APEX-ICS: AUTOMATED PROTOCOL EXPLORATION AND FUZZING FOR CLOSED SOURCE ICS PROTOCOLS

by

Parvin Kumar

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Computer Science

West Lafayette, Indiana

May 2023

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Dongyan Xu, Chair

Samuel Conte Professor of Computer Science Director, CERIAS

Department of Computer Science

Dr. Antonio Bianchi

Department of Computer Science

Dr. Dave (Jing) Tian

Department of Computer Science

Dr. Z. Berkay Celik

Department of Computer Science

Approved by:

Dr. Kihong Park

ACKNOWLEDGMENTS

It is with immense pleasure and gratitude that I express my most profound appreciation to those who have contributed to the successful completion of my master's thesis. First and foremost, I would like to express my sincerest thanks to my principal advisor, Prof. Dongyan Xu, for his invaluable guidance, support, and encouragement throughout my research journey. His brilliant insights and constructive feedback on my ideas have helped me shape my research direction and achieve my goals.

I am also grateful to my co-advisor, Prof. Dave (Jing) Tian, for his invaluable help and efforts in providing me with a broader perspective on the research problem and for his continued support throughout the thesis process. I extend my sincere thanks to my committee members, Prof. Z. Berkay Celik and Prof. Antonio Bianchi, for their guidance, insightful feedback, and valuable time in reviewing the work that has helped me to refine my thesis.

I want to express my heartfelt appreciation to my past and present colleagues in the PurSec group, Sriharsh Etigowni, Abdulellah Alsaheel, Ozgur Ozmen, Hongwei Wu, Hyungsub Kim, Sungwoo Kim, Ashwin Nambiar, Siddharth Muralee, and Prashast Srivastava. Their support, constructive criticism, and insightful discussions have been invaluable in shaping my research work and improving my research skills. I would also like to extend my gratitude to my collaborators outside Purdue, Dr. Andrei, for his insightful feedback and support, and my friends Ashish Gahlot, Abhinav Gupta, and Qian Zhang for their insightful discussions.

Finally, I would like to express my gratitude to my mother for her unwavering love, trust, and support; this journey would not have been possible without you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

A closed-source ICS communication is a fundamental component of supervisory software and PLCs operating critical infrastructure or configuring devices. As this is a vital communication, a compromised protocol can allow attackers to take over the entire critical infrastructure network and maliciously manipulate field device values. Thus, it is crucial to conduct security assessments of these closed-source protocol communications before deploying them in a production environment to ensure the safety of critical infrastructure. However, Fuzzing closed-source communication without understanding the protocol structure or state is ineffective, making testing such closed-source communications a challenging task.

This research study introduces the APEX-ICS framework, which consists of two significant components: Automatic closed-source ICS protocol reverse-engineering and stateful black-box fuzzing. The former aims to reverse-engineer the protocol communication, which is critical to effectively performing the fuzzing technique. The latter component leverages the generated grammar to detect vulnerabilities in communication between supervisory software and PLCs. The framework prototype was implemented using the Codesys v3.0 closed-source protocol communication to conduct reverse engineering and fuzzing and successfully identified 4 previously unknown vulnerabilities, which were found to impact more than 400 manufacturer's devices.

1. INTRODUCTION

1.1 Background

In the background section, the impact of cyber-attacks on the industrial control systems (ICS) field is discussed, highlighting the vulnerability of multiple layers. Additionally, the significance of automatic protocol reverse engineering components to improve the effectiveness of fuzzing is emphasized.

1.1.1 ICS environment and protocols

Industrial Control Systems (ICS) are an integral part of modern society, witnessing a rapid expansion in the era of Industry 4.0 [1]. ICS are computerized control systems that regulate a wide range of industrial sectors, including critical infrastructures [2] such as power grids, oil and gas industries, transportation, and water treatment facilities. These systems rely on the distributed control system (DCS) architecture [3], which involves the integration of information technology (IT) and operational technology (OT) networks for connectivity. A security vulnerability in ICS devices or their associated protocols can have devastating impacts on human lives and physical property, including significant revenue losses, environmental disasters, or even casualties [4]. Therefore, ensuring the security and reliability of ICS in today's interconnected world is crucial.

Vulnerabilities are flaws or weaknesses in software, hardware, or systems that attackers can exploit to compromise the security or functionality of a target [5]. These vulnerabilities can exist due to coding errors, configuration mistakes, design flaws, or other reasons [6]. Attackers often look for vulnerabilities to gain unauthorized access to sensitive data or systems, disrupt operations, steal intellectual property, or cause physical damage [7]. In the context of Industrial Control Systems (ICS), proprietary protocol and device vulnerabilities can have severe consequences, as demonstrated by high-profile attacks such as Stuxnet [4], BlackEnergy [8], and Triton [9].

- Stuxnet targeted Siemens' proprietary protocol communication, which is used in their industrial control systems. The malware was designed to exploit vulnerabilities in the

Siemens Step7 software used to program PLCs, allowing it to spread across the network and ultimately manipulate the control systems. Stuxnet's code was able to reprogram the PLCs to change the operating parameters of the centrifuges, causing them to fail and disrupting Iran's nuclear program [10].

- BlackEnergy is a well-known malware that has been used in several cyber attacks. This malware has exploited vulnerabilities in various software and hardware components, including proprietary protocols used in industrial control systems (ICS). For example, in one of its attacks, BlackEnergy exploited vulnerabilities in GEs Cimplicity HMI to execute arbitrary malicious code for firmware updates of UPS systems. Additionally, the malware also used vulnerabilities in TIA Portal [11] to read arbitrary files via crafted packets. These attacks demonstrate the importance of securing proprietary protocols and highlight the need for effective security measures to protect critical infrastructure against cyber threats [12].

- The Triton malware targeted Safety Instrumented Systems (SIS) controllers used in critical infrastructure. It used the TriStation proprietary protocol to interact with the controllers, including reading and writing programs and functions. The malware injected custom code into the memory of the SIS controllers, allowing it to manipulate their behavior and potentially cause physical damage or harm. Triton's attackers used spear-phishing emails and social engineering tactics to gain initial access to the network. They then conducted extensive reconnaissance and moved laterally within the network to identify and target the SIS controllers. Once the controllers were infected, the attackers had the ability to issue commands and take control of the systems remotely. This attack highlights the importance of securing not only traditional IT systems but also critical infrastructure and operational technology [13].

In an ICS environment, there are three levels of targets: supervision level, control level, and field level [14], as shown in figure 1.1. The supervision level includes management software such as Human Machine Interface (HMI) [15], Configuration Software [16], and a historian server [14]. Configuration Software is also known as supervisory software, as it

allows for the configuration and management of devices at the control level. The control level includes devices such as PLC (Programming Logic Controller) [17], RTU (Remote Telemetry Unit) [18], and PAC (Programmable Automation Controller) [19]. These devices are responsible for directly controlling physical processes at the field level. The field level includes devices such as pumps, motors, and boilers, which are directly involved in controlled physical processes. However, for the purpose of ICS attacks, only the supervision and control level are typically targeted. Attackers often focus on the control level [20], as compromising

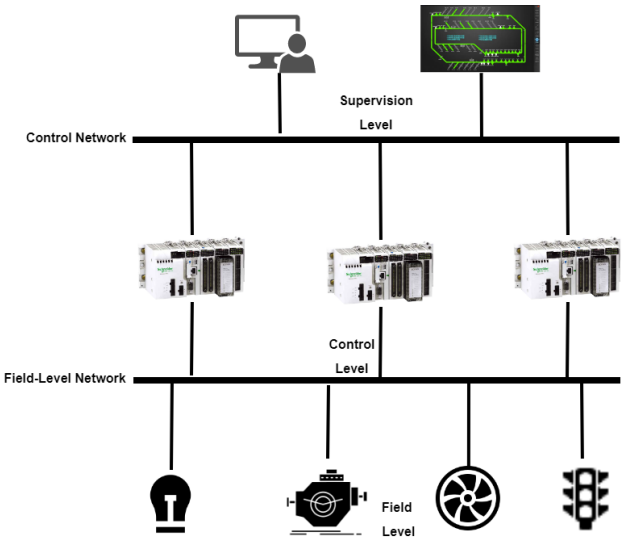

Figure 1.1. A typical ICS network layer architecture

these devices allows direct control of physical processes. However, the supervision level is also targeted as it often contains critical management software that can be used to gain access to supervision and the control level [21].

In the past, the development of industrial control system (ICS) devices and their associated software has been largely shrouded in secrecy due to the proprietary nature of the technologies [22]. This has made it difficult for third-party security researchers to analyze for vulnerabilities, leading to a lack of comprehensive security testing and patching. However, with the increasing computing power of ICS devices and the move towards generic frameworks for both device runtime and engineering workstations, many device vendors are starting to white-label and use these generic frameworks [23]. One such framework is the Codesys runtime [24], which is used by around 80 industrial device vendors across a range

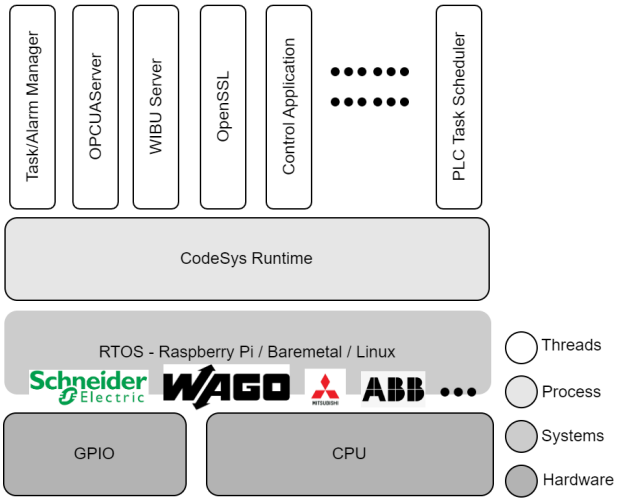

Figure 1.2. Describes system stack for Codesys-based PLCs.

of over 400 devices, including major manufacturers such as Schneider-Electric Automation [25], Wago [26], Mitsubishi [27], and ABB [28]. As a result, the transition to generic OEM runtime frameworks is expected to continue to grow in popularity, with current estimates suggesting that Codesys devices alone make up at least 20% of the active PLCs worldwide [29].

1.1.2 Protocol reverse engineering

The task of reverse engineering network protocols poses a significant challenge in the realm of cybersecurity. It is crucial to clearly comprehend the protocol's semantics to employ various security analysis techniques such as malware analysis, vulnerability analysis, penetration testing, and fuzzing. These techniques are used to test the security robustness of the protocol. [30]. Without the protocol's ground truth, it becomes difficult to accurately detect and mitigate attacks using intrusion detection systems (IDS) [31], intrusion prevention systems (IPS) [32], and security information and event management (SIEM) tools [33]. Therefore, the ability to reverse engineer network protocols and extract their semantics is paramount for effective security analysis and threat mitigation in industrial control systems (ICS) and other critical infrastructure environments.

Understanding the semantics of a protocol is crucial for malware analysis to detect potentially malicious activity and create detection signatures. For instance, if malware is using a protocol to communicate with a command and control server, comprehending the protocol's semantics can aid in identifying unusual traffic patterns and recognizing the specific command and control messages [34]. Also, in the case of fuzzing, without a clear understanding of the protocol's semantics, generated test cases may be invalid or lack coverage of important protocol features. For example, imagine a protocol with specific fields used to control the system's behavior. If these fields are not understood, test cases may not accurately exercise them, which could result in vulnerabilities going undetected. Additionally, fuzzing without knowledge of the protocol's grammar could lead to test cases that do not conform to the protocol's syntax and therefore fail to provide adequate coverage.

Attacking proprietary protocols used in industrial control system (ICS) environments can have serious consequences for the security of these systems. One of the primary challenges is that proprietary protocols are developed by a single vendor, which can make them less transparent and difficult for security researchers to analyze for vulnerabilities. This lack of transparency can result in a lack of comprehensive security testing and patching, leaving the systems vulnerable to exploitation. Therefore, it is important to have security measures in place that allow for identifying and analyzing vulnerabilities in proprietary protocols to ensure the security of ICS environments. For instance, a TriStation protocol is developed by Safety Instrumented Systems (SIS), Schneider Electric develops UMAS [35], Codesysv3.0 is from CODESYS Group [36], and S7Comm is from Siemens [37]. These protocols are proprietary and are not publicly available, as they are developed and owned by specific vendors. This lack of transparency can lead to challenges in understanding the protocol's specifications, potentially leaving them vulnerable to security vulnerabilities and attacks.

1.1.3 Fuzzing

Fuzzing, also known as fuzz testing or mutation testing, is a software testing technique that involves feeding unexpected, random, or invalid data as inputs to a program to detect vulnerabilities, software defects, or crashes [38]. The idea is to simulate real-world scenarios

and attack vectors to find security flaws in software. Fuzzing is particularly useful for identifying and testing complex communication protocols, which are often used in network-based applications and embedded systems [39]. By discovering and fixing these vulnerabilities, fuzzing helps improve software applications' overall security and reliability.

Fuzzing proprietary communication protocols is challenging due to the lack of publicly available specifications and the proprietary nature of these protocols. This makes it difficult to generate input data that adheres to the protocol's specifications, leading to invalid or incorrect data being sent to the target system [40]. In addition, proprietary protocols may have complex or custom message formats, making identifying and mutating the correct fields harder. To address these challenges, a suitable grammar that accurately represents the message format of the proprietary protocol is necessary to generate valid input data for the fuzzing process. Without suitable grammar, the fuzzing process may not be effective or may cause the target system to crash or reset, resulting in potential downtime and loss of revenue for the organization.

There are several specialized fuzzing frameworks developed for testing Industrial Control System (ICS) network protocols. These frameworks utilize different techniques and tools to improve the effectiveness of the fuzzing procedure.

Pulsar [41] is a stateful black-box fuzzing technique that aims to test ICS proprietary network protocols by using the PRISMA [42] tool for automatic protocol reverse engineering. It simulates server-role and focuses on testing the interactions between client and server. Polar [43] uses static and dynamic analysis to identify vulnerable operations and applies semantic-aware mutation to improve the fuzzing procedure. AFLNET [44] is a grey box fuzzer that uses a mutated corpus of recorded network messages and state feedback to guide the fuzzer. However, it requires program source code for instrumentation and only works with server-role programs. Peach* [45] is a coverage-based improvement over the standard Peach fuzzer that targets ICS network protocols to identify vulnerabilities and improve testing coverage. Lastly, ICS3Fuzzer [46] is a framework that focuses on discovering implementation bugs on supervisory software by fuzzing the network communication protocol used to communicate with the field devices. It synchronizes the controls of the GUI operation and network communications to fuzz the entire supervisory software, but it does not fuzz PLC devices.

Table 1.1. Comparison with existing protocol fuzzing frameworks.

Subject	Closed binary	Client-role	Proprietary-protocols	Coverage-guide	Store new response
Pulsar	✓	✓	✓	✗	✗
Polar	✗	✗	✗	✓	✓
ICS3Fuzz	✓	✓	✓	✗	✗
Peach*	✗	✗	✗	✓	✓
AFLNET	✗	✗	✗	✓	✓
APEX-ICS	✓	✓	✓	✓	✓

APEX-ICS [47] is a fuzzing framework for proprietary communication protocols, comprising two key components: 1) automatic protocol reverse engineering and 2) grammar-based mutation fuzzing. The automatic protocol reverse engineering component extracts the semantics and structure of the proprietary communication protocol by analyzing the network traffic of the protocol captured through a packet capture (PCAP) file as shown in the fig 3.1. The extracted protocol structure is used to generate a suitable grammar for the protocol. The comparison with above mentioned related work is shown in 1.1.

APEX-ICS's fuzzing approach for testing proprietary communication protocols consists mainly of four key components as shown in fig 3.4. Firstly, input generation uses a reverse engineering framework to generate valid input data that adheres to the protocol's specifications, reducing the risk of system crashes. Secondly, input mutation utilizes Radamsa to mutate the input data generated by the RE framework. Thirdly, target selection allows for selecting specific components within the target system to be fuzzed, reducing the risk of false positives and unnecessary testing. Finally, the approach employs a feedback loop for monitoring target system responses during fuzzing and categorizes them based on validity for further processing by different components. It also utilizes a hash table to prioritize mutation algorithms for testing new paths and states discovered during the fuzzing process.

Together, these two components of APEX-ICS work to achieve effective and efficient fuzzing of proprietary communication protocols, allowing for discovering potential vulnerabilities and weaknesses in the protocol implementation.

1.2 Problem Statement

This research study addresses the challenge of automatic protocol reverse engineering, which is a necessary step to start meaningful fuzzing of proprietary communication protocols. While many tools and techniques have been developed to tackle this problem, they have not been sufficient (discussed in detail in section 1.3) from the perspective of fuzzing for Netzob [48], Discoverer [49], and Netplier [50]. Two major techniques have been used in the past to generate suitable grammar for fuzzing.

The first technique is leveraging the binary, which can provide a more accurate analysis of how the input buffer is accessed (e.g., PISE [51], Prospex [52], Polyglot [53] and Context-Aware Monitored Execution [54] etc.). Dataflow analysis [55], Angr [56], and other tools and techniques can be used to analyze the binary and extract the necessary information. However, this technique is not always feasible because it requires access to the binary. In some cases, the Programmable Logic Controller (PLC) firmware may be protected or obfuscated, making it difficult to conduct dynamic analysis. Even if the binary for the client application is available, the counterpart on the server side may be much more difficult to acquire. The second technique is using network traces to reverse engineer the protocol, and it is helpful in cases where access to the binary is not possible. However, it can be challenging to extract suitable grammar from network traces because they do not contain information about the structure of the protocol. Additionally, manual analysis of network traces can be time-consuming and error-prone.

To overcome these challenges, this research study proposes a novel approach that leverages both techniques to extract suitable grammar for fuzzing proprietary communication protocols. The automatic reverse engineering framework extracts information from network traces and uses it to generate valid input data adhering to the protocol's specifications. The APEX-ICS framework utilizes a feedback loop to monitor the target system's response during the fuzzing process and categorizes them based on their validity. The approach also uses a hash table to track new states or paths discovered during the fuzzing process, ensuring that the mutation algorithm is prioritized towards testing these new areas.

Overall, this research study presents a comprehensive approach to automatic protocol reverse engineering that overcomes the challenges posed by proprietary communication protocols. The approach uses techniques to extract suitable grammar for fuzzing, making it more efficient and effective than previous methods.

1.3 Research Statement

There are many studies have been done to extract grammar for proprietary protocol communication using different techniques such as leveraging alignment-based algorithms (PIP, ScriptGen [57], Netzob), token-based (Veritas [58], Discoverer), lastly, using multiple sequence alignment (Netplier). This study picked one prominent tool from each category to analyze these tools. Here following are the limitation of the above-mentioned tools.

1. Alignment based: Netzob [48] is a protocol reverse engineering tool using alignment-based clustering methods to group similar messages. This clustering method assumes that messages are of the same type if they have similar sequences of values. However, this assumption may not always hold true because messages of the same type can have different values for the same fields. Similarly, messages of different types can have some standard fields and the same values. Due to this limitation [50], Netzob may not always produce accurate results and may misclassify messages.

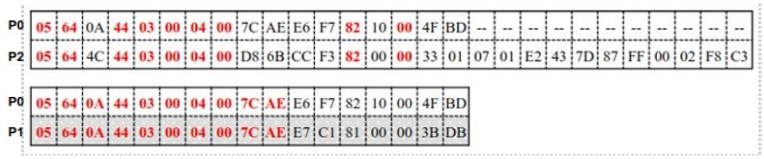

Figure 1.3. Issues with Netzob, which resulted in incorrect clustering.

For instance, in some cases, a single message may contain multiple values or fields, some of which may be unique to that message type. In such cases, Netzob may struggle to cluster messages accurately. Additionally, Netzob may be unable to identify the boundaries between messages in a communication stream, which can result in incorrect clustering.

To overcome these limitations, this study proposed a novel technique that aims to improve the accuracy and efficiency of clustering methods to extract grammar using automatic protocol reverse engineering. The reverse engineering component of APEX-ICS has been used to improve the accuracy of clustering; it also helps to identify the boundaries between messages, enabling more accurate clustering.

2. Token based: Discoverer [49] is a state-of-the-art token-based protocol reverse engineering method that initially conducts clustering based on token type patterns. It considers consecutive bytes with printable ASCII values as a text token, as it has been observed that messages of the same type usually have similar mixtures of binary sequences and textual strings. The resulting tokenization of a network traffic trace is illustrated in Figure 1.3a [50], where the differences between the two patterns are highlighted in red. Based on this tokenization, Discoverer produces two initial clusters, as shown in Figure 1.3b [50]. It then further divides each cluster into sub-clusters by values of potential representative and ultimately produces four clusters, as shown in Figure 1.3c [50]. However, Discoverer has limitations, as each ground-truth type is sub-optimally divided into two smaller types or clusters, which can affect the accuracy of the reverse engineering process.

3. Multiple sequence alignments (MSA): NetPlier [50] followed a packet alignment and field identification approach. However, its process for determining protocol keywords was based on introducing random variables to calculate the likelihood of a field being a keyword. This approach was probabilistic in nature and relied on determining the most likely fields to be protocol keywords. While this approach may have been helpful in some contexts, it may not have been suitable for certain applications, such as protocol reverse engineering for fuzzing. Choosing key fields randomly could potentially lead to fuzzing inputs that do not accurately reflect the target system's behavior, thus compromising the efficacy of the fuzzing process.

   Netplier uses Multiple sequence alignment (MSA), a commonly used method for aligning and clustering protocol messages. However, this method has limitations in the context of protocol reverse engineering. First, MSA methods are computationally in-

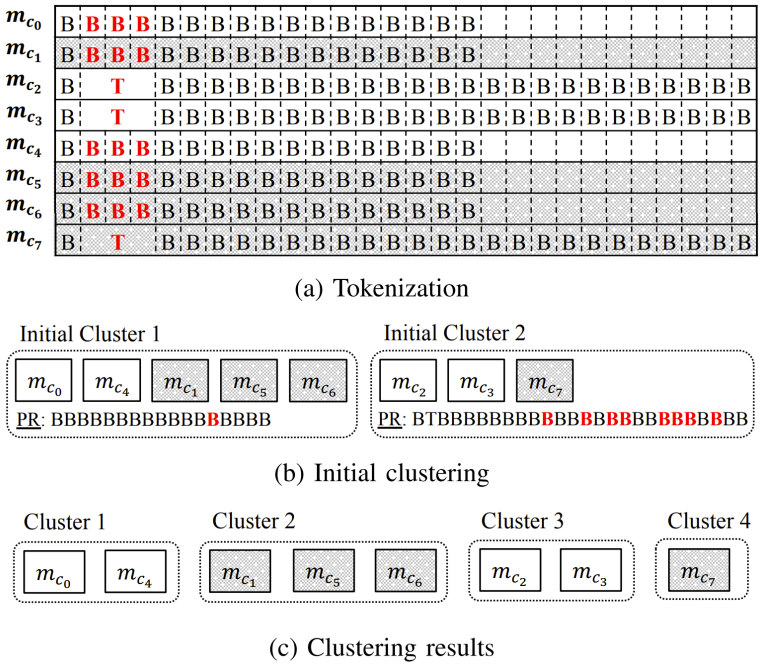

Figure 1.4. Issues with Discoverer, which resulted in incorrect clustering for similar tokens.

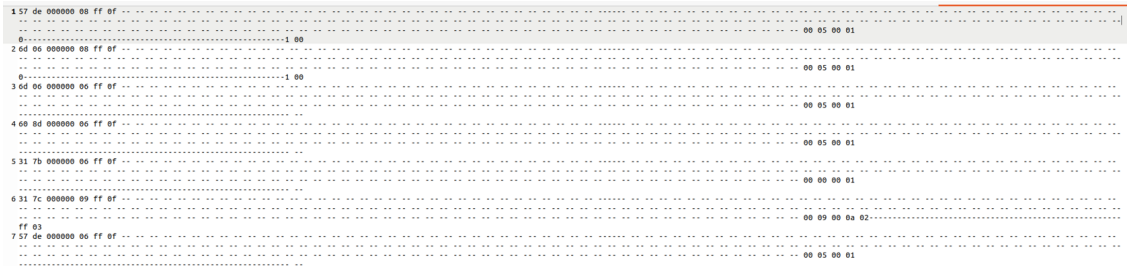

Figure 1.5. Issues with Netplier, which resulted in incorrect grammar generation

tensive and may not be feasible for large datasets or real-time analysis. Additionally, MSA methods require a large amount of memory, which can be a limiting factor for resource-constrained systems. Second, MSA methods require many sequences to produce accurate alignments, which can be difficult to obtain for rare or infrequently used protocols. Furthermore, MSA methods rely on the assumption that sequences of the same type share a common structure, which may not always be the case for protocols with dynamic message formats. Third, MSA methods can produce inaccurate results

when dealing with sequences of different lengths or with gaps in the sequences. This is a common problem in protocol reverse engineering, where message lengths can vary, and message structures may include optional or variable-length fields.

Overall, while MSA methods can be useful for protocol reverse engineering, they have limitations that must be addressed to produce accurate and reliable results.

The proposed approach for generating a protocol grammar for fuzzing involves several steps as mentioned in architecture diagram 3.3. First, it uses the Levenshtein algorithm [59] to calculate the similarity score for each column of the packet as shown in formula 3.2, byte by byte. Based on these similarity scores, it identifies probable key fields. Along with this, it also identifies static and dynamic fields that are essential for fuzzing.

Once the probable key fields are identified, it runs a clustering algorithm to group similar types of packets together. We then calculate similarity scores, intra-scores, inter-scores, and length variance for each cluster, repeating this process for each cluster. This helps to understand each cluster's characteristics better and identify any patterns that may emerge.

In the next phase, APEX-ICS calculates a joint probability for each cluster and declares the final key field (the key field determines the message type) with the highest score. This approach generates a more accurate and complete protocol grammar for fuzzing, addressing the limitations of other tools such as Netzob, Discoverer, and Netplier. By leveraging the strengths of these above-mentioned algorithms and techniques in section 1.3, this proposed approach can overcome the challenges of generating a protocol grammar for proprietary communication protocols, ultimately leading to more effective and efficient fuzzing.

1.4 Assumptions

The assumption that filtered network traces are given to the framework to generate grammar is reasonable, as it is a common practice in the field of protocol reverse engineering. However, it is important to note that the quality and quantity of the network traces can significantly impact the accuracy and completeness of the generated grammar.

The assumption that users did not generate network trace files with many features simultaneously is also reasonable because it is unlikely that a typical user would generate complex

network traffic with a multitude of features simultaneously, for instance, resetting the PLC, upgrading firmware, etc. It is important to understand the structure of the protocol being analyzed clearly. Trying to reverse engineer multiple features simultaneously can lead to confusion and errors in the generated grammar. It is generally better to focus on one feature at a time and generate a grammar that accurately reflects its structure.

Overall, while these assumptions may limit the applicability of the approach to certain scenarios, they are reasonable and necessary to ensure the accuracy and effectiveness of the protocol reverse engineering process.

1.5 Limitation

One potential limitation of the RE component's approach is its reliance on network traces to generate the protocol grammar. The resulting grammar may not accurately reflect the protocol's specifications if the network traces do not fully capture the system's behavior. Additionally, if the network traces are not representative of the entire range of possible inputs, the generated grammar may not be comprehensive enough to cover all possible scenarios during fuzzing.

Another potential limitation is the reliance on the Levenshtein algorithm to calculate similarity scores for identifying probable key fields. While the algorithm is effective in many cases, there may be instances where it fails to accurately identify key fields due to variations in the data or other factors. Additionally, the approach may not be able to handle protocols with complex structures or non-standard formats. Finally, this approach is designed to work specifically with ICS protocols. It may not be applicable to other types of protocols or communication systems. The fuzzing module of APEX-ICS could be the effectiveness of the random mutation algorithm used for input mutation. While the approach prioritizes the mutation algorithm based on the hash value of previously generated test cases, there is still a possibility that the algorithm may not effectively mutate the input data in a way that effectively tests all possible paths and states of the target system. Additionally, the approach's effectiveness may also depend on the complexity and variability of the proprietary protocol being tested and the specific components being fuzzed. Therefore, further testing

and evaluation of the fuzzing module's effectiveness may be necessary to assess its limitations fully. During the fuzzing process, there is a possibility that the generated test cases can violate the security policy or cause operational issues [60]. These issues can range from shutting down the system to triggering false alarms or causing data corruption. Detecting these types of security issues isn't covered in this study. To address this limitation, observing the supervisory system during the fuzzing process is important.

1.6 Delimitation

The delimitations of the RE component's approach include the following:

- Limited input generation capabilities: While RE component generates input data adhering to the protocol's specifications, it may still have limitations in terms of the range and complexity of input data it can generate. This could potentially limit the effectiveness of the tool in detecting certain types of vulnerabilities or attacks.
- Dependence on network traces: The effectiveness of RE component is largely dependent on the quality and quantity of the network traces. Insufficient or poor-quality traces can lead to the generation of inaccurate grammar and, therefore, inaccurate testing results. For example, if the network traces are captured with a low sampling rate or are filtered with only a subset of packets, then it may not be able to capture the entire grammar.
- Limited coverage of protocol states: APEX-ICS relies on a feedback loop to monitor the target system's response during the fuzzing process and track new states or paths discovered. It makes efforts to identify new states to get coverage. But still, there may still be limitations regarding the range of protocol states that can be covered. To address this issue, the key field can be chosen for fuzzing to cover all hidden states. Note that APEX-ICS is only focusing on those key fields only which were generated by the RE component.

To effectively use APEX-ICS, after generating the grammar, it is recommended that the user has a basic understanding of the network trace and the dynamic fields, such as

checksums, session IDs, and SYN numbers. By understanding these fields well, the user can make informed decisions about which fields to fuzz and which to leave untouched. This knowledge can greatly improve the effectiveness and efficiency of the fuzzing process. In this sense, understanding the network trace is not a limitation but rather a valuable asset that can help the user achieve better results in fuzzing.

1.7 Contribution of the study

The contributions of this study can be summarized as follows:

1. Automatic Protocol Reverse Engineering: This study proposes a novel automatic protocol reverse engineering approach that leverages network traces to extract suitable grammar for fuzzing. The approach uses a token-based clustering and Levenshtein distance algorithm to calculate the similarity score for each column byte by byte to identify probable key fields. The approach also identifies static and dynamic fields that are essential for fuzzing.

2. Fuzzing Module: The study proposes an efficient and comprehensive fuzzing module for proprietary protocols used in industrial control systems. The approach includes input generation, input mutation, target selection, and monitoring. The input generation uses an automatic reverse engineering framework to extract a suitable grammar for the proprietary protocol under test, which is then used to generate input data for the fuzzing process. The input mutation uses a random mutation algorithm based on Radamsa and is prioritized based on the hash value of previously generated test cases. The approach allows for the selection of specific components within the target system to be fuzzed, reducing the risk of false positives and unnecessary testing. The monitoring includes a feedback loop that categorizes responses based on their validity and tracks new states or paths discovered during the fuzzing process.

3. Experimental Evaluation: The study provides an extensive experimental evaluation of the proposed approach using four different protocols from industrial control sys-

tems. The evaluation includes a comparison with state-of-the-art tools and techniques, demonstrating the effectiveness and efficiency of the proposed approach.

4. The contribution of this study is further exemplified by the discovery of 4 previously unknown vulnerabilities in the CodeSys v3.0 proprietary protocol using the APEX-ICS approach, which was found to impact more than 400 manufacturers devices. This highlights the approach's effectiveness in identifying security vulnerabilities in industrial control systems, which can have serious consequences if exploited by attackers.

The contributions of this study provide a comprehensive and efficient approach to automatic protocol reverse engineering and fuzzing of proprietary protocols used in industrial control systems, with significant potential for enhancing the security and resilience of these systems.

2. LITERATURE REVIEW

This section presents and discusses related work that is relevant to the study. Significant research has been conducted on automating fuzzing and optimizing the process, and several well-known fuzzers that are useful in the ICS domain will be discussed. As ICS protocols differ significantly from standard IT protocols, they require in-depth analysis before they can be correctly fuzzed. Therefore, the section will also cover relevant work done in the area of reverse engineering protocols. The study will investigate the results of these studies and discuss the issues pertinent to the current study. It is essential to highlight the gaps and issues these studies aim to address and find solutions to existing problems in the field of ICS security.

2.1 Reverse Engineering Frameworks

This section provides an in-depth analysis of the approach and limitations of the popular reverse engineering frameworks used to analyze Industrial Control Systems (ICS) protocols.

2.1.1 Netzob

Netzob [48] is an open-source tool for reverse engineering, traffic generation, and fuzzing protocols. The security auditors of AMOSSYS and the CIDRE research team of Supélec developed it. One of the first studies used network traffic to analyze the protocol structure. Netzob supports different protocols like text, delimiter-based, fixed, and variable-length field protocols. It examines the protocol using PCAPs, live communications, or text files as input. Based on the type of protocol to be fuzzed, it splits the data utilizing a delimiter or byte-by-byte and then analyses each protocol field to be static or dynamic. Once it analyzes the field type, it can further cluster similar messages together by choosing one of the fields as the required field.

Although Netzob has a function to cluster using the required field, it has no logic for choosing the best key area from all the probable key fields. To generate static and dynamic fields, NetZob has parameter options to decide whether to merge them. However, it does not

have the smartness to determine whether to join or not based on the payload values, which is essential to understand the protocol structure correctly. This indicates that the user needs to have some basic understanding of the protocol to analyze it accurately and may not give accurate results for a completely unknown protocol, as shown in figure 1.3.

2.1.2 Netplier

NetPlier is a tool for protocol reverse engineering; it uses network traces as input and infers the keyword by multiple sequence alignment and probabilistic inference. Netplier addresses a key challenge in network protocol reverse engineering keyword identification, which allows the clustering of network messages correctly and enables high precision in downstream analysis, such as field identification and state machine reconstruction. They formulate keyword identification as a probabilistic inference problem, allowing them to model the inherent uncertainty naturally. It takes network traces as input and produces the final message format.

Netplier mainly uses advanced techniques to cluster the messages based on the MAFFT tool's alignment [61] in generating the fields. It functions mainly like Netzob for generating the fields, i.e., byte-by-byte analysis. The key-field inference logic combines multiple ideas to identify the final key field, making it widely accurate. However, the output provided by Netplier is not very intuitive in terms of having all the needed fields to start fuzzing (except key fields). Also, it does not provide the final key field used for clustering, as shown in figure 1.5. **APEX-ICS is designed to furnish all the essential protocol fields in addition to the identified key fields, whereas Netplier only supplies the key fields. In other words, APEX-ICS offers a comprehensive range of protocol fields along with the necessary key fields, whereas Netplier only provides the essential key fields.**

2.2 Fuzzing Frameworks for ICS

This section aims to examine the approaches and limitations of existing fuzzing frameworks for industrial control system (ICS) protocols. Various approaches have been developed

to automatically reverse-engineer and fuzz protocols, with the goal of generating adequate inputs based on protocol formats and states. However, as mentioned in 1.1, many of these approaches have limitations. For example, some of these approaches rely on access to source code, making them less useful for closed-source systems. Additionally, many of these approaches only handle server-role programs, which limits their usefulness in testing client-role programs. Such limitations can be observed in studies like Polar, AFLNET, and Peach*.

It is worth noting that the limitations of existing fuzzing frameworks can pose significant challenges to testing ICS protocols effectively. However, ongoing research continues to explore alternative approaches to overcome these limitations and improve the effectiveness of fuzzing for ICS protocols.

2.2.1 ICSFuzz

The study focuses on the security evaluation of industrial control systems (ICS) and Programmable Logic Controllers (PLCs), which have seen a rapid proliferation in the last decade with the advent of the 4th Industrial Revolution [62]. The paper highlights the lack of depth in the security evaluation of industrial devices such as PLCs and the threats from the information technology domain that can be readily ported to industrial environments. The paper presents a detailed analysis of the composition of control applications based on all available IEC 61131-3 languages, highlighting the unique characteristics and intricacies introduced by different languages and compiling tools. The authors develop a fuzzing framework for PLC applications to uncover existing binary vulnerabilities that would lead to crashes or exploitation and extend the framework to include system functions that belong to the host software of the PLC application. To prove the correctness of their fuzzing tool, the authors use a database of in-house developed binaries (a collection of vulnerable PLC binaries which can be used by future researchers in industrial control systems security) and functional control applications collected from online repositories. They showcase the efficacy of their technique by demonstrating uncovered vulnerabilities in control application binaries and their runtime system. Furthermore, they demonstrate an exploitation methodology for an in-house as well as a regular control binary based on the uncovered vulnerabilities. This study highlights the

usefulness of fuzzing as a tool for uncovering vulnerabilities in industrial control systems, even in the presence of heavy I/O and scan cycles. Overall, the paper presents a comprehensive study on the feasibility of exploiting PLC binaries and their surrounding environment in industrial control systems and provides a valuable contribution to the field of industrial control systems security.

As described above, this study uses in-house developed IEC application binaries to detect vulnerabilities; however, it only interacts with a few components. It also has several issues; Firstly, it is limited by using the KBUS subsystem for input delivery to the control application, necessitating using a physical WAGO PLC. This limits its scalability as it cannot be used to fuzz multiple PLCs simultaneously, and also, a few more vendors support Codesys. Secondly, due to the loss of synchronization with the scan cycle of the control application, ICSFuzz periodically drops fuzzing inputs, resulting in slower performance. This can be a significant drawback, especially when testing large and complex control systems. Thirdly, ICSFuzz lacks automation for observing the state of the control program, which means that crash monitoring is manual. This limitation can increase the time and effort required to identify vulnerabilities in the control system. Lastly, ICSFuzz performs limited stateless fuzzing of shared library functions, specifically those of the Codesys runtime on WAGO PLC. Although it can discover some crashes, the stateless and out-of-context nature of the fuzzing means it can miss vulnerabilities that require the execution context of the runtime. Overall, these limitations underscore the need for a more scalable and automated fuzzing tool that can effectively identify vulnerabilities in industrial control systems.

2.2.2 ICS3Fuzzer

The ICS^{3}Fuzzer [46] framework is designed to discover vulnerabilities in the communication protocols between supervisory software and field devices in industrial control systems (ICSs). One of the main challenges in fuzzing ICS supervisory software is using proprietary protocols, making it difficult to enter deep states for effective fuzzing without extensive reverse-engineering efforts. To overcome this challenge, the ICS^{3}Fuzzer framework constructs a state book based on the readily-available execution trace of the supervisory

software and corresponding inputs. The state selection algorithm then identifies the protocol states that are more likely to contain bugs, distributing the fuzzer's budget to these states to increase the chances of discovering vulnerabilities. To reach these interesting states quickly, the ICS^{3}Fuzzer framework synchronously manages external events, such as GUI operations and network traffic during the fuzzing loop. This approach addresses the challenge of time-sensitive communication protocols, which may not work effectively with traditional snapshot-based methods. The ICS^{3}Fuzzer framework has been implemented as a prototype and has been used to fuzz the supervisory software of four popular ICS platforms. In this testing, the framework identifies vulnerabilities and affects different products. Overall, the ICS^{3}Fuzzer framework is a customizable and effective tool for detecting protocol implementation vulnerabilities in different supervisory software from various vendors. By building a communication template to simulate the session based on captured messages and using automated synchronization of network and GUI behavior, the framework can reach any input state and feed mutated inputs directionally, making it a powerful tool for improving the security of ICSs.

The proposed ICS^{3}Fuzzer framework has several limitations that should be taken into account. Firstly, it is tailored specifically for the supervisory software, not the PLCs. Additionally, synchronizing GUI events with network traffic requires manual effort and can be time-consuming. The framework also requires a lot of computational resources, which may limit its scalability to larger-scale ICS systems. Moreover, the input generation strategy used by the framework may not be able to generate all possible inputs, which could limit the effectiveness of the fuzzing process. The state selection algorithm used by the framework may not cover the entire state space, which could result in the framework missing some vulnerabilities. Additionally, the framework may miss some vulnerabilities due to limitations in the input generation strategy and the state selection algorithm, resulting in false negatives. Finally, the framework may not be easily generalized to other types of supervisory software or ICS platforms, which could limit its applicability to a wider range of systems.

2.2.3 Pulsar

Pulsar [41] is a stateful black-box fuzzing technique that is designed to test proprietary network protocols. It leverages the PRISMA tool to gain insight into the protocol grammar and then uses this knowledge to fuzz the protocol based on the extracted grammar. PRISMA uses a token-based clustering algorithm that splits messages into tokens based on predefined delimiters or n-grams. It then searches for the tokens with the most frequent values, which are then used to cluster similar messages together. This clustering approach helps identify patterns and structures within the protocol, which can be used for further analysis and testing.

One limitation of Pulsar is that it does not involve synchronized control and states during the fuzzing loop. This may result in some limitations in detecting certain types of vulnerabilities or in capturing protocol states that may lead to more efficient fuzzing.

2.2.4 Other Fuzzing Frameworks

For the literature review, other fuzzers were also studied based on the source-code instrumentation, such as Peach* [45], AFLNET [44], and Polar [43]. These studies, along with the above-mentioned fuzzers are covered in section 1.1.3 and also compared (in table 1.1) with APEX-ICS.

3. METHODOLOGY

This section of the document provides an in-depth insight into the methodology adopted by APEX-ICS for automatic reverse engineering and fuzzing. Reverse engineering is a process that involves analyzing a system or component to understand how it works and how it can be altered or exploited. The reverse engineering component of APEX-ICS is responsible for generating the protocol grammar by analyzing the proprietary communication protocol. The generated grammar is then used in the fuzzing component for test case generation.

Fuzzing, on the other hand, is a software testing technique that involves the input of invalid, unexpected, or random data into a system or component to discover potential vulnerabilities. The fuzzing component of APEX-ICS utilizes various mutation algorithms, including bit-flip, byte-flip, arithmetic mutation, and random mutation, to generate test cases that cover a broad range of possible input scenarios.

The APEX-ICS framework integrates both reverse engineering and fuzzing components to automate the process of vulnerability discovery in the proprietary communication protocol. By combining these two techniques, the framework can generate more diverse and effective test cases, leading to the identification of previously unknown vulnerabilities.

3.1 Automatic Protocol Reverse Engineering

To get a holistic view of the protocol for fuzzing, it is crucial to analyze it in multiple ways, including the protocol binary and the payload transfer done using the protocol. The framework is designed and developed to handle these different approaches and provide a platform to fuzz the required protocol smartly. With regard to network flow analysis, the framework can parse a protocol as an unknown protocol to infer the protocol semantics. This section describes the detailed design architecture for each of the flows that have been used to develop the framework. Fig: 3.1 shows the details of the protocol reverse engineering framework.

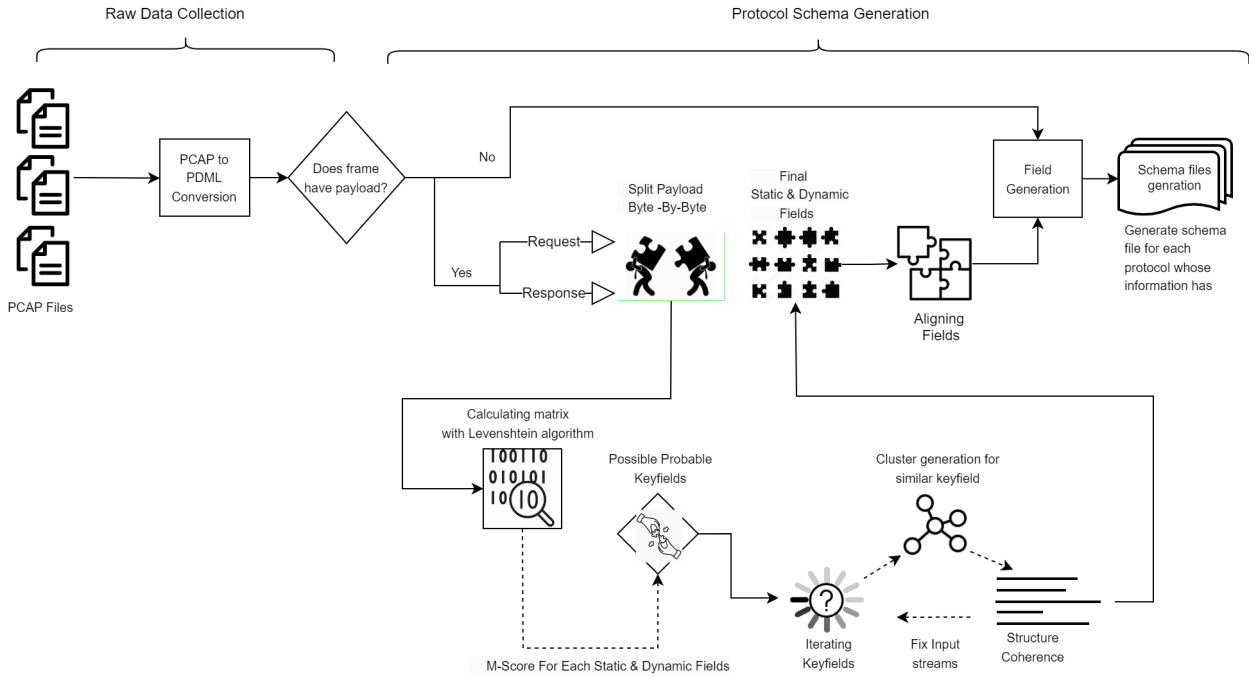

Figure 3.1. Architecture for Proprietary Protocol Reverse Engineering

3.1.1 Input Parsing

The APEX-ICS framework uses a PCAP file and the port number on which the proprietary protocol is expected to run to extract the payload. The PCAP file should contain the packet captures of the protocol to be analyzed. Once the input is provided, the framework parses the protocol and filters out the TCP payloads of communication with the specified port number. This enables the framework to accurately analyze the grammar and structure of the protocol's payload.

3.1.2 PCAP to PDML conversion

The PCAP file provided as input is converted to a PDML file to extract the payload pertaining to the required protocol. A PDML file is treated as an XML file and can be easily parsed in Python using various parsers.

3.1.3 Proprietary protocol parsing

This section discusses the detailed flow of parsing a proprietary protocol. It is assumed that the tool does not have any prior knowledge of the protocol being parsed.

Step 1: Payload Filtration

To conduct a detailed grammar analysis of a proprietary protocol communication, it is important to filter the protocol's payload before further analysis. This is done by generating a PDML XML file, which contains the entire payload of the communication. However, the generated file includes other network traffic as well, so it is necessary to filter out the TCP payloads, including communication with the necessary port, to obtain the details of the protocol communication. In addition, it is crucial to separate out the request and response packets, as the grammar and structure of the two tend to differ significantly. If the packets are not separated, it may result in inaccurate grammar analysis.

Step 2: Field-type Identification

After filtering the protocol payload, the next step in the protocol reverse engineering process is to identify the various fields within the protocol. This is done by performing an in-depth analysis of the payload, focusing on the request/response packets. The process of field-type identification is carried out in multiple stages. Each stage involves analyzing the payload more granularly, extracting relevant information, and identifying the corresponding fields in the protocol:

1. **Byte-by-byte analysis:** In fuzzing, identifying static and dynamic fields in the protocol payload is crucial for generating effective and meaningful test cases. The first step in the process is to perform a byte-by-byte analysis of the payload. Each byte is analyzed, and based on its value, it is categorized as either static or dynamic:

    - **Static:** Static fields refer to fields that have a constant value across all packets. These fields do not change, and their values remain the same regardless of the message content. For example, fields such as protocol version, packet length, and

message type are typically static fields. Since the values of static fields do not change, it is important to ensure that these fields are properly initialized in the test cases.

- **Dynamic:** Dynamic fields, on the other hand, refers to fields that can have multiple values within a packet or across different packets. These fields can include message IDs, sequence numbers, and timestamps. Since dynamic fields can have multiple values, it is important to ensure that the test cases generated cover as many values as possible to increase the chances of detecting vulnerabilities.

Identifying static and dynamic fields in the protocol payload is an important step in fuzzing because it helps to identify the fields that need to be fuzzed. By focusing on dynamic fields, the test cases can cover a wider range of possible values, increasing the chances of detecting vulnerabilities. On the other hand, by identifying static fields, the test cases can ensure that the basic functionality of the protocol is tested and validated.

2. **Similarity score calculation:** The similarity score calculation is performed to identify the field type for the protocol. The Levenshtein distance is a widely used method for calculating the similarity score between two strings. The score is between 0 and 1, where 1 means the same values. In this case, it is used to calculate the similarity score between the values of the dynamic fields.

   The Levenshtein distance is calculated by counting the number of edits required to transform one string into another. The edits include insertion, deletion, and substitution of a single character. The formula used to calculate the Levenshtein distance is shown in Figure 3.2.

   Fig. 3.2 [63] shows the formula used to calculate Levenshtein distance.

   The importance of calculating the Levenshtein distance lies in its ability to identify the variations in dynamic fields, which are essential for fuzzing. Fuzzing involves sending input data to the system under test with the goal of finding vulnerabilities.

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\big(\operatorname{tail}(a), \operatorname{tail}(b)\big) & \text{if } a[0] = b[0], \\ 1 + \min \begin{cases} \operatorname{lev}\big(\operatorname{tail}(a), b\big) \\ \operatorname{lev}\big(a, \operatorname{tail}(b)\big) \\ \operatorname{lev}\big(\operatorname{tail}(a), \operatorname{tail}(b)\big) \end{cases} & \text{otherwise,} \end{cases}$$

Figure 3.2. Levenshtein Distance Formula.

By identifying the dynamic fields and their variations, the fuzzing process can be optimized to target these fields, increasing the chances of finding vulnerabilities.

The Python library *Levenshtein* is used to support the calculation of Levenshtein distance, making it easy to implement in the field-type identification process [64].

3. **Similar field-type merging:** To properly identify fields of a protocol, it is important to consider that the actual fields may be greater than one byte in size. Therefore, in the analysis process, the payload is divided into individual bytes, and it is crucial to merge consecutive byte fields that have similar Levenshtein distances. This merging process involves combining consecutive static fields and consecutive dynamic fields with a difference of 0.2 or less in their Levenshtein distance. This approach helps to ensure that the identified fields are accurate and representative of the protocol's structure, improving the effectiveness of the subsequent fuzzing process.

Step 3: Probable Key-fields Identification

Once the fields have been analyzed, the next step is identifying the probable fields that can be the key field. This is the unique identifier for each payload that is used by the protocol to decode the message type. To finalize this unique identifier, probable candidate key fields must be chosen, which can be further analyzed to choose the final key field. The dynamic fields that lie within the range of the minimum length of the payload observed are considered.

The unique identifier is a field that is expected to be dynamic but whose value does not vary too drastically. The Levenshtein distance score is used to identify the probable key fields. The ideal score for the key field has been observed to lie between 0.8 and 0.95. However, it heavily depends on the PCAP file and the communication that has been captured. Thus, based on the PCAP file, it is possible that the similarity score of the fields does not lie in the ideal range. If no probable key field is found in the ideal range, the range start is decreased till at least one probable key field is identified in the specified range.

Step 4: Final Key-field Identification

During the clustering process, a unique identifier called the key field is selected from the list of probable key fields identified in the previous step. The payload is then clustered based on the value of the key-field candidate. To ensure the correct formation of clusters, several factors are calculated for each cluster:

1. **Intra score:** This is the similarity of the payload packets that are in the same cluster. The higher the intra-score, the more similarity is between the packets in the cluster, which indicates the formation of correct clusters.

2. **Inter score:** This is the similarity of the payload packets that are in different clusters. The lower the inter-score, the more difference is between the packets in different clusters, which indicates the formation of correct clusters.

3. **Length Variance:** Length variance is calculated to determine the average length of packets in the same cluster with respect to the maximum length of packets in the cluster. The value of length variance ranges from 0 to 1, where a score of 1 indicates that all packets in the cluster are of the same length. A higher score indicates the formation of correct clusters.

4. **Cluster Count:** The cluster count is determined by counting the number of clusters created for the key field. An ideal key field should have a low number of clusters. The higher the number of clusters, the lower the probability that the field is the key field.

The above factors are calculated for each cluster in the probable key field. The candidates where $\geq 50\%$ of clusters contain individual packets are dropped. Using these, the probability of the key field being the final is calculated using the formula:

$$Probability = InitialScore * IntraScore * (1 - InterScore) * LengthVariance * (1 - ClusterCount)$$

The key field with the highest probability is selected as the final identifier for clustering similar messages. This identifier is considered as unique and used to group similar messages together.

Step 5: Final cluster generation

In the next step of the protocol reverse engineering process, the packets are clustered into groups based on the key field identified in the previous step. This key field acts as a unique identifier to cluster similar messages together. Each cluster is considered as a different message type, and grammar is generated separately for each group. To categorize the packets in each cluster, the byte-by-byte analysis is performed again to identify whether each field is static or dynamic.

After generating the grammar for the request payload, the response payload is analyzed to generate the response grammar. The same key field used in the analysis of the request payload is utilized to form the clusters for the response payload. This is because it is assumed that the location of the key field should be the same in both request and response payloads.

The protocol structure is completely identified by analyzing both the request and response payloads and generating their respective grammars. This allows for a better understanding of the protocol and helps create a more effective fuzzing campaign.

3.1.4 Grammar Generation

After the clusters are generated for the protocol, the schema for each cluster is generated to optimize the fuzzing script and improve the bug discovery process. This schema generation

involves gathering several important pieces of information, which are used to generate an efficient and effective fuzzing script.

The schema generation process captures the following characteristics:

1. **Default Value:** This field captures one of the values observed in the payload that can be used as a default value to generate the fuzzing script. This value is identified as a static field in the byte-by-byte analysis phase and can be used to provide a starting point for fuzzing.

2. **Field Length:** The observed length of the field is also recorded in the schema. This information is essential for generating the correct input size for each field and helps ensure that the input data adheres to the protocol's specifications.

3. **Field Type:** The identification of the category of the field, indicating whether the field is static or dynamic, is also included in the schema. This information is essential for determining the type of mutation required for each field and helps ensure that the input data is valid and adheres to the protocol's specifications.

4. **Field similarity score:** The Levenshtein distance of the field is also recorded in the schema to analyze the variance in the field values. This score is used to determine the type and intensity of the mutation required for each field and helps ensure that the input data covers a wide range of possible values.

By capturing these characteristics and incorporating them into the schema, an optimized fuzzing script can be generated, which can direct the fuzzing process toward smart fuzzing and lead to faster bug discovery.

Fig: 3.3 shows the detailed state diagram for proprietary protocol reverse engineering.

3.1.5 APEX-ICS Script Generation

After the protocol reverse engineering component has generated the grammar, the next step is to write the fuzzing script. The output of the reverse engineering process provides essential information about the protocol, including the static fields or magic bytes, dynamic

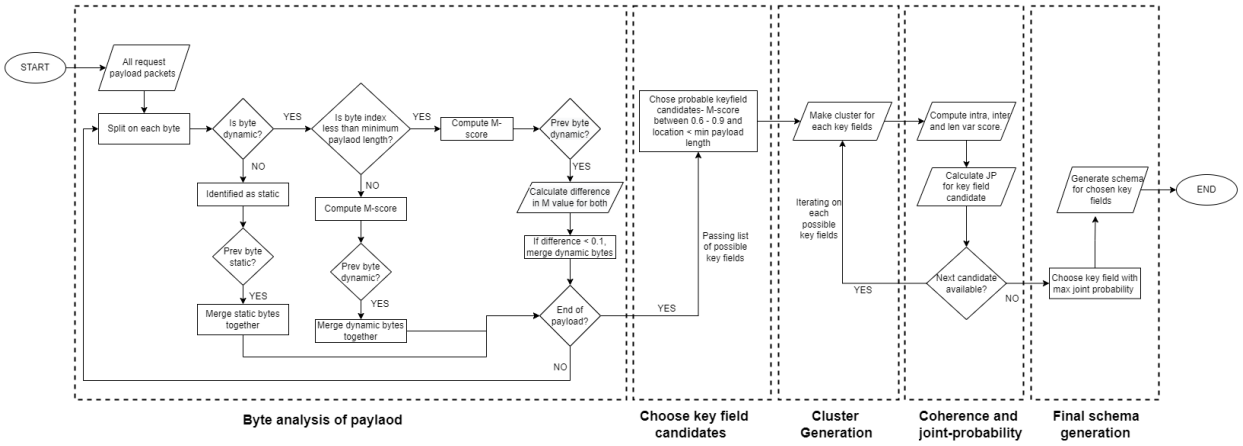

Figure 3.3. State Diagram for proprietary protocol reverse engineering.

fields, the size of each field, and the payload section, among others. With this information, the user can easily craft packets that adhere to the protocol's specifications, including checksum calculation, payload size, ACK number, or any other field that needs to be handled properly to communicate with the target.

Crafting the packet correctly is crucial in the fuzzing process, as it ensures that the generated test cases are valid and adhere to the protocol's specifications. A well-crafted packet will also help efficiently test the target system's various components, reducing the risk of false positives and unnecessary testing. Additionally, the fuzzing script can be optimized to prioritize testing the most critical components of the target system, leading to faster bug discovery and reducing the overall testing time.

The process of generating a fuzzing script is simplified with the output of the reverse engineering process, and this can also be usable for other fuzzers. It provides all the necessary information for crafting packets and testing the target system efficiently.

3.2 Fuzzing

APEX-ICS is a comprehensive fuzzing framework specifically designed to test the security of proprietary industrial control system protocols. The framework consists of several different components that work together to generate and execute a comprehensive fuzzing campaign.

The architecture of the APEX-ICS framework is shown in the figure below.

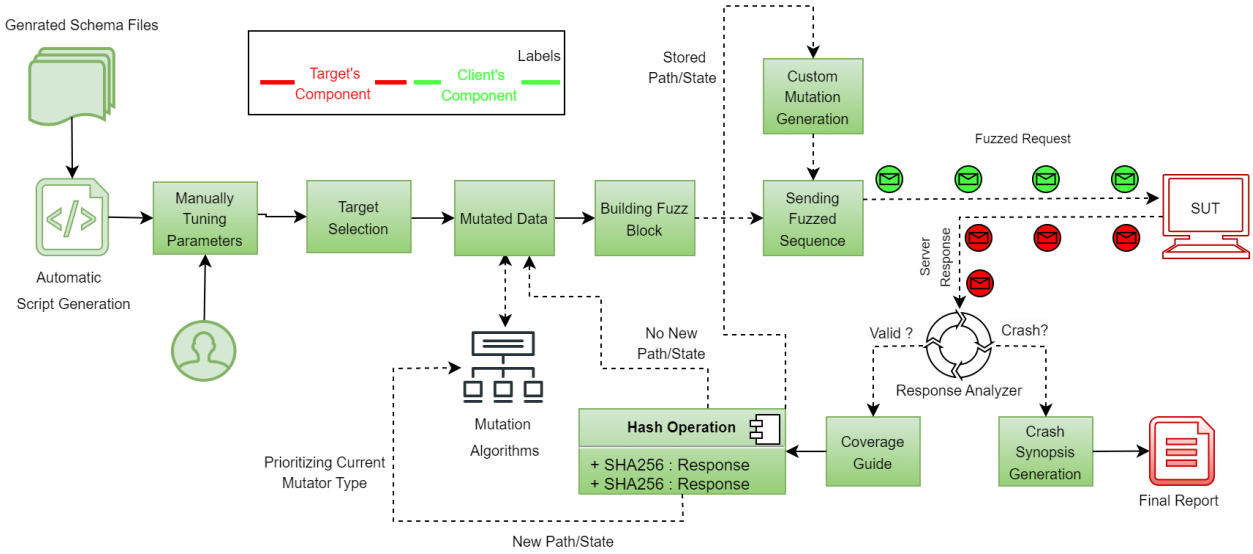

Figure 3.4. APEX-ICS, proprietary protocol fuzzing architecture.

As you can see from the architecture diagram, the APEX-ICS framework consists of several different modules, each of which plays a specific role in the fuzzing process. These modules include input generation, input mutation, target selection, monitoring and feedback, custom mutation generation, execution and analysis, and vulnerability detection and reporting.

In the following sections, we will provide a more detailed overview of each of these modules and how they work together to execute a successful fuzzing campaign.

3.2.1 Input Generation:

Input Generation is the first step in the fuzzing process, where the fuzzing engine utilizes the corpus provided by the reverse engineering component to generate input data. This input data is used to simulate real-world scenarios to identify vulnerabilities in the target system. If the protocol communication involves a checksum number, SYN/ACK number, or the size of the payload, the user needs to carefully craft the packet based on preliminary analysis of network traces in tools such as Wireshark. Once the user clearly understands the communication process, they can establish a connection with the target system before

initiating the fuzzing process. This connection is necessary to correct the fuzzing process on the target system. It is crucial to accurately craft the packet to ensure the target system responds correctly and identifies any vulnerabilities efficiently.

3.2.2 Target Selection:

The APEX-ICS tool allows users to select specific components within the target system to be fuzzed in the target selection phase. This can be achieved by choosing the corresponding key fields identified in the reverse engineering phase, which are then used to cluster similar messages together. The user can choose to fuzz specific key fields or all of the identified key fields. Additionally, the user can also specify which type of mutation algorithm should be used for the selected key fields. This approach ensures that only the relevant components are tested, reducing the risk of false positives and unnecessary testing.

In the context of fuzzing, key fields in a protocol can be seen as representing the state of the target system. By manipulating the values of these key fields, the fuzzer can induce different states in the target system, which can help to uncover vulnerabilities that are specific to certain states.

To leverage this aspect of fuzzing, users can use an Integrated Development Environment (IDE) to generate test data with multiple states. For example, the user can design test cases that reset the Programmable Logic Controller (PLC), log in and out of the system, start and stop a program, and so on. By creating test cases with different states, the user can ensure that the fuzzer explores all possible paths and states in the target system, increasing the likelihood of discovering vulnerabilities.

3.2.3 Input Mutation:

There are several types of mutation algorithms used in fuzzing, including [65]:

1. Bit-flip: This algorithm randomly selects a bit within the input data and flips its value from 0 to 1 or vice versa [66].

2. Byte-flip: This algorithm randomly selects a byte within the input data and flips its value from 0 to 255 or vice versa [67].

3. Arithmetic mutation: This algorithm performs mathematical operations on the input data, such as addition, subtraction, multiplication, and division. The goal is to introduce unexpected values into the input data and test how the system handles them [68].

4. Random mutation: This algorithm generates random input data by using various techniques, such as generating random bytes or strings or modifying existing input data in random ways [69].

It's important to note that each algorithm serves a different purpose and can be used to target specific parts of the input data. The bit-flip algorithm, for example, is useful for testing how the system handles small changes in the input data, while arithmetic mutation is useful for introducing unexpected values and testing the system's ability to handle them and similarly other algorithms.

The "mutation algorithms" block plays a critical role in the input mutation phase of APEX-ICS. This block lets the user specify which mutation algorithms to use and which fields to target. The default configuration selects algorithms randomly, but users can specify specific algorithms based on their requirements. This approach gives users greater control over the fuzzing process and allows them to tailor it to their specific needs.

3.2.4 Monitoring/Feedback Loop:

The "Monitoring/Feedback Loop" phase [70] is a critical aspect of the fuzzing process in APEX-ICS. During this phase, the fuzzer continuously monitors the target system's response to the generated input and provides feedback about the system's status. This feedback enables the fuzzer to make decisions about the response's validity and whether it contains any errors. If the response is valid, it passes to the "Hash operation" block. If the response contains an error, it is marked as a crash.

The "Hash operation" block maintains a dictionary that stores the hash of each response received from the target system along with the actual response packet in a JSON file. If the calculated hash already exists in the JSON file, no entry is made, and the current mutator selection procedure continues. However, if the hash is new, an entry is added to the JSON file with the newly calculated hash and actual response, and the mutator is prioritized towards testing this new area.

This approach of APEX-ICS utilizes a hash table to ensure new paths and states are tested during the fuzzing process. The "Hash operation" block calculates a hash for each response received, and if it differs from any previously encountered values, it is considered a new state or path and is added to the hash table along with the response. The mutation algorithm then prioritizes testing these new areas, ensuring that the fuzzing process covers as much of the target system as possible.

This feedback can be used to adapt the fuzzer's input generation and mutation algorithms to maximize the chances of discovering vulnerabilities.

3.2.5 Custom Mutation Generation:

In the fuzzing process, the initial seed input is mutated using various mutation algorithms to generate new inputs that can explore new paths in the target system. The Radamasa library [71] is one such tool used for custom mutation generation in APEX-ICS for fuzzing module. Radamasa is a Python library [72] that generates custom mutations based on the structure and syntax of the input. It takes a string as input and generates new structurally similar strings but with different values. This helps generate realistic inputs that are more likely to trigger vulnerabilities in the target system.

The importance of custom mutation generation lies in allowing the fuzzer to explore different areas of the target system that may not be covered by traditional mutation algorithms. This can help in discovering complex vulnerabilities that may not be detected through random or simple mutations.

In APEX-ICS, the custom mutation generation is used in the "Custom Mutation Generation" block, which takes the stored response from the "Hash Operation" block as the seed

input for further fuzzing rounds. This ensures that the fuzzing process continues to explore new paths in the target system based on previously encountered states and responses, increasing the likelihood of discovering vulnerabilities.

3.2.6 Execution and Analysis:

During execution, the fuzzer will record various metrics such as response, error messages, and crashes. These metrics are then analyzed to evaluate the overall stability of the target system and identify potential vulnerabilities as discussed in detail in 4.2 section. It is important to note that the execution phase may take a significant amount of time to complete, especially when dealing with complex target systems. The fuzzer may need to execute multiple rounds of testing to ensure that it has covered as many paths and states in the target system as possible.

Once the execution phase is complete, the fuzzer will analyze the recorded metrics to identify any anomalies or patterns that may indicate the presence of a vulnerability. This analysis may involve manual inspection of the logs and error messages, as well as the use of automated tools to identify common vulnerability patterns. Overall, the Execution and Analysis phase is critical in the fuzzing process as it provides insights into the overall stability of the target system and the presence of potential vulnerabilities. It allows the fuzzer to identify areas of the target system that require further testing and refinement, ultimately improving the efficiency and effectiveness of the fuzzing process.

4. RESULTS

In this section, the results obtained by the execution of the framework will be analyzed. All the aspects of the framework, such as details of protocol reverse engineering, etc., will be covered. Payloads of Various protocols have been used to test the framework to give a holistic review of the framework. It is important to understand and validate the framework results and compare the results of the reverse engineering framework with the actual fields to get an insight into the accuracy of the framework.

4.1 Protocol Reverse Engineering

To show the effectiveness and evaluate the automatic proprietary protocol reverse engineering component of APEX-ICS, multiple ICS protocols were used, but this section will contain mainly two protocols: Modbus/TCP and DNP3 protocol PCAPs for testing and analysis purposes. These two protocols were selected because they have widely different characteristics that provide insights into the framework's performance. Modbus/TCP is an ICS protocol with the shortest length and mainly same-length payload sizes. This protocol was chosen due to its simplicity and ease of testing, as it is commonly used in industrial control systems. On the other hand, DNP3 is the largest protocol in the ICS field that is commonly used in the energy sector for communication between control systems and field devices. The selection of DNP3 allows for the testing of the framework on a more complex and lengthy protocol.

Using both protocols evaluated the framework's performance for different types of protocols, allowing for a better understanding of the framework's output and ability to handle varying protocol characteristics. This approach helps to ensure that the framework is suitable for a range of industrial control systems with different communication requirements and can be used to identify and analyze vulnerabilities in various protocols.

4.1.1 Static / Dynamic Field Identification

When conducting fuzzing on protocols, it is essential to distinguish between static and dynamic fields. This is because many protocols have fields that serve as identifiers, such as protocol identifiers or magic fields, that must remain constant during the fuzzing process. If these values change, the server may not recognize the payload accurately, resulting in the payload being dropped. On the other hand, dynamic fields refer to values that can change during the protocol communication process, which are crucial to the protocol's operation. Therefore, analyzing how much these fields vary is necessary to ensure that the fuzzing process includes them. For example, fields such as messages, SYN and ACK can vary significantly in value, whereas fields like flags, function codes, and session IDs tend to remain constant. It is worth noting that the extent to which a field varies can depend on various factors, including the specific protocol being tested, the nature of the application, and the network environment. For this reason, conducting a thorough analysis of the protocol is essential to identify the fields that are most likely to vary and prioritize their inclusion in the fuzzing process.

By adequately categorizing fields as static or dynamic and determining the extent to which dynamic fields vary, researchers can conduct more effective and comprehensive fuzzing tests that identify vulnerabilities in the protocol. This approach helps to ensure that the protocol can operate correctly under different circumstances, including situations where dynamic fields are likely to change. For example, Fig. 4.1 shows identified dynamics fields for Codesys v3.0 protocol.

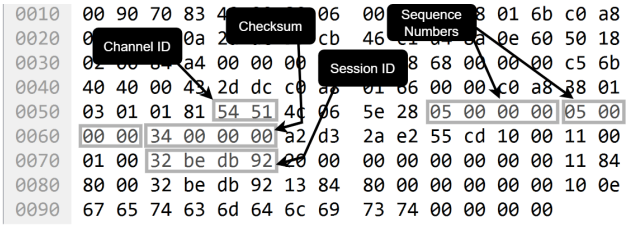

Figure 4.1. Dynamic field identification for Codesys v3.0 protocol

For the Modbus/TCP PCAP file used for testing, the framework identified eight fields. Three of them were identified as static and the remaining as dynamic, with similarity scores

ranging from 0.1 to 0.85. Fig. 4.2 shows the details of the similarity scores for each of the fields.

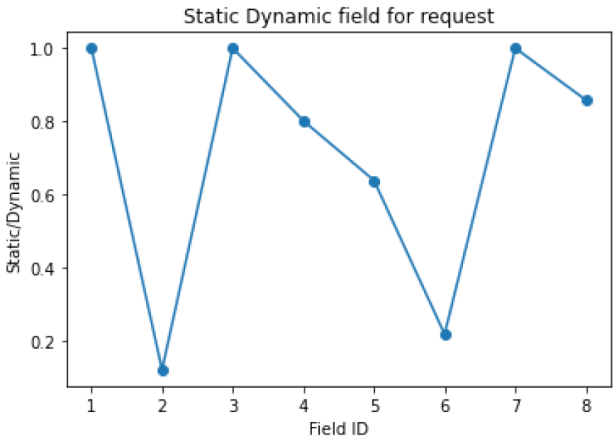

Figure 4.2. Modbus/TCP static/dynamic field identification

For the DNP3 PCAP file used for testing, the framework identified 15 fields. Five of them were identified as static and the remaining as dynamic, with similarity scores ranging from 0.2 to 0.9. Fig. 4.3 shows the details of the similarity scores for each of the fields.

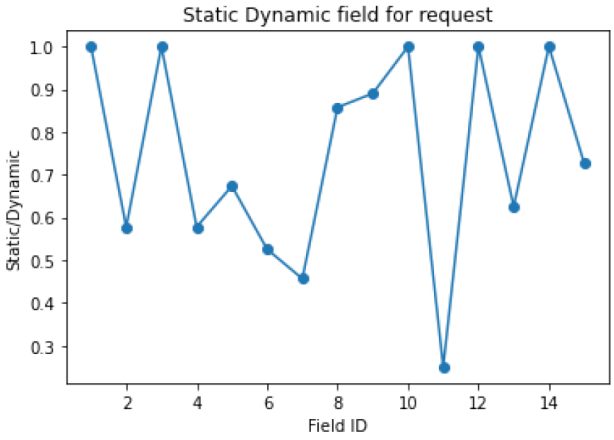

Figure 4.3. DNP3 static/dynamic field identification

4.1.2 Probable Key-fields Identification

It is important to cluster similar messages together as different message types have different structures. For this, probable key fields to clustering messages together are identified,

from which one of the fields is chosen as the final key field. / For Modbus/TCP, two fields were chosen as probable key fields with similarity scores of 0.80 and 0.85, respectively. Both the fields are of length one byte. For the first probable key field, three clusters are formed, with the number of packets in each cluster varying from 15 to 150. For the second probable key field, five clusters are formed, with the number of packets in each varying from 1 to 185. Fig: 4.4 shows the details of the probable key fields for Modbus/TCP protocol.

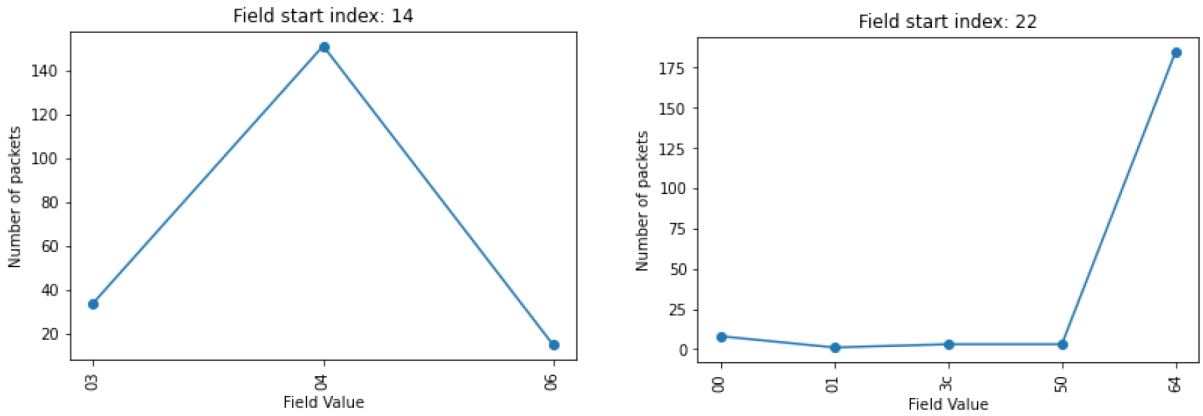

Figure 4.4. Modbus/TCP probable key-field identification

For DNP3, two fields were chosen as probable key fields with similarity scores of 0.85 and 0.89, respectively. The first probable key field is of one byte, whereas the second one is of length two bytes. Three clusters are formed for the first probable key field, with the number of packets in each cluster varying from 4 to 45. Five clusters are formed for the second probable key field, with the number of packets in each varying from 1 to 49. Fig: 4.5 shows the details of the probable key fields for the DNP3 protocol.

4.1.3 Intra-Score Similarity Analysis

Intra-score shows the similarity in the packets that belong to the same cluster. Packets that belong to the same cluster should have high similarity indicating that the intra-score should be high.

For Modbus/TCP, the first probable key field shows a higher and more consistent intra-score value as compared to the second field suggesting the first probable key field to be the

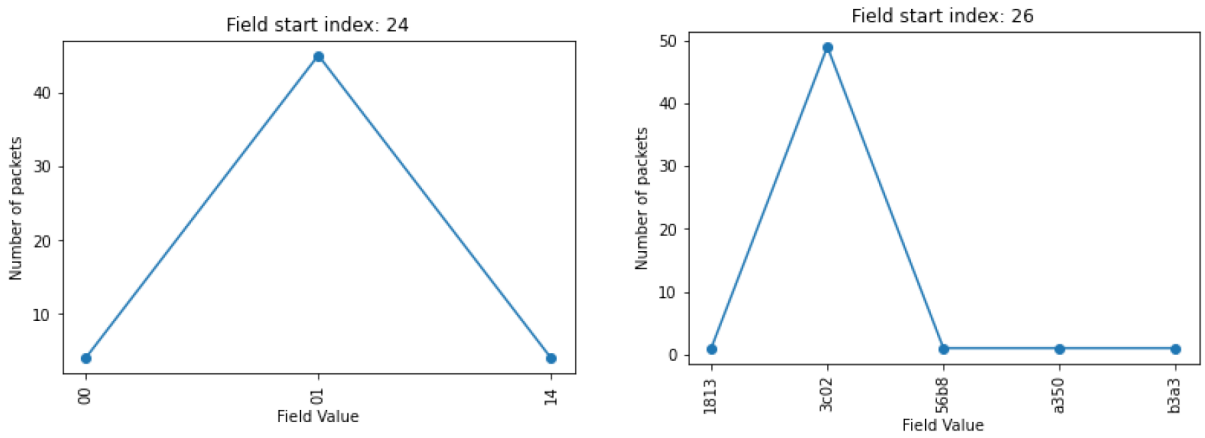

Figure 4.5. DNP3 probable key-field identification

final key field. Fig: 4.6 shows the details of the intra-score values for clusters generated by different probable key fields.

Figure 4.6. Modbus/TCP intra-score analysis

For DNP3, the first probable key field shows a higher and more consistent intra-score value as compared to the second field suggesting the first probable key field be the final key field. Fig: 4.7 shows the details of the intra-score values for clusters generated by different probable key fields.

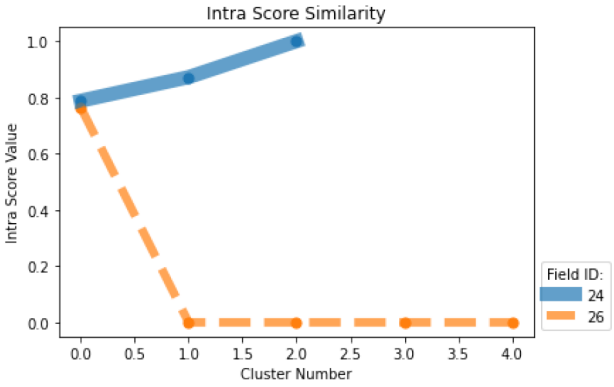

Figure 4.7. DNP3 intra-score analysis

4.1.4 Inter Score Similarity Analysis

Inter-score shows the similarity in the packets that belong to a different cluster. Packets that belong to different clusters should have low similarity indicating that the inter-score should be low.

For Modbus, the first probable key field shows a higher inter-score value as compared to the second field, suggesting that the second probable key field is a better candidate to be the final key field. Fig: 4.8 shows the details of the inter-score values for clusters generated by different probable key fields.

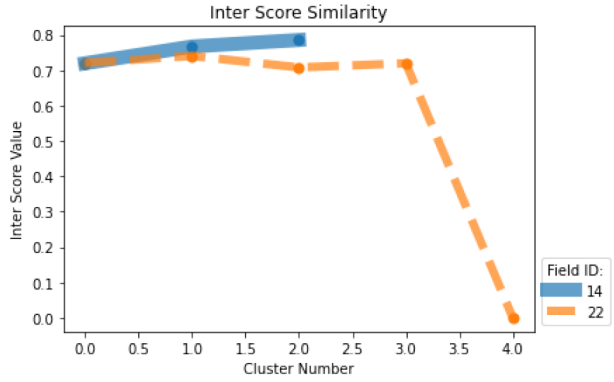

Figure 4.8. Modbus/TCP inter score analysis

For DNP3, the first probable key field shows a higher inter-score value as compared to the second field, suggesting that the second probable key field is a better candidate to be the final key field. Fig: 4.9 shows the details of the inter-score values for clusters generated by different probable key fields.

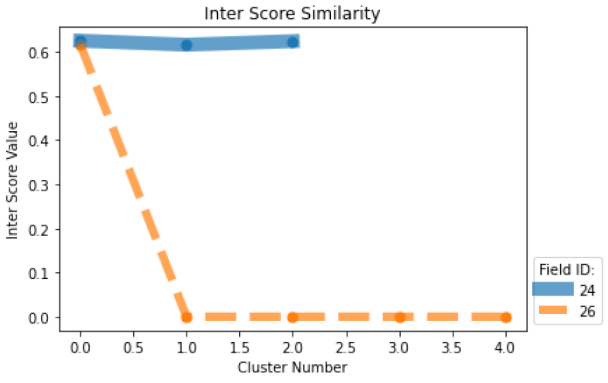

Figure 4.9. DNP3 inter score analysis

4.1.5 Length Variance Analysis

Length variance shows how the length of the packets that belong to the same cluster varies. Packets that belong to the same cluster should have similar lengths indicating that the length variance should be high.

For Modbus/TCP, the first probable key field, the clusters have a length variance of one, indicating that all the packets in each of the clusters are of the same length. For the second probable key field, since there is one cluster that has only one packet, the length variance for that particular cluster is calculated as 0. This suggests that the first field is a better candidate to be the final key field. Fig: 4.10 shows the details of the length variance for clusters generated by different probable key fields.

For DNP3, the first probable key field, the clusters have a length variance ranging between 0.8 and 1. For the second probable key field, four of the clusters have only one packet, and the length variance for them is calculated as 0. This suggests that the first field is a better candidate to be the final key field. Fig: 4.11 shows the details of the length variance for clusters generated by different probable key fields.

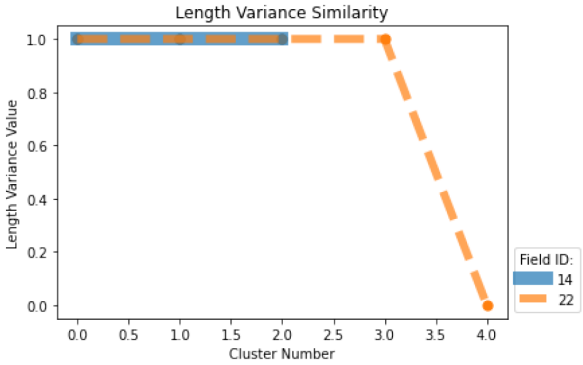

Figure 4.10. Modbus/TCP length variance analysis

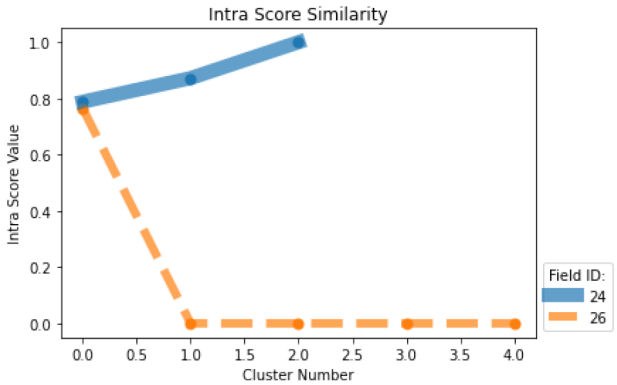

Figure 4.11. DNP3 length variance analysis

4.1.6 Final Joint Probability Calculation

The final joint probability is calculated to decide the final key-field candidate for the protocol. It is calculated using the intra-score, inter-score, length variance, and the number of clusters formed for a candidate field.

For Modbus/TCP, the first probable key field has the final joint probability of 0.67, and the second one has 0.48. Hence, the first field is chosen as the final key field. Fig: 4.12 shows the details of the joint probability for Modbus.

For DNP3, the first probable key field has a final joint probability of 0.65, and the second one has 0.01. Hence, the first field is chosen as the final key field. Fig: 4.13 shows the details of the joint probability for DNP3.

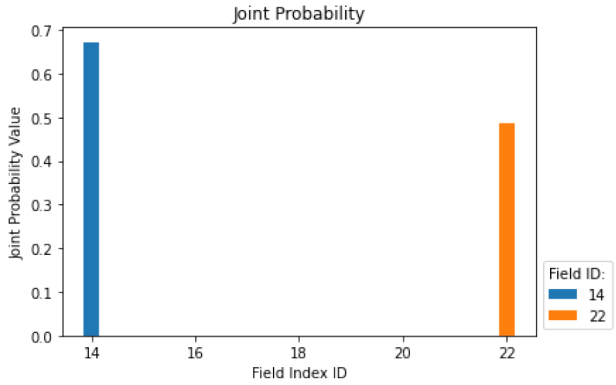

Figure 4.12. Modbus/TCP joint probability analysis

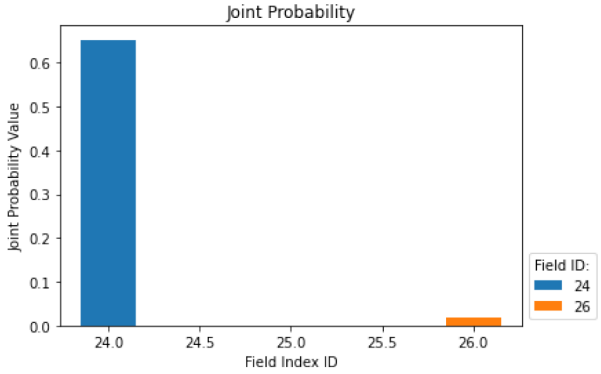

Figure 4.13. DNP3 joint probability analysis

4.1.7 Evaluation with existing reverse engineering frameworks

In order to evaluate the effectiveness of APEX-ICS compared to the existing frameworks Netplier and Netzob, this study considered multiple proprietary and non-proprietary industrial control system (ICS) protocols. The protocols chosen for evaluation included Modbus/TCP, IEC-104, Ethernet/IP (Non-ConMgr), BachMann/3500, DNP3, CodeSys v3.0, and Ethernet/IP (ConMgr). Table: 4.1 shows the details of the comparison with existing frameworks.

During the evaluation process, this study did not use all key fields in the recorded network trace according to the ground truth of the protocol. Instead, the selection of key fields was based on the researcher's interest, such as which transitions needed to be fuzzed. For

example, while Modbus/TCP has 15 key fields, the network traffic in the evaluation only contained communication of key fields 0x03 (Read Holding Registers), 0x04 (Read Input Registers), and 0x06 (Force Single Register).

By focusing on specific key fields, this study was able to provide a targeted evaluation of the frameworks' effectiveness in identifying vulnerabilities and potential attack vectors. This approach allowed for a more comprehensive comparison of the frameworks' detection capabilities under specific conditions rather than a general evaluation of their performance across all key fields.

Table 4.1. Comparison with existing protocol reverse engineering frameworks

Protocol	Framework	Total Fields	Actual Key Fields	Identified Key Fields	Detection Capability	Proprietary	Private
Modbus/TCP	APEX-ICS	6	(0x03), (0x04), (0x06)	(0x03), (0x04), (0x06)	High	✓	✗
	NetZob	6	(0x03), (0x04), (0x06)	(0x00), (0x01), (0x04)	Low		
	Netplier	6	(0x03), (0x04), (0x06)	(0x03), (0x04), (0x06)	Moderate		
IEC-104	APEX-ICS	8	(0x2d), (0x3a), (0x2e), (0x1f), (0x2f), (0x30), (0x31), (0x32), (0x33)	(0x2d), (0x3a), (0x2e), (0x1f), (0x2f), (0x30), (0x31), (0x32), (0x33)	High	✓	✗
	NetZob	6	(0x2d), (0x3a), (0x2e), (0x1f), (0x2f), (0x30), (0x31), (0x32), (0x33)	(0x00)	Low		
	Netplier	634	(0x2d), (0x3a), (0x2e), (0x1f), (0x2f), (0x30), (0x31), (0x32), (0x33)	(0x04), (0x0e), (0x0e), (0x11), (0x10), (0x12)	Moderate		
Ethernet/IP (Non-ConMgr)	APEX-ICS	10	(0x64), (0x68), (0x69), (0x73), (0x77), (0x8e), (0xac)	(0x8e)	High	✓	✗
	NetZob	8	(0x64), (0x68), (0x69), (0x73), (0x77), (0x8e), (0xac)	(0x00), (0x01)	Low		
	Netplier	8	(0x64), (0x68), (0x69), (0x73), (0x77), (0x8e), (0xac)	(0x01), (0x02), (0x04), (0x0f)	Moderate		
Bachmann/3500	APEX-ICS	10	(0010), (0740)	(0010), (0740)	High	✗	✓
	NetZob	11	(0010), (0740)	(0x00), (0x02)	Low		
	Netplier	14	(0010), (0740)	(0x02), (0x5a), (0xe8)	Low		
DNP3.0	APEX-ICS	9	(0x00), (0x01), (0x14)	(0x00), (0x01), (0x14)	High	✓	✗
	NetZob	6	(0x00), (0x01), (0x14)	(0x00), (0x14), (0xc4)	Low		
	Netplier	14	(0x00), (0x01), (0x14)	(0xc4), (0x44)	Moderate		
CODESYS	APEX-ICS	23	(0x03), (0x40)	(0x03), (0x40)	High	✗	✓
	NetZob	30	(0x03), (0x40)	(0x80), (0x9f), (0x4c), (0x2e), (0x81), (0x54)	Low		
	Netplier	28	(0x03), (0x40)	(0x6b), (0xc3), (0xdc), (0xed), (0xeb)	Low		
Ethernet/IP (ConMgr)	APEX-ICS	18	(0x52), (0x55), (0x4e)	(0x04), (0x52), (0x54), (0x8e)	High	✓	✗
	NetZob	15	(0x52), (0x55), (0x4e)	(0x00), (0x93x01), (0x1bx00), (0x14x01), (0x1ax00), (0x1ex00), (0x1cx00)	Low		
	Netplier	26	(0x52), (0x55), (0x4e)	(0x00), (0x03), (0x05), (0x07), (0x0b)	Moderate		

To compare APEX-ICS with Netplier and NetZob in a systematic and objective manner, a well-defined methodology was followed. Initially, the packets were filtered out based on the desired direction using the Wireshark filter, and then the filtered packets were converted into PDML XML format using tshark. The resulting XML file was then passed as an argument to the reverse engineering engine of APEX-ICS, which provides detailed information about the length of the field, magic values, static/dynamic fields, and key fields with a corresponding index number. In cases where there were multiple key fields, the tool generated multiple blocks for each identified key field with the required information, which could be used to write a fuzzer script.

To execute Netzob for this evaluation, the guidelines provided on their GitHub repository were followed [73]. However, unlike APEX-ICS and Netplier, the output of Netzob does not provide information about identified key fields. Instead, it provides the number of fields along with their corresponding values. For this reason, this study considered only the low-varying fields from the output for evaluation purposes. This means that for the comparison table 4.1,

only the number of fields identified by Netzob as low varying were considered in the "Total Fields" column, and the corresponding number of identified key fields was considered for the "Identified Key Fields" column. This methodology allowed for a fair comparison between the capabilities of the three frameworks despite the differences in their output formats.

In the same way that the methodology for APEX-ICS and Netzob was followed, the guidelines provided in the GitHub repository of Netplier were also followed to perform reverse engineering on the same network trace [74]. After executing Netplier, the resulting files (msa-fields-info, msa-fields-visual, msa-input.fa, msa-output, msa-output-oneline, prob-request, prob-response) are stored in one directory. The "msa-fields-info" file contains type information for each identified field, such as whether it is static, dynamic, or varying. In Table 4.1, the column for "Total Fields" only counts the number of static and dynamic fields from the "msa-fields-info" file, and the results for the "Identified Key Fields" column are taken from the generated output. This approach ensures consistency in the evaluation process and allows for a fair comparison between the frameworks being evaluated.

Once the reverse engineering was completed, the resulting reverse-engineered data from all three frameworks were compared based on the "Identified Key Fields" and "Actual Key Fields". Fig: 4.14 shows the results between APEX-ICS, Netplier, and Netzob. The identified key fields were then compared with the actual key fields based on the ground truth of each protocol. Additionally, the detection capability of each framework was compared by assessing how accurately it could detect key fields. Fig: 4.14 shows the results between APEX-ICS, Netplier, and Netzob for proprietary and non-proprietary ICS protocols.

This methodology ensured that the comparison was performed on an equal footing and that any differences observed could be attributed to the underlying capabilities of the frameworks being compared. Following a systematic and objective approach, this evaluation study performed a fair and comprehensive evaluation of APEX-ICS against Netplier and NetZob.

4.2 Fuzzing

This section presents the results of the fuzzing experiments performed using APEX-ICS, focusing on evaluating the effectiveness of the input generation and mutation techniques, as

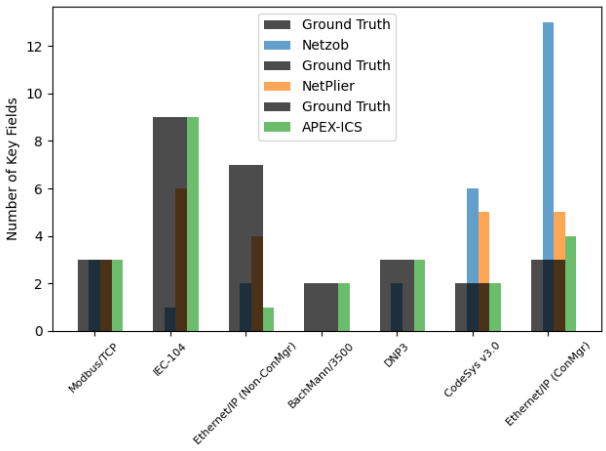

Figure 4.14. Evaluation of Netplier and Netzob with ICS protocols

well as the feedback loop and custom mutation generation features. Finally, it discusses the performance of APEX-ICS to demonstrate the effectiveness of the fuzzing approach.

4.2.1 Evaluation of Fuzzing for APEX-ICS

The fuzzing approach used in the APEX-ICS framework identified several new paths and generated a significant number of packets. The input mutation 4.2.1 technique was effective, with Bit Flip and Arithmetic Mutation algorithms generating the newest paths, while Byte Flip generated the least, as shown in 4.15 and 4.16. The results obtained from the Custom Mutation Generation component were limited due to the packet size constraint, but it could still leverage identified new paths for further testing. Overall, the APEX-ICS framework successfully identified four new vulnerabilities (discussed in the Appendix section) in the target system. These results demonstrate the effectiveness of the fuzzing approach and suggest that it can be a valuable tool for identifying security issues in industrial control systems.

The following subsections detail the experimental setup and provide insights into the performance of APEX-ICS during the fuzzing process.

Input Generation

APEX-ICS leverages the output of the automatic proprietary protocol reverse engineering component, which produces a grammar that defines the structure of the protocol messages. This generated grammar is used to generate valid protocol messages that can be used for the fuzzing individual states, as shown in table 4.2. For instance, to trigger the "Service Layer", multiple states for Datagram and Channel layer should be specified exactly in the same order as shown in the table 4.2. The effectiveness of the generated grammar can be evaluated in terms of the coverage achieved during the fuzzing process. The grammar should be able to generate messages that exercise a significant portion of the target system's functionality. The coverage can be measured by the number of messages generated, the number of unique message types, the code paths exercised, and the number of errors triggered.

Table 4.2. Identified Codesys runtime states for each layer

Subject	Component	Codesys Runtime State	Establish Communication?
Codesys v3.0	Datagram Layer	0x03, 0x40	✓
	Channel Layer	0x02, 0xc3	✓
	Service Layer	(0100, 0100 - GetIdent), (0100, 0a00 - Session), (0100, 0200 - Login), (1100, 0100 - PLCShell)	✓

Moreover, the correctness of the generated grammar is also crucial for effective input generation. Without the correct grammar, the Codesys runtime will not be able to understand the communication format, leading to reset communication. Therefore, the grammar should maintain the correct order of static and dynamic fields like checksum, syn/ack, payload size, etc.

Overall, APEX-ICS uses the generated grammar to generate valid protocol messages for fuzzing. The effectiveness of the grammar can be evaluated based on the coverage achieved during the fuzzing process, while correctness ensures that the Codesys runtime can correctly interpret the protocol messages.

Target Selection

In the target selection step, the fuzzer uses the key fields identified during the input generation step to select specific target components of the Codesys runtime to be fuzzed. For example, to fuzz the datagram layer, the recognized states (0x03) and (0x40) can trigger

this specific state. Similarly, for fuzzing the channel and service layers, the fuzzer must ensure that the datagram layer is correctly implemented and that both the datagram and channel layers are implemented, respectively, as described in table 4.2.

When fuzzing the Codesys runtime, several challenges need to be addressed. One of these challenges is maintaining the Channel ID field in the channel layer. The fuzzer needs to keep track of the response from the target to update the Channel ID field, which is a dynamic field that can change based on the current state of the target. In addition, the fuzzer needs to maintain the BLK Transmission ID and Session ID fields in the service layer. The Codesys runtime sends these IDs in response to a request, and the fuzzer needs to send the correct IDs in each packet and keep track of the responses to update the IDs accordingly.

These challenges can make the fuzzing process more complex and time-consuming, but they are essential for successful fuzzing and can result in more thorough testing of the target system.

Input Mutation

APEX-ICS uses various mutation algorithms during the input mutation phase to modify the input data based on the user's selection of which fields to fuzz. The default configuration randomly selects mutation algorithms, but users can specify specific ones.

The mutation algorithms used in APEX-ICS include bit-flip, byte-flip, arithmetic mutation, and random mutation. The bit-flip algorithm randomly selects a bit within the input data and flips its value from 0 to 1 or vice versa. The byte-flip algorithm randomly selects a byte within the input data and flips its value from 0 to 255 or vice versa. The arithmetic mutation algorithm performs mathematical operations on the input data, such as addition, subtraction, multiplication, and division. The goal is to introduce random values into the input data and test how the system handles them. Finally, the random mutation algorithm generates random input data using various techniques, such as generating random bytes or strings or modifying existing input data unexpectedly.

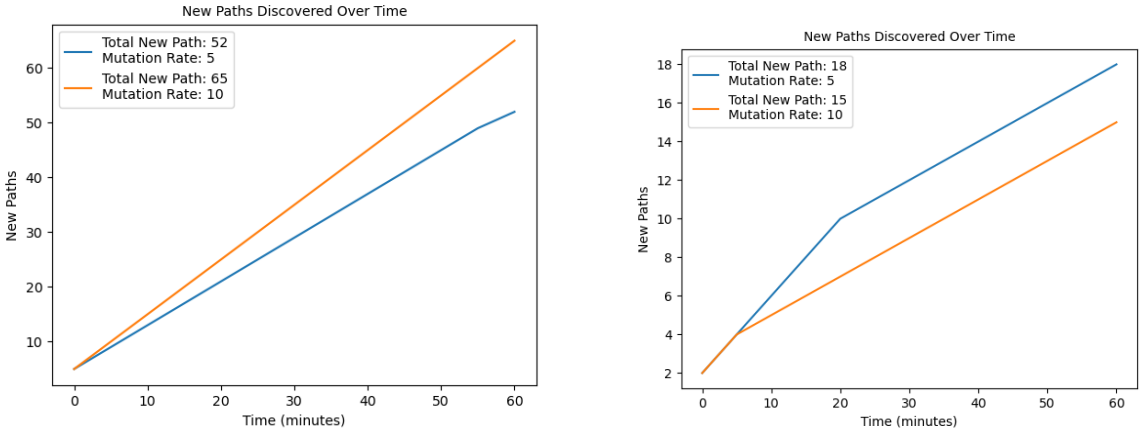

Figure 4.15. Performance for Bit/Byte flip mutation algo with diff mutation rate.

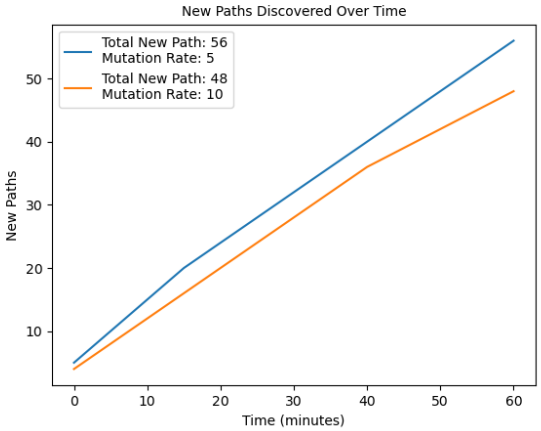

Figure 4.16. Performance for arithmetic operation algo with diff mutation rate.

Three experiments were performed using the Bit Flip, Arithmetic, and Byte Flip algorithms to evaluate the effectiveness of the mutation algorithms in APEX-ICS. For each algorithm, two experiments were conducted with mutation rates of 5 and 10.

For the Bit Flip mutation algorithm, 10,000 packets sent to the target system with a mutation rate of 5 triggered 52 new paths as shown in fig: 4.15. With a mutation rate of 10, 21,262 packets were sent, and 65 new states were triggered. Similarly, for the Arithmetic algorithm, 15,868 packets were sent with a mutation rate of 5, resulting in 56 new paths triggered, as shown in fig: 4.16. With a mutation rate of 10, 16,800 packets were sent, and

48 new states were triggered. Finally, for the Byte Flip algorithm, 8,961 packets were sent with a mutation rate of 5, resulting in 18 new paths triggered. With a mutation rate of 10, 11,500 packets were sent, and 15 new states were triggered.

In addition to the number of new paths or states triggered, graphs were generated for each algorithm to visualize the relationship between the mutation rate and the number of unique paths triggered. These results demonstrate the effectiveness of the mutation algorithms in discovering new paths or states in the target system and provide insight into the optimal mutation rates for each algorithm.

From the results, It can be observed that the effectiveness of each mutation algorithm varies depending on the specific protocol being fuzzed. For example, the bit Flip algorithm was the most effective, triggering the highest number of new paths for both mutation rates tested. On the other hand, the byte flip algorithm was the least effective, triggering the lowest number of new paths for both mutation rates.

Additionally, the number of new paths triggered by each mutation algorithm did not always increase linearly with the mutation rate. For instance, the arithmetic mutation algorithm triggered more new paths with a mutation rate of 5 than with a rate of 10. This indicates that there may be a threshold for the mutation rate beyond which the algorithm's effectiveness declines.

Monitoring/Feedback Loop

In the case of APEX-ICS, the feedback loop was used to monitor the responses generated by the target system and analyze them to identify any unexpected behavior or errors triggered by the input packets.

The results obtained from the input mutation technique can be analyzed to evaluate the effectiveness of the feedback loop. As described in the Input Mutation section 4.2.1, the number of new paths triggered and the quality of the responses were evaluated to determine the fuzzing techniques' effectiveness. The feedback loop played a critical role in this evaluation process by providing insights into the system's behavior in response to the input packets.

The feedback loop also identified any crashes or errors triggered by the input packets, which were subsequently analyzed to determine their root cause. This information was used to refine the input generation and mutation techniques further, leading to the identification of additional vulnerabilities in the target system.

Custom Mutation Generation

The Custom Mutation Generation component 3.2.5 utilizes the identified new paths in another round of fuzzing as shown in the fuzzing architecture 3.4. In addition, this component creates customized input data to test the target system's response to specific scenarios. The "Radamsa" library generates various mutated input data to achieve this.

It's worth noting that the input data size is a critical factor in the effectiveness of the fuzzing process. In this case, the target system (Codesys runtime) expects a maximum of 512 bytes in each packet. Therefore, any generated input data exceeding this size will result in a communication reset. As a result, the custom mutation generation component is programmed to only create test cases with a maximum length of 512 bytes, ensuring that the test data is compatible with the target system.

The use of customized input data generated by the custom mutation generation component can provide valuable insights into the target system's vulnerabilities and areas for improvement.

4.2.2 Effectiveness of Fuzzing Component

This section discusses obtained results of the fuzzing component for the APEX-ICS framework. In addition, this section will discuss the different components of each layer of Codesys runtime.

Case Study 1 (CWE-20: Improper Input Validation)

A valid protocol grammar of Codesys v3.0 was obtained from the reverse engineering component of APEX-ICS. It generated key fields based on the state execution of the Codesys runtime, as shown in Table 4.2.

The CmpRouter [24] component is a crucial part of the Codesys network architecture responsible for detecting nodes in the network and routing packets to their intended destinations. The RouterHandleData function in CmpRouter parses several fields in incoming packets, including the MagicNumber, HopInfo, PacketParameter, ServiceID, LengthField, SenderAddress, ReceiverAddress, and Padding. The ServiceID field is significant, as it forward packets to different components depending on their values. For example, a ServiceID value of 0x03 represents network service, and 0x40 indicates that packets should be forwarded to the CmpChannelManager component in the channel layer. In this case study, the key field 0x03 was used to fuzz SenderAddress and ReceiverAddress. However, a vulnerability was discovered in the datagram layer, allowing an attacker to connect with Codesys-based PLC with forged values of SenderAddress and ReceiverAddress.

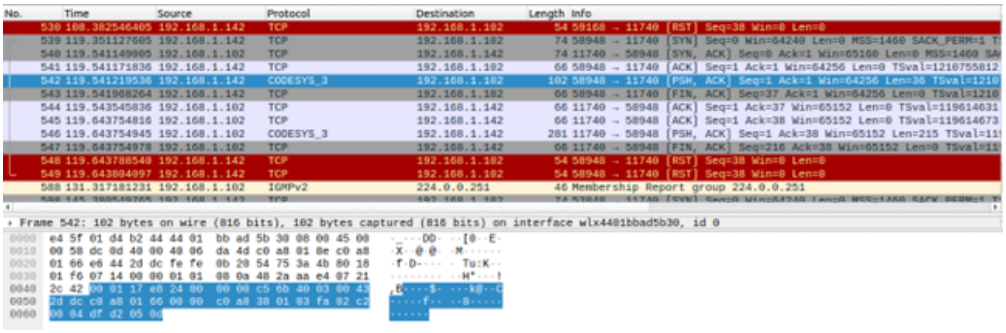

Figure 4.17. A highlighted section in the Wireshark image shows the attacker uses a different IP address and port number.

Successful exploitation of this vulnerability could lead to unauthorized access to Codesys-based PLCs and a loss of system integrity. Once exploited, the Codesys runtime would send confirmation over the IGMPv2 protocol [75], indicating that the attacker has gained unauthorized access. The vulnerability highlights the importance of proper data validation and integrity checking in network communication components, especially in critical systems like PLCs.

Case Study 2 (CWE-700: Allocation of Resources Without Limits or Throttling)

As mentioned in the table in Table 4.2, the next state is the channel layer. To trigger channel layer component Codesys runtime, the fuzzer has key fields 0x40, which belong to the ServiceID field of the datagram layer as described in section 4.2.2. To open a new channel, APEX-ICS emulates the channel layer; it implements these fields to send the packet: ChannelID 0xc3, ChannelFlags 0x00, and a random 2-Bytes value for ChannelID. Key field block 0xc3 (ChannelID) was used to fuzz the channel layer. It observed that a legitimate user of Codesys IDE (supervisory system) couldn't connect with Codesys-based PLC, as shown in the following figure 4.18.

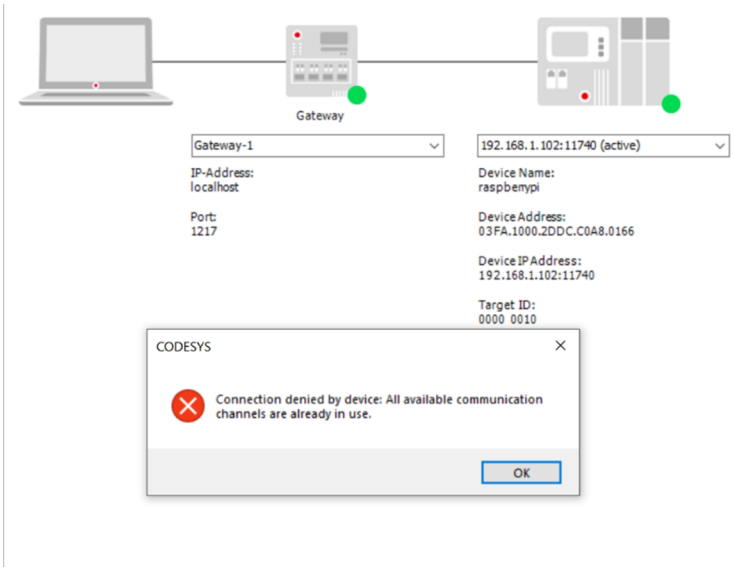

Figure 4.18. A legitimate Codesys IDE user trying to connect with PLC.

The vulnerability is located at the Channel Layer of Codesys v3.0 protocol. An attacker can exploit this vulnerability by continuously sending new channel IDs ranging from 0x0000 to 0xFFFF. If this attack is successful, legitimate users can't connect to Codesys-based PLC because the maximum number of allowed channels may have already been reached. This vulnerability can have severe consequences, as it can prevent authorized users from accessing the Codesys-based PLC to perform necessary tasks. Additionally, if the attacker can flood the system with enough channel IDs, it could potentially cause a denial of service (DoS) attack, disrupting the system's normal functioning.

Case Study 3 (CWE-400): Uncontrolled Resource Consumption ('Resource Exhaustion')

APEX-ICS has emulated a datagram and the channel layer to trigger service layer components. To implement the service layer, APEX-ICS sent a packet with these field values: ProtocolID, HeaderSize, ServiceID, CommandID, SessionID, PayloadSize, AdditionalData, and Payload. In the case of emulation for the service layer, APEX-ICS has to finish some other states before interacting with service layer components, which is extracted by reverse engineering component as mentioned in table 4.2. Firstly, 0x0100 (ServiceID) and 0x0100 (CommandID) to get information on Codesys-based PLC. Secondly, 0x0100 (ServiceID) and 0x0a00 (CommandID) to create a session, and lastly, 0x0100 (ServiceID) and 0x0200 (CommandID) to Login. After execution of these stated, APEX-ICS interacts with the service layer's components, such as PLCShell, Settings, Alarms, etc. This study focused on the PLCShell component. To interact with PLCShell, APEX-ICS used 0x1100 and 0x0100. While interacting with the service layer, the channel layer will have the same ChannelID 0x01 through the fuzzing, ACKNumber, and Checksum (calculated on the whole service layer using the CRC32 algorithm) also managed. Login and logout crashed in the fuzzing process, which is responsible for running the application.

5. SUMMARY

The APEX-ICS framework is a tool that allows security researchers to test the security of industrial control systems (ICS) systems by simulating attacks against them. The framework consists of several components, including a reverse engineering component and a fuzzing component.

The reverse engineering component is responsible for automatically generating a grammar for a proprietary protocol used by the ICS system. It does this by analyzing network traffic between the development IDE system and a PLC, and using this information to infer the structure and format of the protocol. The grammar generated by the reverse engineering component can then be used by the fuzzing component to generate valid test cases.

The fuzzing component of APEX-ICS is designed to test the ICS for vulnerabilities by generating inputs that are designed to trigger unexpected behavior. It uses various mutation algorithms, including bit-flip, byte-flip, arithmetic mutation, and custom mutation generation, to generate these inputs. The fuzzing component also includes a monitoring and feedback loop that analyzes the responses from the ICS to determine whether a particular input has caused unexpected behavior or crashed the system.

Overall, the APEX-ICS framework is a powerful tool for testing the security of ICS systems. By combining automated reverse engineering with sophisticated fuzzing techniques, the framework can identify unknown vulnerabilities in PLCs, helping to improve their overall security.

REFERENCES

[1] R. Davis and I. M. Duhaime, "Diversification, vertical integration, and industry analysis: New perspectives and measurement," *Strategic Management Journal*, vol. 13, no. 7, pp. 511–524, 1992.

[2] S. J. Shackelford, M. Sulmeyer, A. N. C. Deckard, B. Buchanan, and B. Micic, "From russia with love: Understanding the russian cyber threat to us critical infrastructure and what to do about it," *Neb. L. Rev.*, vol. 96, p. 320, 2017.

[3] D. Trentesaux, "Distributed control of production systems," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 971–978, 2009.

[4] T. M. Chen and S. Abu-Nimeh, "Lessons from stuxnet," *Computer*, vol. 44, no. 4, pp. 91–93, 2011.

[5] *Vulnerability*, Apr. 2023. [Online]. Available: https://en.wikipedia.org/wiki/Vulnerability.

[6] *Vulnerabilities*. [Online]. Available: https://owasp.org/www-community/vulnerabilities/.

[7] S. L. Cutter, "Vulnerability to environmental hazards," *Progress in human geography*, vol. 20, no. 4, pp. 529–539, 1996.

[8] R. Khan, P. Maynard, K. McLaughlin, D. Laverty, and S. Sezer, "Threat analysis of blackenergy malware for synchrophasor based real-time control and monitoring in smart grid," in *4th International Symposium for ICS & SCADA Cyber Security Research 2016 4*, 2016, pp. 53–63.

[9] A. Di Pinto, Y. Dragoni, and A. Carcano, "Triton: The first ics cyber attack on safety instrument systems," in *Proc. Black Hat USA*, vol. 2018, 2018, pp. 1–26.

[10] W. K. Miller, "An examination of us policy toward iranian nuclear proliferation," ARMY WAR COLL CARLISLE BARRACKS PA, Tech. Rep., 2007.

[11] H. Berger, *Automating with SIMATIC S7-400 inside TIA portal: configuring, programming and testing with STEP 7 Professional*. John Wiley & Sons, 2014.

[12] J. A. Lewis, *Assessing the risks of cyber terrorism, cyber war and other cyber threats*. Center for Strategic & International Studies Washington, DC, 2002.

[13] D. J. Hickson, D. S. Pugh, and D. C. Pheysey, "Operations technology and organization structure: An empirical reappraisal," *Administrative science quarterly*, pp. 378–397, 1969.

[14] K. Stouffer, J. Falco, K. Scarfone, *et al.*, "Guide to industrial control systems (ics) security," *NIST special publication*, vol. 800, no. 82, pp. 16–16, 2011.

[15] S. Debernard, C. Chauvin, R. Pokam, and S. Langlois, "Designing human-machine interface for autonomous vehicles," *IFAC-PapersOnLine*, vol. 49, no. 19, pp. 609–614, 2016.

[16] J. Kramer and J. Magee, "Self-managed systems: An architectural challenge," in *Future of Software Engineering (FOSE'07)*, IEEE, 2007, pp. 259–268.

[17] M. M. Lashin, "Different applications of programmable logic controller (plc)," *International Journal of Computer Science, Engineering and Information Technology (IJCSEIT)*, vol. 4, no. 1, pp. 27–32, 2014.

[18] G. Suciu, A. Vulpe, O. Fratu, and V. Suciu, "M2m remote telemetry and cloud iot big data processing in viticulture," in *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, IEEE, 2015, pp. 1117–1121.

[19] I. Bell, "The future of control [programmable automation controllers]," *Manufacturing Engineer*, vol. 84, no. 4, pp. 36–39, 2005.

[20] N. Govil, A. Agrawal, and N. O. Tippenhauer, "On ladder logic bombs in industrial control systems," in *Computer Security: ESORICS 2017 International Workshops, CyberICPS 2017 and SECPRE 2017, Oslo, Norway, September 14-15, 2017, Revised Selected Papers 3*, Springer, 2018, pp. 110–126.

[21] D. U. Case, "Analysis of the cyber attack on the ukrainian power grid," *Electricity Information Sharing and Analysis Center (E-ISAC)*, vol. 388, pp. 1–29, 2016.

[22] W. M. Cohen, R. Nelson, and J. P. Walsh, *Protecting their intellectual assets: Appropriability conditions and why us manufacturing firms patent (or not)*, 2000.

[23] D. Tychalas, H. Benkraouda, and M. Maniatakos, "Icsfuzz: Manipulating i/os and repurposing binary code to enable instrumented fuzzing in ics control applications.," in *USENIX Security Symposium*, 2021, pp. 2847–2862.

[24] A. Nochvay, "Security research: Codesys runtime, a plc control framework," *Kaspersky ICS CERT*, pp. 56–61, 2019.

[25] D. H. Hanssen, *Programmable logic controllers: a practical approach to IEC 61131-3 using CODESYS*. John Wiley & Sons, 2015.

[26] D. Tychalas and M. Maniatakos, "Iffset: In-field fuzzing of industrial control systems using system emulation," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2020, pp. 662–665.

[27] N. Bargal, A. Deshpande, R. Kulkarni, and R. Moghe, "Plc based object sorting automation," *International Research Journal of Engineering and Technology, IRJET*, vol. 3, no. 07, 2016.

[28] S. Adams, "The automation of abb: Breakers and switches spindle milling machine," 2012.

[29] D. Tychalas, H. Benkraouda, and M. Maniatakos, "Icsfuzz: Manipulating i/os and repurposing binary code to enable instrumented fuzzing in ics control applications.," in *USENIX Security Symposium*, 2021, pp. 2847–2862.

[30] B. D. Sija, Y.-H. Goo, K.-S. Shim, H. Hasanova, and M.-S. Kim, "A survey of automatic protocol reverse engineering approaches, methods, and tools on the inputs and outputs view," *Security and Communication Networks*, vol. 2018, pp. 1–17, 2018.

[31] R. G. Bace, P. Mell, *et al.*, "Intrusion detection systems," 2001.

[32] N. Desai, "Intrusion prevention systems: The next step in the evolution of ids," *Security Focus*, vol. 27, 2003.

[33] S. Bhatt, P. K. Manadhata, and L. Zomlot, "The operational role of security information and event management systems," *IEEE security & Privacy*, vol. 12, no. 5, pp. 35–41, 2014.

[34] J. A. Marpaung, M. Sain, and H.-J. Lee, "Survey on malware evasion techniques: State of the art and challenges," in *2012 14th International Conference on Advanced Communication Technology (ICACT)*, IEEE, 2012, pp. 744–749.

[35] L. Martn-Liras, M. A. Prada, J. J. Fuertes, A. Morán, S. Alonso, and M. Domnguez, "Comparative analysis of the security of configuration protocols for industrial control devices," *International Journal of Critical Infrastructure Protection*, vol. 19, pp. 4–15, 2017.

[36] A. Nochvay, "Security research: Codesys runtime, a plc control framework," *Kaspersky ICS CERT*, pp. 56–61, 2019.

[37] C. Lei, L. Donghong, and M. Liang, "The spear to break the security wall of s7comm-plus," *Blackhat USA*, 2017.

[38] M. Sutton, A. Greene, and P. Amini, *Fuzzing: brute force vulnerability discovery*. Pearson Education, 2007.

[39] J. Chen *et al.*, "Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing.," in *NDSS*, 2018.

[40] M. Niedermaier, F. Fischer, and A. von Bodisco, "Propfuzzan it-security fuzzing framework for proprietary ics protocols," in *2017 International conference on applied electronics (AE)*, IEEE, 2017, pp. 1–4.

[41] H. Gascon, C. Wressnegger, F. Yamaguchi, D. Arp, and K. Rieck, "Pulsar: Stateful black-box fuzzing of proprietary network protocols," in *Security and Privacy in Communication Networks: 11th EAI International Conference, SecureComm 2015, Dallas, TX, USA, October 26-29, 2015, Proceedings 11*, Springer, 2015, pp. 330–347.

[42] T. Krueger, H. Gascon, N. Krämer, and K. Rieck, "Learning stateful models for network honeypots," in *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, 2012, pp. 37–48.

[43] Z. Luo, F. Zuo, Y. Jiang, J. Gao, X. Jiao, and J. Sun, "Polar: Function code aware fuzz testing of ics protocol," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–22, 2019.

[44] V.-T. Pham, M. Böhme, and A. Roychoudhury, "Aflnet: A greybox fuzzer for network protocols," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, IEEE, 2020, pp. 460–465.

[45] Z. Luo, F. Zuo, Y. Shen, X. Jiao, W. Chang, and Y. Jiang, "Ics protocol fuzzing: Coverage guided packet crack and generation," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2020, pp. 1–6.

[46] D. Fang *et al.*, "Ics3fuzzer: A framework for discovering protocol implementation bugs in ics supervisory software by fuzzing," in *Annual Computer Security Applications Conference*, 2021, pp. 849–860.

[47] *Github repository for apex-ics*. [Online]. Available: https://github.com/ParvinSoni/APEX-ICS.git.

[48] G. Bossert, Guih, and G. Hiet, "Towards automated protocol reverse engineering using semantic information," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014, pp. 51–62.

[49] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces," in *16th USENIX Security Symposium (USENIX Security 07)*, Boston, MA: USENIX Association, 2007.

[50] Y. Ye, Z. Zhang, F. Wang, X. Zhang, and D. Xu, "Netplier: Probabilistic network protocol reverse engineering from message traces," *Proceedings 2021 Network and Distributed System Security Symposium*, 2021.

[51] R. Marcovich, O. Grumberg, and G. Nakibly, "Pise: Protocol inference using symbolic execution and automata learning," *Proceedings 2023 Workshop on Binary Analysis Research*, 2023.

[52] P. M. Comparetti, G. Wondracek, C. Krügel, and E. Kirda, "Prospex: Protocol specification extraction," *2009 30th IEEE Symposium on Security and Privacy*, pp. 110–125, 2009.

[53] J. Caballero, H. Yin, Z. Liang, and D. X. Song, "Polyglot: Automatic extraction of protocol message format using dynamic binary analysis," in *Conference on Computer and Communications Security*, 2007.

[54] Z. Lin, X. Jiang, D. Xu, and X. Zhang, "Automatic protocol format reverse engineering through context-aware monitored execution," in *Network and Distributed System Security Symposium*, 2008.

[55] P. Cousot, "Syntactic and semantic soundness of structural dataflow analysis," in *Sensors Applications Symposium*, 2019.

[56] C. Casinghino, J. T. Paasch, C. Roux, J. Altidor, M. Dixon, and D. Jamner, "Using binary analysis frameworks: The case for bap and angr," in *NASA Formal Methods*, 2019.

[57] C. Leita, K. Mermoud, and M. Dacier, "Scriptgen: An automated script generation tool for honeyd," *21st Annual Computer Security Applications Conference (ACSAC'05)*, 12 pp.–214, 2005.

[58] Y. Wang, Z. Zhang, D. D. Yao, B. Qu, and L. Guo, "Inferring protocol state machine from network traces: A probabilistic approach," in *International Conference on Applied Cryptography and Network Security*, 2011.

[59] S. Zhang, Y. Hu, and G. Bian, "Research on string similarity algorithm based on levenshtein distance," *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pp. 2247–2251, 2017.

[60] M. Zhang *et al.*, "Towards automated safety vetting of plc code in real-world plants," in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 522–538.

[61] K. Katoh and D. M. Standley, "Mafft multiple sequence alignment software version 7: Improvements in performance and usability," *Molecular biology and evolution*, vol. 30, no. 4, pp. 772–780, 2013.

[62] A. A. Shahroom and N. Hussin, "Industrial revolution 4.0 and education," *International Journal of Academic Research in Business and Social Sciences*, vol. 8, no. 9, pp. 314–319, 2018.

[63] A. U. Rustamovna, "Understanding the levenshtein distance equation for beginners," *The American Journal of Engineering and Technology*, vol. 3, no. 06, pp. 134–139, 2021.

[64] L. Gómez, M. Rusinol, and D. Karatzas, "Lsde: Levenshtein space deep embedding for query-by-string word spotting," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, vol. 1, 2017, pp. 499–504.

[65] H. Kim, M. O. Ozmen, A. Bianchi, Z. B. Celik, and D. Xu, "Pgfuzz: Policy-guided fuzzing for robotic vehicles.," in *NDSS*, 2021.

[66] P. Oehlert, "Violating assumptions with fuzzing," *IEEE Security & Privacy*, vol. 3, no. 2, pp. 58–62, 2005.

[67] Y. Li, B. Chen, M. Chandramohan, S.-W. Lin, Y. Liu, and A. Tiu, "Steelix: Program-state based binary fuzzing," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 627–637.

[68] A. Odena, C. Olsson, D. Andersen, and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," in *International Conference on Machine Learning*, PMLR, 2019, pp. 4901–4911.

[69] M. Woo, S. K. Cha, S. Gottlieb, and D. Brumley, "Scheduling black-box mutational fuzzing," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 511–522.

[70] R. McNally, K. Yiu, D. Grove, and D. Gerhardy, "Fuzzing: The state of the art," 2012.

[71] G. Grieco, M. Ceresa, and P. Buiras, "Quickfuzz: An automatic random fuzzer for common file formats," *ACM SIGPLAN Notices*, vol. 51, no. 12, pp. 13–20, 2016.

[72] S. Hernández Ramos, M. T. Villalba, and R. Lacuesta, "Mqtt security: A novel fuzzing approach," *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1–11, 2018.

[73] *Github repository for netzob.* [Online]. Available: https://github.com/netzob/netzob.

[74] *Github repository for netzob.* [Online]. Available: https://github.com/netplier-tool/NetPlier.

[75] T. Hardjono and B. Cain, "Key establishment for igmp authentication in ip multicast," in *1st European Conference on Universal Multiservice Networks. ECUMN'2000 (Cat. No. 00EX423)*, IEEE, 2000, pp. 247–252.