Modbus Code Documentation

Introduction

The Modbus protocol is widely used for communication in industrial systems and automation processes. This documentation provides detailed instructions for using the Modbus protocol code, which supports packet capture, manipulation, and modification of Modbus messages.

The Modbus protocol implementation is designed to handle both live capture and offline processing of Modbus packets:

- **Live Capture**: Captures packets directly from a network interface.
- **Offline Processing**: Reads and modifies packets from a PCAP file.

System Requirements

Python Version:

- Python 3.x

Required Libraries:

- `scapy`: For crafting, capturing, and manipulating network packets.
- `pyshark`: For live packet capture and filtering.

Install the required libraries using the following command:

```
pip install scapy pyshark
```

Usage Instructions

The Modbus protocol code provides two options for capturing and processing packets: **live capture** from a network interface and **offline processing** of pre-recorded `.pcap` files.

Command Line Options

You can use the following commands to run the code:

1. **Live Capture**:
   Capture Modbus packets directly from a live network interface:

   ```
   python3 script_name.py --livecapture --store <filename.pca
   p>
   ```

   Replace `script_name.py` with the name of your script. The captured packets will be stored in the specified file.

2. **Offline Processing**:
   Process packets from an existing `.pcap` file:

   ```
   python3 script_name.py --pcapfile <input_filename.pcap> --
   output <output_filename.pcap>
   ```

3. **Help**:

   ```
   python3 script_name.py -h
   ```

Network Interface and Filter Setup

To capture only Modbus packets, the script applies a filter based on EtherType:

- **Network Interface**: `eth1` (can be changed based on your system).
- **EtherType Filter**: Modbus EtherType is `0x0800` (for IP packets).

```
networkInterface = "eth1"
filter_string = "ether proto 0x0800"
Modbus_TYPE = 0x0800
```

You can modify the `networkInterface` variable in the code to capture from a different interface.

Manual Code Modifications

The core packet processing logic and modification of Modbus packets are handled within the code. Specific sections are highlighted below:

- The code supports modifying the **function code** and **reference number** for Modbus Query packets.
- For Modbus Response packets, it allows modification of the **function code** and **register value**.

Modification Logic

The core modification logic is implemented in the section where the following fields are updated:

- **Query Packets**:
  - **Function Code**: Modified to `0x02`.
  - **Reference Number**: Modified in the last 4-2 bytes.
- **Response Packets**:
  - **Function Code**: Modified to `0x06`.
  - **Register Value**: Modified in the last 2 bytes.

This part of the logic allows for injecting the modified packets back into the network or saving them to a `.pcap` file using the `wrpcap` function.

Sample Workflow

1. **Live Capture**:
   - The script captures packets from the `eth1` interface, filters for Modbus packets using `ether proto 0x0800`, and stores them in a `.pcap` file.
2. **Filtering Modbus Packets**:
   - The captured packets are filtered to retain only the Modbus packets, which are then processed.
3. **Modifying Packets**:
   - Users can choose specific packets for modification. The script allows for changing the **function code** and **reference number** for Query packets, or

the **function code** and **register value** for Response packets.

4. **Injecting or Saving Packets**:
    - The modified packets can be injected back into the network (optional), or saved in a `.pcap` file for further analysis.

Additional Notes

- **DOS Attack Warning**: When injecting packets at high frequency, consider the potential impact on the network and choose a method accordingly.

  → Refer to documentation on "**Tools for sending packets**" for more information.

- **Wireshark**: To analyze Modbus packets, you can use Wireshark and apply the display filter `ether proto 0x0800` to isolate Modbus traffic.
- **Security Considerations**: This code should be used with caution in live environments. Unauthorized manipulation of Modbus packets can have serious implications for industrial systems.
