IEDFuRL: A Black-box Fuzz Tester for IEC61850-based Intelligent Electronic Devices using Reinforcement Learning

A THESIS

SUBMITTED FOR THE DEGREE OF

Master of Technology (Research)

IN THE

Faculty of Engineering

BY

Kanmani A

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

May, 2025

Declaration of Originality

I, **Kanmani A**, with SR No. **04-04-00-10-22-22-1-21450** hereby declare that the material presented in the thesis titled

IEDFuRL: A Black-box Fuzz Tester for IEC 61850-based IEDs using Reinforcement Learning

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2022-2025**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Friday 28th February, 2025 Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Prof. Vinod Ganapathy Advisor Signature

DEDICATED TO

My Family and Friends

who supported me through this journey

Acknowledgements

I express my deepest gratitude to Dr. Vinod Ganapathy for his unwavering support and guidance throughout this journey. His encouragement to approach problems with a positive mindset and think beyond conventional solutions has been invaluable. His insightful questions have consistently challenged me to refine my ideas and discover more effective solutions. The knowledge and perspective I have gained under his mentorship will continue to benefit me throughout my life, for which I am truly grateful.

I extend my sincere thanks to Dr. Gurunath Gurrala from the Department of Electrical Engineering for accommodating my test devices and workstation in the FIST Lab. I also thank the members of the FIST Lab for the technical support they provided during the test setup of the project. I am grateful to the Powergrid Centre of Excellence in Cybersecurity (PGCoE) for supporting and funding the project, and I would like to express my gratitude to Mr. Bhargav Nerayanoor, Technical Officer at PGCoE, for his valuable assistance in helping me understand the devices and protocols involved in this work.

I am profoundly thankful to my parents for their unconditional support, which has enabled me to pursue my aspirations. I also extend my gratitude to my brother for his strong belief in me and the confidence he has always had in me. A heartfelt thanks to my friend Priyanshu, whose insightful discussions on my project and its methodologies have been instrumental in shaping my work. His motivation during challenging times has been invaluable. I am also grateful to my friend Prashansa for being an incredible support system on campus, helping me navigate academics and friendships. A special thanks to my friend Vijeshwarya for being a constant source of moral support during my low phases.

This work would not have been possible without the encouragement and support of all these individuals, and I am truly grateful for their contributions to my academic journey.

Abstract

Intelligent Electronic Devices (IEDs) are essential components of modern power grids, functioning as microprocessor-based controllers that facilitate communication, monitoring, protection, and control within Supervisory Control and Data Acquisition (SCADA) systems. As these devices operate across power generation, transmission, and distribution, they have become prime targets for cyberattacks, leading to risks such as large-scale power disruptions, unauthorized data access, and critical equipment failures. Communication between these devices is governed by the IEC 61850 standard, which defines the Manufacturing Message Specification (MMS) protocol over TCP/IP network stack. In this thesis, we propose **IEDFuRL**, a black-box fuzz testing tool for IEC 61850-based IEDs. **IEDFuRL** aims to identify vulnerabilities in the communication module of the IEDs. Our approach begins by crafting valid MMS requests targeting various data points within the IEDs and using response packets as feedback for categorization. We develop a reinforcement learning (RL) agent that is rewarded for discovering new category of responses and crashes. The agent learns the optimal sequence of mutations from any specific request packet to generate new category of responses and crashes thereby increasing the fuzz testing coverage.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Intelligent Electronic Devices (IEDs) are microprocessor-based controllers that serve as the backbone of Supervisory Control and Data Acquisition (SCADA) [1] systems in modern power grids. These devices are pivotal for various power grid operations, including data communication, measurement, protection, and control, spanning the generation, transmission, and distribution layers of electricity infrastructure. Given their critical role, IEDs are increasingly targeted by cyberattacks, with potential consequences including widespread power outages, data breaches, equipment failures, and cascading blackouts.

Recent cyberattacks on Intelligent Electronic Devices (IEDs) used in power grids reveal the increasing risks associated with their vulnerabilities, especially those tied to communication protocols like IEC 61850 [2]. For example, the 2016 Industroyer [3] malware attack on Ukraine's power grid exploited weaknesses in IEC 60870-5-101 and IEC 61850 protocols to remotely control substation IEDs, nearly causing large-scale disruptions. Similarly, campaigns like Dragonfly [4] and Havex [5] targeted SCADA systems by manipulating protocols to gain control over grid infrastructure. The Hive ransomware group infiltrated Tata Power Company Limited's communication network, causing operational disruptions and exposing sensitive data [6]. Although primarily aimed at reconnaissance, these attacks exposed the potential for operational disruptions and underscored the importance of securing IED communication channels and firmware.

Further highlighting these risks, persistent cyber threats have been targeting India's power grid infrastructure. In 2021, investigations revealed that Chinese state-sponsored groups may have infiltrated Indian power systems, potentially leading to blackouts [7]. In 2022, Chinese hackers targeted power grid assets in Ladakh, raising concerns over the security of critical infrastructure [8]. Additionally, the Triton [9] malware, which has expanded its scope to target IEDs, poses a direct threat by exploiting protocols such as MMS to intentionally disrupt equip-

ment operations, raising critical safety and reliability concerns. These incidents collectively underline the pressing need for proactive vulnerability detection and mitigation strategies to secure critical infrastructure against evolving cyber threats.

To address the risks posed by these cyber threats, this thesis proposes a technique to test IEDs for security vulnerabilities. More precisely, our goal is to detect security vulnerabilities in the IED that can cause the IED to malfunction by crashing (causing it to become unresponsive over the network) or reveal sensitive information.

Uncovering vulnerabilities in IEDs presents significant challenges, primarily due to their reliance on proprietary firmware developed by major vendors such as Siemens, ABB, and Schneider Electric. The source code of the firmware is not usually available [10]. Additionally, patches and bug fixes issued post-deployment are also encrypted, and information about inputs that could trigger vulnerabilities is never disclosed. This complete lack of transparency complicates vulnerability assessments, rendering traditional techniques such as source code or binary analysis impractical. Furthermore, IEDs are functionally complex, and the technical challenges of emulating their firmware without internal knowledge hinder efforts to proactively identify and mitigate vulnerabilities in these critical systems.

Fuzzing [11] is a widely recognized and effective technique for systematically identifying vulnerabilities in software and hardware systems. By generating diverse inputs and analyzing their effects on a target device, fuzzing can uncover unexpected behaviors, crashes, or anomalies indicative of vulnerabilities. Among the three types of fuzzing—black-box, white-box, and gray-box [12]—black-box fuzzing is particularly well-suited for IEDs, as it does not require prior knowledge of the device's internal architecture or firmware. This approach makes it ideal for testing proprietary systems, where access to source code is unavailable.

When employing fuzzing to test IEDs, one of the key questions to be addressed is *what* software to test? IEDs are functionally complex, consisting of several layers of software, often proprietary and vendor- and device-specific. However, a feature common to all modern IEDs is that they all use the IEC 61850 communication standard. The IEC 61850 communication standard is at the heart of modern SCADA systems, enabling high-speed information exchange and interoperability between devices from various vendors. This standard defines several application-layer protocols, including Sampled Measured Values (SV) [13], Generic Object-Oriented Substation Event (GOOSE) [14], and Manufacturing Message Specification (MMS) [15]. Of these, MMS plays a crucial role at the supervisory layer, facilitating communication between SCADA systems and IEDs for critical functions such as control, monitoring, and data exchange. Therefore, vulnerabilities in the MMS protocol pose significant threats, as adversaries can exploit these weaknesses to disrupt grid operations or gain unauthorized access to critical systems.

The research in this thesis focuses on applying fuzzing to the MMS protocol to uncover vulnerabilities in IED's communication module. By leveraging black-box fuzzing techniques inspired by tools like Snipuzz [16], our proposed framework called **IEDFuRL** enables testing of IEDs with only black-box access to the MMS protocol implementation of the IED device's communication module. Additionally, introduces a novel reinforcement learning-based approach to enhance fuzzing efficiency, allowing a protocol-aware agent to intelligently mutate request fields and learn optimal testing strategies. Our combined approach aims to develop a generic fuzzing solution applicable to any IED, thereby strengthening the security and resilience of power grid infrastructures.

1.1 Contributions

The key contributions of our research, titled **IEDFURL: A Black-box Fuzz Tester for IEC 61850-based IED Devices using Reinforcement Learning**, are summarized as follows:

- **Reinforcement Learning (RL) for Automated IED Fuzzing:** We propose and implement a novel fuzzing framework that leverages reinforcement learning to identify vulnerabilities in IEC 61850-based Intelligent Electronic Devices (IEDs). The RL agent dynamically learns optimal mutation strategies for specific fields in MMS (Manufacturing Message Specification) requests, enabling an efficient exploration of the IED's state space.
- **Protocol-Aware Fuzzing Mechanism:** Our framework incorporates protocol-awareness, allowing the fuzzing agent to understand and target specific fields in MMS requests. By applying tailored mutation strategies to these fields, the agent prioritizes more impactful test cases, improving vulnerability detection efficiency compared to traditional approaches.
- **Applicability Across Diverse IED Configurations:** The proposed framework is adaptable to various IED implementations of the IEC 61850 protocol. By incorporating device-specific configurations into the initial seed set, the agent efficiently targets vulnerabilities unique to individual IED models.
- **Evaluation with Real-World IEDs:** The methodology has been validated through experiments on real-world IEDs, demonstrating its effectiveness in uncovering vulnerabilities, increasing code coverage, and enhancing the robustness of the devices tested.

By addressing the challenges associated with vulnerability assessment in IEDs and targeting the critical MMS protocol, this research contributes to the development of secure and resilient

power grid systems. It provides a pathway for systematically identifying vulnerabilities in proprietary systems, helping to safeguard critical infrastructure against emerging cyber threats.

1.2 Outline of Thesis

The rest of this thesis is organized as follows:

- **Chapter 2:** We explain the IEC 61850 standard and the MMS protocol to establish a foundational understanding. This chapter also illustrates the layered communication structure of MMS request and response packets.
- **Chapter 3:** We review existing fuzz testing methodologies for IEC 61850 and highlight their limitations. We explore the adaptation of Snipuzz to IED fuzzing, detailing its mechanism.
- **Chapter 4:** We introduce reinforcement learning (RL) and justify its application in fuzz testing. We demonstrate a protocol-aware fuzzing mechanism. We describe the details reinforcement learning agent while highlighting key improvements over Snipuzz.
- **Chapter 5:** We describe the operational framework of IEDFuRL, outlining its components and technology stack.
- **Chapter 6:** We present the bugs identified by IEDFuRL in real-world IEDs. We categorize and explain the discovered bugs.
- **Chapter 7:** We evaluate the effectiveness of the RL agent by comparing it with Snipuzz and a random policy agent. We show the improvements achieved by IEDFuRL through performance analysis.
- **Chapter 8:** We conclude with insights into future research directions in automated fuzz testing for power grid security.

Chapter 2

Background

Role of IEDs in Power Grid Communication Intelligent Electronic Devices (IEDs) serve as a critical interface between control systems and physical power grid components, facilitating seamless communication and operational control. These devices are responsible for collecting real-time data from sensors and power equipment, enabling continuous monitoring of grid conditions. By measuring key electrical parameters such as voltage, current, and frequency, IEDs play a vital role in ensuring the stability and reliability of power distribution networks.

Beyond basic data acquisition, IEDs are equipped with advanced functionalities, including event monitoring, disturbance recording, and fault detection, which enhance the resilience of the grid. They also execute control commands, such as circuit breaker operations or load adjustments, to maintain optimal performance and prevent system failures. Additionally, IEDs enable real-time synchronization between various components of the power grid, ensuring coordinated and efficient operation.

Given their central role in modern power grid infrastructure, IEDs are deeply embedded in critical communication pathways, making them essential for grid automation and stability. However, this also exposes them to potential cybersecurity threats, as vulnerabilities within IEDs can lead to disruptions in power transmission and distribution. This underscores the need for robust security mechanisms to safeguard IEDs against cyberattacks and ensure the reliability of power systems.

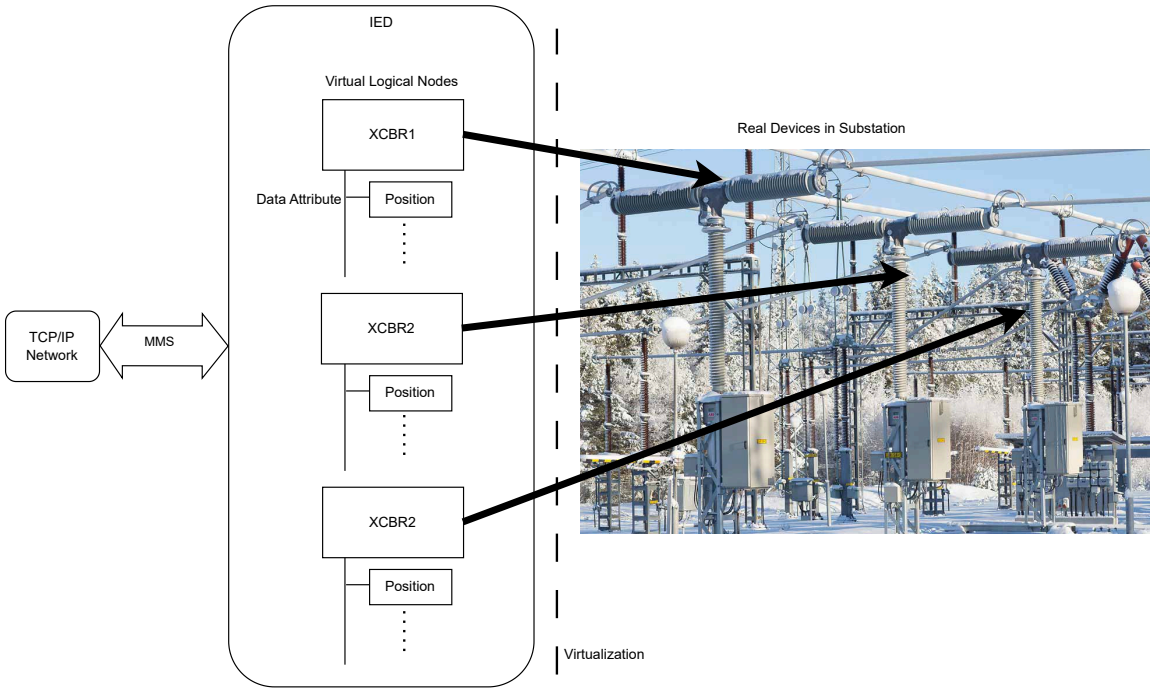


Figure 2.1: Working of MMS

Working of MMS: Explained with an Example The MMS protocol in the IEC 61850 standard enables communication between control systems and Intelligent Electronic Devices (IEDs) in a power grid. In this example [17], three circuit breakers (real power equipment) are virtually mapped to logical nodes within the IED. These logical nodes serve as digital representations of the circuit breakers within the IED which can be accessed via the MMS protocol.

Each logical node contains data attributes that define its state and behavior. As shown in the figure 2.1, one such attribute is Position, which determines whether the circuit breaker is open or closed. Using the MMS protocol, control commands can be sent to modify this Position attribute, thereby remotely operating the circuit breakers. This virtual representation enables efficient monitoring and control of power grid operations, ensuring real-time synchronization and automation within substations.

2.1 MMS Protocol Encoding: Basic Encoding Rules (BER) and ASN.1 Representation

The Manufacturing Message Specification (MMS) protocol follows the Basic Encoding Rules (BER) as defined by the Abstract Syntax Notation One (ASN.1) standard [18, 19]. ASN.1

provides a formal notation to describe structured data, ensuring interoperability across different platforms and devices in IEC 61850-based communication systems. BER employs a Tag-Length-Value (TLV) encoding format, which facilitates structured, efficient, and deterministic parsing of messages.

2.1.1 Tag-Length-Value (TLV) Structure in BER

In BER encoding, each field in an MMS message consists of three key components:

- **Tag (T)**: Identifies the data type, such as an object, integer, sequence, or bit-string.
- **Length (L)**: Specifies the length of the value field.
- **Value (V)**: Contains the actual data being transmitted.

This structured format allows for efficient parsing and decoding of MMS messages, ensuring that receiving devices interpret the data correctly.

2.1.2 Illustration: BER Encoding in an MMS Request

The extracted TLV encoding is as follows for the invokeID field:

Field	Tag (T)	Length (L)	Value (V)
invokeID	0x02	1 byte = 0x01	0x15

Table 2.1: Example of BER TLV Encoding in an MMS Write TrgOps Request

This TLV structure ensures precise communication between the SCADA system and Intelligent Electronic Devices (IEDs), facilitating control, monitoring, and event reporting in the IEC 61850 standard.

2.1.3 Visual Representation

Figure 2.2 below illustrates a real-world MMS request packet encoded using BER TLV format, as observed in a network capture.

Wireshark [20] has an interpreter that understands these rules and converts the hexadecimal bitstring in the packets to human readable form as expressed in the left bottom of the request sample in 2.2 and 2.3.

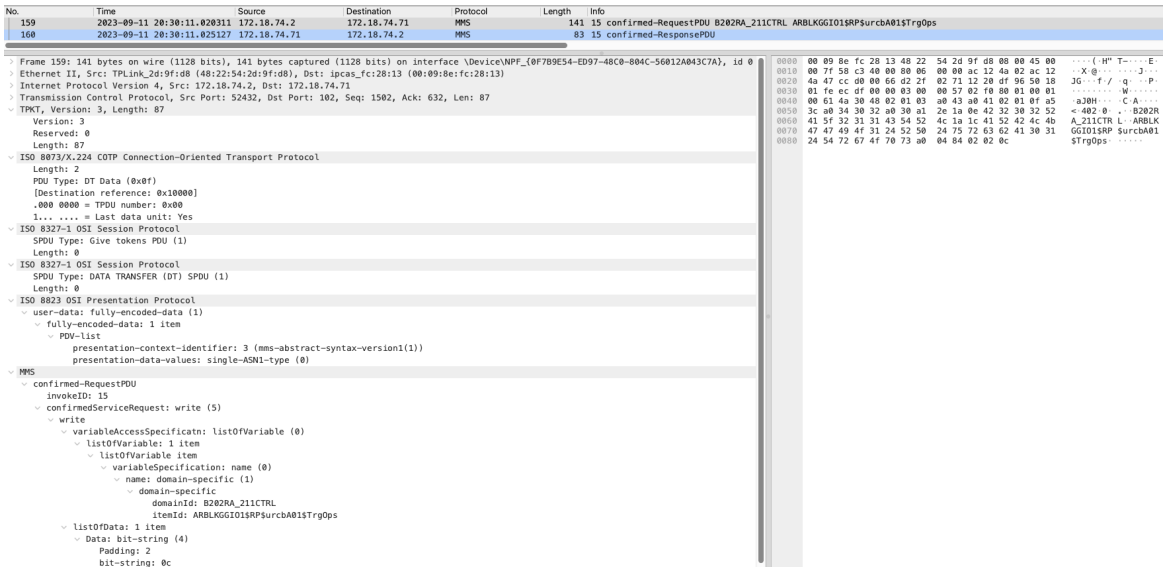


Figure 2.2: MMS Request Example

2.2 Analysis of MMS Request Packets in IEC 61850 Communication

The image 2.2 captures a confirmed service request in an IEC 61850-based communication system, specifically for an MMS write operation on Siemens SIPROTEC 7SA522 [21]. This request is transmitted over TCP port 102, which is standard for MMS communication between Intelligent Electronic Devices (IEDs). This PDU (Protocol Data Unit) is being used to write a value to a specific attribute within the IEC 61850 data model.

2.2.1 Layered Breakdown of the Protocol Stack

MMS (Manufacturing Message Specification) communication in IEC 61850 follows a layered approach, leveraging both modern TCP/IP networks and ISO transport over TCP to ensure reliable data exchange between SCADA systems and Intelligent Electronic Devices (IEDs). Each protocol layer in the MMS request packet plays a crucial role in establishing and managing the communication.

Transmission Control Protocol (TCP) [22] TCP operates at the Transport Layer and ensures the reliable delivery of MMS messages. It provides essential mechanisms such as error detection, retransmission of lost packets, flow control, and in-order delivery of messages. The MMS protocol communicates over `port 102`, a dedicated port. This ensures a stable connection between the SCADA system (MMS client) and the IED (MMS server), enabling seamless data

transfer.

Transport Packet (TPKT – RFC 1006) [23] RFC 1006 defines the TPKT layer, which was added to the traditional MMS protocol stack in the transport layer to construct discrete data units called Transport Protocol Data Units(TPDUs) instead of continuous data stream and act as bridge between ISO protocol stack and TCP/IP base stack. It includes a 4-byte header that specifies the total packet length, helping in the segmentation and reassembly of COTP packets.

Connection-Oriented Transport Protocol (COTP) [24] COTP, as defined by the *ISO 8073* standard, operates at the Transport Layer and is responsible for establishing and maintaining logical transport connections. It supports session multiplexing, ensuring that multiple MMS transactions can take place over a single TCP connection without interference. Additionally, COTP handles connection-oriented communication, ensuring that MMS messages are reliably delivered in the correct sequence.

ISO Session Layer The ISO Session Layer (*ISO 8327-1*) [25] provides structured synchronization of MMS communications, ensuring that each session is independently managed. It plays a key role in logically separating multiple MMS interactions.

ISO Presentation Layer The OSI Presentation Layer (*ISO 8823-1*) [26] ensures that MMS messages are correctly encoded and interpreted by both the client and the server. It uses **Abstract Syntax Notation One (ASN.1)** encoding, specifically following the **Basic Encoding Rules (BER)**, which structures messages in a **Tag-Length-Value (TLV)** format.

Manufacturing Message Specification (MMS) Layer [15] The Manufacturing Message Specification (MMS) operates at the application layer and serves as the primary communication protocol for exchanging data between SCADA systems and Intelligent Electronic Devices (IEDs). MMS facilitates real-time monitoring, control, and data retrieval by defining standard message formats for reading, writing, reporting, and commanding operations on IEDs. It leverages the underlying ISO and TCP/IP layers to ensure structured and reliable communication. MMS gives access to the logical nodes and data attributes in a standardized manner, ensuring interoperability across different vendors' devices in the power grid infrastructure. By defining a consistent way to access and manipulate substation data, MMS plays a crucial role in automation, diagnostics, and event-driven control within IEC 61850-based systems.

All fields belonging to the Presentation (PRES), Association Control Service Element (ACSE), and Manufacturing Message Specification (MMS) layers are considered mutable within our fuzzing framework. These layers contain semantic protocol elements relevant to service re-

quests and device interaction, making them ideal candidates for targeted mutations. On the other hand, fields from lower layers such as the Session Layer and Connection-Oriented Transport Protocol (COTP) are not mutated. Modifying these fields typically disrupts the establishment and maintenance of the communication session between the TCP client and the IED server, resulting in premature connection failures. Therefore, to ensure the stability of the communication interface while maximizing the impact of the fuzzing strategy, mutations are confined to PRES, ACSE and MMS layers.

2.2.2 Breakdown of Confirmed Request Protocol Data Unit

The MMS Application layer in 2.2 shows a **confirmed service request**, which is a type of MMS PDU [27] which refers to a request sent from a client to an IED server where the server explicitly acknowledges receipt of the request and confirms that it will attempt to fulfill the service, essentially guaranteeing a response back to the client regarding the outcome of the operation; this is in contrast to an unconfirmed service where the server may not send a confirmation upon receiving the request. Therefore, Confirmed Service Request request ensures reliable execution of control commands by requiring acknowledgment from the IED server. The structure of the MMS confirmed service request includes the following elements:

- Invoke ID
  - **Value: 15** The invoke ID is an identifier uniquely identifies this request, ensuring that responses are correctly mapped to corresponding requests.
- Service Type: Write Request
  - **Service: confirmedServiceRequest: write (5)**
  - This request is an MMS **write operation**, meaning it modifies a specific data attribute within the IED.
- Variable Access Specification This section defines the variables or attributes within the IED data model being accessed.
- List of Variables:
  - **Domain ID:** B202RA_211CTRL
  - **Item ID:** ARBLKGGIO1RPurcbA01$TrgOps
    1. **B202RA_211CTRL** - Logical Device (LD) name, representing a specific function within the relay configuration.

    - **B202RA** indicates *Bay 202 Relay A*.
    - **211CTRL** suggests that it is associated with *control functions*.
  2. **ARBLKGGIO1** - Logical Node (LN), which represents a specific function within the logical device.
    - **ARBLK** refers to an *auto-reclosing blocking function*.
    - **GGIO1** corresponds to the *Generic Input/Output* (IEC 61850 LN for general-purpose I/O).
  3. **RP** - Report Control Block (RCB) settings, which define how reporting mechanisms function in the IED.
  4. **urcbA01** - Unbuffered Report Control Block (URCB).
    - Unbuffered reports are sent immediately when triggered by specific conditions but are *not stored* in the device.
    - If the connection is lost, these reports are lost as well.
    - **A01** denotes a *specific instance* of the URCB.
  5. **TrgOps** - Trigger Options in the Report Control Block.
    - Defines conditions that trigger a report.
    - SIEMENS 7SA522 supports the following triiger conditions: Integrity, Data change, Quality change, Data update, General Interrogation

- Data Section
  - **Data Type: Bit-String (4 bits)**
  - **Padding: 2 (bit alignment)**
  - **Bit String Value: 0c (0000 1100 in binary)**
    - This value determines the conditions that trigger a report from the IED.
    - "0c" in hexadecimal translates to a combination of two trigger conditions.

2.3 Analysis of MMS Response Packets in IEC 61850 Communication

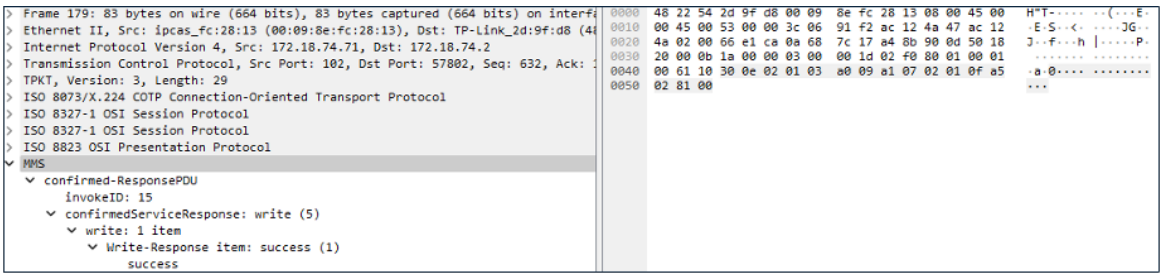


Figure 2.3: Corresponding MMS Response Example

The response packet to the acknowledgment of an MMS request, confirming the execution of a previously sent command to the Intelligent Electronic Device (IED). As seen in the captured response 2.3, the Manufacturing Message Specification (MMS) layer encapsulates a `confirmed-ResponsePDU`, which consists of an `invokeID`, indicating that this response corresponds to a specific request, and a `confirmedServiceResponse` of type `write`. The `write: Response item` field explicitly indicates the outcome of the write operation, showing a result of `success (1)`. This confirms that the IED has successfully processed the modification request and updated the target data attribute accordingly.

The response message ensures the integrity of control commands and data modifications in IEC 61850-based systems, as it allows the client (e.g., SCADA, RTU, or an automated testing framework) to verify that its issued commands have been correctly applied.

In the case of fuzz testing and security evaluation, analyzing such response packets is critical for identifying inconsistencies, unexpected behavior, or deviations from the IEC 61850 standard, which may indicate potential vulnerabilities in the IED's communication module implementation.

In some cases, the request may be rejected, in that case Reject Protocol Data Unit is received. When the connection is aborted, Abort Protocol Data Unit is received. When there is an error executing the request Confirmed Error Protocol Data Unit is received by the Client.

2.3.1 Negative Responses in MMS Communication

In a typical Manufacturing Message Specification (MMS) communication exchange, a client sends requests to an Intelligent Electronic Device (IED) and expects a corresponding response. While successful executions result in a `confirmed-ResponsePDU`, certain conditions may cause

the request to be rejected, aborted, or result in an execution failure. These scenarios are managed through specific Protocol Data Units (PDUs) designed to handle errors and connection terminations in a structured manner.

Reject Protocol Data Unit (RejectPDU) If an MMS request does not conform to the expected protocol format, contains invalid parameters, the IED rejects the request and responds with a `RejectPDU`. This response indicates that the request was deemed unacceptable due to syntactical, semantic, or security-related reasons. The `RejectPDU` typically includes an error code specifying the cause of rejection, helping the client diagnose and correct the issue before resending the request.

Abort Protocol Data Unit (AbortPDU) In cases where an ongoing MMS session encounters a critical failure or an unexpected termination condition, an `AbortPDU` is generated. This PDU signifies the abrupt termination of the communication session, which may occur due to IED-side resource constraints, or protocol violations. Unlike a `RejectPDU`, which applies to a single request, an `AbortPDU` terminates the entire session, requiring the client to establish a new connection before further interactions can take place.

Confirmed Error Protocol Data Unit (Confirmed-ErrorPDU) Even if a request is structurally correct and accepted for processing, the IED may encounter an error during execution. In such cases, the device responds with a `Confirmed-ErrorPDU`, indicating that the requested operation could not be completed successfully. This response typically includes an error classification.

Significance in Fuzz Testing and Security Analysis From a security and robustness testing perspective, analyzing these negative responses is critical for evaluating the resilience of an IED's MMS implementation.

Understanding these response mechanisms allows us to refine fuzzing strategies, ensuring that test cases effectively probe for weaknesses in MMS-based communication while maintaining protocol compliance.

Chapter 3

Related Work

3.1 Existing IEC61850 Black-box Fuzzers

Fuzzing is a well established technique in software testing domain but it has not been explored much in IED testing or industrial control systems domain to ensure the security of critical infrastructure like the powergird system. The comparison of existing IEC61850 fuzzers is showed in the Table 3.1.

Model	Cybersecurity test-bed for IEC 61850 based smart substations [28]	Vulnerability Mining System Based on Fuzzing for IEC 61850 Protocol [29]
Algorithm	Mutation based fuzzing – Algorithm not mentioned	Based on Sulley Fuzzer [30] – Flexible for Protocol Fuzzing
Devices Tested	The IEDs include 10 real protection relays, 8 measurement and control devices for 220kV and above smart substations.	Devices tested on are not specified.
Feedback Mechanism	No Feedback	Least Feedback – Device Alive check is done to assign weight for fields to be fuzzed.
Code	Not Available	Not Available

Table 3.1: Comparison of existing IEC61850 Fuzzers

In IEDFuRL, we incorporate an efficient feedback mechanism that clusters the received response and prioritizes requests for fuzzing based on exploration of new categories of responses.

3.2 Peach Fuzzer

Peach Fuzzer [31] is a generic fuzzing framework that lacks a feedback mechanism. Additionally, only the community edition of Peach Fuzzer has been open-sourced by GitLab, limiting its capabilities for advanced fuzz testing. Configuring Peach Fuzzer to test a complex protocol like IEC 61850 MMS is a labor-intensive process, as the data and states must be modeled using XML (eXtensible Markup Language), which is time-consuming. Conguration files called PeachPit files must be written to describe the structure of data and the nature of different states. The hierarchical and nested structure of MMS objects makes them particularly challenging to model accurately.

To illustrate, consider modeling the following part of an MMS request:

```
invokeID: 15
confirmedServiceRequest:  write (5)
```

The configuration for Peach Fuzzer must be explicitly defined so that it can interpret and mutate the relevant fields.

```
<!-- Template Data Model for any Key Value Pair -->
<DataModel name="KeyValueTemplate">
  <String name="Key" />
  <String value=": " token="true" />
  <String name="Value" />
</DataModel>

<DataModel name="confirmed-ReuestPDU">
  <Block name="RequestID" ref="KeyValueTemplate">
    <String name="Key" value="invokeID" />
    <String name="Value" value="15" />
  </Block>
  <Block name="ConfirmedServiceRequest" ref="KeyValueTemplate">
    <String name="Key" value="comfirmedServiceRequest" />
    <String name="Value" value="write" />
  </Block>
  <String value=" (" token="true" />
  <Number name="requestTypeID" size="8" value="5" />
  <String value=")" token="true" />
</DataModel>
```

State Modeling in Peach Fuzzer involves defining states and transitions explicitly, making it even more difficult to accommodate complex protocol behaviors. Below is an example of a state model designed for IED Client-Server interaction in Peach:

```
<StateModel name="ClientServer" initialState="InitialState">
    <State name="InitialState">

            <!-- Peach will automatically connect to the remote host -->

            <!-- Send data -->
            <Action type="output">
                    <DataModel ref="confirmed-RequestPDU" />
            </Action>

            <!-- Receive response -->
            <Action type="input">
                    <DataModel ref="confirmed-ResponsePDU" />
            </Action>

            <!-- Send data -->
            <Action type="output">
                    <DataModel ref="confirmed-RequestPDU" />
            </Action>

            <!-- Receive response -->
            <Action type="input">
                    <DataModel ref="confirmed-ResponsePDU" />
            </Action>
    </State>
</StateModel>

<Test name="Default">
    <StateModel ref="ClientServer"/>
    <Publisher class="TcpClient">
        <Param name="Host" value="172.18.74.71" />
        <Param name="Port" value="102" />
    </Publisher>
</Test>
```

While Peach Fuzzer allows for monitoring and logging the received responses and device status, it does not use this information to enhance input mutation or generation efficiency. This is a fundamental limitation, as it lacks a feedback mechanism that can guide test case

generation dynamically based on observed responses.

Due to the complexity of MMS data modeling, the rigid state definitions, and the lack of a feedback mechanism, Peach Fuzzer is not a viable solution for effectively fuzzing IEC 61850 MMS-based systems. The manual effort required to model MMS packets and their hierarchical structure makes it impractical for large-scale fuzzing. Additionally, the inability of Peach to use response feedback in real-time leads to inefficient exploration in the fuzz testing.

3.3 AFL based libIEC61850 Fuzzer

In this study [32], the authors utilized the traditional American Fuzzy Lop (AFL) [33] to fuzz-test `libIEC61850` [34], an open-source implementation of the MMS protocol within the IEC 61850 standard, written in C. AFL, a widely used mutation based grey-box fuzzing tool, employs light-weight instrumentation to generate test cases and analyze code coverage effectively. By applying AFL to `libiec61850`, they successfully identified some vulnerabilities in the library. To assess the real-world impact of these findings, the researchers replayed all AFL-generated inputs on actual IEDs and observed that two of these inputs triggered crashes in the physical devices.

However, the MMS protocol implementation in commercial IEDs varies significantly, as vendors develop proprietary communication modules with differing configurations, capabilities, and parameters. These variations extend to data objects, data attributes, naming conventions, and typing schemes. Consequently, while AFL proved effective in identifying vulnerabilities in the open-source `libiec61850`, it may not be universally effective for fuzzing all IEDs. The generic test cases generated may not necessarily trigger failures in actual devices due to their structural and functional differences.

3.4 Applictaion of Snipuzz to IEDs

Snipuzz [16] is a black-box fuzzing tool initially designed for Internet of Things (IoT) devices. It achieves high code coverage and vulnerability detection through advanced response clustering, snippet-based request mutation, and iterative seed augmentation. Its application to IED fuzzing highlights its ability to overcome key challenges:

1. **Absence of Feedback Mechanisms:** Traditional black-box fuzzers lack insight into the internal workings of devices. Snipuzz circumvents this limitation by clustering device responses using similarity metrics, enabling the identification of promising areas for further mutations.

2. **Handling Randomized Device Responses:** IoT and IED devices often include randomized fields in their responses, such as timestamps or session tokens, which disrupt response analysis. Snipuzz addresses this issue by sending duplicate requests and analyzing response similarities to establish robust thresholds for response clustering.

3.4.1 Request - Response Inference

The process of request-response inference in Snipuzz follows a structured methodology to efficiently explore the behavior of an Intelligent Electronic Device (IED) under test. The steps involved in this process are outlined as follows:

Setup Initial Seed Requests Initially, valid MMS requests are selected using either documentation or available configuration information of the target IED. These seed requests form the basis for generating further test cases.

Change/Mutate in the Request Packets Seed request packets are mutated by selecting a mutation strategy and snippet to be mutated in a random way and sent to the device.

Categorize/Cluster the Responses Received The responses obtained from the IED are categorized based on a similarity score into different groups based on the observed behavior. This categorization helps in understanding the various response patterns and aids in guiding future test case generation by identifying interesting requests for further mutation.

Append Interesting Requests to the Seed If a response received belongs to a previously unexplored category, the corresponding request is added to the seed set. The mutation process is then repeated to further explore the new response category, aiming continuous expansion of test coverage.

3.4.2 Response Handler

Snipuzz employs a similarity score to compare responses by calculating the edit distance. The edit distance, also known as the Levenshtein distance [35], represents the minimum number of operations—insertions, deletions, or substitutions—required to transform one string into another.

Consider two response strings, r_1 and r_2. Their similarity score is defined as:

$$\text{Similarity Score}(r_1, r_2) = 1 - \frac{\text{edit\_distance}(r_1, r_2)}{\max(\text{len}(r_1), \text{len}(r_2))} \tag{3.1}$$

where $\text{edit\_distance}(r_1, r_2)$ is the number of edit operations required to transform r_1 into r_2, and $\max(\text{len}(r_1), \text{len}(r_2))$ represents the length of the longer response.

If the similarity score exceeds the threshold, then r_1 and r_2 are classified into the same response category. This threshold is determined by sending the seed request twice and calculating the similarity score of the two responses received, allowing the system to establish a baseline similarity score. Any response must meet or exceed this baseline score to be considered part of the same category, ensuring that minor variations due to randomness in the responses do not lead to incorrect categorizations.

3.4.3 Snippet Determination

Since Snipuzz operates without grammar knowledge of the protocol, it identifies logical components within the request, referred to as *snippets*, which can be mutated independently. The process of snippet determination consists of two key phases: **probing** and **hierarchical clustering**.

The **Probing phase** involves iteratively removing one character at a time from the request packet and sending the modified request to the device to observe the response. If the removal of consecutive bytes results in identical responses, those bytes are grouped together as a single snippet. This phase allows the identification of independent, logical components within the request structure.

The **Hierarchical clustering phase** [36] refines snippet determination by analyzing response similarity. Since responses can exhibit minor variations while still belonging to the same snippet. The adjacent clusters are merged together to contribute to a single snippet. These new snippets are added to the snippet list for further mutation. This clustering mechanism ensures that logically related segments are grouped together, enabling more targeted and efficient mutation during fuzzing.

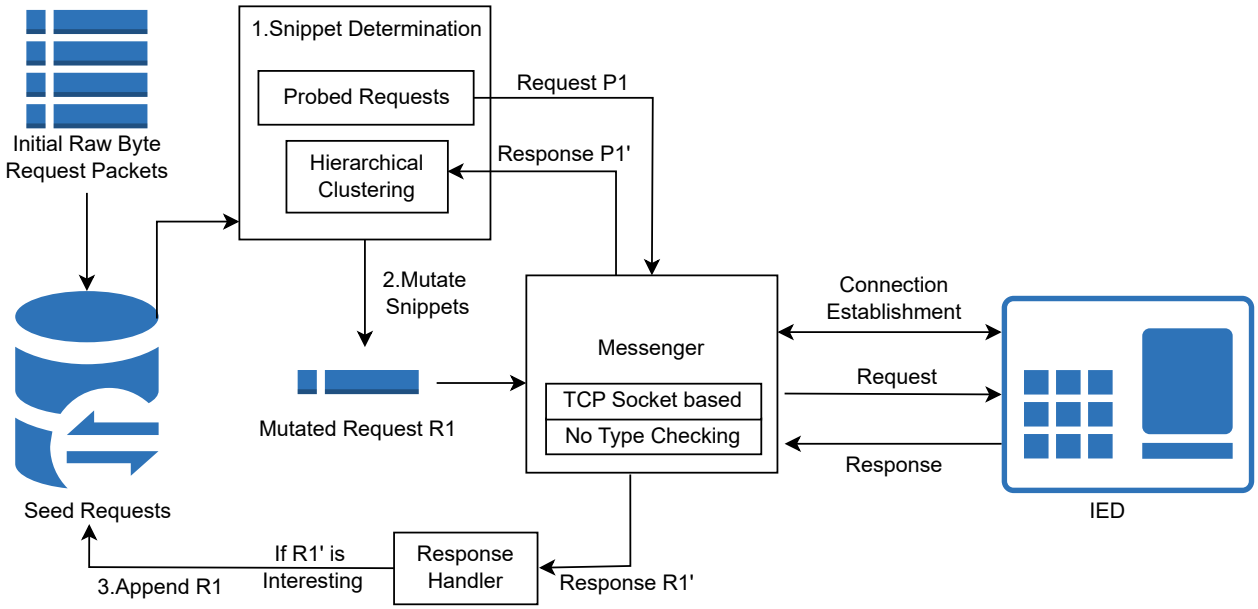


Figure 3.1: Snipuzz adapted to IEDs

The implementation setup of Snipuzz, as shown in Figure 3.1, requires the setup details mentioned in Sections 5.1 and 5.4, similar to the setup of IEDFuRL.

3.4.4 Characteristics of Snipuzz

Snipuzz demonstrates superior performance compared to state-of-the-art black-box fuzzers due to its incorporation of a feedback-driven fuzzing mechanism. This mechanism enhances code coverage by systematically exploring new execution paths, thereby maximizing the number of distinct response types triggered. As a result, Snipuzz has proven effective in uncovering a substantial number of zero-day vulnerabilities.

Therefore, We adapted the core principles of Snipuzz in our initial study to develop a fuzzing methodology specifically tailored for Intelligent Electronic Devices (IEDs). By integrating response clustering, snippet mutation, and iterative seed augmentation, this approach efficiently identified few vulnerabilities without requiring access to proprietary firmware or source code.

What was lacking in Snipuzz? It does not utilize the underlying protocol grammar, which could significantly improve the efficiency of the fuzzing process. By incorporating protocol-specific structures, the fuzzer could apply targeted mutations that align with the expected format of IEC 61850 messages, increasing the likelihood of discovering meaningful vulnerabilities. Instead, Snipuzz randomly selects fields and mutations, without considering the semantics of the protocol. This lack of structured mutation results in redundant test cases, reduced code

coverage, and inefficient exploration of the input space.

Additionally, Snipuzz lacks intelligence in mutation selection for specific fields. Instead of prioritizing critical fields that are more likely to cause system failures, it applies mutations arbitrarily, missing potential vulnerabilities that a more strategic approach could uncover. Another key limitation is the absence of sequence tracking, where Snipuzz fails to recognize patterns of successive mutations that could lead to bugs across related request types.

Moreover, Snipuzz lacks transferability, meaning that the knowledge gained from fuzzing one request type or device cannot be applied to another. This restricts its ability to generalize findings across different IED models or IEC 61850 implementations, requiring the fuzzer to restart the fuzzing process from scratch for each new device. Addressing these shortcomings would enhance the scalability, adaptability, and effectiveness of the fuzzing approach, making it more suitable for identifying security flaws in real-world power grid environments.

Chapter 4

Design of IEDFuRL

This chapter justifies the use of Reinforcement Learning (RL) for fuzz testing, highlighting its advantage over traditional black-box methods (Section 4.1). It introduces protocol-aware fuzzing, which leverages the IEC 61850 MMS protocol to prioritize test cases (Section 4.2). The RL framework is outlined, explaining how the agent interacts with its fuzzing environment(Section 4.4). Key components like Deep Q-Network (DQN), Q-learning, epsilon-greedy policy, and action masking are discussed to enhance learning efficiency (Section 4.5). Finally, the workflow and algorithm of IEDFuRL are presented (Section 4.5.4).

4.1 Why RL for Fuzz Testing?

Reinforcement Learning (RL) [37] is a dynamic machine learning paradigm where an agent interacts with its environment through a series of actions to achieve a specific objective. In this framework, the agent learns by trial and error, optimizing its behavior based on the rewards it receives. The goal of the RL agent is to maximize cumulative rewards, which are determined by the feedback received from the environment after each action. This learning mechanism enables the agent to identify patterns, improve decision-making, and refine its strategy over time, making RL highly effective for complex and dynamic environments.

Reinforcement Learning (RL) presents a promising approach to automated fuzz testing by enabling an agent to interact with its environment, execute a sequence of actions, and learn from feedback. The fundamental goal of the RL agent is to maximize cumulative rewards, which are assigned based on the effectiveness of its actions in identifying vulnerabilities. Unlike static or brute-force fuzzing techniques, RL leverages an adaptive learning mechanism that allows it to refine its strategy dynamically over time. Through repeated interactions, the agent learns which mutation strategies lead to impactful outcomes, ultimately increasing the efficiency of

the fuzzing process.

Applying RL to the fuzz testing of IEC 61850-based IEDs is particularly advantageous due to the highly dynamic state space of these devices. Traditional fuzzing techniques often rely on predefined heuristics or random mutations, making them inefficient when testing complex, structured protocols such as Manufacturing Message Specification (MMS). In contrast, an RL-based agent can systematically explore and adapt to the target device's behavior, ensuring a more intelligent and focused approach. By balancing exploration, where the agent discovers new potential vulnerabilities, and exploitation, where it applies previously learned successful mutations, RL-based fuzzing can improve the overall test coverage and effectiveness.

Furthermore, RL operates with minimal reliance on pre-existing training data, making it particularly suitable for black-box fuzzing scenarios where access to the device's source code or internal architecture is restricted. The agent learns in real time by observing the responses of the target IED to mutated inputs, allowing it to develop an understanding of the system's behavior without requiring explicit prior knowledge. This enables an adaptive fuzzing strategy that continuously evolves based on feedback, as opposed to static rule-based fuzzing approaches.

4.1.1 Input Space Complexity

Exhausting all possible input combinations would require 16^L mutations, where each hexadecimal bit in the input can take 16 possible values (from `0` to `f`), and L represents the maximum length of the MMS Protocol Data Unit (PDU). For the Siemens SIPROTEC 7SA522 6.1.1 device, the maximum PDU size is 65,536 hexadecimal bits. Therefore, the total number of possible inputs would be 16^{65536}, which approximates to infinite (NaN - Not a Number) possibilities. Exploring all these combinations is computationally impossible.

Although it is theoretically possible to discard already tested inputs to avoid redundancy, this is currently not implemented. This is because the IED behaves as a stateful system, where not just the individual inputs but also the sequence of inputs influences the system's behavior. Consequently, input filtering based on previously sent requests would require maintaining and comparing an extensive history, which would further increase computational complexity.

For this reason, we adopt action masking instead of input matching or masking. Action masking prevents the RL agent from selecting actions that have already led to crashes, guiding it toward unexplored and potentially impactful mutations. The RL agent is designed to learn this behavior through the reward mechanism, rather than by explicitly comparing every newly generated input against previously tested ones.

Additionally, prior research such as [38] has shown that it is extremely difficult to estimate exact time durations or convergence points for fuzzing campaigns, as different fuzzers exhibit

varying performance characteristics depending on their strategies and the nature of the system under test.

4.1.2 Key Advantages of RL Over Snipuzz

Comprehensive Mutation Strategy: Unlike Snipuzz, which only chooses a random mutation strategy at any point to generate new responses, our approach targets various fields and mutations, driving requests toward the fuzzing goal.

Experience-Based Learning: The RL agent builds on previous experience to learn the optimal sequence of mutations for specific types of MMS requests, enabling it to explore the most rewarding paths for uncovering vulnerabilities.

Implicit Sequence Tracking in Reinforcement Learning for Fuzzing: Reinforcement Learning (RL), by its very nature, offers the ability to implicitly track and learn mutation sequences as part of its decision-making process. Unlike conventional fuzzing tools such as Snipuzz, which evaluate mutations in isolation, RL agents inherently capture the temporal dependencies between successive actions through their learning mechanism. By interacting with the environment, the RL agent learns to associate specific sequences of mutations with desirable outcomes, such as triggering crashes or discovering new states. This implicit sequence tracking enables the agent to not only explore individual mutations but also to refine its strategy by learning the cumulative impact of a series of mutations. As a result, the RL-based framework achieves a more comprehensive exploration of the mutation space, systematically uncovering vulnerabilities that require complex and interdependent actions. This capability of RL provides a significant advantage in advancing the effectiveness of fuzz testing for Intelligent Electronic Devices (IEDs).

4.2 Protocol-Aware Fuzzing

We propose to enhance our fuzzing suite by incorporating protocol awareness, allowing it to understand specific fields in MMS requests and apply the most suitable mutation strategies to those fields. This approach prioritizes test cases based on protocol knowledge and increases the likelihood of generating valid inputs that traverse deeper execution paths in the code without being rejected due to parsing errors in the initial stages, thereby improving overall fuzzing effectiveness.

4.2.1 Advantages of Protocol-Aware Fuzzing

Efficient Field Identification: Our method eliminates the need to probe individual bits in seed requests by using protocol knowledge to group related bits of a field within a request

packet, improving overall efficiency.

Accurate Field Knowledge: The use of protocol knowledge also avoids the need for hierarchical clustering to identify bits of similar semantic meaning that contribute to a snippet or basically a field in the protocol.

4.3 Objective

The objective is to develop a reinforcement learning-based fuzzing agent that integrates protocol-aware fuzzing to maximize efficiency in vulnerability detection. Reinforcement learning (RL) enables the agent to dynamically learn optimal mutation sequences, adapting its strategy based on feedback to explore deeper execution paths. By leveraging protocol-aware fuzzing, the agent understands the structure of MMS packets and ensures that mutations align with the structure of MMS request packets, reducing parsing errors and increasing the likelihood of reaching critical code regions. This combination allows the agent to systematically identify vulnerabilities while improving code coverage and the overall effectiveness of fuzz testing.

4.4 Environment

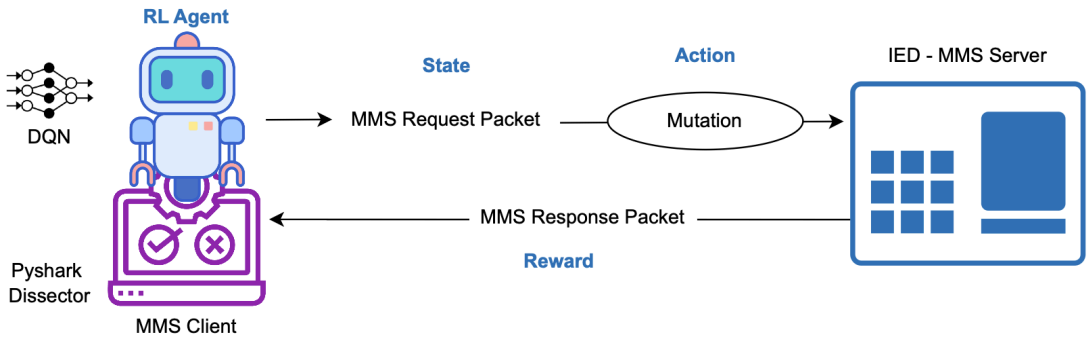


Figure 4.1: Reinforcement Learning Architecture

The Reinforcement Learning (RL) environment is a simulated or real-world system where the RL agent interacts and learns to achieve a goal by maximizing cumulative rewards. The environment consists of states, actions, rewards, transition dynamics, and the initial state distribution [37]. The RL environment of IEDFuRL is illustrated in Figure 4.1.

State Space The state represents the current status of the environment. In this setup, it is defined as the hexadecimal string representation of the MMS request packet. The initial state is a valid MMS request. After each action (mutation), the state updates to the mutated request

packet string.

Action Space Actions are transitions between states, where each action applies a mutation strategy to a selected location in the request packet. There are m mutable locations in the request packet, and n mutation strategies, such as Flipping bit/nibbles/all bits, Swapping bits/nibbles within the same location, Clearing values, Repeating values, Substituting interesting bit string values, then the cardinality of the action space is defined as $m \times n$, and the RL agent selects the action to perform based on the current state(MMS request packet) based on the probability to derive the highest return.

Reward Function The agent receives rewards based on its interaction with the environment, guiding its learning process.

The primary goal of IEDFuRL is to identify crashing scenarios and uncover vulnerabilities in the IED's communication module, ensuring a more effective and targeted fuzz testing approach. To accomplish this, the reward function plays a pivotal role in shaping the agent's learning process by reinforcing actions that contribute to achieving this objective.

Therefore, the rewards are structured as follows to ensure that the agent prioritizes discovering inputs that expose critical vulnerabilities and maximize code coverage in the IED's communication module:

1. A **crash of the EN100 communication module and malformed MMS responses** provides the highest reward, as it signifies the achievement of the agent's goal, and the episode terminates.

2. The agent receives a reward for identifying **new or interesting responses** that differ from previously observed outcomes. A response is classified as new or interesting based on its similarity score with previously explored response categories, as defined in Equation 3.1.

Conversely, the agent is not rewarded when it encounters **previously explored responses**, discouraging redundant exploration.

Transition Dynamics Transition dynamics describe how the environment changes from one state to another. In this case, the IED under test defines the dynamics, as it is configured with the IEC61850 MMS protocol to give predefined responses to the requests made. Therefore, it is different for various IEDs that adapt different versions of IEC61850 and their proprietary communication module that handles these requests.

Initial State Distribution The initial state distribution is the set of states where the agent begins its interaction. It consists of a distribution of various MMS request types, such as write, read, getVariableAccessAttributes, status, etc. We specify the initial valid seed requests on which fuzz testing has to be performed. This also has IED specific parameters has the variables and modules in each IED are different and they have unique configurations.

Terminal State A terminal state signifies the end of an episode, occurring when a predefined condition is met, indicating that the agent has achieved its goal. In IEDFuRL, an episode represents a complete fuzzing sequence, starting from an initial valid MMS request and progressing through a series of mutations applied by the reinforcement learning (RL) agent. The primary objective of the agent is to explore the input space of the Intelligent Electronic Device (IED) and identify vulnerabilities by triggering unexpected behaviors or crashes. The episode continues as long as the IED remains operational and responds to mutated requests. However, the episode terminates immediately when a critical event occurs, such as a system crash, an unresponsive state, or an observed misbehavior in the IED's response patterns. A crash is typically characterized by a failure to respond within a defined timeout window. Misbehavior, on the other hand, includes responses that deviate significantly from expected behavior, such as protocol violations, malformed responses, inconsistent error messages, or unexpected modifications to IED parameters. When such events are detected, the episode ends, and the agent receives the maximum reward, reinforcing its learning strategy to prioritize mutations that maximize the likelihood of discovering vulnerabilities. By structuring the RL environment in this manner, IEDFuRL ensures efficient and targeted exploration, effectively uncovering critical security flaws in IEC 61850-based communication modules.

4.4.1 Illustration of RL Environment with an example

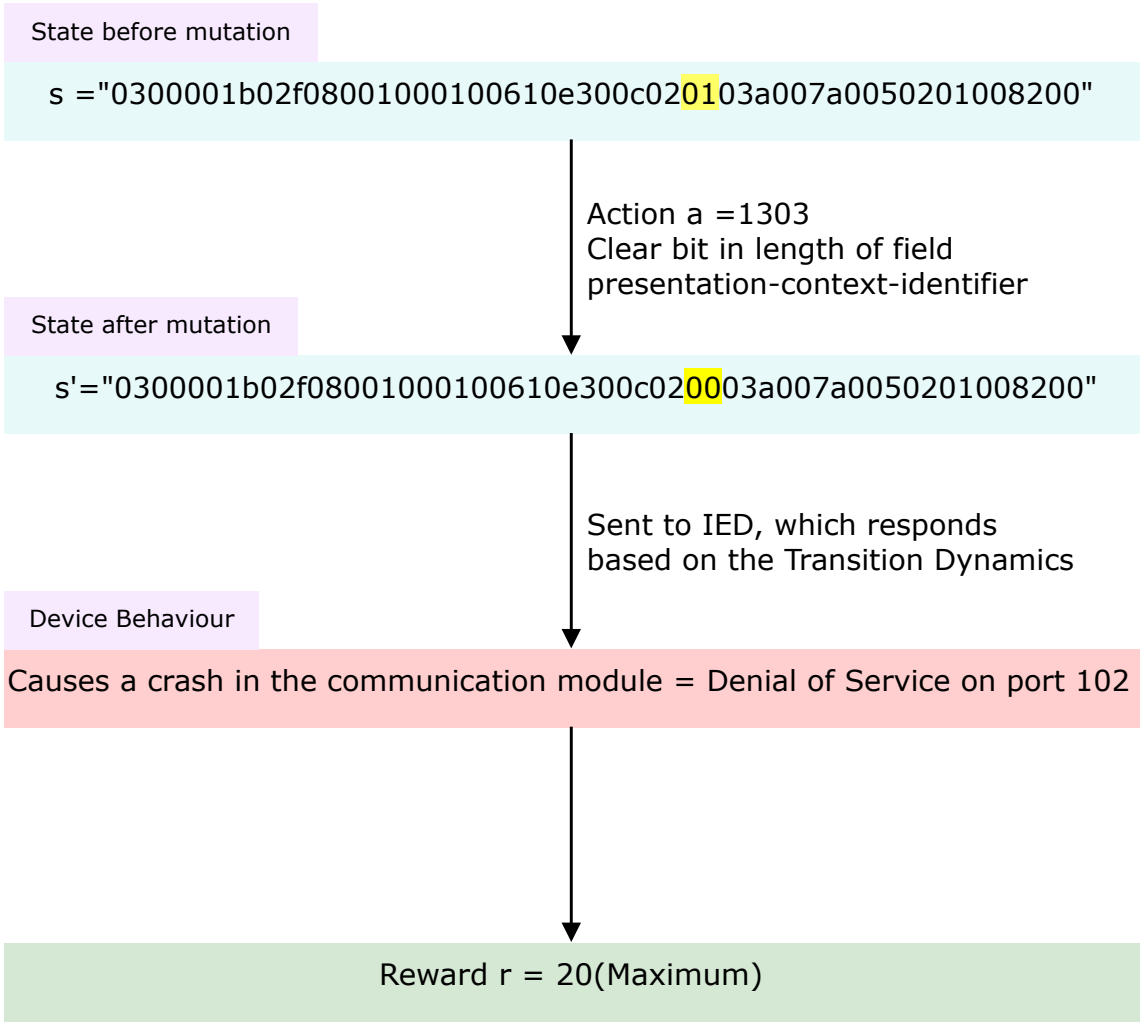


Figure 4.2: Working of the RL Environment

A RL agent is the decision maker in the RL environment, it can control the actions in the environment, observe the rewards and be trained to perform better. Figure 4.2 illustrates an example of how the reinforcement learning (RL) agent interacts with the fuzzing environment. The agent begins with an initial state s, representing a valid MMS request in hexadecimal format, starting from the TPKT layer, which was introduced in section 2.2. It then selects an optimal action a for that specific state, where a consists of both the **mutation strategy** and the **target field**. In this case, the agent applies a **clearing value** mutation to the **presentation-context-identifier length** field within the Presentation Layer of the packet. The mutated request s' is then transmitted to the IED.The Wireshark interpretations of the hexadecimal

bitstrings s and s' as MMS packets are shown in Figure 4.3 and Figure 4.4, respectively.

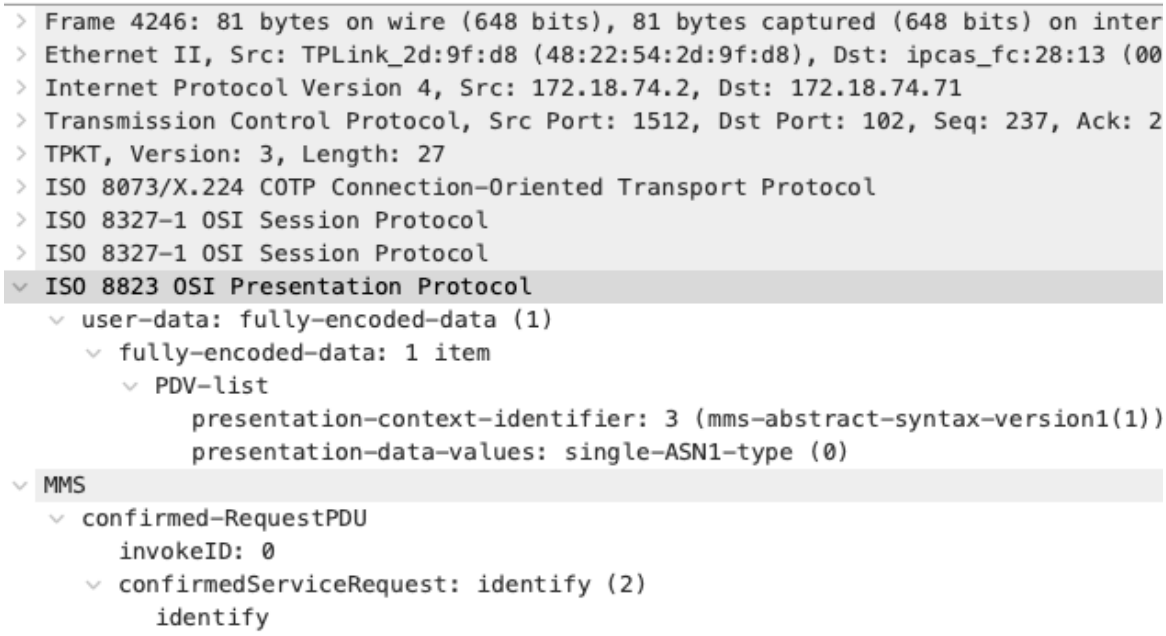


Figure 4.3: MMS packet of state s, as interpreted by wireshark

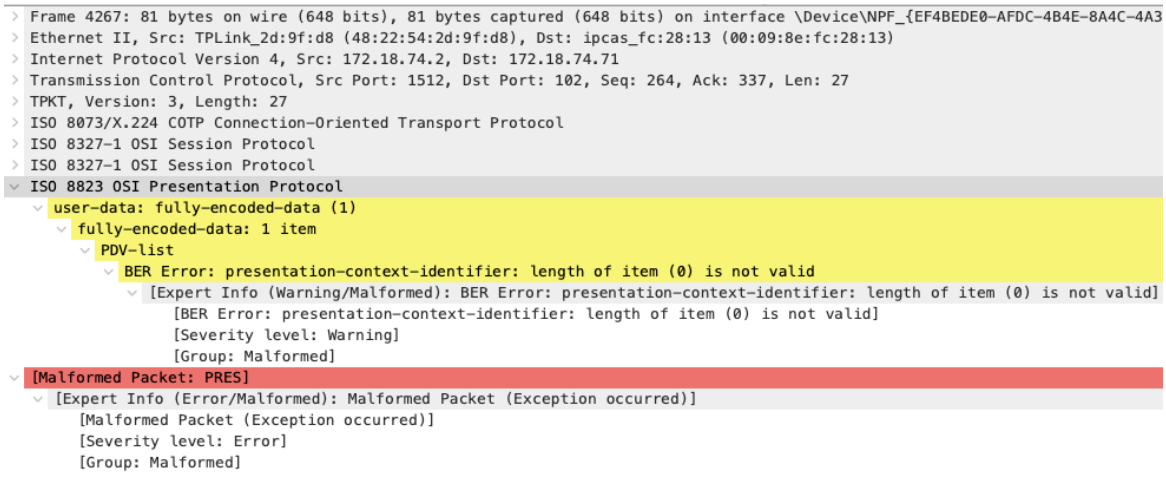


Figure 4.4: MMS packet of state s', as interpreted by wireshark

As a result, a crash is observed in the IED's communication module and it fails to respond, which leads to a **Denial of Service (DoS) on port 102**. Since the agent successfully triggered a crash, it receives the **maximum reward** of 20. This process represents a single transition in the RL framework, where the agent moves from state s to a new state s' by taking action a

and receiving reward r. This transition forms the learning tuple (s, a, r, s'), contributing to the agent's ability to refine its fuzzing strategy and improve vulnerability detection.

4.5 Reinforcement Learning Agent - DQN

Our RL agent operates in a high-dimensional state space, as the total number of states—comprising all possible request packet strings that any IED can support—is extremely large. It is also dependent on the IEC61850 version implemented and also the functional capabilities of the IED. Enumerating all possible states is impractical as the MMS request in the form of hexadecimal string, therefore each hexadecimal bit can take 16 possible values and the request can be of varying lengths.

We do not explicitly know the model of the environment, as the IED acts as part of our environment and transitions to the next state based on the IEC61850 protocol implementation in its communication module. Therefore, a model-free RL agent is suitable for this problem. There are three types of model-free RL agents: value-based, policy-based, and actor-critic agents.

4.5.1 Action Masking

Since the RL agent is designed to handle various types of MMS requests, an action mask is utilized to filter valid actions based on the fields present in the seed MMS request. Only actions applicable to these fields are considered valid.

Additionally, when a crash is triggered, the action responsible for the crash is masked to prioritize the discovery of new crashes instead of repeatedly causing the same one. The process of allowing only the valid actions and blocking other actions is called action masking [39]. Action masking is implemented as follows:

$$\text{mask}(action) = \begin{cases} 1 & \text{if } action \text{ is valid,} \\ 0 & \text{if } action \text{ is invalid.} \end{cases} \tag{4.1}$$

The mask information is stored as an array for all actions configured in the RL agent.

4.5.2 Selection of Reinforcement Learning Approach for IEDFuRL

Choosing the appropriate reinforcement learning (RL) approach is crucial for the efficiency and effectiveness of IEDFuRL. Given the high dimensional state space of IEC 61850 MMS requests and the discrete action space involved in mutation strategies, a well-suited RL method is required to optimize fuzzing performance. This subsection explores different RL approaches, evaluating their suitability for IEDFuRL.

Why Not a Policy-Based Approach? Our action space is discrete, and only a limited number of actions are valid at any given time, making a policy-based approach unsuitable. Policy-based methods, such as REINFORCE [40] or Proximal Policy Optimization (PPO) [41], sample actions from a probability distribution, which can be inefficient in environments where many actions are invalid.

Limitations of the Actor-Critic Approach The actor-critic [42] method introduces additional complexity by requiring two separate networks: an actor network responsible for selecting actions and a critic network for evaluating them. Training and tuning both networks simultaneously can be computationally expensive and unstable, especially in high-dimensional environments where state transitions are complex. Given the need for a structured and stable learning process, the added overhead of an actor-critic model is not justified for IEDFURL.

Why a Value-Based Approach? Since our objective is to approximate the expected reward for each mutation without learning an explicit policy, a value-based, model-free RL agent is the optimal choice. Q-learning [43] is a well-established value-based RL algorithm that estimates the Q-values for state-action pairs, allowing the agent to determine the best possible action for a given state. The Q-value is updated using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{4.2}$$

where,

- $Q(s,a)$ is the Q-value for state s and action a.
- α is the learning rate, controlling how much new information influences the existing value.
- r is the reward received for taking action a in state s.
- γ is the discount factor that determines the importance of future rewards.
- $\max_{a'} Q(s',a')$ represents the highest future reward possible from the next state s'.

However, given the high-dimensional state space in fuzz testing, traditional Q-learning is not computationally feasible, as storing and updating explicit Q-tables becomes impractical.

Deep Q-Network (DQN) with Action Masking for IEDFuRL To overcome the limitations of traditional Q-learning, we adopt Deep Q-Networks (DQN) [44], which approximate Q-values using a neural network. Instead of maintaining a Q-table, DQN estimates Q-values through function approximation, where the network's parameters θ are updated using the loss function:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} (y_i - Q(s, a; \theta))^2 \tag{4.3}$$

where,

- θ represents the neural network parameters.
- N is the batch size fetched from the training buffer.
- y_i is the target Q-value computed as:

$$y = \begin{cases} r & \text{for terminal } s', \text{ mask action } a \text{ using equations 4.1 \& 4.5} \\ r + \gamma \max_{a'} Q(s', a'; \theta') & \text{for non-terminal s'} \end{cases} \tag{4.4}$$

where $\theta^{'}$ refers to the target network parameters, which are periodically updated to stabilize learning.

DQN efficiently handles large state spaces by leveraging the neural network to estimate the Q-values of each action from a specific state and choosing the action with the highest Q-value. The epsilon-greedy exploration strategy further allows the agent to balance exploration and exploitation, making it well-suited for fuzz testing IEDs.

Moreover, action masking in DQN ensures that the agent only considers valid mutations, improving learning efficiency and avoiding redundant or invalid actions. This is achieved by multiplying Q-values with a binary mask:

$$Q(s, a) = Q(s, a) \cdot \text{mask}(a) \tag{4.5}$$

where mask(a) is the action mask, defined by the Equation 4.1

This structured approach significantly enhances the agent's ability to discover new vulnerabilities while reducing unnecessary state explorations. This ensures the RL agent effectively explores new vulnerabilities instead of taking the same sequence of actions to reach already identified vulnerabilities. Given these considerations, DQN with action masking is the approach we chose for IEDFuRL, enabling systematic and efficient fuzzing of IEC 61850-based devices.

The RL agent inherently prioritizes its exploration based on the defined reward function. As detailed in Section 4.5.1, we use action masking to block specific actions that have already resulted in a crash, thereby preventing redundant exploration. Our primary objective is to reach

crash-inducing inputs by navigating through new and interesting response categories. Therefore, explicitly segmenting the testing process into separate phases (e.g., first for crashes, then for malformed responses) does not necessarily steer the fuzzing experiment more effectively toward discovering crashes. It is expected that a greater number of new responses and comparatively fewer crashes will be encountered as the agent learns.

4.5.3 Epsilon-Greedy Policy in Deep Q-Networks

The epsilon-greedy policy [44] is used by the DQN agent in IEDFuRL to balance exploration (trying new actions) and exploitation (choosing actions that yield the highest rewards). Instead of always choosing the action with the highest estimated reward, which could cause the agent to get stuck in a local suboptimal state, the epsilon-greedy approach introduces a probability ϵ for selecting a random action. Initially, a high ϵ value encourages exploration to discover new vulnerabilities, while over time as training progresses, ϵ is decayed to prioritize exploitation of optimal mutations. This ensures that IEDFuRL efficiently traverses new execution paths in MMS request fuzzing while refining effective mutation strategies for maximizing vulnerability detection.

$$a = \begin{cases} \text{random action} & \text{with probability } \epsilon, \\ \arg\max_a Q(s, a; \theta) & \text{with probability } 1 - \epsilon \end{cases} \tag{4.6}$$

The epsilon-greedy policy enables the DQN agent in IEDFuRL to balance exploration and exploitation when selecting mutation strategies.

4.5.4 Training and Evaluation Algorithm

The training and evaluation process in **IEDFuRL** is adapted from [44] and presented as an algorithm 1 consisting of three phases: (1) **Collection Phase**, where the agent interacts with the environment by applying mutations to MMS request packets using an *epsilon-greedy policy*, storing state transitions and rewards in a training buffer; (2) **Training Phase**, where the agent samples experiences, computes target Q-values, and updates network weights via gradient descent to minimize loss; and (3) **Evaluation Phase**, where the trained agent applies learned mutation strategies, observes cumulative rewards, and assesses fuzzing effectiveness.

.

Input: Initial state s_0 (valid MMS request), training buffer $\mathcal{B}$ with sample batch size N, discount factor γ, learning rate α, exploration rate ϵ

Output: Trained DQN RL Agent

for *training_step t=1* **to** T **do**

Collection Phase: Gathering Experience

for *mutation_step m=1* **to** k **do**

Select action a using the epsilon greedy policy 4.6.

Execute action a, observe reward r, and transition to the next state s';

Store experience tuple (s, a, r, s') in $\mathcal{B}$;

end

Training Phase: Improving the DQN

Sample a mini-batch of experiences (s, a, r, s') from $\mathcal{B}$;

Compute the target Q-value for all actions according to equation 4.4;

Compute loss using equation 4.3;

Update weights θ using gradient descent to minimize the loss;

Evaluation Phase: Fuzzing

for *mutation_step m=1* **to** k **do**

Select action a using the learned agent policy;

Execute a, observe reward r_m, and transition to the next state s';

end

Calculate the cumulative reward for the evaluation episode:

$$R_t = \sum_{m=1}^{k} \gamma^{m-1} r_m \tag{4.7}$$

end

return *Trained DQN agent with optimized weights* θ

Algorithm 1: DQN-Based RL

Chapter 5

Implementation of IEDFuRL

The implementation of the IEDFuRL integrates robust mechanisms to perform automated fuzz testing on Intelligent Electronic Devices (IEDs). It is designed as an MMS client, communicating with target IEDs using the IEC 61850 MMS protocol over TCP. This section provides an in-depth discussion of its setup, technology stack, ensuring a comprehensive understanding of the fuzzer's implementation.

5.1 Communication Framework of IEDFuRL

The core objective of IEDFuRL is to perform fuzz testing on the communication module of the Intelligent Electronic Device (IED) by interacting with it through its designated system interface. The IED's communication capabilities are primarily determined by its interface and communication module, which establish seamless integration with power grid automation and control systems.

Interface: Serial System Interface The IED is equipped with a Serial System Interface [45] that has a unique IPv4 address, serving as the primary communication medium with the control center. This interface facilitates the exchange of critical operational data between the IED and the supervisory control systems, enabling real-time monitoring, control, and automation of power system functions. Given its role as the main entry point for command execution, this interface is a crucial medium for fuzz testing with IEDFuRL.

Communication Module IEDs that support the IEC 61850 standard are typically equipped with an Ethernet communication module, facilitating seamless interoperability across devices from different vendors. The Ethernet communication module enables the IED to exchange MMS requests and responses, making it a key component for fuzz testing. The Siemens test device (Section 6.1.1) includes a communication module called EN100 [45], which integrates

the IED into 100 Mbps Ethernet communication networks, enabling efficient interaction with process control and automation systems. The Hitachi Energy RET670 device (Section 6.1.2) has its own proprietary Ethernet communication module.

In the IEDFuRL framework, the communication module is subjected to targeted fuzzing by generating and injecting malformed or mutated MMS request packets. The response packets received from the IED serve as feedback for analyzing protocol handling, detecting deviations, and identifying potential vulnerabilities. By systematically fuzz testing the EN 100 module via the Serial System Interface, IEDFuRL aims to uncover security flaws that could be exploited in real-world attack scenarios, ensuring the resilience and reliability of IED communications.

To enable communication between the IED server and the IEDFuRL client, an Ethernet switch with standard Layer 2 packet switching capabilities is used as an intermediary network device. The switch is equipped with LC-type SFP (Small Form-Factor Pluggable) transceivers, allowing high-speed fiber-optic connectivity to the IED. The IEDFuRL client, running on a PC, connects to the switch via an RJ45 port to establish a wired connection. Alternatively, the switch can be replaced with an SFP-to-Ethernet media converter to accommodate the RJ45 port on the client device.

For successful communication, the IPv4 address of the Serial System Interface in the IED and subnet mask of the PC running the IEDFuRL client must be configured to ensure network compatibility with the IED server. Both the client and server must reside within the same subnet, and the default gateway configured on the switch must be aligned with the corresponding subnet address. This setup ensures seamless packet transmission between the fuzzing framework and the target IED, enabling effective injection of MMS request mutations and real-time response monitoring.

The IEDFuRL establishes communication with the target IED using a TCP socket connection, adhering to the IEC 61850 MMS protocol. Each IED is associated with a unique system port IP address, typically accessible via port 102, which is reserved for MMS communication. The fuzzer initializes a connection and generates MMS requests to be sent to this port. Upon receiving these requests, the IED is expected to respond synchronously with predefined responses according to its configuration.

To test any request type, the process begins with establishing a TCP connection by sending a `SYN` packet to open a socket on port 102 of the IED's IP address. This is followed by setting up a COTP session by sending a `Connect Request` to the device and receiving a `Connect Confirm` COTP packet. Once the transport connection is established, communication is initiated by transmitting an MMS `initiate-Request` Protocol Data Unit (PDU) and awaiting an MMS `initiate-ResponsePDU`. After completing this handshake to establish MMS communication

between the client and the IED, the actual request being fuzzed is sent to the IED. However, when fuzzing the MMS`initiate-RequestPDU`, only the TCP connection and COTP session are established before sending the mutated packet.

The responses are analyzed to detect anomalies. If the IED provides unexpected outputs, fails to respond, or exhibits functional failures, it suggests potential vulnerabilities in its request-handling mechanisms.

To manage the complex state space and the structured nature of MMS requests, the fuzzer leverages reinforcement learning and packet analysis tools, as described in the subsequent sections.

5.2 Request and Response Handler - Pyshark

Effective fuzzing of Intelligent Electronic Devices (IEDs) requires an in-depth understanding of the MMS protocol structure and its encoding. To facilitate this, IEDFuRL integrates Pyshark [46], a Python wrapper for Wireshark, to parse and analyze MMS request and response packets in real time. This section details how Pyshark aids in structuring the fuzzing process by decoding, extracting, and manipulating protocol-specific fields.

5.2.1 Pyshark Library for MMS Packet Analysis

IEDFuRL is designed as a protocol-aware fuzzing tool, requiring a comprehensive understanding of the IEC 61850 MMS protocol. To achieve this, Pyshark is employed due to its built-in ASN.1-based MMS dissector, which interprets all protocol fields following the **Tag-Length-Value (TLV)** structure defined by **Basic Encoding Rules (BER)** (Section 2.1). This enables the fuzzer to systematically analyze request and response packets, facilitating the efficient mutation of fields within MMS request packets. Any hexadecimal representation of an MMS request can be transmitted to the IED via a socket connection, captured by Pyshark, and subsequently analyzed to extract relevant field-value pairs. These extracted values are crucial for identifying mutation targets within the MMS request structure. Pyshark also provides a variety of functions for manipulating and retrieving packet information, making it a powerful tool for packet inspection.

By leveraging Pyshark, the reinforcement learning agent within IEDFuRL can monitor interactions between the client and the IED. The dissected packet fields guide the agent in selecting optimal mutation strategies, improving the efficiency of fuzzing by focusing on high-impact request modifications.

The hexadecimal bit-string response received from the device is interpreted by Pyshark and presented as key-value pairs to IEDFuRL, enabling it to compute the similarity score of the

response with previously explored responses using Equation 3.1. This process helps the RL agent within IEDFuRL allocate rewards for specific mutations applied to the corresponding request packet. If a packet does not conform to the correct BER encoding, Pyshark flags it as malformed, providing immediate feedback to the fuzzing framework. This feedback is used to reward the RL agent in IEDFuRL whenever the IED responds with a malformed MMS packet.

Figure 5.2 presents an example of Pyshark's dissection of an MMS packet, which is originally captured and displayed in Wireshark, as shown in Figure 5.1. Pyshark processes the packet by parsing all its fields into key-value pairs, as illustrated in Figure 5.2a. Additionally, it extracts each field's raw hexadecimal value and its position within the packet, represented by the `raw_value` and `pos` fields, as shown in Figure 5.2b.

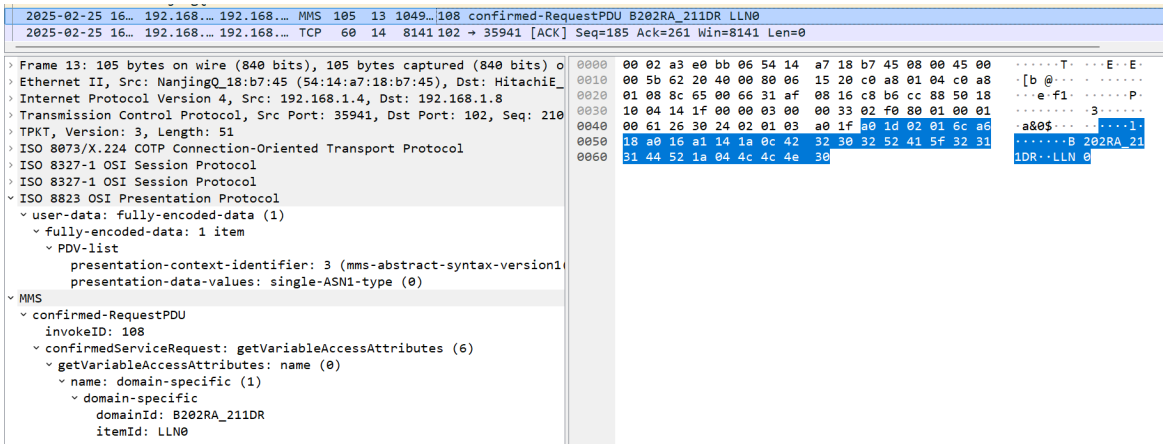
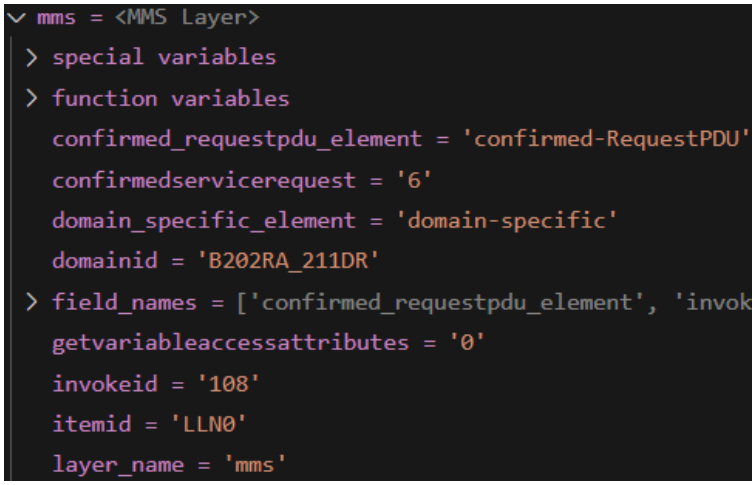
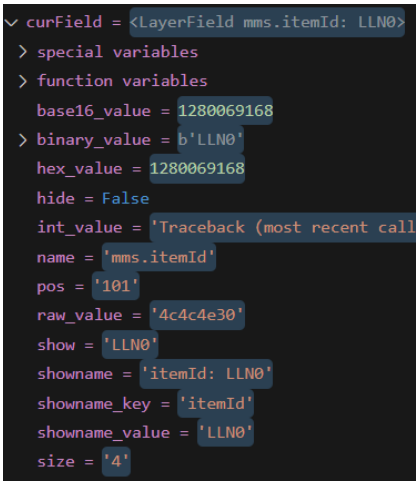


Figure 5.1: MMS request packet.



(a) Pyshark dissection of the MMS request



(b) Pyshark field extraction

Figure 5.2: Pyshark dissection of the above MMS request packet

The ability to interpret MMS requests and responses in real-time ensures that the fuzzer not only generates meaningful test cases but also adapts to different IED implementations based on observed response behaviors. Thus, Pyshark serves as the interface for handling request and response packets in IEDFuRL, enabling structured, informed, and efficient fuzz testing of IEC 61850-based IEDs.

5.2.2 Configuring Field Names for MMS Request Mutation

For IEDFuRL to effectively perform protocol-aware fuzzing, it must be aware of the fields that can be mutated in an MMS request packet. The RL agent selects one of these fields from a given seed request—an initial valid MMS request—as part of its mutation actions. The DQN in the RL agent determines the optimal action to perform from a given state, requiring it to be configured with all possible fields that can undergo mutations using different mutation strategies.

While Pyshark can identify the fields present in any request, it does not provide a predefined compilation of fields that may appear in MMS packets. To equip IEDFuRL with knowledge of all possible fields that can be considered for mutation, a manually curated list of mutable field names spanning various protocol layers, starting from the Presentation layer, is compiled in a Python list and provided as input to IEDFuRL. This configuration enables IEDFuRL to equip the RL agent with mutation options for all these fields, identify their presence in a given packet using Pyshark analysis (as discussed in Section 5.2.1), and apply mutations accordingly. Currently, 116 fields have been configured as mutable fields in the request packet. A portion of this list is shown below.

```
mutableFields=[
    #Presentation Layer fields
    "mode-selector",
    "calling-presentation-selector",
    "called-presentation-selector",
    "presentation-context-identifier",
    "abstract-syntax-name",
    "transfer-syntax-name",
    #Association Control Service Layer fields
    "aSO-context-name",
    "called-AP-title",
    "called-AE-qualifier",
    "called-AP-invocation-identifier",
    "called-AE-invocation-identifier",
    "calling-AP-title",
    "calling-AE-qualifier",
```

```
    "user-information",
    #Manufacturing Message Specification Layer fields
    "invokeID",
    "listOfModifier",
    "confirmedServiceRequest",
    "cs-request-detail",
    "attach-To-Event-Condition",
    "domainId",
    "itemId",
    "localDetailCalling",
    "proposedMaxServOutstandingCalling",
    "proposedMaxServOutstandingCalled",
    "proposedDataStructureNestingLevel",
    "proposedVersionNumber",
    "proposedParameterCBB",
    "servicesSupportedCalling",
    "objectClass",
    .
    .
    .
    ]
```

Alternatively, these fields can be extracted from Wireshark's dissection modules [20] by parsing the `.asn` files to retrieve the field names that may be present in any MMS packet. The mutable fields can be extracted from Wireshark's dissection modules [20], which points to all the MMS fields. Fields from the ASN.1 Presentation (PRES) and Association Control Service (ACSE) layers can also be configured from the same repository. For now, we have included only a few fields from the PRES and ACSE layers, as those were relevant to the devices we worked with, whereas all fields from the MMS layer have been included.

5.3 Characteristics of Neural Network used in DQN of IEDFuRL - TFAgents

5.3.1 Deep Q-Network Algorithm Implemented with TensorFlow

To navigate the high-dimensional state space of MMS request mutations, IEDFuRL employs a Deep Q-Network (DQN) algorithm implemented using TensorFlow. TensorFlow has a library called TF Agents which provides a flexible and efficient framework for designing and training reinforcement learning agents, with well tested modular components that can be customized for specific application, it also enables fast code iterations [47]. We use Gymnasium [48], an

open source python library for developing and comparing reinforcement learning algorithms that provides API for modeling our RL environment and facilitating communication between the environment and learning algorithms. TF agents have a gymnasium wrapper that acts as an interface between our environment modelled with gymnasium and algorithm and RL agent designed with TF agents

5.3.2 Neural Network Architecture of the DQN Agent

The Deep Q-Network (DQN) agent in IEDFuRL utilizes a neural network called Q-network to approximate Q-values for each action in a given state. The Q-network is designed to process MMS request packets, represented as hexadecimal bitstrings, and learn meaningful patterns to guide the fuzzing process. The network consists of multiple layers customized using TensorFlow's Keras API [49], each serving a specific role in transforming the raw MMS request into a structured representation that can be effectively used for decision-making. The architecture comprises the following layers:

1. Input Layer – Text Vectorization The Input layer is responsible for converting the raw MMS request bitstring into a numerical representation that the neural network can process. Since the state space consists of hexadecimal strings representing MMS request packets, this layer ensures efficient tokenization and transformation of input data. The TextVectorization layer is responsible for transforming raw hexadecimal strings into sequences of integer tokens, enabling efficient processing within the neural network. It performs normalization and tokenization to convert the input into a structured format. The vocabulary size, defined by `max_tokens`, specifies the maximum number of unique tokens that can be recognized. Additionally, the sequence length is controlled by `output_sequence_length`, ensuring that all input sequences maintain a consistent length through padding or truncation. The vocabulary consists of valid hexadecimal values, each mapped to a specific token index, allowing for structured and meaningful representation of MMS request packets.

2. Embedding Layer – Token Representation Once the input is tokenized, it is passed through the embedding layer, which converts each token into a dense vector representation of dimension `embed_dim`. This embedding allows the model to capture semantic relationships between different protocol fields, ensuring that tokens with similar functionalities are mapped closer in the learned vector space.

3. Positional Embedding Layer – Context Awareness In MMS request packets, the same hexadecimal token may have different meanings depending on its position within the packet. The Positional Embedding Layer incorporates positional information into the token

embeddings, ensuring that the network can differentiate between protocol fields based on their location within the structured message.

4. Dense Layers – Feature Extraction A series of **fully connected (dense) layers** process the extracted features to refine the learned representations. These layers progressively transform the input feature vectors into a more compact and informative representation, capturing complex relationships between protocol fields and their effects on system behavior.

5. Q-Values Layer – Action Selection The final layer of the network is the Q-values layer, which produces a Q-value for each possible action in the action space. These values represent the estimated future rewards for taking a given action in a specific state, allowing the RL agent to choose the most effective mutation strategy.

The final Q-network model is constructed by sequentially stacking all the layers above 5 layers. This architecture ensures that the DQN agent in IEDFuRL can efficiently process MMS request packets, extract meaningful representations, and compute Q-values to determine optimal mutation actions. The combination of **text vectorization, embedding, positional embedding, and dense layers** allows the model to intelligently explore and exploit vulnerabilities in IED communication protocols.

5.3.3 Low Level Details on DQN Implementation

To realize the Deep Q-Network (DQN) architecture described in Section 4.5, IEDFuRL implements its neural network using TensorFlow and Keras. This subsection provides the concrete layer-wise construction of the Q-network used by the RL agent. The corresponding Python implementation is shown below.

```
input_layer = tf.keras.layers.TextVectorization(max_tokens,
                output_mode='int',
                output_sequence_length,
                vocabulary)

embedding_layer = tf.keras.layers.Embedding(input_dim=max_tokens,
                    output_dim=embed_dim)

pos_embedding_layer = PositionalEmbedding(max_len, embed_dim)

dense_layers = []
for num_units in dense_layer_parameters:
    dense_layers.append(tf.keras.layers.Dense(num_units, activation_func='
   relu'))
    dense_layers.append(tf.keras.layers.Dropout(droup_out_value))
```

```
15
16 q_values_layer = tf.keras.layers.Dense(
17     num_actions, activation_func, kernel_initializer, bias_initializer)
18
19 q_net = sequential.Sequential([
20     input_layer,
21     embedding_layer,
22     pos_embedding_layer
23 ] + dense_layers + [q_values_layer])
24
25 optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
26
27 td_errors_loss_fn=common.element_wise_squared_loss
```

5.3.4 Hyperparameter Tuning for Learning the Weights in DQN

Optimizing the hyperparameters in the Deep Q-Network (DQN) agent was essential to ensuring stable training and effective learning. Several key parameters, including the learning rate, mutation sequence count, discount factor, and reward structure, were fine-tuned through iterative experimentation.

Learning Rate Optimization Initially, the learning rate was set to e^{-3}, but this caused the loss function to increase significantly after each episode. Reducing the learning rate to e^{-4} stabilized the loss, ensuring better convergence. However, further decreasing it to e^{-5} resulted in slower learning, reducing the agent's efficiency in adapting and obtaining results. Therefore, e^{-4} was selected as the optimal learning rate.

Mutation Sequence Count The number of sequential mutations applied to the seed MMS hexadecimal bitstring was initially set to 5. However, this limited the discovery of vulnerabilities requiring multi-byte mutations. Increasing it to 10 improved the diversity of responses found, yet the number of unique responses remained less. Further increasing it to 15 balanced the trade-off between uncovering new responses and avoiding prolonged exploration of invalid states that produced redundant responses. Setting it to 20 led to prolonged episodes where the agent remained in the same invalid state without discovering additional vulnerabilities. Therefore, 15 was determined to be the optimal value.

Discount Factor The discount factor γ was set to 1, ensuring that new responses received at any point were assigned equal reward values. This approach allowed the agent to equally weigh all newly discovered response categories.

Reward Structure The reward values played a crucial role in guiding the agent's learning process. A discrete reward system is particularly well-suited for protocol fuzzing tasks like those in IEDFuRL because the goals of the fuzzing process—such as triggering a crash, discovering a malformed response, or identifying a new response category—are inherently categorical. Assigning fixed reward values to these outcomes (e.g., +20 for a crash/malformed response, +1 for a new response, 0 for a repeated response) makes the learning process more stable, interpretable, and efficient. Unlike continuous rewards, which can be noisy or ambiguous in such environments, discrete rewards provide clear feedback that helps the RL agent focus on impactful mutations. This approach also aligns well with the sparse-reward nature of fuzzing, where most inputs do not produce interesting behaviour, and only specific cases should be strongly reinforced. As a result, a discrete reward system ensures the agent remains goal-driven and prioritizes meaningful vulnerabilities during its exploration. To explore the impact of different reward structures, the following configurations were also tested:

Table 5.1: Reward Structure - Optimal

Response Type	**Reward Value**
Already Explored Response	0
New Response	1
Crash Triggered	20
Malformed Response	20

Table 5.2: Reward Structure - High Difference

Response Type	**Reward Value**
Already Explored Response	-1
New Response	15
Crash Triggered	200
Malformed Response	200

Table 5.3: Reward Structure - Low Difference

Response Type	Reward Value
Already Explored Response	-1
New Response	0.5
Crash Triggered	1
Malformed Response	1

The reward values in Table 5.2 destabilized training, as the large difference in reward values led to highly fluctuating loss values, making it difficult for the agent to learn effectively. On the other hand, the values in Table 5.3 assigned lower weights to crash and malformed responses, reducing the agent's incentive to prioritize actions leading to critical failure states. This resulted in an inefficient fuzzing strategy.

Ultimately, the reward structure in Table 5.1 was selected. It ensured that crash and malformed responses were given higher priority while keeping the reward difference moderate to maintain training stability and prevent excessive fluctuations in the loss function.

5.4 Implementation of Network Monitor

To effectively detect crashes or failures in the communication module, IEDFuRL continuously monitors the status of the device throughout the fuzzing process. A *ping alive* check is performed on the IP address of the communication module within the serial system interface described in Section 5.1. This check runs in the background while mutated inputs are sent to the IED, allowing the system to observe whether the communication module remains operational, experiences temporary failures before recovering, or completely denies service until a manual reboot is required. Thus, the *ping alive* check provides a continuous assessment of the communication module's availability.

This network monitoring mechanism enables real-time detection of failures triggered by fuzz testing. By tracking the communication module's response behavior, the system can assess whether a particular mutation on a request packet triggers a crash. This information is integrated into the reinforcement learning framework to appropriately reward the RL agent when it successfully induces a failure or a crash in the device.

Chapter 6

Bugs Found

The experimental evaluation of IEDFuRL uncovered critical vulnerabilities in the tested IED devices, demonstrating its effectiveness in automated fuzz testing. IEDFuRL was executed on two real-time IEDs, as detailed in Section 6.1, leading to the identification of 12 distinct bugs. These vulnerabilities fall into three primary categories: denial of service, information disclosure, and data boundary violations. This chapter presents a comprehensive analysis of the discovered bugs, including their triggering inputs, observed effects, and potential implications.

6.1 Test Devices

To evaluate IEDFuRL in a real-time environment, we tested two Intelligent Electronic Devices (IEDs) from different vendors, each with distinct specifications, firmware versions, and IEC 61850 implementations. These devices operated independently, without connections to any external physical or control systems. Despite the absence of external components, IEDFuRL successfully uncovered multiple diverse bugs, demonstrating its effectiveness in detecting flaws.

According to the IEC 61850 standards, the IEDs are expected to respond with one of the negative responses described in Section 2.3.1, indicating that such erroneous packets are properly handled. However, if the response matches any of the conditions defined in the reward function subsection of Section 4.4, it is considered a buggy scenario—representing a successful vulnerability-triggering incident discovered by the fuzzer.

6.1.1 SIEMENS SIPROTEC 7SA522

The Siemens SIPROTEC 7SA522, shown in Figure 6.1, is a distance protection device equipped with a powerful microprocessor system designed for power line protection [45]. It enables fast and selective fault clearance on transmission cables and overhead lines, facilitating quick fault detection, isolation, and resolution. The device processes all its functions digitally, from

receiving measured values to executing control commands for circuit breakers.

The SIPROTEC 7SA522 is equipped with four current and four voltage inputs. It features a system interface that connects to SCADA control centers for status reporting and remote control operations. As detailed in Section 5.1, IEDFuRL was specifically designed to conduct fuzz testing on this interface to assess its resilience. The primary function of the device is to determine the distance to a fault and assist in fault clearance. Additionally, it supports complex fault identification through multiple distance protection measurement elements. Beyond distance protection, the device is also used for overcurrent protection, voltage protection, frequency protection, and circuit breaker failure protection.

Device 1 **SIEMENS SIPROTEC 7SA522**

Firmware Version V04.76.02

IEC61850 Version V04.29.01

Figure 6.1: Siemens Device Tested

6.1.2 Hitachi Energy RET670

The Hitachi Energy RET670 [50], shown in Figure 6.2, is designed for the reliable protection and control of power transformers and shunt reactors. Power transformers are critical components used to transfer electrical energy at varying voltage levels [51], while shunt reactors are employed in high-energy transmission systems to absorb reactive power, thereby enhancing system efficiency [52].

The RET670 offers a pre-configured protection solution for all types of transformers, It is optimized for power transmission applications. It features fast differential protection, sensitive earth-fault protection, distance protection, circuit breaker failure protection, current protection, frequency protection, voltage protection, and impedance protection. Additionally, the

device includes built-in disturbance and event recorders, ensuring comprehensive monitoring and analysis of power system events.

Device 2	**HITACHI ENERGY RET670**
Firmware Version	RET670ver2.2.6
IEC61850 Version	IEC61850-8-1:1 Protocol Edition 2.1

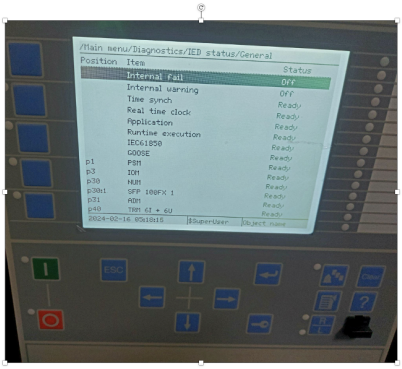


Figure 6.2: Hitachi Energy Device Tested

6.2 Overview of Identified Vulnerabilities and Bugs

The IEDFuRL successfully identified nine distinct crash scenarios, one information disclosure event and one data boundary issue during the testing phase. These findings underscore the tool's capability to systematically uncover vulnerabilities in IEC 61850-based IEDs. Table 6.1 provides a comprehensive overview of the identified vulnerabilities, highlighting the specific protocol layers, crafted fields, and their resulting effects. The mutations were applied across diverse protocol layers, including PRES, MMS, and ACSE of the MMS packet, and involved different request types, further demonstrating the adaptability and effectiveness of the fuzzing framework. By mutating single or multiple fields in various MMS request packets, IEDFuRL triggered a range of issues, including denial-of-service (DoS) conditions, information disclosure flaws and data boundary issue. The detected vulnerabilities span two distinct test devices, namely Siemens 7SA522 and HitachiEnergy RET670, showcasing the robustness and versatility of the fuzzing framework.

Table 6.1: Identified Bugs in IEC 61850-based IEDs

Bug No.	**Layer**	**Fields Crafted**	**Effect**	**Devices Affected**
1	PRES	`presentation-context-identifier` length = 00	DoS	Siemens 7SA522
2	MMS	`invokeID` length = 00	DoS	Siemens 7SA522
3	MMS	length = 81	DoS length=Next Byte of length	Siemens 7SA522
4	MMS	`localDetailCalling` length = 00 in `initiate` request	DoS	Siemens 7SA522
5	MMS	`proposedVersionNumber` length = 00 in `initiate` request	DoS	Siemens 7SA522
6	MMS	`proposedMaxServOutstandingCalled` length = 00 in `initiate` request	DoS	Siemens 7SA522
7	MMS	`proposedDataStructureNestingLevel` length = 00 in `initiate` request	DoS	Siemens 7SA522
8	MMS	`invokeID` length = 84, following 5 bytes: `ff 7c __ __ 00 (to) ff ff __ __ 00` in `write` request	DoS	Siemens 7SA522
9	MMS	`objectClass` length = 00 in `getNameList` request	DoS	Siemens 7SA522
10	MMS	`invokeID` length = 84 following 5 bytes: `ff 89 __ __ 00 (to) ff ff __ __ 00` in `fileDirectory` request	DoS	Siemens 7SA522
11	ACSE	`called-AP-title, calling-AP-title` with invalid `ap-title-form2` in `initiate` request	Information Disclosure	HitachiEnergy RET670, Siemens 7SA522
12	MMS	`proposedMaxServOutstandingCalled` with negative value in `initiate` request	Data Boundary Violation	Siemens 7SA522

6.3 Denial of Service

This section outlines the bugs that led to a Denial of Service in the communication module of the tested IED, providing a comprehensive analysis of their impact.

6.3.1 Bug 1

Seed Input - Initial Valid MMS Request Packet Fuzz testing begins with a valid MMS request packet encoded as a TPKT hexadecimal bit string. The request can be any `confirmedServiceRequest`, as described in Section 2.2.2. This packet is transmitted from the client to the IED, following the setup detailed in Section 5.1.

6.3.1.1 Fuzzer Action

IEDFuRL performs a action on the MMS request packet, consisting of two key decisions:

- **Chosen Field:** Length of `presentation-context-identifier`, a field in the Presentation(PRES) layer of the packet.
- **Chosen Mutation:** Clearing the value.

IEDFuRL modifies the packet by setting the length of `presentation-context-identifier` to `00`. This mutation affects a single byte within the request structure.

6.3.1.2 Effects Observed by the Fuzzer

This bug was identified in the Siemens 7SA522 device. Upon sending the mutated request, the following behaviors were recorded:

1. **ACK-Only Behavior:** The IED responded with an Acknowledgment (ACK) packet, confirming receipt of the request but failing to return a `responsePDU`, `rejectPDU`, `abortPDU`, or `confirmedErrorPDU`. According to the MMS protocol (Section 2.2.2), a confirmed request requires the IED to either fulfill the service or explicitly reject it with a negative response. Since the length field of a mandatory parameter cannot be `00`, the device should have responded with one of the negative response PDUs defined in Section 2.3.1. The lack of such a response indicates improper handling of malformed requests.
2. **Socket Closure:** Immediately after sending the ACK packet, the IED abruptly terminated the active socket connection on port 102, preventing any further communication through that connection.

3. **Ping Failure:** The network monitor (Section 5.4) detected a temporary loss of connectivity to the IED's communication module. Ping requests to the module's IP address failed for a brief duration, indicating that the module was unreachable. After a short interval, connectivity was restored, and subsequent pings were successful.

These observations suggest that the malformed MMS request triggers an unexpected failure state in the IED's communication module.

6.3.1.3 Extensive Analysis

Further investigation revealed that the failure module and failure channel, visible in the IED's Human Machine Interface (HMI), were toggling ON and OFF in response to the malformed request. Whenever the failure module and failure channel switched ON, the IED's communication module became unresponsive, leading to packet loss and frequent retransmissions, as illustrated in Figure 6.3.

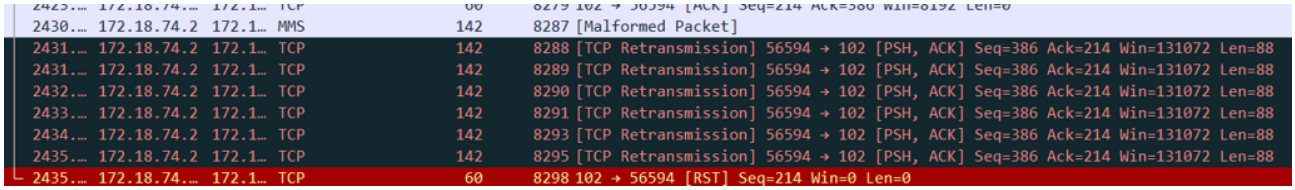


Figure 6.3: Packet Loss and Retransmissions.

Replaying the same request multiple times caused frequent toggling of the failure module and failure channel, exhibiting erratic system behavior, as shown in Figure 6.4.

Figure 6.4: Fluctuations in Failure Module and Channel.

To further analyze the impact of this failure, the mutated packet was replayed multiple times at different time intervals. Initially, the communication module exhibited temporary failures, with the failure module and failure channel toggling ON and OFF intermittently. However, after a certain number of iterations, the failure module and failure channel stayed ON, leading to a persistent communication breakdown as shown in Figure 6.5. Ping requests to the device consistently failed, and all further communication attempts were unsuccessful until the device was manually rebooted.

Figure 6.5: Prolonged Denial of Service.

Ultimately, this behavior resulted in a persistent denial-of-service (DoS) vulnerability on port 102 of the serial system interface. The ability to trigger such a failure remotely poses a

severe risk, as an attacker with network access could exploit this vulnerability to disrupt critical power grid operations.

To determine the conditions under which the IED transitioned into a denial-of-service state, we systematically replayed the malformed MMS request at varying time intervals. Using a divide-and-conquer strategy, we identified the sequence of state transitions leading to persistent failure conditions. The automaton in Figure 6.6 represents the most probable state transition model based on our testing.

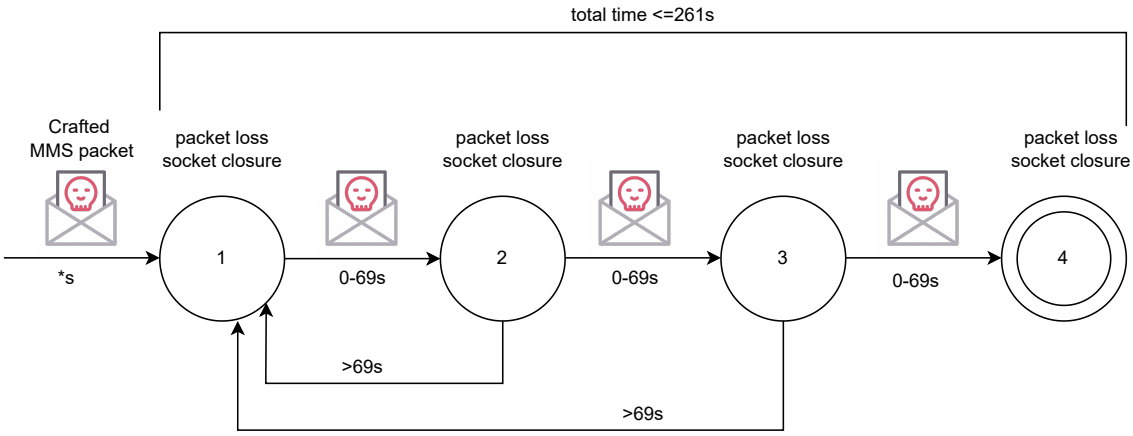


Figure 6.6: Attack Scenario causing Denial of Service

When the specially crafted MMS packet is sent for the first time, the IED reaches State 1, experiencing packet loss or socket closure. If the malformed packet is replayed within 0-69 seconds, the IED transitions to State 2. Further replay within the next 69-second interval, leadthe IED to State 3. Each state serves as an intermediate step, with the IED reverting to State 1 if no malformed packets are sent within the time window of 69 seconds. In States 1, 2, and 3, the failure module and failure channel toggle ON momentarily before turning OFF again. While these transient failures indicate instability, they do not immediately disrupt device operation. If the malformed MMS request is injected four times within a total duration of 261 seconds, the IED enters State 4, a critical failure state where the failure module and failure channel remain permanently ON. At this stage, the device experiences continuous packet loss, effectively rendering the communication module unresponsive to further interactions, permanently blocking communication with the device.

6.3.1.4 Security Implications

This vulnerability poses a serious risk to power grid operations. An attacker with network access to the IED can exploit this flaw by repeatedly sending the malformed MMS request, forcing the

device into an unresponsive state. Since the IEC 61850 protocol is crucial for real-time data exchange and remote control of grid infrastructure, a DoS attack on an IED's communication module could significantly disrupt substation automation and grid reliability.

The malformed request leads to unexpected ACK-only behavior, deviating from the MMS protocol specifications. Replay of specially crafted packet within specific time intervals results in a denial-of-service condition. The device remains unresponsive until a manual reboot is performed. Attackers exploiting this vulnerability can disable communication with the IED, potentially disrupting critical infrastructure.

6.3.2 Bug 2

Seed Input - Initial Valid MMS Request Packet Fuzz testing begins with a valid MMS request packet encoded as a TPKT hexadecimal bit string. The request can be any `confirmedServiceRequest`, as described in Section 2.2.2. This packet is transmitted from the client to the IED, following the setup detailed in Section 5.1.

6.3.2.1 Fuzzer Action

IEDFuRL performs an action on the MMS request packet, consisting of two key decisions:

- **Chosen Field:** Length of `invokeID`, a field in the MMS layer of the packet.
- **Chosen Mutation:** Clearing the value.

IEDFuRL modifies the packet by setting the length of `invokeID` to `00`. This mutation affects a single byte within the request structure.

This bug was identified in the Siemens 7SA522 device. Its effects, extensive analysis, and security implications are identical to those of Bug 1 (Section 6.3.1).

6.3.3 Bug 3

Seed Input - Initial Valid MMS Request Packet The fuzzing process starts with a valid MMS request packet represented as a TPKT hexadecimal bit string. The request can be any `confirmedServiceRequest`, as detailed in Section 2.2.2. The packet is transmitted from the client to the IED, following the setup outlined in Section 5.1.

6.3.3.1 Fuzzer Action

IEDFuRL performs an action on the MMS request packet, consisting of two key decisions:

- **Chosen Field:** MMS `length` field.
- **Chosen Mutation:** Substituting with interesting bit string values.

IEDFuRL modifies the packet by assigning a length of `81`. The bitstring `81` is particularly interesting because it represents -127 in decimal, which is the lowest possible value a byte can take within the hexadecimal byte range of -127 to +127. Notably, decimal -128 (hexadecimal `80`) is interpreted the same as decimal 0 (hexadecimal `00`).

6.3.3.2 Effects Observed by the Fuzzer

This bug was identified in the Siemens 7SA522 device. It causes the next byte to be misinterpreted as the actual length value, potentially leading to parsing inconsistencies. This vulnerability can be exploited in scenarios where setting a specific length value results in an anomaly. By setting the length to 81, followed by the targeted value, the same anomaly can be triggered. Other effects, extensive analysis, and security implications of this bug are identical to those of Bug 1 (Section 6.3.1).

6.3.4 Bug 4

Seed Input - Initial Valid MMS Request Packet A valid MMS `initiate` request is used as the seed input, encoded as a TPKT hexadecimal bit string. The transmission follows the procedure described in Section 5.1.

6.3.4.1 Fuzzer Action

IEDFuRL performs an action on the MMS request packet, consisting of two key decisions:

- **Chosen Field:** Length of `localDetailCalling`, a field in the MMS layer.
- **Chosen Mutation:** Clearing the value.

The length of `localDetailCalling` is set to `00`. This mutation affects a single byte within the request structure. This bug was identified in the Siemens 7SA522 device. Its effects, extensive analysis, and security implications are identical to those of Bug 1 (Section 6.3.1).

6.3.5 Bug 5

Seed Input - Initial Valid MMS Request Packet The fuzzer starts with a valid MMS `initiate` request packet, structured as a TPKT hexadecimal bit string and transmitted as per Section 5.1.

6.3.5.1 Fuzzer Action

IEDFuRL performs an action on the MMS request packet, consisting of two key decisions:

- **Chosen Field:** Length of `proposedVersionNumber`, an MMS field.

- **Chosen Mutation:** Clearing the value.

The packet is modified by setting the length of `proposedVersionNumber` to `00`. This mutation affects a single byte within the request structure. This bug was identified in the Siemens 7SA522 device. Its effects, extensive analysis, and security implications are identical to those of Bug 1 (Section 6.3.1).

6.3.6 Bug 6

Seed Input - Initial Valid MMS Request Packet The fuzzer starts with a valid MMS `initiate` request packet, structured as a TPKT hexadecimal bit string and transmitted as per Section 5.1.

6.3.6.1 Fuzzer Action

IEDFuRL performs an action on the MMS request packet, consisting of two key decisions:

- **Chosen Field:** Length of `proposedMaxServOutstandingCalled`, an MMS field.
- **Chosen Mutation:** Clearing the value.

By setting the length of `proposedMaxServOutstandingCalled` to `00`. This mutation affects a single byte within the request structure. This bug was identified in the Siemens 7SA522 device. Its effects, extensive analysis, and security implications are identical to those of Bug 1 (Section 6.3.1).

6.3.7 Bug 7

Seed Input - Initial Valid MMS Request Packet The fuzzer starts with a valid MMS `initiate` request packet, structured as a TPKT hexadecimal bit string and transmitted as per Section 5.1.

6.3.7.1 Fuzzer Action

IEDFuRL performs an action on the MMS request packet, consisting of two key decisions:

- **Chosen Field:** Length of `proposedDataStructureNestingLevel`, an MMS field.
- **Chosen Mutation:** Clearing the value.

By setting the length of `proposedDataStructureNestingLevel` to `00`, this mutation affects a single byte within the request structure.

This bug was identified in the Siemens 7SA522 device. Its effects, extensive analysis, and security implications are identical to those of Bug 1 (Section 6.3.1).

6.3.8 Bug 8

Seed Input - Initial Valid MMS Request Packet A valid MMS `write` request packet is used as the input, transmitted following the setup in Section 5.1.

6.3.8.1 Fuzzer Action

IEDFuRL applies a series of six mutation actions to the MMS request packet, each involving the selection of a specific field and an associated mutation strategy. In this case, IEDFuRL modifies the length of `invokeID` to `84` and alters the subsequent five bytes to `ff ff 00 11 00`. This mutation affects multiple bytes within the request structure.

This bug was identified in the Siemens 7SA522 device. Its effects, extensive analysis, and security implications are identical to those of Bug 1 (Section 6.3.1).

6.3.8.2 Additional Analysis

Further analysis revealed that the same bug could be triggered by setting the length of `invokeID` to `84` and modifying the following five bytes to values within the range `ff 7c __ __ 00` to `ff ff __ __ 00`. The specific byte value ranges responsible for triggering the bug were systematically identified using a divide-and-conquer binary search strategy.

6.3.9 Bug 9

Seed Input - Initial Valid MMS Request Packet A valid MMS `getNameList` request packet is used as the input, transmitted following the setup in Section 5.1.

6.3.9.1 Fuzzer Action

IEDFuRL performs an action on the MMS request packet, consisting of two key decisions:

- **Chosen Field:** Length of `objectClass`, an MMS field in the request.
- **Chosen Mutation:** Clearing the value.

Setting the length of `objectClass` to `00` modifies the request structure, potentially leading to an unintended response or failure.

This bug was identified in the Siemens 7SA522 device. Its effects, extensive analysis, and security implications are identical to those of Bug 1 (Section 6.3.1).

6.3.10 Bug 10

Seed Input - Initial Valid MMS Request Packet A valid MMS `fileDirectory` request is used, formatted as a TPKT hexadecimal bit string and transmitted according to Section 5.1.

6.3.10.1 Fuzzer Action

IEDFuRL applies a series of six mutation actions to the MMS request packet, each involving the selection of a specific field and an associated mutation strategy. In this case, IEDFuRL modifies the length of `invokeID` to `84` and alters the subsequent five bytes to `ff ff 11 11 00`. This mutation affects multiple bytes within the request structure.

This bug was identified in the Siemens 7SA522 device. Its effects, extensive analysis, and security implications are identical to those of Bug 1 (Section 6.3.1).

6.3.10.2 Additional Analysis

Further analysis revealed that the same bug could be triggered by setting the length of `invokeID` to `84` and modifying the following five bytes to values within the range `ff 89 __ __ 00` to `ff ff __ __ 00`. The specific byte value ranges responsible for triggering the bug were systematically identified using a divide-and-conquer binary search strategy.

6.4 Information Disclosure

This section outlines the bugs that caused information leaks in the communication module of the tested IEDs, providing a comprehensive analysis of their impact.

6.4.1 Bug 11

Seed Input - Initial Valid MMS Request Packet Fuzz testing begins with a valid MMS `initiate` request packet encoded as a TPKT. This packet is transmitted from the client to the IED, following the setup detailed in Section 5.1.

6.4.1.1 Fuzzer Action

IEDFuRL performs an action on the MMS request packet, involving two key decisions:

- **Chosen Field:** `called-ap-title` with the subfield `ap-title-form2`, a field in the Association Control Service (ACSE) layer of the packet.
- **Chosen Mutation:** Substituting with an interesting bitstring value.

IEDFuRL modifies the packet by replacing the first byte of the subfield `ap-title-form2` in `called-ap-title` with `ff 01 87 67 01`, instead of the original value `29 01 87 67 01`. The bitstring `ff` is particularly interesting as it consists entirely of bits set to 1 in its binary representation. This mutation alters a single byte within the request structure.

6.4.1.2 Effects Observed by the Fuzzer

This bug was identified in both tested devices, the Siemens 7SA522 and the Hitachi Energy RET670. The response generated by the IED was flagged as a malformed response by the Pyshark dissector, as described in Section 5.2.1. A malformed response indicates that the packet does not conform to the Basic Encoding Rules (BER), which define the encoding format for MMS communication, as explained in Section 2.1. This deviation from the expected BER structure is considered an unexpected behavior, suggesting a potential flaw in the device's handling of request packets.

6.4.1.3 Extensive Analysis

During the analysis of the malformed response, we observed that the last 87 bytes of the response were identical to the `initiate-Response` previously sent by the IED. This raised the possibility that the malformed response contained residual data from the last processed response of the device. Upon further inspection, we confirmed that the malformed response consistently included the last 87 bytes of the response to the most recently processed request. If the length of the previous response was shorter than 87 bytes, the malformed response contained additional leading bytes along with the entire previous response.

To examine the persistence of this behavior, we tested whether closing the current socket connection would clear the leaked data in the malformed response. However, even after terminating the socket connection and reopening it, the malformed response still contained data from the last processed request. This observation enabled us to simulate a potential attack vector where two separate clients: one being a legitimate client that initiates and maintains communication with the IED, and the other being an attacker exploiting the vulnerability, as illustrated in Figure 6.7. The attacker operates from a different system with a different IP address. However, this issue can also be replicated on the same system by establishing a separate socket connection. For instance, Client B, using the same IP address, can initiate a new TCP socket connection to exploit the vulnerability in a similar manner.

The attack scenario can be described as follows:

Legitimate User - Client A:

1. Opens a TCP socket connection and establishes a Connection-Oriented Transport session through `ClientA` with IP Address `IP_A`.
2. Sends an `MMS initiate-Request` packet to the IED.
3. Receives an `MMS initiate-Response` packet from the IED.

4. Sends `MMS Request R1` to the IED.

5. Receives `R1'` (the response to `R1`) from the IED.

Attacker – Client B:

1. Opens a new TCP socket connection and establishes a Connection-Oriented Transport session through `ClientB` with IP Address `IP_B`.

2. Sends the specially crafted `MMS initiate-Request` packet.

3. Receives a malformed ACSE response containing the last 87 bytes of `R1'`.

Legitimate User – Client A:

6. Sends `MMS Request R2` to the IED.

7. Receives `R2'` (the response to `R2`) from the IED.

Attacker – Client B:

1. Sends another specially crafted `MMS initiate-Request` packet.

2. Receives a malformed ACSE response containing the last 87 bytes of `R2'`.

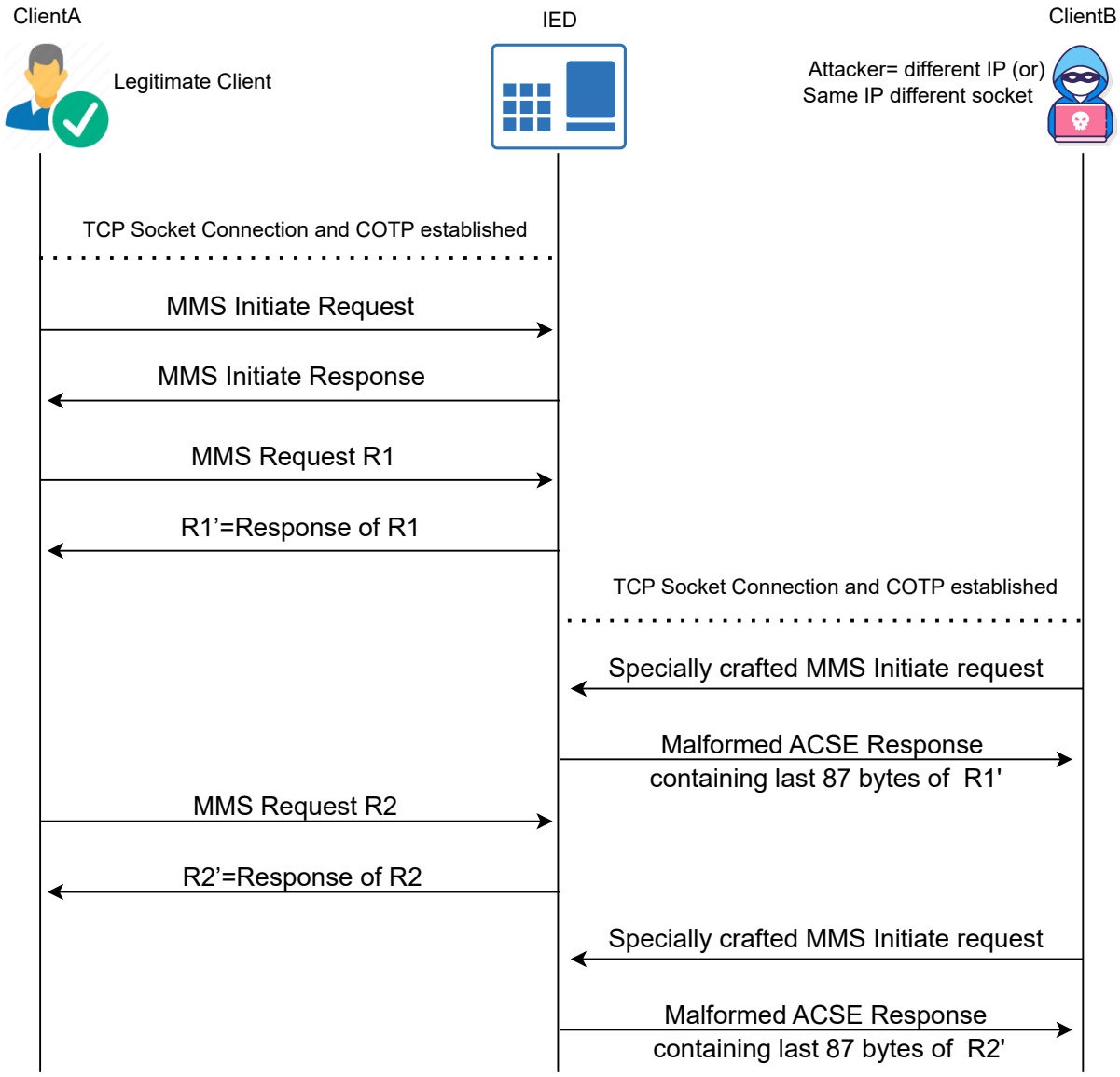


Figure 6.7: Attack scenario causing Information Disclosure

This behavior allows an attacker to continuously retrieve response data from previous legitimate client requests. By repeatedly sending the malformed `MMS InitiateRequest`, the attacker gains access to response data originally intended for another client. This poses an information disclosure risk.

6.4.1.4 Security Implications

An attacker with network access to the IED can exploit this specially crafted initiate request to capture a portion of the last legitimate response processed by the device. By continuously send-

ing the crafted initiate request after each legitimate response, the attacker can incrementally extract parts of all responses sent to legitimate users. This enables a passive man-in-the-middle (MitM) attack, allowing the attacker to access the last 87 bytes of all the previously processed responses.

IEDFuRL effectively detects such anomalies by analyzing responses and identifying deviations from protocol specifications. This capability helps uncover protocol inconsistencies and implementation vulnerabilities, strengthening the security evaluation of IEC 61850-based devices.

6.5 Data Boundary Violation

This section outlines the bug that caused the IED to violate data boundary of one of the data attributes.

6.5.1 Bug 12

Seed Input - Initial Valid MMS Request Packet Fuzz testing begins with a valid MMS `initiate` request packet encoded as a TPKT. This packet is transmitted from the client to the IED, following the setup detailed in Section 5.1.

6.5.1.1 Fuzzer Action

IEDFuRL performs an action on the MMS request packet, involving two key decisions:

- **Chosen Field:** `proposedMaxServOutstandingCalled`, a field in the MMS layer of the packet.
- **Chosen Mutation:** Flipping a bit.

IEDFuRL modifies the packet by flipping the most significant bit of the `proposedMaxServOutstandingCalled` field, changing its value from `04` (binary `00000100`) to `84` (binary `10000100`). This mutation affects a single bit within the request structure. Here, `84` corresponds to -124 in decimal representation.

6.5.1.2 Effects Observed by the Fuzzer

The IED responds with a well-formed `initiate-Response` Protocol Data Unit (PDU), where the field `negotiatedMaxServOutstandingCalled` contains the value -124 (or the corresponding negative value derived from the `proposedMaxServOutstandingCalled` field in the request). This value results from the negotiation process carried out by the device in response to the `initiate-Request`. However, assigning a negative value to an allocated resource is indicative

of erroneous behavior within the device's communication module, as it is illogical to negotiate resources with a negative count. IEDFuRL classifies this response as a new category and rewards the RL agent accordingly to encourage the discovery of similar anomalies.

6.6 Reported and Acknowledged Vulnerabilities

The identified bugs were reported to Siemens ProductCERT and Hitachi Energy's Product Security Incident Response Team. Some of the reported bugs were already discovered and documented in Siemens Security Advisory SSA-635129, which references two known vulnerabilities: CVE-2018-11452 with a CVSS v3.0 Base Score of 7.5 and CVE-2018-11453 with a CVSS v3.0 Base Score of 5.9.

For reference, Figure 6.8 presents a sample response from Siemens regarding the report submitted for Bug 8.

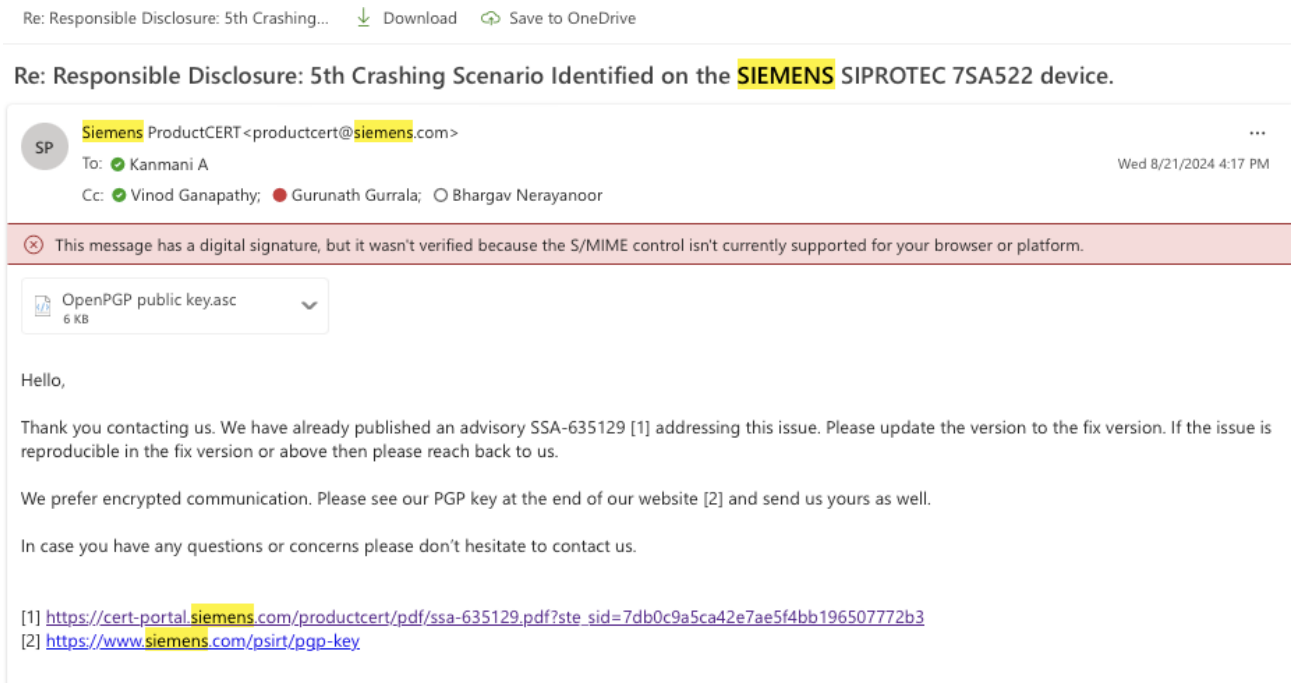


Figure 6.8: Acknowledgment Email from Siemens ProductCERT.

The reporting and acknowledgment details of all identified bugs are summarized in Table 6.2.

Table 6.2: Bug Report Details

Bug No.	Reporting Date	Response Date	Response Description
1, 2, 3	26.03.2024	04.02.2024	Bug confirmed, but already addressed in SSA-635129. Fixed in version v4.33 and higher.
4, 5, 6, 7	Not Reported	-	These bugs were not reported to Siemens as their standard response to such issues was to patch the device and retest.
8	14.08.2024	14.08.2024	Bug confirmed, but already addressed in SSA-635129. Fixed in version v4.33 and higher.
9	21.08.2024	21.08.2024	
10	28.10.2024	28.10.2024	
11	12.11.2024	22.11.2024	The bug was confirmed, and both Siemens and Hitachi Energy acknowledged it as a newly identified issue. They are actively working on a fix. However, no CVE was issued, as the disclosed information could be obtained through standard MMS requests, given that MMS lacks encryption by design.

6.7 Discussion

IEDFuRL is designed to discover vulnerabilities in IEC 61850-based IEDs through protocol-aware fuzzing guided by reinforcement learning. The primary categories of bugs identified using IEDFuRL include Denial of Service (DoS), Information Disclosure, and Data Boundary Violations. These categories were identified by interpreting the IED's responses based on the reward structure defined in the reinforcement learning environment, as described in Section 4.4.

IEDFuRL specifically targets **crashing bugs** and **malformed response bugs**, which are interpreted as buggy scenarios—each representing a successful vulnerability-triggering incident. The agent receives the highest reward when such a response is observed, making these bugs the most prioritized during training. This reward structure essentially defines the failure oracle for IEDFuRL, allowing it to detect vulnerabilities without requiring detailed domain-specific knowledge about correct MMS behavior.

In the broader context of fuzzing, a fuzzing engine's core responsibility is to continuously generate and supply mutated inputs to the system under test. It does not inherently interpret the outcome of a test as a success or failure; instead, this determination is delegated to an external *failure oracle*. Traditional fuzzers often rely on oracles such as segmentation faults or crash dumps. However, fuzzing is capable of producing any input that may trigger a failure, assuming a mechanism exists to detect such failure conditions.

In IEDFuRL's case, the simplest and most effective failure oracle selected is the crash or instability of the communication module, detected using methods outlined in Section 5.4. All major categories of bugs discovered during testing—including information disclosure and data boundary violations—were initially detected using this oracle. For instance, the malformed response that led to the discovery of an information leak was first flagged because it did not conform to expected encoding rules and thus triggered the malformed response reward. Similarly, the data boundary violation was detected due to an anomalous negative value in the IED's response, which was treated as a new response category and rewarded accordingly.

It is important to note that IEDFuRL can theoretically be extended to incorporate more sophisticated failure oracles.

Another potential failure oracle is based on the behavior of the IED's Human Machine Interface (HMI). While not currently used in IEDFuRL, such an indicator could be leveraged in future iterations of the framework to expand the categories of detectable bugs.

While IEDFuRL presently focuses on a limited set of bug categories defined by its reward function and associated oracles, the architecture is flexible enough to accommodate a broader spectrum of vulnerabilities as more robust failure detection mechanisms or specifications become available.

While the number of bugs discovered through our experiments offers valuable insight into the presence of vulnerabilities, defining a definitive security metric to compare one IED against another remains a complex challenge. In fact, the broader problem of establishing quantitative security metrics for software systems is still an open research question. This difficulty arises due to the challenges in exhaustively enumerating the system's attack surface and accounting for the complex interactions between internal components. As highlighted in Manadhata and Wing's work on attack surface metrics [53], the attack surface of a system is often vast and dynamic, making it difficult to measure and compare security quantitatively across systems.

Chapter 7

Performance Evaluation of RL in IEDFuRL

To assess the effectiveness of the reinforcement learning (RL) agent in IEDFuRL, we conducted a comparative evaluation against a baseline random policy agent.

7.1 Performance Comparison

To systematically analyze the differences in performance, we tested the following two fuzzing approaches:

IEDFuRL The RL agent in IEDFuRL employs a Deep Q-Network (DQN) implemented using TensorFlow. The DQN-based agent intelligently selects mutation strategies that maximize the likelihood of discovering vulnerabilities by learning from previous interactions with the IED. Through reinforcement learning, the agent continuously refines its approach by prioritizing mutations that yield higher rewards, leading to a more structured and efficient exploration of the input space.

RandomFuzzer To serve as a baseline comparison, we developed `RandomFuzzer`, a fuzzing agent that is protocol-aware but lacks intelligent decision-making capabilities. Unlike the RL-based approach, RandomFuzzer applies mutations randomly across the MMS request fields without considering their impact on triggering faults in the IED. This agent helps evaluate the advantages of reinforcement learning over a purely random mutation strategy. The implementation setup of RandomFuzzer is very similar to that of IEDFuRL. It requires all the components described in Chapter 5, except for the DQN model details outlined in Section 5.3.

While both IEDFuRL and RandomFuzzer are protocol-aware, the key difference lies in their mutation selection approach. IEDFuRL leverages reinforcement learning to prioritize

effective mutations based on past interactions, whereas RandomFuzzer applies mutations in a random, unstructured manner without any adaptive learning.

Cybersecurity testbeds for IEC 61850-based smart substations [28], does not provide algorithmic or implementation details, making direct comparison infeasible. Similarly, the system presented in [29] lacks publicly available code and employs a protocol-based fuzzing approach with limited feedback. In that work, a device-alive check is used to assign weights to protocol fields targeted for fuzzing. The fuzzing strategy used in [29] is based on the Sulley fuzzer [30], which requires manual configuration of protocol fields and extensive data modeling. This modeling process suffers from the same limitations as Peach fuzzer, including high setup overhead and lack of scalability across diverse devices. To provide a fair comparison, we implemented a RandomFuzzer that is protocol-aware but lacks intelligent decision-making. It selects actions randomly and can therefore serve as a suitable baseline against systems such as those in [50], [48], and [6], where feedback mechanisms and learning strategies are limited or absent. However, since the codebases for these systems were not publicly available and setting up data modelling for each IED would have demanded significant effort, we did not reimplement them. Instead, we chose to implement Snipuzz for IEDs because its source code is available, allowing us to compare it and RandomFuzzer with IEDFuRL. This approach enabled a technically grounded evaluation while acknowledging the practical constraints of reproducing prior work.

7.2 Characteristics of Test Experiments

The experiment begins with a valid MMS request as the initial seed input for fuzzing. In each step, 15 sequential mutations are applied to the request,each of which is transmitted to the IED under test. The response is analyzed, and in the case of the RL-based fuzzer, the agent receives a reward to refine future mutation strategies. This process is repeated for 50 steps to ensure thorough exploration of the input space. Three different seed requests were selected to evaluate performance comprehensively. The weights of the DQN in IEDFuRL were reset to their initial values before each test run to ensure a consistent evaluation. In future work, an approach incorporating accumulated training across multiple runs could be explored to further enhance performance.

The following table summarizes the test runs used for evaluation:

Table 7.1: Experimental Runs for Performance Evaluation

Run No.	Seed Request Type	Device Tested
1	`identify`	Siemens 7SA522
2	`getNameList`	Siemens 7SA522
3	`initiate`	Hitachi Energy RET670

The obtained results include graphs comparing the average rewards, cumulative rewards, mutation efficiency of each approach. The results are analyzed to determine the effectiveness of reinforcement learning in the fuzzing process compared to a random mutation strategy.

7.2.1 Average Reward

The average reward is calculated as the total reward accumulated per episode, providing insight into how effectively the agent explores the input space and applies meaningful mutations. A higher average reward indicates that the agent is consistently identifying impactful mutations and optimizing its fuzzing strategy.

The RL-based IEDFuRL agent significantly outperformed the random policy agent in terms of average reward, as illustrated in Figure 7.1, which compares reward trends over 50 steps for the three test runs listed in Table 7.1. While the RandomFuzzer may have achieved a higher reward in the initial steps due to randomly triggering a crash scenario, over the long run, the RL-based IEDFuRL consistently demonstrated a steady increase in reward, emphasizing its ability to learn optimal mutation patterns and focus on high-rewarding input regions. Its structured learning mechanism enabled it to refine mutation strategies, ensuring a more efficient and targeted fuzzing process compared to the purely random approach.

In contrast, the random policy agent exhibited significantly lower and more fluctuating rewards, as it lacked any learning mechanism to guide its mutation choices.

(a) Run 1 (b) Run 2 (c) Run 3

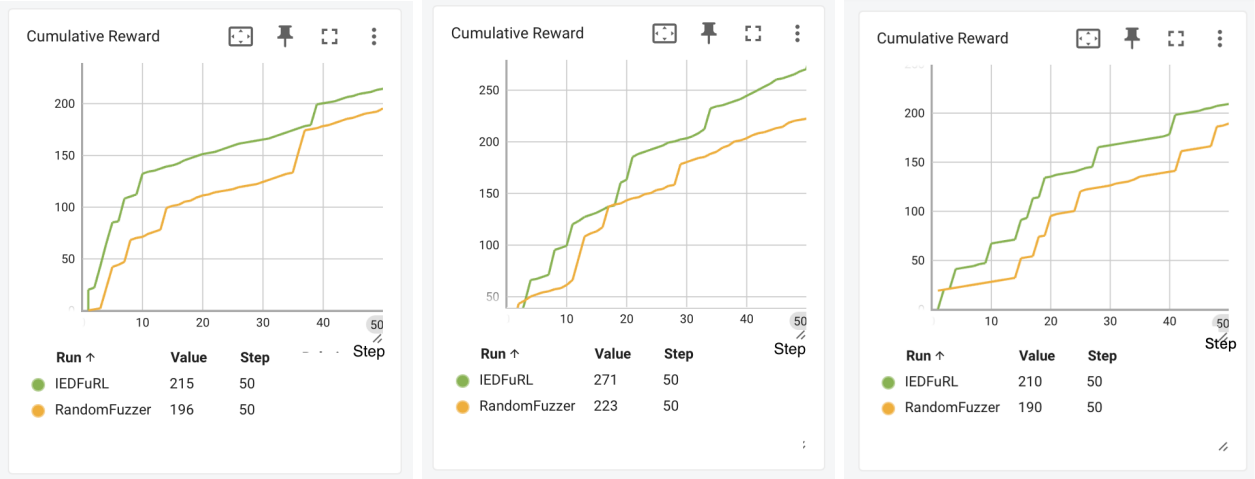
Figure 7.1: Comparison of Average Reward

7.2.2 Cumulative Reward Graph

The cumulative reward graph provides insight into the learning progress and effectiveness of the RL agent over multiple episodes. A steadily increasing cumulative reward indicates that the agent is not only identifying high-impact mutations but also prioritizing them effectively, leading to a more efficient fuzzing process.

Figure 7.2 compares the cumulative reward of the RL agent and the random policy agent over 50 steps for the three test runs listed in Table 7.1. A key observation from the results is that the RL agent demonstrated a consistent upward trend, showcasing its ability to refine its mutation strategy, learn from past interactions, and maximize rewards over time. This reinforces its effectiveness in adapting to the fuzzing environment and systematically uncovering vulnerabilities.

In contrast, the random policy agent exhibited a much slower growth in cumulative reward, as it lacked a structured learning mechanism to optimize its mutation selection. This highlights the advantage of reinforcement learning in fuzz testing, as the RL agent efficiently navigates the input space and prioritizes high-rewarding mutations, unlike the random approach, which relies solely on unstructured exploration.



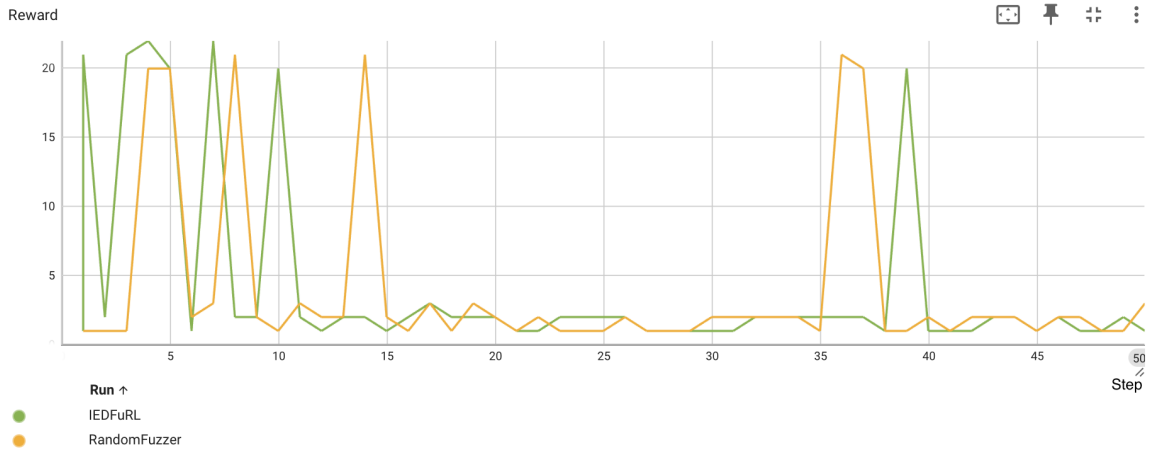
(a) Run 1 (b) Run 2 (c) Run 3

Figure 7.2: Comparison of Cumulative Reward

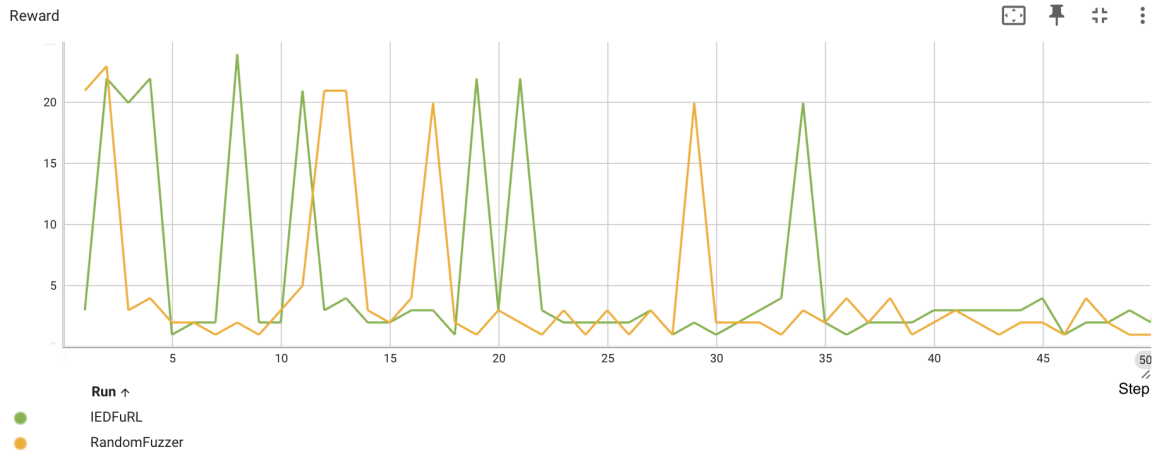
7.2.3 Reward Graph

The reward graph shown below exhibits peaks whenever the agent discovers a crash or a malformed response, in accordance with the reward function of IEDFuRL, as described in Section 5.3.4. Once a crash-triggering action is identified, it is subsequently masked, as explained in Section 4.5.1, preventing redundant exploration of already discovered crashes. The fluctuations in the graph, characterized by peaks and dips, result from the RL agent's balance between exploration and exploitation.

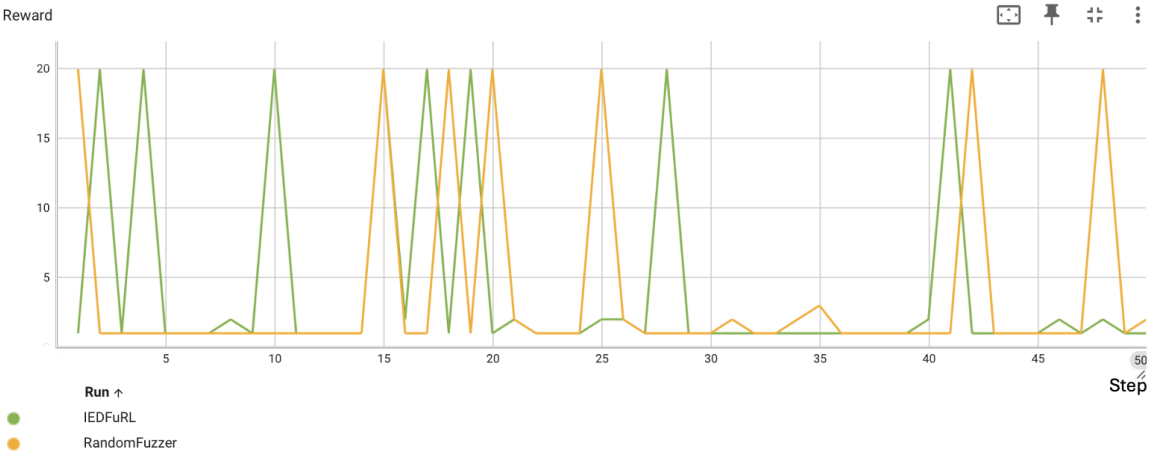
A key observation is that IEDFuRL identifies crashes significantly earlier in the fuzzing process, detecting more crashes within the first few steps compared to the random agent. This highlights the RL agent's ability to quickly adapt, learn effective mutation strategies, and efficiently navigate towards high-impact inputs, reinforcing its advantage over a purely random approach.



(a) Run 1



(b) Run 2



(c) Run 3

Figure 7.3: Rewards Obtained by IEDFuRL and RandomFuzzer

7.3 Efficiency in Triggering Bugs

Comparison with Snipuzz and Random Agent The comparison of the number of mutations required to discover unique bugs for the first time across different fuzzing strategies demonstrates the efficiency of IEDFuRL. Initial seed input - MMS Request packet- is same for all the three setups. Figure 7.4 presents a comparative analysis of the number of mutations needed to trigger a bug using three different approaches: IEDFuRL, Snipuzz, and RandomFuzzer.

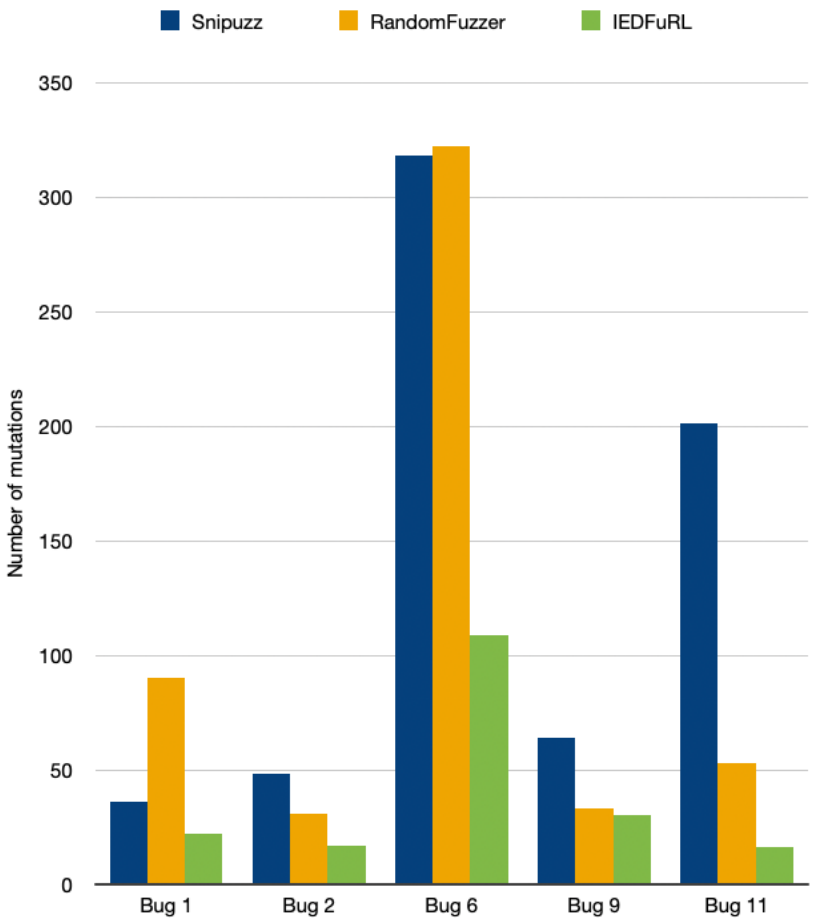


Figure 7.4: Number of mutated requests required to encounter the bugs for the first time

Among the twelve identified bugs in the Siemens IED described in Chapter 6, all three methods successfully detected five common bugs. However, the RL-based IEDFuRL agent consistently required fewer mutations to uncover these vulnerabilities, showcasing its ability to prioritize high-impact test cases. The occurrence of bugs during fuzzing is inherently proba-

bilistic in most cases, except for specific scenarios where Snipuzz identifies a bug during its deterministic probing stage—by sequentially setting bits to 0 from left to right. Consequently, the results may vary across different runs. To address this variability, we chose to analyze a representative case in detail, as repeating the same process for every run may not lead to consistent observations. At the same time, we evaluated all five bugs instead of just one to gain a broader and more general understanding of each fuzzer's performance across diverse scenarios. This finding underscores the advantage of reinforcement learning in fuzz testing, as it systematically guides the fuzzing process towards crashes and vulnerabilities with greater efficiency.

Chapter 8

Conclusion and Future Directions

This research presented IEDFuRL, a protocol-aware reinforcement learning (RL)-based fuzzing framework designed for automated vulnerability detection in IEC 61850-based Intelligent Electronic Devices (IEDs). The RL agent dynamically learns optimal mutation strategies for MMS request fields, enabling efficient state-space exploration while prioritizing impactful test cases. This approach significantly enhances vulnerability detection compared to traditional black-box fuzzing techniques. Furthermore, IEDFuRL is designed as a generalized methodology, making it adaptable across various IED implementations that conform to the IEC 61850 standard.

The IEDFuRL has been rigorously evaluated on real-world IEDs, demonstrating its effectiveness in identifying communication module vulnerabilities and underscoring its potential to enhance the security of critical power grid infrastructure. While IEDFuRL presents significant advancements in protocol-aware fuzz testing using reinforcement learning, it currently focuses solely on issues within the communication modules of IEDs. Exploring vulnerabilities in other components, such as device firmware, remains an open area for future research.

The framework is adaptable and can be applied to any client-server protocol that is not encrypted. However, adapting it to other protocols will require modifications to the Request and Response Handler (Section 5.2), as different protocols have varying request and response field structures. Future enhancements of IEDFuRL may also focus on incorporating state awareness and temporal dynamics into the fuzzing process. This would enable the detection of a wider range of anomalies, including subtle deviations in protocol behavior and state-dependent issues, beyond just malformed responses and crashes. Moreover, future work will aim to further optimize the learning process and improve fuzzing efficiency.

Bibliography

[1] Stuart A. Boyer. *SCADA: Supervisory Control And Data Acquisition*. International Society of Automation, Research Triangle Park, NC, USA, 4th edition, 2009. ISBN 1936007096. 1

[2] International Electrotechnical Commission (IEC). *IEC 61850: Communication Networks and Systems for Power Utility Automation*, 2013. URL https://webstore.iec.ch/publication/6028. 1

[3] Anton Cherepanov. Industroyer: Biggest Threat to Industrial Control Systems Since Stuxnet. *ESET Research*, 2017. URL https://www.welivesecurity.com/2017/06/12/industroyer-biggest-threat-industrial-control-systems-since-stuxnet/. 1

[4] Symantec Security Response Team. Dragonfly: Cyber Espionage Attacks Against Energy Sector. *Symantec Threat Intelligence*, 2017. URL https://www.security.com/blogs/threat-intelligence/dragonfly-energy-sector-cyber-attacks. 1

[5] Cybersecurity and Infrastructure Security Agency (CISA). Tactics, techniques, and procedures of indicted state-sponsored russian cyber actors targeting the energy sector, 2022. URL https://www.cisa.gov/news-events/alerts/2022/03/24/tactics-techniques-and-procedures-indicted-state-sponsored-russian-cyber-actors-targeting-energy. 1

[6] Jagmeet Singh. Tata Power confirms cyberattack on IT infrastructure, 2022. URL https://techcrunch.com/2022/10/25/tata-power-hive-ransomware/. 1

[7] David E. Sanger and Nicole Perlroth. China Appears to Warn India: Push Too Hard and the Lights Could Go Out. *The New York Times*, 2021. URL https://www.nytimes.com/2021/02/28/us/politics/china-india-hacking-electricity.html. 1

[8] The Economic Times. Chinese hackers target Indian power grid assets in Ladakh. *The Economic Times*, 2022. URL https://economictimes.indiatimes.com/industry/

energy/power/chinese-hackers-target-indian-power-grid-assets-in-ladakh/articleshow/90950986.cms?from=mdr. 1

[9] McAfee Labs. Triton Malware Spearheads Latest Generation of Attacks on Industrial Systems. *McAfee Blogs*, 2018. URL https://www.mcafee.com/blogs/other-blogs/mcafee-labs/triton-malware-spearheads-latest-generation-of-attacks-on-industrial-systems/. 1

[10] Paria Shirani, Louis Collard, Basile L. Agba, Benoit Lebel, and Mourad Debbabi. BINARM: Scalable and Efficient Detection of Vulnerabilities in Firmware Images of Intelligent Electronic Devices. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 2122–2131. ACM, 2019. doi: 10.1145/3297280.3297531. 2

[11] Barton P. Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of UNIX utilities. *Commun. ACM*, 33(12):32–44, December 1990. ISSN 0001-0782. doi: 10.1145/96267.96279. URL https://doi.org/10.1145/96267.96279. 2

[12] Hongliang Liang, Xiaoxiao Pei, Xiaodong Jia, Wuwei Shen, and Jian Zhang. Fuzzing: State of the Art. *IEEE Transactions on Reliability*, 67(3):1199–1218, 2018. doi: 10.1109/TR.2018.2834476. 2

[13] International Electrotechnical Commission (IEC). IEC 61850-9-2: Communication networks and systems for power utility automation – Part 9-2: Specific Communication Service Mapping (SCSM) – Sampled values over ISO/IEC 8802-3, 2022. URL https://webstore.iec.ch/en/publication/66549. [Online; accessed February 17, 2025]. 2

[14] Carl Kriger, Shaheen Behardien, and John-Charly Retonda-Modiya. A detailed analysis of the GOOSE message structure in an IEC 61850 standard-based substation automation system. *International Journal of Computers Communications & Control*, 8(5):708–721, 2013. 2

[15] SISCO. Manufacturing Message Specification (MMS) Overview, 2016. URL https://www.sisconet.com/wp-content/uploads/2016/03/mmsovrlg.pdf. [Online; accessed February 17, 2025]. 2, 9

[16] Xiaotao Feng, Ruoxi Sun, Xiaogang Zhu, Minhui Xue, Sheng Wen, Dongxi Liu, Surya Nepal, and Yang Xiang. Snipuzz: Black-box fuzzing of iot firmware via message snippet inference. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pages 337–350, 2021. 3, 17

[17] Saurabh Amin, Alvaro A. Cárdenas, and S. Sastry. Safe and secure networked control systems under the iec 61850 standard, 2010. URL http://seclab.illinois.edu/wp-content/uploads/2011/03/iec61850-intro.pdf. Accessed: 2024-02-25. 6

[18] Nilotpal Mitra. Efficient Encoding Rules for ASN. 1-Based Protocols. *AT&T Technical Journal*, 73(3):80–93, 1994. 6

[19] SISCO. MMS Abstract Syntax, 2016. URL https://www.sisconet.com/wp-content/uploads/2016/03/mms_abstract_syntax.txt. [Online; accessed February 17, 2025]. 6

[20] Wireshark Community. MMS ASN.1 Encoding - Wireshark Interpreter, 2013. URL https://github.com/boundary/wireshark/blob/master/asn1/mms/mms.asn. 7, 40

[21] Siemens. *7SA522 MMS PIXIT Manual*, 2020. URL https://cache.industry.siemens.com/dl/files/387/109743387/att_903668/v1/7SDSAVK_PIXIT_A3_V042001_en.pdf. 8

[22] J. Postel. Transmission Control Protocol - DARPA Internet Program Protocol Specification. Request for Comments (RFC) 793, 1981. URL https://www.ietf.org/rfc/rfc793.txt. 8

[23] Marshall T. Rose and Dwight E. Cass. RFC 1006 - ISO Transport Service on Top of the TCP. Technical report, Internet Engineering Task Force (IETF), 1987. URL https://www.ietf.org/rfc/rfc1006.txt. 9

[24] International Organization for Standardization. ISO/IEC 8073:1997 - Information Technology — Open Systems Interconnection — Protocol for Providing the Connection-mode Transport Service. International Standard, 1997. URL https://www.iso.org/standard/24077.html. 9

[25] International Organization for Standardization (ISO). ISO 8327: Information Processing Systems - Open Systems Interconnection - Session Protocol Specification, 1987. URL https://cdn.standards.iteh.ai/samples/15466/214207c0253841818258eb8e145988cf/ISO-8327-1987.pdf. 9

[26] International Organization for Standardization. ISO/IEC 8823-1:1994 - Information technology — Open Systems Interconnection — Connection-oriented presentation protocol: Protocol specification, 1994. URL https://www.iso.org/standard/20270.html. 9

[27] Jan Tore Sorensen and Martin Gilje Jaatun. An Analysis of Manufacturing Message Specification Protocol. *CiteSeerX*, 2008. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4a5be26509557f0a1a911e639868bfe9d002d664. 10

[28] Yi Yang, HT Jiang, Kieran McLaughlin, L Gao, YB Yuan, W Huang, and Sakir Sezer. Cybersecurity Test-bed for IEC 61850 based smart substations. In *2015 IEEE Power & Energy Society General Meeting*, pages 1–5. IEEE, 2015. 14, 67

[29] Tengfei Tu, Hua Zhang, Boqin Qin, and Zhuo Chen. A vulnerability mining system based on fuzzing for IEC 61850 protocol. In *2017 5th International Conference on Frontiers of Manufacturing Science and Measuring Technology (FMSMT 2017)*, pages 589–597. Atlantis Press, 2017. 14, 67

[30] OpenRCE Community. Sulley Fuzzing Framework, 2009. URL https://github.com/OpenRCE/sulley. 14, 67

[31] Peach Tech. Peach Fuzzer - Community Edition, 2024. URL https://peachtech.gitlab.io/peach-fuzzer-community/. 15

[32] Claroty Team82. MMS Under the Microscope: Examining the Security of a Power Automation Standard, 2022. URL https://claroty.com/team82/research/mms-under-the-microscope-examining-the-security-of-a-power-automation-standard. 17

[33] Michal Zalewski. American Fuzzy Lop (AFL), 2015. URL http://lcamtuf.coredump.cx/afl/. 17

[34] MZ Automation GmbH. libIEC61850: Open-source IEC 61850 Client/Server Library, 2025. URL https://libiec61850.com/. Accessed: 2025-02-26. 17

[35] VI Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Proceedings of the Soviet physics doklady*, 1966. 18

[36] Frank Nielsen and Frank Nielsen. Hierarchical Clustering. *Introduction to HPC with MPI for Data Science*, pages 195–211, 2016. 19

[37] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998. 22, 25

[38] George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. Evaluating fuzz testing. CCS '18, page 2123–2138, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356930. doi: 10.1145/3243734.3243804. URL https://doi.org/10.1145/3243734.3243804. 23

[39] Anssi Kanervisto, Christian Scheller, and Ville Hautamäki. Action Space Shaping in Deep Reinforcement Learning. In *2020 IEEE Conference on Games (CoG)*, pages 479–486, 2020. doi: 10.1109/CoG47356.2020.9231687. 30

[40] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999. 31

[41] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 31

[42] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999. 31

[43] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992. 31

[44] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 31, 33

[45] Siemens AG. *7SA522 Distance Protection Relay - Manual A9 V047100*, 2021. URL https://cache.industry.siemens.com/dl/files/402/109743402/att_903694/v1/7SA522x_Manual_A9_V047100_en.pdf. 35, 46

[46] Kennet Shibata and Contributors. Pyshark: Python Wrapper for Wireshark, 2015. URL https://github.com/KimiNewt/pyshark. Accessed: 2024-02-17. 37

[47] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Efi Kokiopoulou, Luciano Sbaiz, Jamie Smith, Gábor Bartók, Jesse Berent, Chris Harris, Vincent Vanhoucke, and Eugene Brevdo. TF-Agents: A library for Reinforcement Learning in TensorFlow. https://github.com/tensorflow/agents, 2018. URL https://github.com/tensorflow/agents. [Online; accessed 25-June-2019]. 40

[48] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A Standard Interface for Reinforcement Learning Environments. *arXiv preprint arXiv:2407.17032*, 2024. 40

[49] Google Brain and TensorFlow Contributors. TensorFlow Keras API Documentation, 2015. URL https://www.tensorflow.org/api_docs/python/tf/keras. 41

[50] Hitachi Energy. RET670 - Transformer Protection, 2024. URL https://www.hitachienergy.com/in/en/products-and-solutions/substation-automation-protection-and-control/products/protection-and-control/transformer-protection/ret670. 47

[51] A. E. Fitzgerald, Charles Kingsley, and Stephen D. Umans. *Electric Machinery*. McGraw-Hill, 2003. 47

[52] Hitachi Energy. Shunt Reactors and Inductors - Hitachi Energy, 2024. URL https://www.hitachienergy.com/in/en/products-and-solutions/transformers/reactors-and-inductors/shunt-reactors. 47

[53] Pratyusa K. Manadhata and Jeannette M. Wing. An attack surface metric. *IEEE Transactions on Software Engineering*, 37(3):371–386, 2011. doi: 10.1109/TSE.2010.60. 65