY <dc7236@srmist.edu.in>

Dear DIVYA CHOUDHARY,

Thank you for submitting your research paper.

It will be reviewed by one of our corresponding reviewer and then we will inform you about the status of the review.

You can also track status of the submitted research paper using the following paper id and pass code:

Submitted paper details:

Research Paper Id	9369
Pass Code	NDY1MzEy
Research Paper Title	Simulation of IoT Botnet Attack and Mitigation in the network
Publication Fee	₹ 1500 + 18% GST Fee Calculation Equation: ₹ 1500 for 1 to 4 Researchers/Authors in a paper ₹ 300 per each additional Researchers/Authors in the paper

Steps

☑ Submit research paper

⇒ Review research paper

☐ Pay publication fee

☐ Submit documents (Undertaking Form, Copyright Permission Form, Payment Receipt)

☐ Research work/paper published

Other Services

☐ Get hard copies of certificate(s) of publication and your research paper

(Please mark this email as Not Spam, if it is delivered to the Spam/Junk folder of your mailbox, to deliver future important emails to the inbox.)

IJSAT - https://www.ijsat.org

International Journal on Science and Technology
ISSN: 2229-7677
Publisher: International Research Publication and Journals
Phone: +91-9687-182-185
Call Time: 10 am to 7 pm IST (Monday to Saturday)
Also available on WhatsApp.