UNIVERSITÀ DI PISA

Computer Engineering

Advanced Network Architectures And Wireless Systems

Malicious Flows Quarantine

Arsalen BIDANI

Academic year: 2024/2025

Contents

1. Introduction

This project implements a quarantine mechanism in SDN networks to handle malicious flows. Marked flows are rerouted to a quarantine switch, which forwards packets to the controller for buffering. The controller then either drops or flushes them to their destinations.

This document covers the design and implementation of a Floodlight module with a RESTful API for managing these flows, along with testing in a Mininet emulation.

2. Objectives

The primary objectives of this project are:

- **Mark a Flow as Malicious**: Implement a RESTful API to allow external users to mark a flow as malicious and configure its quarantine size.
- **Unmark a Flow and Flush the Buffer**: Provide an API to unmark a flow and flush the buffered packets back to their destination.
- **Unmark a Flow and Clear the Buffer**: Implement an option to unmark a flow and clear its buffered packets.
- **Retrieve Buffered Packet Information**: Implement an API to query the total number of packets buffered for a given malicious flow.

3. Floodlight Module

3.1. Malicious Flows

The MaliciousFlow class is responsible for managing malicious flows in the system. It tracks flows using a **unique identifier (UUID)**, which is later used to identify and manage marked flows.

To handle packet buffering, the class uses a **queue-based buffer** that stores incoming **PACKET_IN messages**. The buffer has a configurable maximum size, and when it reaches capacity, the oldest packet is removed to accommodate new ones.

Additionally, the class provides several utility functions, including:

- **Adding packets to the buffer** while maintaining size constraints.
- **Flushing the buffer**, returning all stored packets.
- **Clearing the buffer**, and discarding all packets.
- **Retrieving the buffer size** to check the number of stored packets.
- **Adjusting the buffer size dynamically**, removing excess packets if needed.

3.2. Flows Management

This process is handled by the MaliciousFlowsQuarantine and FlowUtils class, which provides utility methods for managing flow entries in the network switches.

3.2.1. Installing Flow Rules

When a flow is marked as malicious, the following steps occur:

1. The system checks if the flow already exists; if so, it updates the buffer size.
2. If the flow is new, it is added to the list of active malicious flows.
3. The system iterates through all switches in the network:
    - If the switch is the quarantine switch, it installs a rule to redirect packets to the controller.

    - If the switch is a regular switch, it determines the next-hop port toward the quarantine switch and installs a forwarding rule.

Installing a flow rule is handled by a method in FlowUtils class:

- It creates a match structure based on source and destination IPs.
- It sets flow timeouts and priority.
- It installs a forwarding action to either the controller or the next-hop switch.
- If the packet is buffered in the switch, it is forwarded immediately.

3.2.2. Handling Packet-In Events

1. If a malicious packet reaches the quarantine switch, it is stored in the quarantine buffer.
2. If the packet arrives at a regular switch and its flow is marked as malicious, the redirection rule is reapplied, this is viable in case of switch disconnection and reconnection.

3.2.3. Deleting Flow Rules

When a malicious flow is **unmarked**, the following occurs:

1. The flow entry is removed from active tracking.
2. The system initiates rule deletion for all switches to remove the corresponding flow rule.
3. Buffered packets are either cleared or forwarded to the quarantine switch based on the specified method (flush or clear).

The deletion process constructs a removal rule using the same match structure as the installation process and applies it to the switch.

3.2.4. Flow Rule Expiration Handling

Switches are configured to send `FLOW_REMOVED` messages when flow rules expire.
Upon receiving such messages, the system checks if the removed flow is still active in the malicious flow list, If active, it re-applies the redirection rules.

3.3. Flushing & Clearing Operations

If a malicious flow is unmarked with a **clear** method simply the buffer will be cleared, instead if method was to **flush** packet, a packet-out message must be constructed for each packet present in the buffer to be reinjected into the data plane:

1. The system extracts the packet's metadata either via bufferID or raw packet if the bufferID is `NO_BUFFER`.
2. A new **packet-out** message is created, encapsulating the packet along with its metadata.
3. The packet is then reinjected into the network via the controller to the quarantine switch, specifying the controller as an ingress port to ensure proper forwarding

4. RESTful API

The RESTful API provides several endpoints to manage and interact with flows in the quarantine process

4.1. API Endpoints

When a flow is marked as malicious using the /quarantine/mark endpoint in the RESTful API, the server generates and returns a unique **Flow UUID**. This **Flow UUID** serves as the identifier for that specific flow throughout its lifecycle within the quarantine process. The UUID can then be used in subsequent API calls, such as checking the number of buffered packets associated with that flow.

Action	**HTTP Method**	**URL**	**Description**
Mark Flow as Malicious	POST	/quarantine/mark	Mark a flow as malicious and set the buffer size.
Unmark & Flush Buffer	DELETE	/quarantine/unmark/flush	Unmark a flow and send the stored packets back to the data plane.
Unmark & Clear Buffer	DELETE	/quarantine/unmark/clear	Unmark a flow and drop the stored packets.

Check Buffer Size	GET	/quarantine/buffer/flow_id	Get the number of buffered packets for a specific flow.

4.2. Request Payloads & Responses

The request payload refers to the data sent by the client to the server in the body of the request. These payloads are typically structured in **JSON** format and contain key-value pairs representing the necessary information to execute the desired operation.

To manage these responses, the utility functions in the `JsonResponseUtil` class are used. These functions handle the serialization of the response objects into JSON format.

1. **MarkFlow (POST /quarantine/mark)** The payload for this function requires:
   - **clientIP**: The IP address of the client initiating the flow.
   - **serverIP**: The IP address of the server the flow is interacting with.
   - **bufferSize**: The size of the buffer to hold the flow's packets if it is marked as malicious.

   Response Format:
   Responses are returned as JSON, and they are structured to indicate success or failure. For success, the response includes a `status` of `success`, a `message`, and a `uuid` (the flow identifier). If there is an error, the response contains `status` as `error` with a message detailing the issue.

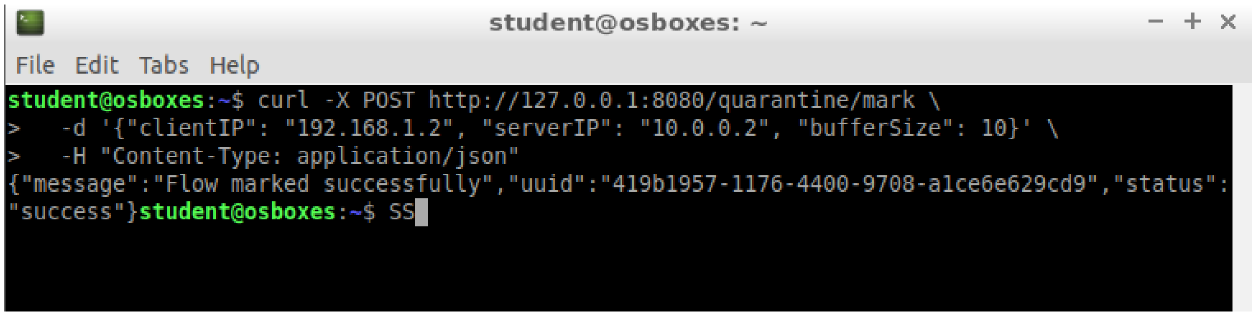
2. **UnmarkFlow (DELETE /quarantine/unmark/{method})** This function requires the following attributes:
   - **clientIP**: The IP address of the client whose flow is to be unmarked.
   - **serverIP**: The IP address of the server involved in the flow.
   - **method**: The action to take—either flush (send the buffered packets back to the data plane) or clear (drop the buffered packets).

Response Format:
Similar to **MarkFlow**, the response is returned in JSON format, with either a success message or an error message

3. **CheckBufferSize (GET /quarantine/buffer/{flow_id})**
   - This function does not require a traditional payload, but it needs the **flowId** in the URL to identify the specific flow and retrieve its buffer size.

   Response Format:
   The response is a JSON object containing the status, a message (e.g., the flow exists), and, in the case of success, the number of bufferedPackets for that flow.

Below is an example of success messages on marking a malicious flow showcasing the POST request on endpoints /quarantine/mark ,its payload and JSON response format:



```
student@osboxes:~$ curl -X POST http://127.0.0.1:8080/quarantine/mark \
>   -d '{"clientIP": "192.168.1.2", "serverIP": "10.0.0.2", "bufferSize": 10}' \
>   -H "Content-Type: application/json"
{"message":"Flow marked successfully","uuid":"419b1957-1176-4400-9708-a1ce6e629cd9","status":
"success"}student@osboxes:~$ SS
```

/quarantine/mark endpoints success request

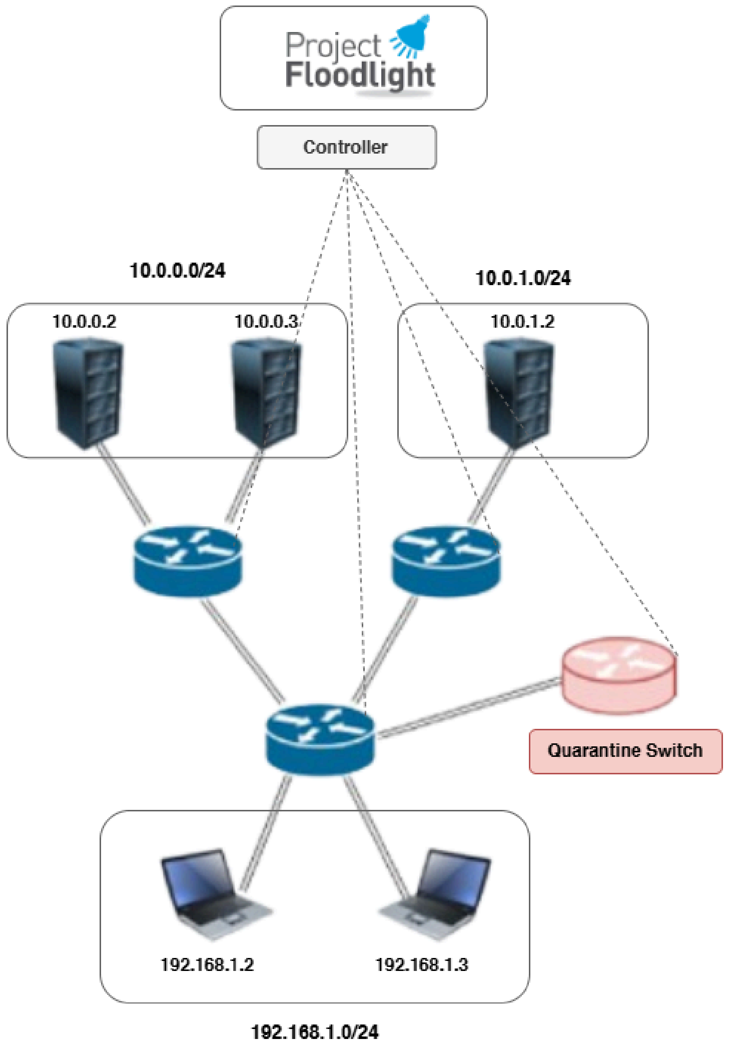
5. Mininet Setup

The system is built around an SDN-based network, where malicious flows are managed through a Floodlight controller. The components of the architecture include:

- **Floodlight Controller**: Acts as the central control plane, exposing a RESTful API for marking/unmarking flows and managing the quarantine mechanism.
- **Quarantine Switch**: A dedicated switch responsible for rerouting packets of malicious flows to the controller. It buffers these packets before they are either flushed or cleared.

- **Client and Server Nodes**: These are the endpoints where flows originate and terminate. When a flow is detected as malicious, its packets are rerouted via the quarantine switch.
- **RESTful API**: Provides external users the ability to interact with the system by marking flows as malicious, unmarking them, and querying the number of buffered packets.

As instructed by project requirements, we created a custom Mininet topology to simulate an SDN-based network for handling malicious flows.

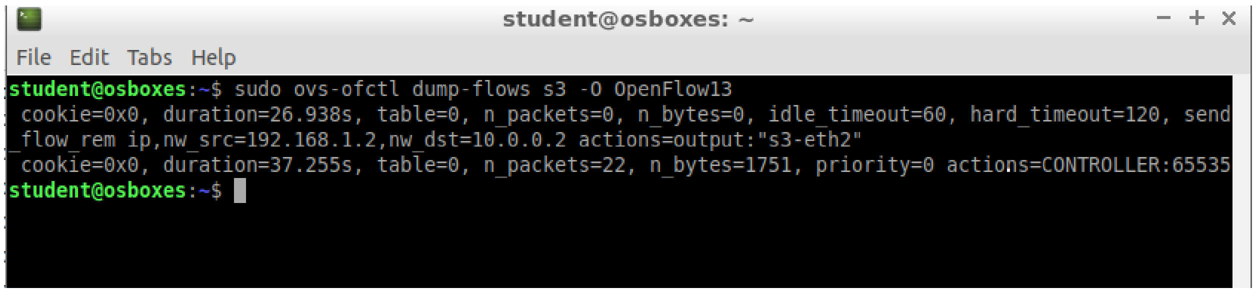


Current Floodlight controlled SDN network

A Python script was developed to construct the network, consisting of multiple switches and hosts across three different subnets: **10.0.0.0/24, 10.0.1.0/24, and 192.168.1.0/24**.

The topology includes a **3 central switch**, a **quarantine switch**, and 3 server and 2 client hosts.
the quarantine switch was assigned a fixed **DPID (Datapath ID)**, to be identified later in the modules

Below is an example of a flow rule being present on one of the central switches (s3) redirecting the matched flow to the quarantine switch (s4):



```
student@osboxes:~$ sudo ovs-ofctl dump-flows s3 -O OpenFlow13
 cookie=0x0, duration=26.938s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, hard_timeout=120, send
_flow_rem ip,nw_src=192.168.1.2,nw_dst=10.0.0.2 actions=output:"s3-eth2"
 cookie=0x0, duration=37.255s, table=0, n_packets=22, n_bytes=1751, priority=0 actions=CONTROLLER:65535
student@osboxes:~$ 
```

Malicious flow rule and its action present on central switch s3

6. Conclusion

One potential use case is mitigating **TCP SYN flood attacks**, where malicious connections overwhelm a server with incomplete handshake requests.
The system effectively allows manual marking and unmarking of malicious flows through a RESTful API. As a future enhancement, **automated malicious flow detection** could be integrated, enabling real-time identification and quarantine of suspicious traffic without manual intervention.