Real-Time Self-Adaptive Distributed Firewall System

Abstract:
This project presents a real-time, self-adaptive distributed firewall system designed to dynamically detect and respond to network anomalies across multiple endpoints. Leveraging Deep Packet Inspection (DPI) tools such as Zeek, Suricata, and Snort, the system identifies threats up to Layer 7 (L7) of the OSI model. Upon detection, the Central Policy Engine generates nftables rules using a trie-based structure for efficient pattern matching and rule deduplication. Rules are dispatched via WebSocket (preferred) or REST API (fallback). Logs are streamed from endpoints to the central server using WebSockets, with periodic cronjob-based REST pings to ensure availability. The entire framework is built using Python and FastAPI, targeting Linux environments like Kali and Ubuntu.

Tools and Technologies (T&T):

Component	Technology/Tool	Purpose
Programming Language	Python	Core development
Framework	FastAPI	REST & WebSocket API framework
Firewall Enforcement	nftables	Rule enforcement at endpoint level
Pattern Matching	Custom Trie (Python)	Fast rule lookup and duplicate prevention
Detection Engines	Zeek, Suricata, Snort	L7 network anomaly detection
DPI Helper	PolarProxy	Decrypt HTTPS for DPI tools
Packet Capture	Scapy	Additional anomaly detection
Communication	REST + WebSockets	Controller ↔ Agents for rules & logs
OS	Kali Linux, Ubuntu	Test environment OS

Functional Requirements:

1. Real-time monitoring and alert generation using DPI tools.
2. Centralized rule generation using trie-based pattern matching.
3. Dispatch of firewall rules via WebSocket (primary) or REST (fallback).
4. Rule enforcement at agents using nftables.
5. Centralized log storage from all agents.
6. Rule ACKs via REST from agent to controller.
7. Continuous health check using cronjobs + REST ping.
8. Real-time WebSocket log streaming to controller.
9. Rule deduplication and tamper-evident logging.

Non-Functional Requirements:

1. Scalability to handle multiple endpoints.
2. High availability through fallback mechanisms.
3. Security through API key authentication.
4. Modularity for easy addition of new detection engines.
5. Efficient rule storage and retrieval via trie.
6. Extensibility for cloud logging and archival.
7. Minimal resource consumption on endpoints.
8. Operability on Debian-based Linux systems.

Test Environment Setup:

- OS: Kali Linux or Ubuntu
- Python 3.9+
- Tools to Install:
  - Zeek
  - Suricata
  - Scapy
  - nftables
  - PolarProxy
  - FastAPI + Uvicorn
  - Requests, PyYAML, Websockets

Testbed Requirements:

- **VM1 (Controller)**:
  - Runs central_engine (main.py)
  - Stores logs from agents
  - Generates and dispatches rules
  - Listens to alerts from agents
- **VM2, VM3, VM4 (Agents/Test Users):**
  - Run Zeek, Suricata, or Snort
  - Run agent FastAPI server (REST & WS)
  - Receive rules from controller and apply via nftables
  - Send alerts/logs to controller via WebSocket/REST
- **Network Configuration:**
  - Controller and agents on same subnet or reachable via IP
  - WebSocket and REST ports opened and forwarded correctly
- **Startup:**
  - A single script or command should setup dependencies
  - Agents should auto-start monitoring and connection services

Network Topology for Real-Time Self-Adaptive Distributed Firewall System

Component	Hostname	IP Address	Services/Modules	Communication Role
Central Policy Engine	VM1	192.168.56.10	- Dispatcher (WebSocket + REST Push) - Zeek/Suricata Alert Receiver - Central Log Collector	- Sends rules to agents - Receives alerts & logs
Agent Endpoint	VM2	192.168.56.11	- Zeek/Suricata - nftables Agent - WS & REST Server	- Sends logs and alerts - Receives rules
Agent Endpoint	VM3	192.168.56.12	- Zeek/Suricata - nftables Agent - WS & REST Server	- Sends logs and alerts - Receives rules
Agent Endpoint	VM4	192.168.56.13	- Zeek/Suricata - nftables Agent - WS & REST Server	- Sends logs and alerts - Receives rules

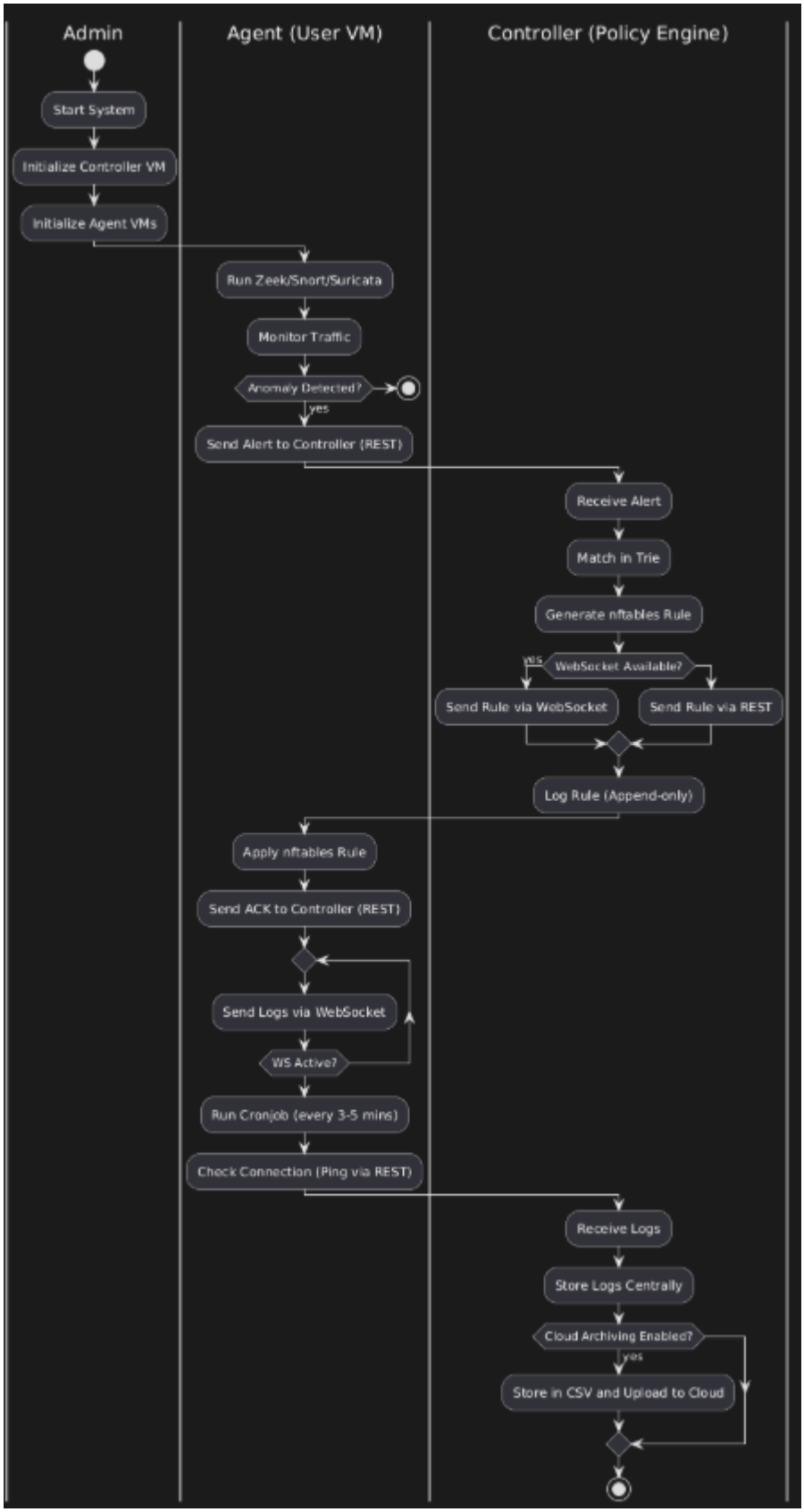
Communication Summary

- → **WebSocket (VM1 ➔ VM2-4)**: Preferred real-time rule push from controller to agents
- ↻ **REST Fallback (VM1 ➔ VM2-4)**: Rule push and acknowledgment in case WebSocket fails
- ⇀ **Log Streaming (VM2-4 ➔ VM1)**: Logs are streamed to VM1 over WebSocket
- □ **Health Cronjobs (VM1 ➔ VM2-4)**: Periodic REST pings to verify endpoint availability or re-establish WS

Testbed Deployment Notes

- All machines reside in the **192.168.56.0/24** internal network
- WebSocket and REST services are exposed on separate ports per agent
- Each agent starts Zeek/Suricata locally to monitor traffic
- Centralized logging and rule dispatch are fully automated from VM1

Activity Diagram



Sequence Diagram