vPLC: A scalable PLC testbed for IIoT security research

Syed Ali Qasim*
Grand Valley State University
Grand Rapids, Michigan, USA
qasims@gvsu.edu

Muhammad Taqi Raza
University of Massachusetts Amherst
Amherst, MA, USA
taqi@umass.edu

Irfan Ahmed
Virginia Commonwealth University
Richmond, Virginia, USA
iahmed3@vcu.edu

ABSTRACT

The rapid expansion of the Industrial Internet of Things (IIoT) has brought forward critical research challenges concerning scalability, robustness, and security. Traditional IIoT infrastructures, confined to lab environments with a limited number of Programmable Logic Controllers (PLCs), hamper large-scale research due to cost and scalability constraints. We propose a solution in the form of a highly scalable virtual PLC (vPLC) for IIoT applications. Our vPLC, as a software-as-a-service (SaaS), can generate hundreds of thousands of virtual PLC instances to simulate a large-scale IIoT network. The vPLC mimics the functionalities of an actual PLC by learning protocol semantics from network dumps of real PLCs and generating PLC templates. This ability allows users to initiate various PLC instances, thereby accurately replicating PLC functionalities like session initiation and control logic handling. In essence, our vPLC serves as an economical, scalable, and flexible research tool that enhances IIoT research and paves the way for advanced applications such as forensic analysis and threat intelligence within IIoT.

KEYWORDS

Industrial Internet of Things (IIOT), Cyber-Physcial Systems, Industrial Control Systems, Programmable Logic Controllers (PLC)

1 INTRODUCTION

The advent of the Industrial Internet of Things (IIoT) has revolutionized the industry sectors, creating a new wave of innovation often referred to as Industry 4.0 [26]. This transformation is brought about by the integration of internet connectivity with traditionally isolated devices, particularly Programmable Logic Controllers (PLCs) [4]. As essential components in automation, PLCs have now become integral elements of IIoT, leading to substantial operational and economic benefits. The integration of these devices has opened up an era of cyber-physical systems, where digital and physical components seamlessly interact in real-time to optimize processes and improve efficiency [30].

However, the rise of IIoT is not without its challenges. The interconnected nature of the IIoT makes it an attractive target for cyber-attacks, leading to a surge in security issues [2, 3, 5–8, 10, 16, 17, 23, 28]. The potential consequences of these attacks range from disruptions in industrial operations to severe damage to physical assets, creating an urgent need for robust security measures [15].

Addressing these security challenges requires extensive testing and experimentation. However, the lack of comprehensive and scalable IIoT testbeds poses a significant barrier. Most current testbeds are either confined to laboratory environments with a limited number of PLCs [14] or situated within industries where data access is heavily restricted [13]. These testbeds are costly, complex to set up, and prone to permanent damage from experimental attacks. This situation curtails large-scale studies and impedes progress in IIoT security research.

To address inherent challenges in traditional PLC systems, we introduce an innovative solution: a virtual PLC (vPLC) designed specifically for IIoT applications. This vPLC is born from the softwarization approach, positioned as a Software as a Service (SaaS) capable of generating and overseeing countless virtual PLC instances. This allows it to emulate an extensive IIoT network, thereby furnishing a scalable, economical, and reusable platform for IIOT research. Unlike conventional hardware-based PLCs, vPLC instances can be swiftly re-instantiated after a cyber-attack, substantially reducing the likelihood of irreversible damage.

In the vPLC's design, the first step is to capture network data from real PLCs. This data is closely examined to understand protocol semantics, which means we pinpoint the exact locations and details of different message fields. From this deep understanding, the vPLC builds a PLC template. This template, which combines protocol semantics and PLC's typical responses to different requests, is at the heart of our system.

After setting up these basics, the vPLC starts acting like a real PLC server. For this simulation to run, it needs the PLC template and some old network data from a real PLC. When someone connects to the vPLC, they're given the impression they're communicating with a real PLC. In reality, the vPLC checks the old network data, finds the right response for the user's request, tweaks it, and sends it back.

This approach stands out due to its adaptability. Instead of simply mimicking a real PLC's function, it enables the launch of multiple vPLC versions concurrently. Depending on the research scenario, each instance can utilize either consistent network data from a single PLC or diverse data sets.

We demonstrate a significant application of the vPLC: forensic analysis of attacks on PLCs in an IIoT environment [25, 29]. These applications underscore the potential of the vPLC as a potent tool in advancing IIoT research and enhancing security measures.

This paper details the development and evaluation of the vPLC, along with its applications in IIoT security research. The findings presented here pave the way for more extensive and diverse research in the field of IIoT, contributing to the resilience and robustness of Industry 4.0. A preliminary version of vPLC is available at [27].

2 INDUSTRIAL IOT

Historically, Cyber-Physical Systems (CPS) have operated in their own silos, performing repetitive operations with minimal optimization or external interaction. With the advent of Industry 4.0, however, the demand for intelligent manufacturing has intensified. This

*Qasim completed this work while he was at Virginia Commonwealth University.

has led to the emergence of the Industrial Internet of Things (IIoT), which seeks to bridge the gap between Operational Technology (OT) and the world of data and business operations. Rather than functioning independently, IIoT utilizes a cooperative approach. Data is collected from individual CPS using sensors, actuators, and other devices, and processed in the IT domain. This data analysis gives rise to effective industrial strategies to optimize resources, decrease expenses, and augment operational efficacy. Once devised, these strategies are executed back into the CPS. As shown in figure 1 the Gartner IIoT architecture [11], a universally acknowledged model, divides IIoT into three distinct tiers: Edge, Platform, and Enterprise.

The Edge tier includes the actual cyber-physical systems, sensors, actuators, and Programmable Logic Controllers (PLCs). Its main functions include data gathering, monitoring, and control of physical processes.

The Platform tier acts as a liaison, offering services such as data filtering, compilation, and storage. This tier preps and transmits the data to the Enterprise level.

The Enterprise tier is where the central IIoT applications are housed. This layer applies machine learning and data processing algorithms to the received data, contributing to the optimization of industrial operations.

Each of these tiers employs different communication networks tailored to their unique needs. The Edge tier uses a serial or proximity network for connecting different devices, the Platform tier leverages Wide Area Networks (WANs) to establish a link between the edge and the platform, and high-speed internet technologies such as 5G are used to connect the Platform and Enterprise tiers.

Despite the array of research challenges posed by IIoT, we have chosen to focus on the Cyber-Physical Systems (CPS) within the Edge tier that blends physical and computational components in this work.

2.1 PLC in IIoT

Cyber-Physical Systems (CPS) are integrative entities that connect the physical world with the digital realm. They are composed of sensors, actuators, and control devices, with Programmable Logic Controllers (PLCs) being a predominant example. In a typical CPS setup, sensors capture signals from physical processes. These input signals are then processed by the PLC in accordance with user-defined control logic (that can be written in and downloaded to the PLC using the engineering software). After processing, the PLC directs the output to the actuators, which in turn manipulate the physical processes. Additionally, PLCs possess network communication capabilities and use ICS protocols such as Modbus, ENIP, S7Comm, etc. The PLCs acting as a server communicate in the form of request and response, i.e. for every request message the PLC receives it sends a response. This enables them to interact with other PLCs, engineering software (proprietary software to monitor and program a PLC), Human Machine Interfaces (HMIs), and the platform tier services within the IIoT framework.

PLCs serve as pivotal embedded devices within the Industrial Internet of Things (IIoT) framework. They establish a critical link between Operational Technology (OT) and Information Technology (IT), making them indispensable to the contemporary industrial landscape. However, their central role also makes PLCs prime targets for numerous cyber-attacks in IIoT. As a result, continuous research is needed to improve security and carry out investigations when attacks occur [18–22]. This ensures that PLCs, and by extension IIoT networks, remain secure and reliable.

3 MOTIVATION

As the Industrial Internet of Things (IIoT) continues to evolve, Programmable Logic Controllers (PLCs) serve as the critical link between the digital and physical realms. However, ensuring their security requires realistic testbeds to rigorously refine our defensive strategies. Existing PLC testbeds, whether in industrial environments or laboratories, often fall short of the complexities inherent in IIoT networks. Industrial testbeds present significant data acquisition challenges and cannot support potentially disruptive experiments. Laboratory-based testbeds, though more accessible, lack the scale to facilitate comprehensive studies.

To serve as a fitting testing ground, a test bed should ideally embody scalability, configurability, and durability. However, conventional physical testbeds often fail to meet these criteria, mainly due to limited scalability, restricted configurability, and high susceptibility to damage, leading to steep repair or replacement costs.

To tackle these issues, we propose the vPLC test bed concept, where instead of using physical PLCs, we simulate them in a software environment. This approach provides a robust, scalable, and cost-efficient platform, ideal for extensive IIoT network research.

Our vPLC mechanism uses packet replay, leveraging network dumps from real PLC communications, to effectively mimic the network behavior of actual PLCs. This innovative shift towards 'softwarization' offers a flexible and resilient platform for developing and perfecting security and forensic solutions, shaping a safer IIoT landscape.

4 vPLC DESIGN

4.1 Overview

The vPLC testbed can consist of one or more virtual PLCs. Given the network dump of a real PLC's communication, the objective of the virtual PLCs is to replay this network dump, replicating the same network abstraction as a real PLC, and providing the application-level functionalities of a real PLC. This process necessitates three functions: (1) Data Management: Using the network dump, the virtual PLC should be capable of organizing the packets to facilitate packet replay and template extraction; (2) Template Generation: The virtual PLC should learn various session-dependent fields and their semantics to update them for new sessions (replay); and (3) Communication Server: In a manner similar to a physical PLC, the virtual PLC should include a communication server and be capable of responding to various request messages.

Figure 2 provides an overview of the vPLC testbed. In the data management phase, the vPLC extracts different sessions from the network dump, identifies request and response messages, and subsequently stores them in a database for further analysis. During the template generation block, the vPLC performs various analytical operations on the communication data stored in the database. This analysis aims to extract the message structure, location, and semantics of different fields in the messages to generate a PLC template.

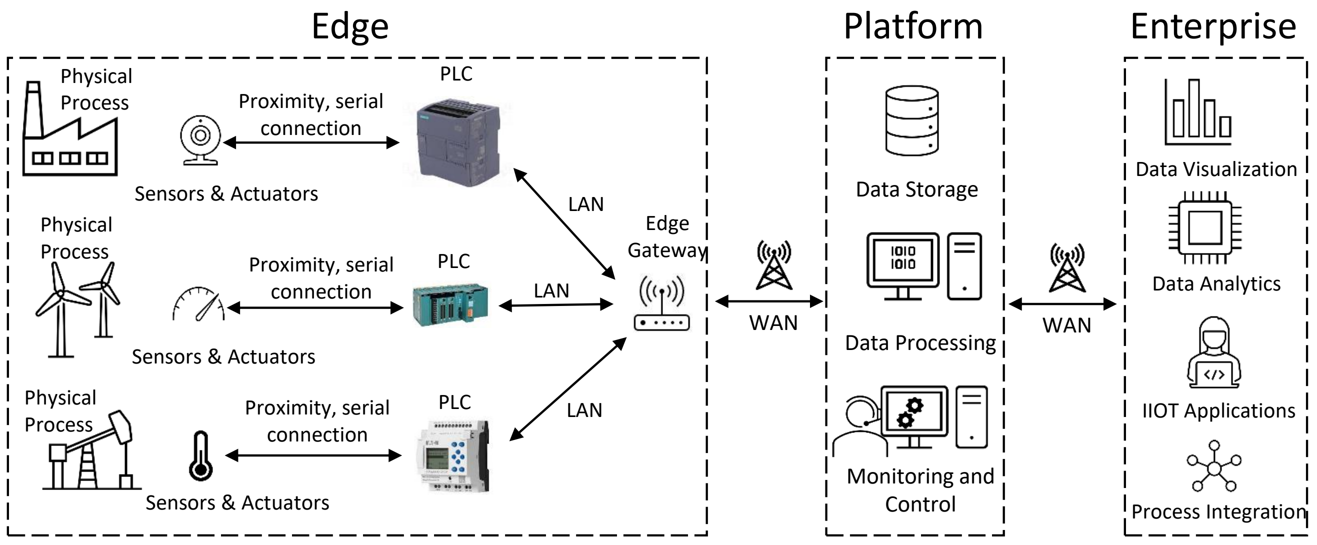

Figure 1: Three tier Gartner IIoT architecture, the Cyber-Physical Systems (physical process, sensors, actuators, and the PLC) reside in the Edge tier

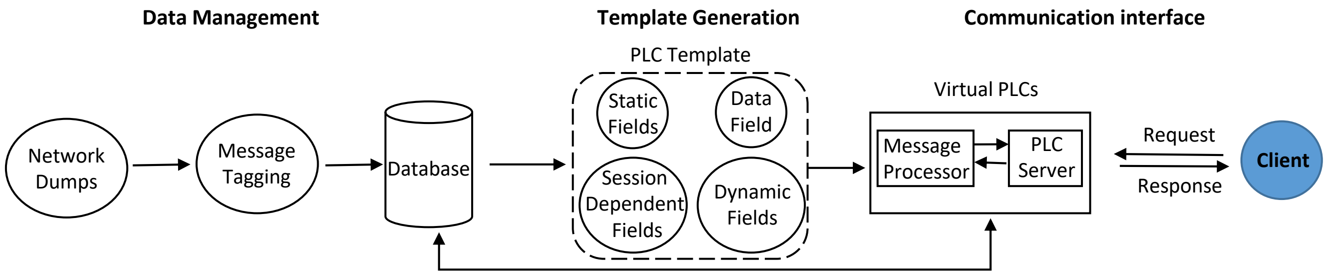

Figure 2: An overview of vPLC

Finally, in the communication interface, the vPLC uses the PLC template and the network dumps to instantiate various instances of virtual PLCs, thereby mimicking a cyber-physical system.

4.2 Data processing

The initial step for the vPLC involves processing network dumps, collected from the active communication of a genuine PLC. These dumps, primarily based on proprietary ICS protocols like Modbus, S7Comm, ENIP, and others, pose significant challenges due to their binary nature. The intricacy deepens as these protocols are not only proprietary but many PLCs, such as Modicon M221, employ multiple nested protocols; for instance, UMAS embedded within the Modbus protocol. Distinguishing individual messages within this layered communication and subsequently organizing them into corresponding request-response pairs is far from straightforward. Without a clear understanding or specialized tools, extracting useful semantic information from these packet captures demands a rigorous, structured methodology.

Message tagging: This begins with identifying distinct communication sessions and recognizing the application-level request and response messages. The vPLC can extract these separate communication sessions from the network dumps based on the IP address and port of the PLC. We then identify the request and response messages within these dumps. Packets whose destination IP and port match the PLC's are tagged as request messages, while those whose source IP and port align with the PLC's are tagged as response messages.

Database: Once the messages have been tagged, they are organized into an efficient database. We treat the application message as a complex binary structure, storing the messages as hex strings, allowing us to remain agnostic of the ICS protocol. To pair the request and response messages together, we've developed a queue system for request messages. Each request message enters a queue and, when a response message is identified, it retrieves the first message in the queue. This paired message is then stored as a key-value pair in the database. Algorithm 1 outlines the process of creating databases.

Algorithm 1 Data Management: Storing the request-response pairs

```
Require: $filename, src\_ip, plc\_ip, p\_port$
 1: $pcap \leftarrow$ rdpcap($filename$)
 2: $table \leftarrow$ empty dictionary
 3: $req \leftarrow$ empty string
 4: $stack \leftarrow$ empty stack
 5: for each $pkt$ in $pcap$ do
 6:     if $pkt$ contains TCP then
 7:         $hex\_payload \leftarrow$ hexlify(payload of $pkt$)
 8:         if source of $pkt$ == $src\_ip$ and destination of $pkt$ == $plc\_ip$ and length of payload > 10 then
 9:             push $hex\_payload$ to $stack$
10:         else if source of $pkt$ == $plc\_ip$ and destination of $pkt$ == $src\_ip$ then
11:             $req \leftarrow$ pop first element from $stack$
12:             $table[req] \leftarrow hex\_payload$
13:         end if
14:     end if
15: end for
```

This methodology ensures we maintain an organized and effective system for interpreting and learning from the communication data.

4.3 PLC Template

The vPLC utilizes a packet replay technique to simulate a real PLC's behavior but replaying network traffic poses numerous challenges. Firstly, several session-dependent fields may exist within a message. The vPLC needs to identify these and establish their relationships in request-response messages. Secondly, according to [1], the message structure changes based on the operation performed, e.g., writing to the PLC memory yields a different request-response structure than reading data from it. Therefore, if the network dump provided to the vPLC was captured during a control logic upload, it could reuse the message after updating the session-dependent fields. However, if captured during a download, the vPLC needs to ascertain the upload message structure and use the downloaded traffic to populate and transmit the upload message. So vPLC solves these challenges by creating a template for each PLC it is mimicking.

Identifying Session Dependent Fields: To identify session dependent fields, the vPLC uses two network dumps where the same PLC operation was executed. These dumps generate separate databases and then form tuples of matching request and response messages from the two different sessions. Given that messages are stored as hex strings, string similarity, and message length are used to identify matching pairs.

Every tuple then undergoes a differential analysis to identify differing bytes between the two request messages, thereby locating the session-dependent fields. The session-dependent fields in request and response messages are compared to establish their relationship. If the session-dependent fields match, the value from the new request message is used to update the fields in the old response message, enabling the replay of the old response message. The identification process is summarized in Algorithm 2.

Algorithm 2 Session Dependent Fields Identification

```
 1: Let $D_1$, $D_2$ be two databases of network dumps
 2: Let $Tuples$ be an empty list
 3: Let $Indices$ be an empty dictionary
 4: for each $req_{1i}$ in $D_1$ do
 5:     Find $match_{req}$ using findMaxMatch($req_{1i}$, $D_2$)
 6:     $Tuples$.append([$req_{1i}$,$match_{req}$,$res_{1i}$,$res_{match_{req}}$])
 7: end for
 8: for each $T$ in $Tuples$ do
 9:     for $index$ = 0 to length($T[0]$) do
10:         if $T[0][index] \neq T[1][index]$ then
11:             if $index$ in $Indices$ then
12:                 $Indices[index] += 1$
13:             else
14:                 $Indices[index] = 1$
15:             end if
16:         end if
17:     end for
18: end for
19: $threshold$ = tn * length($Tuples$)
20: $Indices$ = {index: count for index, count in $Indices$.items() if count > $threshold$}
        return $Indices$
```

Extracting the Message Structure Depending on the operation, the structure of the request and response messages changes. When writing data to memory, the client sends a request message containing the write function code, memory address to write to, size of data to write, and the data or control logic itself. In response, the PLC sends a success message if the operation was successful, or an error message otherwise. To read data from the PLC memory, the client sends a request message containing the read function code, the memory address to read from, and the size of the data to read. The PLC then responds with a success message containing the requested data or an error message. Therefore, if the network dumps the vPLC is replaying contains read operations, the vPLC can send the response after updating the session-dependant fields. However, if the replaying dump only contains write operations and the vPLC is requested to read, the vPLC must extract the structure of the read response message, fill it using the messages in the dump, and then send it as a response to the read request. To extract the read response structure, the vPLC processes two network dumps captured while reading and writing the same control logic. This allows the vPLC to identify the different fields present in a read response message. Generally, there are four types of fields: static fields, dynamic fields, session-dependent fields, and control logic or data fields.

Identifying Static Fields. Algorithm 3 explains the process of identifying the locations of static fields, the vPLC compares all the request messages in a network dump and labels all the fields that remain constant throughout the session (i.e., fields that are identical in all request messages) as static in the request. This is repeated for the response messages, and the locations of static fields in both request and response messages are compared to identify their relationships

Algorithm 3 Static Fields Identification

```
1: Let $D$ be the database of network dumps
2: Let $StaticFields_{Req}$, $StaticFields_{Res}$ be empty lists
3: Initiate $index = 0$
4: while $index < length(D[0])$ do
5:     Let $value$ be the value at $index$ in the first request in $D$
6:     if all request messages in $D$ have the same value at $index$ then
7:         Append $index$ to $StaticFields_{Req}$
8:     end if
9:     Increment $index$
10: end while
11: Repeat the same process for response messages and update $StaticFields_{Res}$
        return $StaticFields_{Req}$, $StaticFields_{Res}$
```

Identifying Control Logic fields. Identifying the control logic or data field in the read response message involves observing that during the reading or writing of a control logic binary, the engineering software consistently divides the binary into equal-sized chunks. These chunks are written to and read from the same memory locations in both operations. Therefore, write requests and read responses containing the same control logic chunk are identified, the longest common sub-sequence of the two is taken and its position in the read response is marked.

Identifying Dynamic fields. To locate dynamic fields such as the length of the request message, a heuristic-based approach is employed. A window of two bytes (a common size of length fields in ICS protocols) is rolled on a message and the value inside the window is compared with the length of the message outside the window. If they match, the location of the window is marked as the potential location of the length field. This process is carried out on each read response message in the database, and the location that appears in all request messages is labeled as the length field. Algorithm 4 gives the overview of the dynamic field identification process. A similar process can be used to identify other dynamic fields like the checksum etc.

After identifying the different types of fields, the vPLC compiles them to determine the structure of the read response message. In the end, all this information, including the message structure, the location of different fields in the message, and their relationships in request and response messages, is stored in the PLC template. This template can then be utilized by the vPLC to accurately replay the network dump.

Algorithm 4 Dynamic Fields Identification

```
1: Let $D$ be a database of network dump
2: Let $PotentialLocations$ be an empty list
3: Let $LengthFieldLocations$ be an empty list
4: for each $msg$ in $D$ do
5:     $msgLength$ = length($msg$)
6:     for $index = 0$ to $msgLength - 2$ do
7:         $windowValue$ = convertToInteger($msg[index : index + 2]$)
8:         if $windowValue = msgLength - index - 2$ then
9:             $PotentialLocations$.append([$index$,$index$+2)
10:        end if
11:    end for
12: end for
13: for $L$ in $PotentialLocations$ do
14:     if $PotentialLocations$.count($L$) = length($D$) then
15:         $LengthFieldLocations$.append($L$)
16:     end if
17: end for
        return $LengthFieldLocations$
```

4.4 Communication Interface- Virtual PLCs

Having organized the network dump in the database and generated the PLC template, the next and final component of the vPLC is the communication interface. The user can instantiate hundreds of virtual PLC instances, providing them with the network dumps to replay (database) and the PLC template. Each virtual PLC mimics the behavior of a real PLC, offering a network abstraction of a real PLC.

PLC server: The virtual PLC consists of two main components: the PLC Server and the Message Processor. The virtual PLC operates a server (running on the same port as a real PLC), enabling communication with clients who can send different request messages to it. Upon receiving a request message, the PLC Server forwards it to the Message Processor, which is tasked with generating an appropriate response message.

Message Processor: Generating Response Messages: For each request message, the Message Processor searches the database for a similar request message, using the message size and string similarity as search parameters. In the first cycle, the Message Processor looks for all request messages in the database with the same length as the current request and finds the message with the highest string similarity. It retrieves the associated response, updates the session-dependent fields for the new session, and sends it to the PLC Server to respond to the client.

If the network dump populates the database and the current operation aligns, the Message Processor will likely find a similar, same-length request in the database. If not, the Message Processor won't find any requests of similar length in the database. In such cases, during the second round, the Message Processor calculates the similarity between the current request message and stored request messages up to the size of the current message.

The Message Processor then selects the message with the highest similarity from the database, retrieves the associated response, and uses it to generate a new response message in line with the PLC template. This response is then sent to the PLC Server, which replies to the client. In this way, by responding to all incoming request messages, the virtual PLC effectively mimics the behavior and operations of a real PLC.

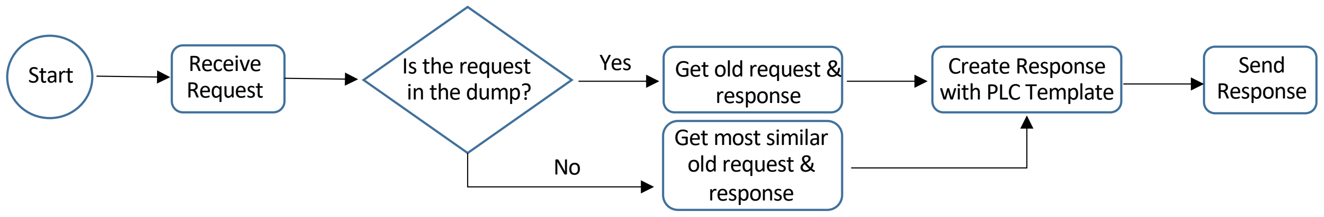
Start
Receive Request
Is the request in the dump?
Yes
No
Get old request & response
Get most similar old request & response
Create Response with PLC Template
Send Response

Figure 3: Flowchart of virtual PLC communication

5 IMPLEMENTATION

The vPLC system was implemented using Python due to its extensive suite of libraries and ease of readability.

The first step of our process involved dealing with network dumps, and for this, we relied on Scapy, a powerful Python-based interactive packet manipulation program and library. Scapy[24] enabled us to analyze and dissect the network packets, providing the foundational framework upon which we built our system.

Creating databases was our next step, and for this, we utilized Python's built-in dictionary data structure. In our database, each key corresponds to a request message, with the corresponding response message stored as its value. This key-value structure provided an efficient means of retrieving and updating the response messages associated with each request.

Identifying similar messages required us to calculate the similarity between strings. This was accomplished using Python's SequenceMatcher function from the difflib library[9]. The SequenceMatcher function allowed us to compare two input sequences, in our case the hex strings of our messages, and determine their level of similarity.

For the virtual PLC server, we used Python's socket library. We used the socket.socket() function to create the server and handle the networking layer of the vPLC.

Lastly, to detect incoming requests, we employed the sniff function from Scapy. The sniff function passively listens for packets on the network, triggering our Message Processor every time a packet arrives. This allowed our vPLC to dynamically respond to incoming requests, thereby mimicking the behavior of a real PLC.

6 EVALUATION

The vPLC has the capability to generate multiple virtual PLCs, each imitating real PLC behavior by replaying captured network traffic. This ability allows vPLC to create a realistic Industrial Internet of Things (IIoT) testbed. We primarily wanted to ensure that the virtual PLC could accurately impersonate real PLCs, effectively replay the captured network dumps, and process the request messages within a short time frame. These three aspects serve as key matrices for the evaluation of our proposed system.

Experimental Setup: The evaluation of the Virtual Programmable Logic Controllers (vPLCs) encompassed their impersonation of three PLCs: the Allen-Bradley MicroLogix 1400 and 1100, and Schneider Electric Modicon M221. Corresponding engineering software, RSLogix 500 and MachineExpertBasic were used for communication. The MicroLogix PLCs employ the PCCC protocol within EtherNet/IP (ENIP), while Modicon M221 uses the UMAS protocol within Modbus.

The vPLC ran on an Ubuntu 18 VM, while the engineering software operated on a Windows 10 VM. All real PLCs, vPLC, and the Windows VM were networked together, facilitating accurate vPLC testing and response evaluation.

Experimental Methodology: Experimental Methodology: To simulate a genuine IIoT scenario, we utilized 20 different control logic programs of varying sizes and complexities for each PLC. Through the engineering software, we connected to a real PLC and executed the 'upload control logic' operation, where the software reads the control logic program running on the PLC and then captures the ensuing network traffic. This captured data was subsequently fed into the vPLC, which instantiated a virtual PLC. We methodically repeated this process for each of the three PLCs: MicroLogix 1400, MicroLogix 1100, and Modicon M221.

6.1 Impersonation of a real PLC

The primary role of the vPLC is to emulate the communication characteristics of a real PLC. Consequently, the vPLC must be recognized as a real PLC by the PLC engineering software, maintain a viable communication session with the software, and execute application tasks, such as control logic transfers.

Results: In our evaluation, the vPLC was successfully recognized as the real PLCs (MicroLogix 1400, 1100, and Modicon M221) by the corresponding engineering software. In addition, the vPLC effectively established and maintained a communication session. Importantly, control logic upload operations were successful, with the vPLC uploading all the control logic presented in the network dumps accurately.

Table 1 summarizes the results of upload operations. We manually compared the control logic programs (20 each MicroLogix 1400, 1100, and Modicon M221) uploaded by virtual PLC and the real PLCs. Our experiment showed that they contained the same number of rungs (blocks of PLC code). This demonstrates the vPLC's capacity to effectively impersonate a real PLC and successfully perform PLC operations such as control logic upload.

6.2 Ability to Replay Network Traffic

The efficacy of the vPLC, which employs a packet replay technique to mimic a real PLC, relies on two key factors. The first is the completeness of the database, meaning that the network dump captured during any PLC operation will contain all messages necessary to recreate the session, without generating any new messages from the engineering software. The second is successful database lookups,

Table 1: Control Logic Upload Accuracy of vPLC

PLC	# of control logic files uploaded	Original Program (Rungs)	vPLC Program (Rungs)	Upload Accuracy %
MicroLogix 1400	20	109	109	100%
MicroLogix 1100	20	235	235	100%
Modicon M221	20	211	211	100%

Table 2: Request messages received by the virtual PLC and their identification in the Database

PLC	# of Experiments	# of Request Messages Received	# of Request Messages Found in DB	Lookup Success %
MicroLogix 1400	20	2060	2060	100%
MicroLogix 1100	20	1440	1440	100%
Modicon M221	20	3500	3500	100%

i.e., the vPLC's ability to identify the identical request in the database for each received request message. This task is challenging due to the proprietary nature of ICS protocols and the possibility of multiple session-dependent fields.

Results: Our results indicate that the network dump indeed contains all the messages required to successfully recreate a communication session and that the vPLC can effectively locate the exact message in the database populated by the previous network dump. Table 2 summarizes the results of our experiments.

During 20 different upload experiments, the MicroLogix 1400 received 2060 messages, all of which were successfully identified by the vPLC in the database. For the MicroLogix 1100, across 20 experiments, the vPLC received 1440 request messages and located all of them in the database. Lastly, for the Modicon M221, the vPLC received 3500 request messages across 20 upload experiments and successfully identified all the messages in the database. This suggests a successful implementation of the packet replay technique in the vPLC design.

6.3 Processing Time Comparison

6.3.1 Processing Efficiency Evaluation. For seamless integration into the IIOT ecosystem, it's crucial that our vPLC responds to requests in a timely manner, akin to a real PLC. Therefore, we compared the request processing times of real and virtual PLCs under different scenarios tied to the nature of operations conducted during network dump capture.

We compared the request processing time of a real PLC with the vPLC under two scenarios: when the network dump was generated during an upload (read) operation, and when it was created during a download (write) operation. Our rationale was that the processing

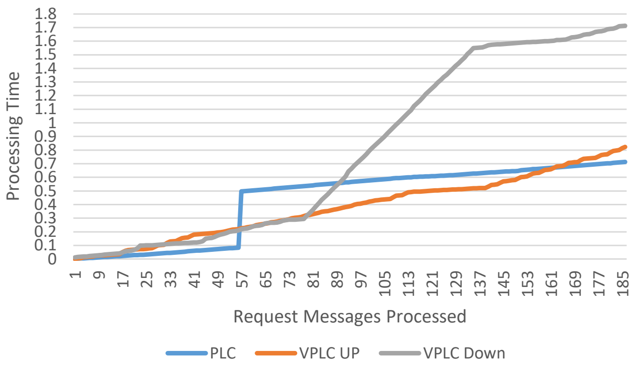

Figure 4: Comparison of Message Processing Time between a Real PLC and virtual PLC using upload and download network dumps

complexity differs based on the nature of the operation performed while capturing the network dump.

Our findings, depicted in Figure 6, reveal that while the vPLC takes slightly more time to process a request compared to a real PLC, the delay is acceptable. Specifically, a real M221 PLC processes a request message in an average of 0.0038 seconds. Meanwhile, the vPLC takes an average of 0.0044 seconds to process a request message given a network dump of an upload operation and 0.0092 seconds given a network dump of a download operation. These results confirm our assumption that the vPLC processes requests faster when the network dump's operation matches the current operation. Although there is a slight delay with the vPLC, we observed no connection timeouts or interruptions during our tests. Therefore, these findings demonstrate that the vPLC is capable of efficiently replaying network traffic to effectively impersonate a real PLC.

7 CASE STUDY: INVESTIGATING IIOT ATTACKS USING VPLC

The IIoT realm is evolving rapidly, with emerging sophisticated cyber-physical system threats like Denial of Engineering Operations (DEO) attacks [25]. These attacks compromise a PLC's remote maintenance capabilities, as evidenced on an Allen-Bradley MicroLogix 1400-B via RSlogix 500 engineering software.

In the DEO I attacks, as illustrated in figure 5, an attacker sets up a 'man-in-the-middle' interception between the PLC and the engineering software. When a control engineer attempts to download control logic onto the PLC, the attacker captures this communication, altering the control logic that gets loaded onto the PLC. Later, to hide this malicious intervention, when the control engineer tries to view the control logic currently operating on the PLC, the attacker steps in once more. They deceive the engineer by displaying the original, benign version of the control logic. As a result, the engineer remains unaware that a malicious control logic has been running on the PLC throughout.

Attack Scenario: The system under attack is governed by a Micrologix 1400 PLC, which is tasked with controlling a traffic signal

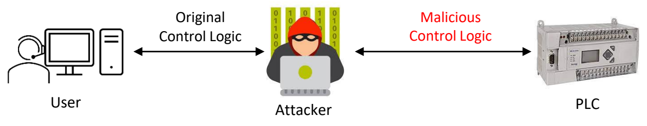

Figure 5: Denial of Engineering Operations Attack (DEO I): Concealing compromised ladder logic from the engineering software

light. The PLC is remotely managed by the engineering software, RSLogix, installed on a computer situated at the control center. Through RSLogix, a user has the capability to not only monitor the PLC's activities but also exert control and make modifications to the control logic running on the PLC from a remote location.

Attack Methodology: The assault is initiated when an attacker successfully establishes a 'man-in-the-middle' position between the traffic signal-controlling Micrologix PLC and its associated engineering software. This is achieved through ARP poisoning. Upon the engineer's attempt to download control logic to the PLC, the attacker captures this communication, specifically targeting the timer instruction responsible for controlling the green light's duration. With the use of an Ettercap filter [12], the preset value within this timer instruction—essentially setting the green light's wait time—is maliciously adjusted, changing it from the standard 20 seconds to an extended 40 seconds. To ensure the subterfuge remains undetected, when the control logic is uploaded back from the PLC for review, the attacker intercepts the data once more, restoring the timer's preset value to its original state. This cunning approach ensures the corrupted control logic continues to run on the PLC, while the engineer remains unsuspecting of any tampering.

Challenges in DEO Attack Forensics: Investigating DEO attacks is a rigorous task, accompanied by several complexities. Even if one manages to capture the network communication during the onset of the attack, the deciphering process is intricate. The primary evidence of a successful intrusion would be the malicious control logic downloaded on the PLC. However, two primary challenges obstruct the seamless extraction of this control logic from the network dump:

Protocol Obfuscation: Micrologix 1400 PLC utilizes the PCCC protocol wrapped within the ENIP protocol. Given that these protocols are binary and proprietary, it necessitates the reverse engineering of their structure and semantics. This task becomes formidable without access to the original design documents or specialized tools.

Decompiling the Control Logic: The control logic, once compiled, is stored in a binary format – a non-human-readable form. To understand its operations, it must be reverted to its source, typically in formats like ladder logic. This decompilation is challenging because it demands an understanding of the precise compilation process, which the engineering software undertakes. Hence, reverse engineering the logic becomes imperative.

In light of these challenges, our approach leverages the vPLC system to navigate and potentially overcome these barriers.

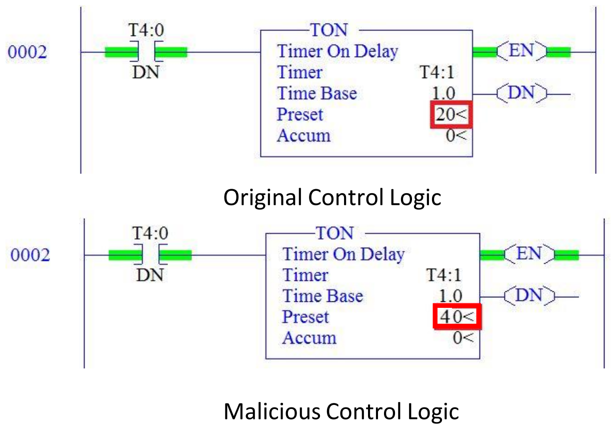

Figure 6: Control logics retrieved from the network dumps of DEO I attack

Forensic Analysis of DEO I Attack:

To investigate the DEO I attack, we leveraged the capabilities of vPLC. A crucial feature of the vPLC is its ability to replay captured network traffic, which proved instrumental in our analysis. By integrating the compiler present within the engineering software, vPLC managed to effectively reconstruct the communication that transpired during the attack. In doing so, vPLC bore the burden of reverse engineering the proprietary ICS protocols.

This integration enabled us to recreate the communication stream and replay it back to the engineering software. As a result, the vPLC handled the intricacies of the ICS protocols, while the engineering software facilitated the decompilation of the machine-readable control logic into a more comprehensible ladder logic form.

Following this process, we separated the communication streams, discerned through MAC addresses, into two categories: from the PLC to the attacker, and from the attacker to the engineering software. We then replayed these streams individually on vPLC, connected the engineering software, and uploaded the control logic. This step transformed the previously obfuscated control logic into a human-readable format, as evidenced in figure 6.

The outcome of our forensic approach underscored vPLC's indispensable role in dissecting complex IIoT cyber-attacks. Not only did the replay function of vPLC provide a window into the attacker's methods, but it also paved the way for the development of potential countermeasures.

8 CONCLUSION

The increasing prevalence of the Industrial Internet of Things (IIoT) has created an urgent need for scalable testbeds. In this paper, we introduce the virtual Programmable Logic Controller (vPLC) as a solution. Through its software-based approach, the vPLC is capable of creating multiple virtual PLC instances, simulating extensive IIoT networks.

At its core, the vPLC uses a network replay mechanism, enabling it to mimic real PLCs by replaying network dumps captured from these devices. It learns the structure of session-dependent fields and messages from the network traffic, enhancing its replay accuracy.

We validated the vPLC using three different PLCs and demonstrated its capability to effectively impersonate real PLC operations. Additionally, we illustrated how the vPLC can aid IIoT research, particularly in the forensic analysis of cyberattacks on IIoT networks, highlighting its potential to advance cybersecurity research.

In conclusion, the vPLC provides a scalable and versatile platform for IIoT research, driving the development of stronger security measures in this expanding field.

REFERENCES

[1] 2023. PREE: Heuristic builder for reverse engineering of network protocols in industrial control systems. *Forensic Science International: Digital Investigation* 45 (2023), 301565. https://doi.org/10.1016/j.fsidi.2023.301565

[2] Irfan Ahmed, Sebastian Obermeier, Martin Naedele, and Golden G. Richard III. 2012. SCADA Systems: Challenges for Forensic Investigators. *Computer* 45, 12 (2012), 44–51. https://doi.org/10.1109/MC.2012.325

[3] Irfan Ahmed, Sebastian Obermeier, Sneha Sudhakaran, and Vassil Roussev. 2017. Programmable Logic Controller Forensics. *IEEE Security & Privacy* 15, 6 (2017), 18–24. https://doi.org/10.1109/MSP.2017.4251102

[4] Irfan Ahmed, Vassil Roussev, William Johnson, Saranyan Senthivel, and Sneha Sudhakaran. 2016. A SCADA System Testbed for Cybersecurity and Forensic Research and Pedagogy. In *Proceedings of the 2nd Annual Industrial Control System Security Workshop* (Los Angeles, CA, USA) *(ICSS '16)*. Association for Computing Machinery, New York, NY, USA, 1–9. https://doi.org/10.1145/3018981.3018984

[5] Adeen Ayub, Wooyeon Jo, Syed Ali Qasim, and Irfan Ahmed. 2023. How Are Industrial Control Systems Insecure by Design? A Deeper Insight Into Real-World Programmable Logic Controllers. *IEEE Security & Privacy* 21, 4 (2023), 10–19. https://doi.org/10.1109/MSEC.2023.3271273

[6] Adeen Ayub, Hyunguk Yoo, and Irfan Ahmed. 2021. Empirical Study of PLC Authentication Protocols in Industrial Control Systems. In *2021 IEEE Security and Privacy Workshops (SPW)*. 383–397. https://doi.org/10.1109/SPW53761.2021.00058

[7] Adeen Ayub, Nauman Zubair, Hyunguk Yoo, Wooyeon Jo, and Irfan Ahmed. 2023. Gadgets of Gadgets in Industrial Control Systems: Return Oriented Programming Attacks on PLCs. In *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 215–226. https://doi.org/10.1109/HOST55118.2023.10132957

[8] T. M. Chen and S. Abu-Nimeh. 2011. Lessons from Stuxnet. *Computer* 44, 4 (2011), 91–93.

[9] diflib. [n. d.]. https://docs.python.org/3/library/difflib.html

[10] Dragos and Dragos. 2022. CRASHOVERRIDE: Analyzing the malware that attacks power grids. https://www.dragos.com/resource/crashoverride-analyzing-the-malware-that-attacks-power-grids/

[11] Ettercap. [n. d.]. https://www.missionsecure.com/blog/purdue-model-relevance-in-industrial-internet-of-things-iiot-cloud

[12] Ettercap. [n. d.]. https://www.ettercap-project.org/

[13] Haihui Gao, Yong Peng, Kebin Jia, Zhonghua Dai, and Ting Wang. 2013. The Design of ICS Testbed Based on Emulation, Physical, and Simulation (EPS-ICS Testbed). In *2013 Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. 420–423. https://doi.org/10.1109/IIH-MSP.2013.111

[14] Joseph Gardiner, Barnaby Craggs, Benjamin Green, and Awais Rashid. 2019. Oops I Did It Again: Further Adventures in the Land of ICS Security Testbeds. In *Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy* (London, United Kingdom) *(CPS-SPC'19)*. Association for Computing Machinery, New York, NY, USA, 75–86. https://doi.org/10.1145/3338499.3357355

[15] Nickolaos Koroniotis, Nour Moustafa, Francesco Schiliro, Praveen Gauravaram, and Helge Janicke. 2023. The SAir-IIoT Cyber Testbed as a Service: A Novel Cybertwins Architecture in IIoT-Based Smart Airports. *IEEE Transactions on Intelligent Transportation Systems* 24, 2 (Feb 2023), 2368–2381. https://doi.org/10.1109/TITS.2021.3106378

[16] Robert M. Lee and Dragos. 2023. Trisis: Analyzing safety system targeting malware. https://www.dragos.com/resource/trisis-analyzing-safety-system-targeting-malware/

[17] Syed Ali Qasim, Adeen Ayub, Jordan Johnson, and Irfan Ahmed. 2022. Attacking the IEC 61131 Logic Engine in Programmable Logic Controllers. In *Critical Infrastructure Protection XV*, Jason Staggs and Sujeet Shenoi (Eds.). Springer International Publishing, Cham, 73–95.

[18] Syed Ali Qasim, Juan Lopez, and Irfan Ahmed. 2019. Automated Reconstruction of Control Logic for Programmable Logic Controller Forensics. In *Information Security*, Zhiqiang Lin, Charalampos Papamanthou, and Michalis Polychronakis (Eds.). Springer International Publishing, Cham, 402–422.

[19] Syed Ali Qasim, Jared M. Smith, and Irfan Ahmed. 2020. Control Logic Forensics Framework using Built-in Decompiler of Engineering Software in Industrial Control Systems. *Forensic Science International: Digital Investigation* 33 (2020), 301013. https://doi.org/10.1016/j.fsidi.2020.301013

[20] Muhammad Haris Rais, Muhammad Ahsan, and Irfan Ahmed. 2023. FRoMEPP: Digital forensic readiness framework for material extrusion based 3D printing process. *Forensic Science International: Digital Investigation* 44 (2023), 301510. https://doi.org/10.1016/j.fsidi.2023.301510 Selected papers of the Tenth Annual DFRWS EU Conference.

[21] Muhammad Haris Rais, Rima Asmar Awad, Juan Lopez, and Irfan Ahmed. 2021. JTAG-based PLC memory acquisition framework for industrial control systems. *Forensic Science International: Digital Investigation* 37 (2021), 301196. https://doi.org/10.1016/j.fsidi.2021.301196

[22] Muhammad Haris Rais, Rima Asmar Awad, Juan Lopez, and Irfan Ahmed. 2022. Memory forensic analysis of a programmable logic controller in industrial control systems. *Forensic Science International: Digital Investigation* 40 (2022), 301339. https://doi.org/10.1016/j.fsidi.2022.301339 Selected Papers of the Ninth Annual DFRWS Europe Conference.

[23] Muhammad Haris Rais, Ye Li, and Irfan Ahmed. 2021. Dynamic-thermal and localized filament-kinetic attacks on fused filament fabrication based 3D printing process. *Additive Manufacturing* 46 (2021), 102200. https://doi.org/10.1016/j.addma.2021.102200

[24] Scapy. [n. d.]. https://scapy.net/

[25] Saranyan Senthivel, Shrey Dhungana, Hyunguk Yoo, Irfan Ahmed, and Vassil Roussev. 2018. Denial of Engineering Operations Attacks in Industrial Control Systems. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy* (Tempe, AZ, USA) *(CODASPY '18)*. ACM, New York, NY, USA, 319–329. https://doi.org/10.1145/3176258.3176319

[26] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. 2018. Industrial Internet of Things: Challenges, Opportunities, and Directions. *IEEE Transactions on Industrial Informatics* 14, 11 (Nov 2018), 4724–4734. https://doi.org/10.1109/TII.2018.2852491

[27] vPLC. [n. d.]. https://gitlab.com/sqasim1/vplc/-/blob/main/VirtualPLC.py

[28] Nauman Zubair, Adeen Ayub, Hyunguk Yoo, and Irfan Ahmed. 2022. Control Logic Obfuscation Attack in Industrial Control Systems. In *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*. 227–232. https://doi.org/10.1109/CSR54599.2022.9850326

[29] Nauman Zubair, Adeen Ayub, Hyunguk Yoo, and Irfan Ahmed. 2022. PEM: Remote forensic acquisition of PLC memory in industrial control systems. *Forensic Science International: Digital Investigation* 40 (2022), 301336. https://doi.org/10.1016/j.fsidi.2022.301336 Selected Papers of the Ninth Annual DFRWS Europe Conference.

[30] Muhammad Zubair Islam, Shahzad, Rashid Ali, Amir Haider, and Hyungseok Kim. 2021. IoTactileSim: A Virtual Testbed for Tactile Industrial Internet of Things Services. *Sensors* 21, 24 (2021). https://doi.org/10.3390/s21248363