By the Numbers: Towards Standard Evaluation Metrics for Programmable Logic Controllers' Defenses

Efrén López-Morales
elopezmorales@islander.tamucc.edu
Texas A&M University-Corpus Christi
Corpus Christi, TX, USA

Jacob Hopkins
jhopkins2@islander.tamucc.edu
Texas A&M University-Corpus Christi
Corpus Christi, TX, USA

Alvaro A. Cardenas
alacarde@ucsc.edu
University of California, Santa Cruz
Santa Cruz, CA, USA

Ali Abbasi
abbasi@cispa.de
CISPA Helmholtz Center for Information Security
Saarbrücken, Saarland, Germany

Carlos Rubio-Medrano
carlos.rubiomedrano@tamucc.edu
Texas A&M University-Corpus Christi
Corpus Christi, TX, USA

Abstract

Our modern society relies on important utility infrastructures such as water treatment plants and electric energy distribution grids. These infrastructures are managed by Industrial Control Systems (ICS), which include devices such as sensors, actuators and Programmable Logic Controllers (PLCs). PLCs are a key component of ICS as they serve as a *bridge* connecting the cyber and physical worlds. A cyberattack on a PLC could have disastrous real-world consequences, such as longstanding energy blackouts.

Researchers have produced a plethora of security defenses in order to safeguard PLCs from cyberattacks, e.g., PLC-specific Intrusion Detection Systems (IDS). However, most of these defenses report incomplete or no performance evaluation metrics. Worse, the defenses that do report metrics evaluate them in an, *ad-hoc* way without providing details. As a consequence, PLC defenses cannot be compared or built upon, which is one of the main ways science progresses. It also makes it difficult to assess the effectiveness of such defenses against attacks.

In this paper, we propose a standard set of performance evaluation metrics designed specifically for PLC security defenses. We propose three types of metrics: security, overhead, and effectiveness metrics. We then lay out what are the challenges faced when collecting these metrics, e.g., the heterogeneity of PLC architectures, and provide recommendations on how these challenges can be addressed to obtain accurate metrics. Obtaining and reporting these metrics will enable researchers to move PLC security research forward ultimately improving the security of ICS and our critical infrastructure.

CCS Concepts

• **Computer systems organization → Embedded software**; **Embedded and cyber-physical systems**.

RICSS '24, October 14–18, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1226-5/24/10
https://doi.org/10.1145/3689930.3695204

Keywords

Programmable Logic Controllers, Cyber-Physical Systems, Industrial Control Systems, Security

ACM Reference Format:
Efrén López-Morales, Jacob Hopkins, Alvaro A. Cardenas, Ali Abbasi, and Carlos Rubio-Medrano. 2024. By the Numbers: Towards Standard Evaluation Metrics for Programmable Logic Controllers' Defenses. In *Proceedings of the 2024 Workshop on Re-design Industrial Control Systems with Security (RICSS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3689930.3695204

1 Introduction

Industrial Control Systems (ICS) underpin the infrastructure of many of the utilities that we use on a daily basis, from water treatment plants to nuclear power facilities. One of the most important elements of ICS are Programmable Logic Controllers (PLCs) [25]. PLCs control physical processes by reading sensor inputs and executing control logic to produce outputs destined for actuators.

PLCs have been the target of multiple cyberattacks, including Stuxnet [25], Industroyer [48], and IronSpider [45]. In order to secure PLCs from such attacks, extensive research has been conducted on developing defensive security methods or simply *defenses*, e.g., control-flow integrity (CFI) methods that defend against control-flow hijacking attacks [2]. However, currently, the PLC security literature is facing a *reproducibility* crisis. A recent study found that only 16% of PLC research papers included any research artifact, let alone functionality or reproducibility [39]. Without the ability to reproduce vulnerabilities and attacks in PLCs, it becomes more difficult for researchers to further investigate and develop countermeasures. This represents a major research gap within the current literature that needs to be addressed. There are two main ways to overcome the reproducibility gap: research artifacts and standardized evaluation metrics. Currently, few PLC security articles include research artifacts of any kind [39]. Some reasons for the lack of public PLC security research artifacts include funding and distribution restrictions [59]. Consequently, despite the importance of sharing PLC research artifacts, an alternative that allows researchers insight into other PLC research is needed. This alternative is standardized performance evaluation metrics.

Standardized performance evaluation metrics would allow PLC security methods to be experimentally compared to each other,

which is not readily done due to the reproducibility issue in PLC security research. Also, standardized evaluation metrics would prove to be an alternative to having access to the original research artifacts because relaying the results of these metrics would be simpler than sharing the research artifacts. However, this requires the original PLC defense paper to share clear evaluation metrics that can be compared. Regrettably, there are very few reported evaluation metrics for PLCs, standard or otherwise. This is not a new problem; in 2021, Sun et al. recommended the development of PLC security benchmarks [56], and in 2024, Lopez-Morales et al. again recommended that PLC security papers report evaluation metrics [39].

With that in mind, we ask the following research questions:

Q-1 **What are the key evaluation metrics for PLC security defenses?**

Q-2 **What are the challenges in obtaining these evaluation metrics?**

Q-3 **How can these challenges be addressed?**

In this paper, we aim to address these questions by introducing a set of quantitative performance evaluation metrics that will allow us to compare PLC defenses even if the corresponding research artifacts are not available. Specifically, we propose three sets of evaluation metrics: *security*, *overhead*, and *effectiveness* metrics. Security metrics will evaluate well-established security principles and techniques, e.g., the Principle of Least Privilege. For example, whether the defense increases (o decreases) the attack surface of the ICS in which the PLC resides. Conversely, Effectiveness metrics will measure how *well* the defense accomplishes its task. For example, if the defense performs anomaly detection, then its effectiveness can be measured by its accuracy. Finally, Overhead metrics will measure how *expensive* it is for the PLC to run the defense method. For example, increasing the PLC scan cycle time.

In addition, we point out challenges that researchers might encounter when measuring our proposed metrics. For example, due to the variety in PLC's architecture it is difficult to find the required benchmarking tools that measures an specific metric. Finally, we provide recommendations on how these challenges can be addressed so that quality metrics can be obtained and reported.

2 Background

This section introduces key background concepts including PLCs, PLC architecture, PLC security defenses, existing evaluation metrics, and profiling and benchmarking.

2.1 Programmable Logic Controllers

A Programmable Logic Controller (PLC) is a small industrial computer designed to run control logic based on input provided by sensors such as temperature sensors. PLCs control complex industrial processes, making them ubiquitous in ICS and SCADA environments [15]. Popular PLC manufacturers include Siemens, Rockwell and Wago [39].

PLCs can be categorized as *HardPLCs* and *SoftPLCs* [39]. A HardPLC is a traditional PLC which includes proprietary software are hardware with an unknown architecture, for example, the Siemens S7-300 PLC. SoftPLCs on the other hand provide a portable software runtime environment that executes control logic programs, for example, programs that follow the IEC 61131-3 standard [58]. SoftPLCs are portable and compatible with multiple hardware such as Raspberry Pis [43]. The two leading SoftPLC platforms are CODESYS [21] and OpenPLC [43].

2.2 PLC Architecture

In general, PLCs have the basic components depicted in Fig. 2. We now describe each component and their security characteristics.

Control Logic. A control logic program contains the instructions that the PLC executes to interact with its environment. Modern control logic programs follow the IEC 61131-3 standard [58] and are written in one of the standard's supported languages, e.g., Structured Text. Depending on the PLC platform, control logic can be compiled into machine code before runtime or it can be translated to machine code during runtime via just-in-time compilation [12]. For example CODESYS' control logic is compiled before runtime and Siemens' PLCs compile control logic during runtime [31].

Runtime Environment. The runtime environment executes the control logic [26] and interacts with the Input/Output modules. It can be proprietary, e.g., CODESYS, or open source like the OpenPLC runtime [7].

Operating System. Most PLCs have a Real-Time Operating System (RTOS) [55]. RTOS are operating systems that meet strict processing time requirements and support real-time applications. Vendors support a variety of RTOS in their platforms. For example, the Siemens S7-1200 PLC uses the Linux-based ADONIS RTOS [3]. However, new runtime options such as OpenPLC run on top of non-RTOS such as Raspian and Windows [42].

Firmware. The firmware bridges the gap between the PLC hardware and software. While simple PLCs might run applications as bare metal (without an OS) [14, 63], modern PLCs use the firmware under an RTOS.

CPU. The CPU interprets the input signals and executes the logic instructions saved in memory. PLCs use different CPU or processor cores depending on the manufacturer and model. Some PLCs, such as the Wago PFC200, include an ARM Cortex A8 processor core [60] while other PLCs might use real-time processors such as the Cortex-R processor family [9].

Memory. It stores the program that the CPU will execute along with input data. Depending on the PLC processor, memory is protected by either a Memory Management Unit (MMU) or a Memory Protection Unit (MPU) [57].

Network Module. Modern PLCs have one or more ports to communicate with the supervisory control network (regular computers monitoring the process) or fieldbus (actuators and sensors).

Physical I/O Modules. These include input modules with metal pins that receive information (via a voltage or current analog signal) from sensors. The output modules send analog data to actuators such as servo motors.

PLC Scan Cycle. The PLC scan cycle refers to the cyclic execution of the PLC control logic, reading inputs, and updating outputs [5]. The control logic contains a set of executable tasks, which vary in size and nature. These tasks affect the scan cycle time based on performance and configuration [29]. The scan cycle time can vary depending on several factors, including control logic program complexity, I/O load, and communication tasks [47].

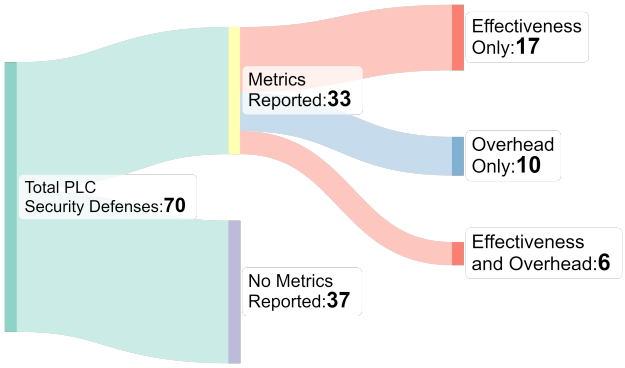

Figure 1: Distribution of 70 PLC security defenses and their reported evaluation metrics. Encompasses PLC security research literature from 2007 to 2023. Effectiveness metrics include accuracy and false positives. Overhead metrics include CPU and scan cycle overhead.

Defense/ Metric	Reported Overhead	Overhead Type	Tool Used
ECFI [2]	1.5%	CPU	perf [22]
D-Box [41]	2.0%	CPU	DWT [11]
Ghostbuster [1]	1%	CPU	PMU Driver [10]
Shade [62]	2%	CPU	N/S
C^2 [40]	3.6%	Scan Cycle	N/S
SnapShotter [30]	54 µs	Scan Cycle	N/S
ICSPatch [47]	17.4 µs	Scan Cycle	N/S
Smart I/O [44]	200 ms	CPU	N/S
Zeus [28]	0%	N/S	N/S

Table 1: Comparison of PLC security defenses that explain how overhead evaluation was performed. N/S = Not Specified.

2.3 PLC Security Defenses

As of 2023, the security community has produced approximately 70 PLC defenses [39]. These include a wide array of approaches such as anomaly detection [5, 37, 49], intrusion detection systems (IDS) [27], and vulnerability patching [47] just to mention a few.

To implement these defenses, different components of the PLC, discussed in Sec. 2.2, need to be modified. For example, ECFI, a control flow integrity (CFI) defense, modified the PLC runtime using assembly code instrumentation [2]. Other defenses utilize the PLC network module to implement their approach. For example, Snapshotter, an IDS, introduces an IDS agent between the PLC and a server [30]. Regardless of the PLC component that a particular defense modifies, these modifications might introduce undesirable side effects such as additional CPU processing, increased privileged processes, or false positives, among others [2]. This is where performance evaluation metrics come into the picture. We can use these metrics to quantify the performance of a particular PLC defense.

2.4 Existing Evaluation Metrics

Multiple metrics have been used to evaluate PLC defenses throughout the years. These include scan cycle overhead, CPU overhead, false positive rate, and accuracy. As depicted in Fig. 1, more than half of PLC defenses in the literature did not report any evaluation metrics and the ones that do report either overhead, e.g., CPU overhead, or effectiveness metrics, e.g., false positives. Only 6 or 8.5% of all PLC defenses in the literature reported both. Worse, as far as we know, there is *only one* PLC defense that reported security evaluation metrics, namely, D-Box [41].

In addition to the metrics discussed above, there are two works that introduced metrics relevant to PLCs that informed this research.

BenchIoT Metrics. BenchIoT is an open-source benchmarking tool designed for IoT security defenses. BenchIoT introduced a set of evaluation metrics designed specifically for Internet of Things (IoT) devices. These metrics include security, performance, and memory and energy metrics [6]. IoT devices share many characteristics with PLCs. For example, they use RTOS to meet strict timing requirements, and they use processors similar to ARM's Cortex-M series. However, there important differences, IoT devices, as their name implies, rely heavily on network communication while PLCs do not necessarily need network communication to accomplish their task. Another important difference is that IoT devices do not use control logic, which is an essential characteristic of PLCs. These differences highlight the importance of designing evaluation metrics specific to PLCs as not all metrics proposed in BenchIoT are directly transferable to PLCs.

Metrics by Lopez-Morales et al. In their PLC-focused Systematization of Knowledge (SoK) work, Lopez-Morales et al. [39] introduced some evaluation metrics to compare PLC security defenses. These metrics include PLC overhead and defense effectiveness, among others. However, these metrics are limited as they are not specific enough to make a quantitative comparison between PLC defenses. For example, the PLC overhead metric categorizes each defense with either zero, negligible, or considerable overhead. However, these metrics do not consider anything specific about CPU, scan cycle, or memory overhead. They also introduced the effectiveness category which includes metrics such as accuracy and false positives. However, such a metric is not reported by any of the defenses that were explored as a part of their SoK work.

2.5 Profiling and Benchmarking

In order to obtain any evaluation metrics there are two main methods: profiling and benchmarking.

Profiling provides measurements for the performance of software applications. Profiling provides fine-grained information for the components of an application, such as how often a function is called, how long a routine takes to execute, and how much time is spent on different parts of the code. With this information, we can identify performance bottlenecks and poorly implemented code that can later be improved [18]. Profiling can be implemented via *instrumentation*. Instrumentation refers to inserting special code at the beginning and end of a routine to record when the routine starts and ends. The profiling result shows the actual time taken by the routine on each call. If the application's source code is available,

profiling is accomplished by inserting the instrumentation code in the source code itself. However, if the source code is unavailable, the instrumentation code is inserted into the application's executable code once it is in memory [2, 18].

Benchmarking is a critical aspect of evaluating computer hardware and software. Benchmarking is a process that compares the performance of various alternative tools and technologies through a standard test (or series of standard tests) [54]. A benchmark comprises three components: a motivating comparison, a task sample, and performance measures [54]. The motivating comparison defines both the reason for the benchmark and the comparison to be used to judge the technologies' performances. The task sample represents the sampling of tasks the subject is expected to solve. Lastly, the performance measures lay out the measurements needed to record the subject's fitness to perform the specified tasks. Often, when a community develops a benchmark for their domain, it encourages the maturation of that domain because a benchmark represents an operationalization of the domain's scientific paradigms [54].

3 Design Objectives

To effectively address the research questions described in Sec. 1, we designed our evaluation metrics with the following objectives in mind.

3.1 Applicable to Multiple PLC Architectures

Our metrics must be compatible with the multiple architectures that PLCs have, as discussed in Sec. 2.2 so that they can be used to compare the widest array of PLC defenses possible. Specifically, we designed our metrics to work for both Hard and SoftPLCs.

3.2 Applicable to All PLC Security Defenses

Our performance metrics must be as general as possible so that we can compare as many defenses as possible. As we discussed in Sec. 2.3, PLC defenses include a wide range of methods that include IDS and CFI. Designing our metrics to be specific to either one of these would not allow for a direct comparison. The current literature has no basis for this type of comparison between PLC defenses, so we want to create that basis.

3.3 Simple and Straightforward

The evaluation metrics that we propose are the ones we believe are fundamental for evaluation and comparison with other PLC defenses. We do not aim to provide an exhaustive set of evaluation metrics for two main reasons. First, if we introduce very specific metrics, this will make our metrics non-general as some PLCs or PLC defenses will not qualify for some specific metrics. The second reason is that introducing too many evaluation metrics creates an additional burden for researchers to conduct several new evaluation experiments. This is not always feasible as researchers might have time or bandwidth constraints. If the metrics are too burdensome to obtain, researchers might just skip them, defeating the purpose of this work.

3.4 Focused on Defenses

Finally, our metrics must focus on the evaluation of *defensive* PLC security methods. An example of a PLC security defense would be *Ghostbuster* [1]. *Ghostbuster* is a countermeasure designed to detect PIN control attacks [1]. In this case, our evaluation metrics would apply to *Ghostbuster* but not PIN control attacks. Additionally, our evaluation metrics are not designed to evaluate the security of control logic programs. Although control logic programs are sometimes part of a defense and can be included in the evaluation process, our proposed metrics are designed for the defense methods, not the control logic. The evaluation metrics proposed in this work are meant for all programmable logic controllers, including HardPLCs and SoftPLCs. Finally, our evaluation metrics are not designed for IEDs, RTUs, or other CPS devices.

4 Evaluation Metrics

Based on the above research questions and design considerations, we now propose three types of evaluation metrics for PLC defenses: *security* metrics, *overhead* metrics, and *effectiveness* metrics.

4.1 Security Metrics

Security metrics quantify if the defense follows the least-privilege principle. They quantify how the defense method affects the security of the system.

(1) **Number of ROP Gadgets**. Return-Oriented Programming (ROP) is an exploit technique that allows an attacker to gain control of the call stack to hijack the program control flow and execute instruction sequences that are already in memory; these instructions are called *ROP gadgets*. [16]. PLCs are also vulnerable to this type of attack [13]; however, not all control logic compilers generate new ROP gadgets. As we discussed in Sec. 2.2, control logic code can be compiled into machine code or translated via just-in-time compilation. Only control logic compiled into machine code might introduce new ROP gadgets. An example of this metric being used in practice is D-Box [41]. This approach was designed to provide secure DMA operations to compartmentalize MCU-based devices. D-Box used the ROP gadget metric to compare Its performance to FreeRTOS-MPU (F-MPU). D-Box had exposed 13 ROP gadgets in its standard configuration, and F-MPU had exposed 520 ROP gadgets in its standard configuration [41].

(2) **Memory Region Ratio**. This metric measures the efficacy of memory isolation by computing the size ratio of the maximum available code region to an adversary with respect to the total code size of the application binary. A lower value is better [6]. To calculate the Memory Region Ratio (MRR) we can follow Equation 1. The maximum available code region size can be obtained by analyzing the PLC defense binary and identifying all the executable code regions. The application binary code size can be obtained by parsing the application itself. Once again, D-Box is an example of the memory region ration metric in practice [41]. The authors evaluated D-box using this metric and demonstrated that D-Box reduced the memory region ratio in various sub-regions, i.e., user space flash.

$$MRR = \frac{MaximumAvailableCodeRegionSize}{ApplicationBinaryCodeSize} \quad (1)$$

(3) **Number of Privileged Cycles**. Privileged cycles may occur during user threads with elevated privileges or I/O operations.

Reducing the number of privileged cycles improves the security of the PLC; thus, PLC defense methods should avoid introducing additional privilege cycles [6]. To our knowledge, no PLC defenses have been evaluated using this metric. This would be a new metric within the PLC security domain.

4.2 Overhead Metrics

Overhead metrics quantify the additional or indirect resources the defense requires to function.

(1) **Scan Cycle Time**. This metric measures the time the PLC takes to complete its scan cycle while running the defense. The scan cycle time should be reported in microseconds (μs). An example of this metric in use can be observed in the evaluation of C^2 [40]. The scan cycle time metric was to determine the performance cost increase that would incur on a PLC from C^2 mediating access to physical assets, e.g., assembly line. This portion of the evaluation was completed on a Raspberry Pi with a 700 MHz processor, and the evaluation demonstrated an increase of between 0.1 to 0.2 ms of scan cycle time [40].
(2) **Total Runtime Cycles**. The total runtime, or wall time, measures the total runtime of running the PLC defense application. The total runtime may include active CPU cycles, waiting for I/O, waiting for the network, etc. The total runtime cycles should be reported in milliseconds (ms). An example of this metric in practice occurred with ICSPatch [47]. During the evaluation of ICSPatch, 24 vulnerable binaries were analyzed to identify the vulnerability and patch said vulnerability. The runtime cycles (milliseconds) for each process phase, in which vulnerability localization, patch generation, and patch verification were recorded and reported.
(3) **Total CPU Cycles**. This metric measures only the time the CPU is actively working on the PLC defense application. Unlike the total runtime cycles metric, this metric does not include other factors, such as waiting for I/O or the network. The total CPU cycles should be reported in milliseconds (ms). A related example of this metric in practice comes from ECFI [2]. The authors of this work reported the CPU cycles used by their tool in terms of worst-case and average-case scenarios.
(4) **Total RAM Usage**. This metric measures the memory (RAM) the defense application utilizes during its execution. This includes all the memory the application allocates for its various components and processes, such as code, data, stack, heap, and any dynamically allocated memory. For example, Linux memory management refers to this metric as resident set size (RSS) [61]. The total RAM usage should be reported in kilobytes (KiB). A related example to this metric in use comes from ICSPatch [47]. The authors only recorded the additional memory (Bytes) required for the generated patches. They did not record the total amount of memory used during the execution of the entire patching process. To abide by this proposed metric, future research would need to record the entire memory used by the proposed defense and not just from its generated output.

4.3 Effectiveness Metrics

Effectiveness metrics quantify how well the defense performs its task, e.g., anomaly detection. The effectiveness metrics we propose are straightforward and are based on the confusion matrix used in the field of Machine Learning [36].

(1) **True Positive**. This is the occurrence of a positive sample that has been correctly identified as being positive.
(2) **True Negative**. This is the occurrence of a negative sample that has been correctly identified as being negative.
(3) **False Positive**. This is the occurrence of a negative sample incorrectly identified as a positive sample. This is commonly referred to as being a Type 1 error.
(4) **False Negative**. This is the occurrence of a positive sample that has been incorrectly identified as being a negative sample. This is commonly referred to as being a Type 2 error.
(5) **Accuracy**. This is the total number of correctly identified positive and negative samples averaged over the total number of samples.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

These metrics are commonly used for classification tasks, e.g., anomaly detection and malware classification. For example, Salehi and Bayat-Sarmadi proposed PLCDefender that combined remote attestation with a physics-based simulation to determine if the target PLC was operating correctly [50]. Part of their method was detecting if the PLC's behavior was operating maliciously or within operating norms. PLCDefender used all five of the above metrics to measure the effectiveness of their solution. If any PLC defense is designed to detect and identify an attack, then that PLC defense's effectiveness can be measured using these metrics.

However, if a PLC defense is not designed to incorporate classification abilities, these metrics may not be applicable. Further, metrics needed to be proposed that can measure the effectiveness of non-classification-based PLC defenses such as AttkFinder [17].

4.4 Case Study: Security Metrics for ICSPatch

ICSPatch is a PLC security defense that automatically locates control logic vulnerabilities and non-intrusively hot-patches vulnerabilities in the control application directly in the main memory of the PLC without stopping its operation [47]. This case study explains how our security metrics can be applied to ICSPatch. We selected ICSPatch because it is one of the most recent PLC defenses.

Security Metrics. ICSPatch does not report security metrics, allowing us to explore how they could be obtained.

Number of ROP Gadgets: ICSPatch relies on a local patch server running on the deployed PLC. The local patch server is written in C code, and the compiled binary file could introduce exploitable ROP gadgets, which can be quantified using our proposed metric.

Memory Region Ratio: ICSPatch increases the binary memory utilization by introducing the local patch server. This may add to the memory available code region for adversaries, increasing the attack surface of the PLC. To quantify this increase, the local patch server binary code size must be measured to calculate the MMR described in Equation 1.

Number of Privileged Cycles: ICSPatch reports the number of CPU cycles. However, there is no distinction between privileged and userspace cycles. To obtain our proposed metric, the number of privileged cycles, the authors of ICSPatch would have to modify

Table 2: Summary of processor cores used by the top 10 most studied PLCs in security literature.

Manufacturer	PLC Model	Processor Core
Siemens	S7-300	Unknown
Siemens	S-1200	ARM Cortex-R4 [3]
Schneider Electric	Modicon M221	Unknown
Siemens	S7-1500	Unknown
Rockwell	MicroLogix 1400	Unknown
Rockwell	ControlLogix 5571	Unknown
Rockwell	MicroLogix 1100	Unknown
Wago	PFC200	ARM Cortex A8 [60]
Wago	750-881	Unknown

their benchmarking tool to report privileged cycles only in addition to the total CPU cycles.

5 Challenges Obtaining Evaluation Metrics

In this section, we describe some of the most critical challenges researchers encounter when measuring evaluation metrics.

5.1 Lack of Standard Benchmarking and Profiling Tools

As we discussed in Sec. 2.5, in order to obtain metrics, we need to use profiling or benchmarking. However, these tools are sometimes not readily available to perform measurements, and even if some of them are available, they might not be compatible with all types of PLC. For example, ARM's PMU is exclusively available for ARM processors as depicted in Table 1. This means other architectures, such as x86, will not have access to this benchmarking feature. Another consequence of the lack of standard benchmarking and profiling tools is that if there is no tool to measure a specific metric for a specific PLC, then researchers have to implement their own profilers via instrumentation and maybe even their own benchmarks. Finally, even if there are some tools available, researchers might not be aware of them. Table 1 includes the tool used for metric measures for several PLC defenses. As we can see, only three papers reported what tool was used to obtain their reported overhead values.

5.2 Proprietary PLC Hardware and Software

As we discussed in Sec. 2.1 and 2.2, some PLCs have proprietary hardware and software that makes metric measurement difficult. For example, we do not know the CPU core for many PLCs. Table 2 depicts the top 10 most common PLCs in the security literature; however, we only know the type of processor of two of them. This lack of information hinders our ability to connect metrics because we do not know the architecture of the processor or the OS. Thus, we do not know what type of profiling or benchmarking tools might be compatible with a particular PLC. This problem is not limited to HardPLCs. CODESYS, one of the main SoftPLC platforms, has a proprietary runtime environment that can make profiling difficult. Specifically, researchers might need to reverse-engineer parts of the CODESYS runtime to insert instrumentation code to implement a profiler, as discussed in Sec. 2.5.

5.3 Different Environment Conditions During Metric Measurement

When we measure any metric via profiling or benchmarking, the resulting measuring is affected by multiple secondary environment conditions and configurations. These conditions include but are not limited to network latency, CPU sleep cycles, multi-threading, and hardware security features such as Address Space Layout Randomization (ASLR) [2, 4]. These conditions directly affect metric measurements by introducing delays and overhead. For example, if we consider the total runtime cycles metric (Sec. 4, the total runtime might be increased by CPU sleep cycles or the overhead introduced by ASLR. However, if ASLR is unavailable or disabled, then the total runtime cycles metric would be different. The end result is that different measurements of the same metric can result in significantly different measurements if the underlying environmental conditions are different.

6 Recommendations

In this section, we provide a set of recommendations aimed at addressing the challenges discussed in Sec. 5.

6.1 Use a Benchmark Control Logic Algorithm

Using a single control logic program or algorithm as a benchmark to test the performance metrics would enable the metrics to remain comparable across multiple defenses. For example, *Abbasi et al.* [2] introduced the control logic depicted in Algorithm 1 that uses all the PLC analog and digital I/O interfaces and performs basic arithmetic operations.

This concept is similar to the use of The Tennessee Eastman Process (TEP) [23]. The TEP is a real industrial process that was modeled in 1993. The TEP has been adopted as a benchmark to test security attack and defense methods [35]. However, the TEP cannot be used as a benchmark for PLC defenses because it does not specify a control logic algorithm.

6.2 Leverage Existing Benchmarking and Profiling Tools

Researchers who want to obtain our proposed metrics should not reinvent the wheel, instead they should make use of existing profiling and benchmarking tools whenever possible. The following are some of the available tools for both Hard and SoftPLCs.

SIMATIC Controller Profiling. The SIMATIC Controller Profiling is an analysis tool that analyzes and evaluates the runtime behavior of the control logic program on a SIMATIC controller which is displayed on a web interface [53]. The SIMACTIC Controller Profiling tool is available for the SIMATIC S7-1500 only. This profiling tool provides the wall time of "OB1 block" which provides the PLC scan cycle metric [52].

CODESYS Profiler. The CODESYS profiler is a tool that enables the detailed measurement of runtime behavior and code coverage of the control logic program [19]. It provides multiple data and measurements, for example, it provides the minimum and minimum

processing time over multiple cycles, which can be used to obtain the scan cycle metric [64]. The CODESYS profiler tool is available for any CODESYS-compatible platform.

OpenPLC's Cycle Time Logs. The OpenPLC runtime logs the real time cycle time (scan cycle time) and latency, the maximum, minimum, and average values in microseconds are printed out in the summary logs when the control logic program stops. These values are also available in the OpenPLC web interface [8].

ARM's Performance Monitor Unit (PMU). ARM processors provide a Performance Monitoring Unit (PMU) as part of their architecture to gather statistics on the operation of the processor and memory system [10]. The PMU can track events on the core via counters such as the Cycle Count Register counter which is one of our overhead metrics (Sec. 4).

ROPgadget Tool. The ROPgadget tool analyzes application binaries and searches for ROP gadgets. It supports multiple architectures, including x86 and ARM [51], which some PLCs support as shown in Table 2.

UNIX Tools. UNIX-based operating systems such as Linux and RTOS provide profiling and benchmarking tools that can be leveraged to obtain PLC defense metrics. These tools are relevant since many PLCs implement UNIX-based operating systems [3]. These tools include the time command [38], the *readelf* command [33], the *perf* command [32] and the *top* command [34].

The *time* command measures an application's total time, which, in this case, is a PLC defense [38]. It provides values such as wall clock time, the same as our *total runtime cycles* metric (Sec. 4).

The *readelf* command [33] allows us to analyze an application binary to identify the executable memory regions. This command is useful when obtaining the *memory region ratio* metric (Sec. 4).

The *perf* command [32] is a versatile profiling and benchmarking tool that provides performance CPU counters and more. These counters allow us to obtain the *total CPU cycles* metric (Sec. 4).

The *top* command [34] is a common Linux utility that monitors the system activity, including the amount of physical RAM a particular process uses, measured in kilobytes. This is relevant to the *total RAM usage* metric (Sec. 4).

6.3 Normalize Environment Configuration

In order to tackle the different environment conditions challenge discussed in Sec. 5, researchers should understand the relevant environment configurations and normalize them before collecting metrics. Specifically, multiple evaluation metrics should be measured under the same environment configuration whenever possible. This is particularly important if multiple PLCs are being evaluated. For example, CODESYS allows users to use one or multiple cores to run control logic [20]. Additionally, researchers should document and share the environmental conditions and configuration when metrics were collected. Sharing these data will allow other researchers to normalize their own environmental conditions so that their metrics are compared optimally. It is worth pointing out that, without proper research artifacts, it is not feasible to reproduce the same exact environment when trying to replicate the metrics of a previous PLC defense.

6.4 Use Worst-Case Execution Time (WCET) for Measuring Overhead Metrics

Worst-Case Execution Time (WCET) is the maximum time a task can take to execute on a specific hardware platform. WCET should be used to verify real-time systems, such as PLCs, where a missed deadline is unacceptable, as this might result in irreparable damage to physical assets and even humans. WCET estimates can be used to verify that the response time of a critical piece of code is short enough, that interrupt handlers finish quickly enough, or that the sample rate of a control loop can be kept [24, 46]. When measuring the metrics we proposed in Sec. 4, researchers should obtain and report WCET for the aforementioned reasons and should not rely on average performance values as these average values are not realistic in the PLC setting where strict real-time performance is necessary [2, 4].

7 Conclusion

In this paper, we propose a set of evaluation metrics for PLC security defenses. Our metrics include security, overhead, and effectiveness metrics specifically tailored to PLCs. We also pointed out potential challenges that researchers might encounter when measuring our proposed metrics and we provide recommendations on how to overcome these challenges. We hope that this work will encourage researchers to report evaluation metrics when publishing new PLC security defenses in addition to providing research artifacts, allowing for PLC research to move towards security by design and not in an ad-hoc manner.

In future work, we plan to leverage the evaluation metrics we have proposed to develop a PLC defense benchmark framework that can automate the whole performance evaluation process. To this end, additional factors need to be considered. For example, how can the evaluation metrics be measured and automated? How many benchmark applications will such a framework include, and what will it include? What are the supported PLC models and processors? Correctly answering these questions will be crucial for evaluating the effectiveness of security defenses for PLCs in the future.

Acknowledgments

This work was partially supported by the US Department of Transportation (USDOT) Tier-1 University Transportation Center (UTC) Transportation Cybersecurity Center for Advanced Research and Education (CYBER-CARE). (Grant No. 69A3552348332), and by the Scholar Achievement in Graduate Education (SAGE) Fellowship from Texas A&M University-Corpus Christi.

References

[1] Ali Abbasi and Andrea Genuise. 2017. Ghost in the PLC vs GhostBuster: on the feasibility of detecting pin control attack in Programmable Logic Controllers. In *Ghost in the PLC vs GhostBuster*. Eindhoven University of Technology.

[2] Ali Abbasi, Thorsten Holz, Emmanuele Zambon, and Sandro Etalle. 2017. ECFI: Asynchronous Control Flow Integrity for Programmable Logic Controllers. In *Proceedings of the 33rd Annual Computer Security Applications Conference* (Orlando, FL, USA) *(ACSAC '17)*. Association for Computing Machinery, New York, NY, USA, 437–448. https://doi.org/10.1145/3134600.3134618

[3] Ali Abbasi, Tobias Scharnowski, and Thorsten Holz. 2019. Doors of durin: The veiled gate to siemens S7 silicon. *BlackHat Europe* (2019).

[4] Ali Abbasi, Jos Wetzels, Thorsten Holz, and Sandro Etalle. 2019. Challenges in designing exploit mitigations for deeply embedded systems. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 31–46.

[5] Chuadhry Mujeeb Ahmed, Martin Ochoa, Jianying Zhou, and Aditya Mathur. 2021. Scanning the cycle: Timing-based authentication on PLCs. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 886–900.

[6] Naif Saleh Almakhdhub, Abraham A Clements, Mathias Payer, and Saurabh Bagchi. 2019. Benchiot: A security benchmark for the internet of things. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 234–246.

[7] Thiago Alves. 2022. OpenPLC Runtime version 3. https://github.com/thiagoralves/OpenPLC_v3 Accessed: 27-09-2023.

[8] Thiago Alves. 2023. webserver/core/main.cpp: Get REAL-TIME time values. https://github.com/thiagoralves/OpenPLC_v3/pull/201

[9] arm. 2024. Cortex-R5. https://www.arm.com/products/silicon-ip-cpu/cortex-r/cortex-r5

[10] ARM Developer. 2024. About the PMU. https://developer.arm.com/documentation/ddi0488/h/performance-monitor-unit/about-the-pmu

[11] ARM Developer. 2024. DWT functional description. https://developer.arm.com/documentation/ddi0337/h/data-watchpoint-and-trace-unit/dwt-functional-description

[12] John Aycock. 2003. A brief history of just-in-time. *ACM Computing Surveys (CSUR)* 35, 2 (2003), 97–113.

[13] 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)*. IEEE, 215–226.

[14] Zachry Basnight, Jonathan Butts, Juan Lopez Jr, and Thomas Dube. 2013. Firmware modification attacks on programmable logic controllers. *International Journal of Critical Infrastructure Protection* 6, 2 (2013), 76–84.

[15] William Bolton. 2015. *Programmable logic controllers*. Newnes.

[16] Erik Buchanan, Ryan Roemer, Stefan Savage, and Hovav Shacham. 2008. Return-oriented programming: Exploitation without code injection. *Black Hat* 8 (2008).

[17] John H. Castellanos, Martin Ochoa, Alvaro A. Cardenas, Owen Arden, and Jianying ZHOU. 2021. AttkFinder: Discovering Attack Vectors in PLC Programs using Information Flow Analysis. In *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses* (San Sebastian, Spain) *(RAID '21)*. Association for Computing Machinery, New York, NY, USA, 235–250. https://doi.org/10.1145/3471621.3471864

[18] Euccas Chen. 2017. CPU Profiling Tools on Linux. https://euccas.github.io/blog/20170827/cpu-profiling-tools-on-linux.html

[19] CODESYS GmbH. 2022. CODESYS Profiler | CODESYS Store International. https://store.codesys.com/en/codesys-profiler.html

[20] CODESYS GmbH. 2024. Multicore. https://help.codesys.com/api-content/2/codesys/3.5.13.0/en/_cds_multi_core/

[21] CODESYS Runtime [n. d.]. CODESYS Runtime. https://www.codesys.com/products/codesys-runtime.html. Accessed: 09-08-2024.

[22] Arnaldo Carvalho De Melo. 2010. The new linux'perf'tools. In *Slides from Linux Kongress*, Vol. 18. 1–42.

[23] James J Downs and Ernest F Vogel. 1993. A plant-wide industrial process control problem. *Computers & chemical engineering* 17, 3 (1993), 245–255.

[24] Jakob Engblom, Andreas Ermedahl, Mikael Sjödin, Jan Gustafsson, and Hans Hansson. 2003. Worst-case execution-time analysis for embedded real-time systems. *International Journal on Software Tools for Technology Transfer* 4 (2003), 437–455.

[25] Nicolas Falliere, Liam O Murchu, Eric Chien, et al. 2011. W32. stuxnet dossier. *White paper, symantec corp., security response* 5, 6 (2011), 29.

[26] David Formby and Raheem Beyah. 2019. Temporal execution behavior for host anomaly detection in programmable logic controllers. *IEEE Transactions on Information Forensics and Security* 15 (2019), 1455–1469.

[27] Dina Hadžiosmanović, Robin Sommer, Emmanuele Zambon, and Pieter H Hartel. 2014. Through the eye of the PLC: semantic security monitoring for industrial processes. In *Proceedings of the 30th Annual Computer Security Applications Conference*. 126–135.

[28] Yi Han, Sriharsha Etigowni, Hua Liu, Saman Zonouz, and Athina Petropulu. 2017. Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1095–1108.

[29] Instrumentation Tools. 2024. Understanding the Scan Cycle of SIEMENS PLC. https://instrumentationtools.com/understanding-the-scan-cycle-of-siemens-plc/#what-is-meant-by-a-scan-cycle

[30] Chenglu Jin, Saeed Valizadeh, and Marten van Dijk. 2018. Snapshotter: Lightweight intrusion detection and prevention system for industrial control systems. In 2018 IEEE Industrial Cyber-Physical Systems (ICPS). *IEEE* 45 (2018), 824–829.

[31] Anastasis Keliris and Michail Maniatakos. 2018. ICSREF: A framework for automated reverse engineering of industrial control systems binaries. *arXiv preprint arXiv:1812.03478* (2018).

[32] Michael Kerrisk. 2024. perf(1) — Linux manual page. https://man7.org/linux/man-pages/man1/perf.1.html

[33] Michael Kerrisk. 2024. readelf(1) — Linux manual page. https://man7.org/linux/man-pages/man1/readelf.1.html

[34] Michael Kerrisk. 2024. top(1) — Linux manual page. https://man7.org/linux/man-pages/man1/top.1.html

[35] Marina Krotofil and Alvaro A Cárdenas. 2013. Resilience of process control systems to cyber-physical attacks. In *Secure IT Systems: 18th Nordic Conference, NordSec 2013, Ilulissat, Greenland, October 18-21, 2013, Proceedings 18*. Springer, 166–182.

[36] Joffrey L Leevy, John Hancock, Richard Zuech, and Taghi M Khoshgoftaar. 2021. Detecting cybersecurity attacks across different network features and learners. *Journal of Big Data* 8 (2021), 1–29.

[37] Efrén López-Morales, Carlos Rubio-Medrano, Adam Doupé, Yan Shoshitaishvili, Ruoyu Wang, Tiffany Bao, and Gail-Joon Ahn. 2020. HoneyPLC: A Next-Generation Honeypot for Industrial Control Systems. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, Virtual Event, USA, 279–291. https://doi.org/10.1145/3372297.3423356

[38] Adam L. Lyon. 2017. Time profiling. https://lyon-fnal.github.io/profiling-book/time-command.html

[39] Efrén López-Morales, Ulysse Planta, Carlos Rubio-Medrano, Ali Abbasi, and Alvaro A. Cardenas. 2024. SoK: Security of Programmable Logic Controllers. arXiv:2403.00280 [cs.CR] https://arxiv.org/abs/2403.00280

[40] Stephen McLaughlin. 2013. CPS: Stateful policy enforcement for control system device usage. In *Proceedings of the 29th Annual Computer Security Applications Conference*. 109–118.

[41] Alejandro Mera, Yi Hui Chen, Ruimin Sun, Engin Kirda, and Long Lu. 2022. D-box: DMA-enabled compartmentalization for embedded applications. *arXiv preprint arXiv:2201.05199* (2022).

[42] Open Source PLC Software. 2024. 1.3 Installing OpenPLC Runtime on Windows. https://autonomylogic.com/docs/installing-openplc-runtime-on-windows/

[43] OpenPLC Overview – OpenPLC [n. d.]. 1.1 OpenPLC Overview – OpenPLC. https://openplcproject.com/docs/openplc-overview/. Accessed: 09-08-2024.

[44] Hammond Pearce, Srinivas Pinisetty, Partha S Roop, Matthew MY Kuo, and Abhisek Ukil. 2019. Smart I/O modules for mitigating cyber-physical attacks on industrial control systems. *IEEE Transactions on Industrial Informatics* 16, 7 (2019), 4659–4669.

[45] R PICKREN, T SHEKARI, S ZONOUZ, and R BEYAH. 2024. Compromising industrial processes using web-based programmable logic controller malware. In *Network and Distributed System Security Symposium (NDSS)*.

[46] Peter Puschner and Ch Koza. 1989. Calculating the maximum execution time of real-time programs. *Real-time systems* 1, 2 (1989), 159–176.

[47] Prashant Hari Narayan Rajput, Constantine Doumanidis, and Michail Maniatakos. 2023. {ICSPatch}: Automated Vulnerability Localization and {Non-Intrusive} Hotpatching in Industrial Control Systems using Data Dependence Graphs. In *32nd USENIX Security Symposium (USENIX Security 23)*. 6861–6876.

[48] Luis Salazar, Sebastián R. Castro, Juan Lozano, Keerthi Koneru, Emmanuele Zambon, Bing Huang, Ross Baldick, Marina Krotofil, Alonso Rojas, and Alvaro A. Cardenas. 2024. A Tale of Two Industroyers: It was the Season of Darkness. In *2024 IEEE Symposium on Security and Privacy (SP)*. 312–330. https://doi.org/10.1109/SP54263.2024.00162

[49] Luis Salazar, Efren López-Morales, Juan Lozano, Carlos Rubio-Medrano, and Alvaro A Cardenas. 2024. ICSNet: A Hybrid-Interaction Honeynet for Industrial Control Systems. In *Proceedings of the 6th Workshop on CPS&IoT Security and Privacy* (Salt Lake City, UT, USA) *(CPSIoTSec '24)*. Association for Computing Machinery, New York, NY, USA.

[50] Mohsen Salehi and Siavash Bayat-Sarmadi. 2021. PLCDefender: Improving Remote Attestation Techniques for PLCs Using Physical Model. *IEEE Internet of Things Journal* 8, 9 (2021), 7372–7379. https://doi.org/10.1109/JIOT.2020.3040237

[51] Jonathan Salwan. 2023. ROPgadget Tool. hhttps://github.com/JonathanSalwan/ROPgadget

[52] Siemens. 2021. OB1 Scan Cycle Time. https://support.industry.siemens.com/forum/WW/en/posts/ob1-scan-cycle-time/253590

[53] Siemens. 2024. SIMATIC Controller Profiling. https://cache.industry.siemens.com/dl/files/245/109750245/att_1165728/v2/109750245_S71500ProfilingTool_DOC_V10_en.pdf

[54] S.E. Sim, S. Easterbrook, and R.C. Holt. 2003. Using benchmarking to advance research: a challenge to software engineering. In *25th International Conference on Software Engineering, 2003. Proceedings.* 74–83. https://doi.org/10.1109/ICSE.2003.1201189

[55] John A Stankovic and Raj Rajkumar. 2004. Real-time operating systems. *Real-Time Systems* 28, 2-3 (2004), 237–253.

[56] Ruimin Sun, Alejandro Mera, Long Lu, and David Choffnes. 2021. SoK: Attacks on Industrial Control Logic and Formal Verification-Based Defenses. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 385–402.

[57] The Memory Protection Unit [n. d.]. The Memory Protection Unit. https://developer.arm.com/documentation/den0042/a/The-Memory-Protection-Unit. Accessed: 09-08-2024.

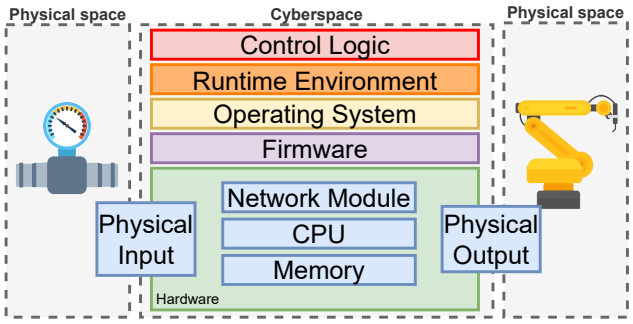

Figure 2: Generic PLC Architecture. Based on [15, 26, 39].

Algorithm 1 Example control logic algorithm that performs temperature Sensor Readings to servo motor PWM Control. Based on [2].

Input: Read In.25 (Temperature Sensor Readings)
Output: Write Out.22 (ServoMotor PWM)
$t \leftarrow 0$
while True **do**
    read input
    if input > 100 **then**
        while input > 100 **do**
            $A, B, C \leftarrow$ Random Int ▷ Set points
            $D \leftarrow A + B + C$
            Update Pulse Width Modulation I/O
            PWM.IO(22) $\leftarrow 1.5 + 0.5 \cdot \sin(t)$
            $t \leftarrow t + D$
        end while
    else if input < 100 **then**
        $A \leftarrow 0.1, B \leftarrow 0.01, C \leftarrow 0.001$ ▷ Set points
        $D \leftarrow A - B - C$
        Update Pulse Width Modulation I/O
        PWM.IO(22) $\leftarrow 0.7 + 0.2 \cdot \sin(t)$
        $t \leftarrow t + D$
    end if
end while

[58] Michael Tiegelkamp and Karl-Heinz John. 2010. *IEC 61131-3: Programming industrial automation systems.* Springer.
[59] Vulnerability Disclosure Policy Template | CISA [n. d.]. Vulnerability Disclosure Policy Template | CISA. https://www.cisa.gov/vulnerability-disclosure-policy-template. Accessed: 09-08-2024.
[60] WAGO. 2024. Controller PFC200 (750-8217/600-000) | WAGO USA. https://www.wago.com/us/controllers-bus-couplers-i-o/controller-pfc200/p/750-8217_600-000
[61] WikiChip. 2020. Resident Set Size (RSS). https://en.wikichip.org/wiki/resident_set_size#:~:text=The%20resident%20set%20size%20(RSS,is%20the%20virtual%20set%20size.
[62] Hyunguk Yoo, Sushma Kalle, Jared Smith, and Irfan Ahmed. 2019. Overshadow plc to detect remote control-logic injection attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 16th International Conference, DIMVA 2019, Gothenburg, Sweden, June 19–20, 2019, Proceedings 16.* Springer, 109–132.
[63] Jonas Zaddach and Andrei Costin. 2013. Embedded devices security and firmware reverse engineering. *Black Hat USA* (2013). https://media.blackhat.com/us-13/US-13-Zaddach-Workshop-on-Embedded-Devices-Security-and-Firmware-Reverse-Engineering-Slides.pdf. Accessed: 03-10-2023.
[64] Gong Zhou. 2023. CODESYS Tutorial: Getting the Actual Cycle Time of a Current Task. https://medium.com/@sean.gongz/how-to-get-current-task-actual-cycle-time-in-codesys-267384bcd3b7

Appendix

Control Logic Algorithm

Algorithm 1 performs temperature sensor readings. This algorithm is given as an example to establish a baseline or benchmark to evaluate the evaluation metrics proposed in this paper.

PLC Architecture

The generic architecture of a PLC is depicted in Fig. 2.