Lab4.DOIP Implementation

1. Overview

This lab provides a hands-on environment to explore the Diagnostics over IP (DoIP) protocol and the Unified Diagnostic Services (UDS) layer that runs on top of it. It simulates a simplified in-vehicle network with three key components:

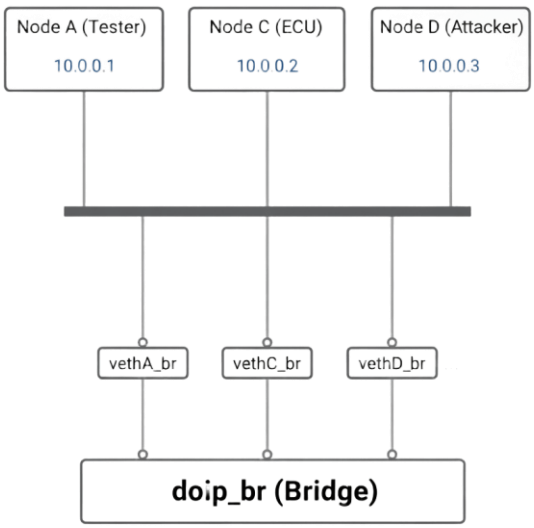
1. **An ECU (Electronic Control Unit):** The target device that provides diagnostic services.
2. **A Legitimate Tester:** A diagnostic tool used by a technician to communicate with the ECU.
3. **An Attacker Node:** A malicious node that attempts to discover, enumerate, and exploit the ECU.

The lab is designed to demonstrate both standard diagnostic communication and common security vulnerabilities, such as logical address enumeration, service ID scanning, and security access brute-forcing.

2. Lab Topology & Network Setup

The lab environment is isolated using Linux network namespaces to simulate three distinct nodes connected to a virtual network bridge.

2.1. Network Diagram



2.2. Setup Script (setup.sh)

The **setup.sh** Bash script automates the creation of this network environment. It performs the following actions:

- Cleans up any previous lab artifacts.
- Creates a Linux bridge named **doip_br**.
- Creates three network namespaces: **NodeA**, **NodeC**, and **NodeD**.
- Creates virtual Ethernet (**veth**) pairs to connect each namespace to the bridge.
- Assigns a unique IP address to each node within the **10.0.0.0/24** subnet.

2.3. Running the Setup

Open a terminal and execute the following commands. You will need **sudo** privileges to manage network namespaces.

Make the script executable

chmod +x setup.sh

Run the script to build the network

sudo ./setup.sh

3. Lab Components

The lab consists of three Python scripts, each representing a different node in the network.

3.1. server_node_c.py - The ECU

- **Role:** Simulates a DoIP-enabled ECU.
- **Namespace: NodeC** (IP: **10.0.0.2**)
- **Key Features:**
    - **DoIP Server:** Listens for UDP discovery requests (0x0001) and TCP routing activation requests (0x0005).
    - **UDS Services:** Implements a subset of standard UDS services:
        - **0x10** Diagnostic Session Control (Default, Extended, Programming)
        - **0x11** ECUReset
        - **0x22** ReadDataByIdentifier (e.g., VIN, Vehicle Model)

- **0x27** Security Access (Seed-Key mechanism)
- **0x2E** WriteDataByIdentifier
- **0x31** RoutineControl (e.g., Self-Test, Checksum calculation)

How to Run:

sudo ip netns exec NodeC python3 server_node_c.py

3.2. NodeA_Tester.py - The Legitimate Tester

- **Role:** Acts as a legitimate diagnostic tool.
- **Namespace: NodeA** (IP: **10.0.0.1**)
- **Key Features:**
  - **Automated Discovery & Routing:** Automatically finds the ECU on the network and establishes a TCP session.
  - **Interactive UDS Session:** After connecting, it provides an interactive prompt where you can manually type UDS requests in hex (e.g., **22 F1 90** to read the VIN).
  - **Automatic Security Handling:** If a security seed is requested, it automatically calculates the correct key and sends it.

How to Run:

sudo ip netns exec NodeA python3 NodeA_Tester.py

3.3. Attacker.py - The Attacker Node

- **Role:** Simulates a malicious actor attempting to compromise the ECU.
- **Namespace: NodeD** (IP: **10.0.0.3**)
- **Key Features:**
  - **DoIP Discovery:** Finds the ECU IP via UDP broadcast.
  - **Logical Address Brute-Force:** Attempts to find a valid Tester Logical Address by brute-forcing the routing activation request.
  - **Attack Modules:** Once connected, it provides a menu with several attack options:

1. **Service ID Enumeration:** Scans through possible UDS service IDs to map which ones are supported by the ECU.
2. **Security Brute-Force:** Attempts to unlock the ECU by brute-forcing the security key.
3. **Manual UDS Command:** Allows for manual injection of any UDS command.

How to Run:

sudo ip netns exec NodeD python3 Attacker.py

4. Step-by-Step Execution Guide

You must run each component in a separate terminal. The ECU must be started first.

Step 1: Open Terminal 1 - Start the ECU (Node C)

Navigate to the directory containing your files

cd /path/to/your/lab/files

Execute the ECU simulator in the NodeC namespace

sudo ip netns exec NodeC python3 server_node_c.py

You should see output indicating that the ECU is listening for UDP and TCP connections.

Step 2: Open Terminal 2 - Start the Legitimate Tester (Node A)

Navigate to the directory containing your files

cd /path/to/your/lab/files

Execute the legitimate tester in the NodeA namespace

sudo ip netns exec NodeA python3 NodeA_Tester.py

The tester will automatically discover the ECU, activate a routing session, and then prompt you for UDS commands.

Step 3: Open Terminal 3 - Start the Attacker (Node D)

Navigate to the directory containing your files

cd /path/to/your/lab/files

Execute the attacker script in the NodeD namespace

sudo ip netns exec NodeD python3 Attacker.py

The attacker will first discover the ECU and then attempt to enumerate a valid logical address. Once successful, it will present you with the attack menu.

5. Understanding the Interactions

- When you start the **Tester** or **Attacker**, they send a DoIP Vehicle Identification Request (0x0001) as a UDP broadcast. The **ECU** receives this and responds with a Vehicle Identification Response (0x0004) containing its VIN and Logical Address.
- After discovery, the **Tester/Attacker** establishes a TCP connection to the **ECU** on port 13400.
- It then sends a Routing Activation Request (0x0005) to establish a diagnostic session. The **ECU** validates the source logical address and responds with a Routing Activation Response (0x0006).
- Once the session is active, all communication uses DoIP Diagnostic Messages (0x8001 for requests, 0x8002 for responses) to encapsulate UDS payloads.