An Automated Reverse Engineering Cyber Module for 5G/B5G/6G

ML-Facilitated Pre-"ret" Discernment Module for Industrial Process Programmable Logic Controllers

Steve Chan
Decision Engineering Analysis Laboratory, VT
San Diego, USA
e-mail: schan@dengineering.org

***Abstract*—Industrial Control System (ICS) components have been subject to heightened cyber risk as hardware/software supply chain vulnerabilities have been illuminated and cyberattacks have become increasingly sophisticated. At the center of this ICS cyber maelstrom is the Programmable Logic Controller (PLC), a key component of Industry 4.0, as it is a main controller for physical processes (e.g., the control of an actuator). Many PLCs were designed for another era; they are resource-constrained, non-optimized, and beset with a variety of legacy facets (e.g., compiler, programming language, etc). This described sub-optimal paradigm also exists within the rubric of standards that specify the time interval between signal ingestion and actuation (e.g., IEEE 1547 specifies 2 seconds) for the operating environment. Hence, the designing/architecting/implementing of a light computational footprint continuous Monitoring/Detecting/Mitigating Module (MDMM) is non-trivial. This paper investigates a specific scenario of an ICS PLC operating within a 5G Ultra-Reliable Low-Latency Communications (URLLC) inter-PLC context and posits a viable MDMM construct that can operate within the paradigm. Central to its viability, the MDMM leverages apriori scan cycle traffic, utilizes Machine Learning (ML)-facilitated PLC logic/code optimization, and endeavors to undertake mitigation via a bespoke Automated Reverse Engineering (ARE) mechanism. The introduced MDMM requires further quantitative benchmarking, but the initial experimental results show promise.**

Keywords-cybersecurity; industrial control system; programmable logic controller; Industry 4.0; Industrial Internet of Things; smart manufacturing; smart grid; 5G; machine learning; artificial intelligence; automated reverse engineering.

I. INTRODUCTION

The benefits of ARE for PLC binaries to reduce the investigation time needed by those in the specialized cybersecurity functional sub-field of Digital Forensics and Incident Response (DFIR) is well documented in the literature [1][2]. Time is of the essence for these DFIR teams, as their task is to quickly comprehend the involved attack vector objective(s) (e.g., PLC exploitation) and effectuate countermeasures post-exploitation analysis. The need to reduce the time needed for exploitation analysis was illuminated by, among other examples, the ICS Stuxnet case study (wherein the PLCs at the involved nuclear facility were targeted). The prolonged non-automated, manual labor-intensive paradigm of that particular reverse engineering process greatly delayed the forensic investigation and articulated the need for an ARE mechanism as well as the need of a digital mirror for supporting such a mechanism.

Yet, ARE, if not properly architected, can also constitute a vulnerability, if it is somehow exploited by attackers. Just as the advisories made available by the National Vulnerability Database (NVD) and Sentient Hyper Optimized Data Access Network (SHODAN) can be used by cyber defenders as early warning indicators, they can also be leveraged by cyber attackers for exploitation opportunities and as attack accelerants [3]. This phenomenon should be of no surprise, as historically, malicious entities have engaged in reverse engineering on two fronts: Hardware Reverse Engineering (HRE) and Software Reverse Engineering (SRE). HRE has long been used by attackers to discern the inner workings of Integrated Circuits (ICs) [4]; indeed, tools, such as HAL – The Hardware Analyzer, have facilitated HRE [5]. On the SRE side, tools include IDA Pro, Radare2, Ghidra (open-sourced by the National Security Agency or NSA), Hopper, and others. When the aforementioned HRE/SRE tools, among others, are utilized as attack accelerants, defending security teams have witnessed the might of reverse engineering attacks, and the detection of these types of attacks has posed an ongoing challenge.

Despite the dilemma and distinct possibility of being utilized as an attack accelerant, the efficacy of ARE constitutes a key capability for forensic investigations. As can be seen by the SolarWinds incident (wherein malicious code was injected into the company's software, which in turn was widely distributed and utilized by client companies for a plethora of Information Technology (IT) management and remote monitoring needs), vulnerable software was quickly propagated throughout an ecosystem of mission-critical organizations. The Time to Response (TTR) was recognized as critical, but non-automated, manual labor-intensive reverse engineering intrinsically has a low TTR. Given the double-edged sword aspect of ARE, the notion of such a mechanism for mission-critical Critical Infrastructure (CI) controllers, such as ICS PLC binaries, has remained an open issue/challenge.

This paper endeavors to respond to that challenge by positing an ARE Cyber Module (ARECM), which is less prone to being utilized as an attack accelerant. Central to the requisite "less prone" protective element is an ML-facilitated Discernment Module (MLDM), which strives to detect that an attack is occurring/has occurred and timely employs (potentially), time-permitting and if still feasible, a bounded active defense mechanism to mitigate against the attack (the

 ISBN: 978-1-61208-893-8

mitigation element is beyond the scope of this paper). Central to this discernment element is yet another module, which also utilizes ML facilitation so as to perform PLC logic/code optimization (a.k.a., ML-facilitated Logic/Code Optimization Module or MLLCOM). The paper utilizes a variety of acronyms, and some of the key ones are provided for the reader's convenience in Table 1 below.

TABLE I. KEY TERMS AND THEIR ACRONYMS

Term	*Acronym*
Anomalous Sample Detection	ASD
Architecture Event Trace	AET
Automated Reverse Engineering	ARE
ARE Cyber Module	ARECM
Branch Trace Store	BTS
Instruction Translation Lookaside Buffer	ITLB
Last Branch Record	LBR
Machine Learning	ML
ML-facilitated "Pre-'ret'" Discernment Module	MLPRDM
ML-facilitated Discernment Module	MLDM
ML-facilitated Logic/Code Optimization Module	MLLCOM
Monitoring/Detecting/Mitigating Module	MDMM
Performance Monitoring Unit	PerMU
PLC Program Execution Context	PLCPEC
Precise Event Base Sampling	PEBS
Prior to the Return	(Pre-Ret)
Return-Oriented Programming	ROP
Return-to-Libc	Ret2Libc
Time to Response	TTR
Translation Lookaside Buffer	TLB

The key components — ARECM, MLDM, and MLLCOM — are delineated within the context of the MDMM Amalgam, as shown in Figure 1 below. The MDMM Amalgam is comprised of three sections: "Monitor," "Detect," and "Mitigate." ARECM is situated in the second dotted box under the "Detect" section. MLDM is also situated in the second dotted box under the "Detect" section. MLLCOM is situated in the first dotted box under the "Detect" section. The MLLCOM helper is situated under the "Monitor" section.

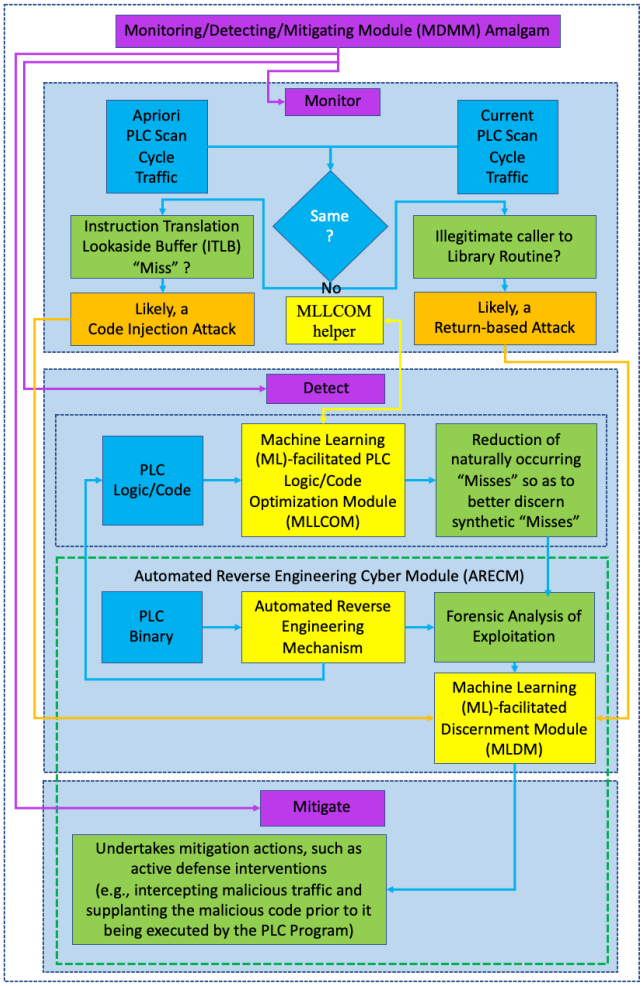

Figure 1. Monitoring/Detecting/Mitigating Module (MDMM) Amalgam: Automated Reverse Engineering Cyber Module (ARECM) with an ML-facilitated Discernment Module (MLDM) and ML-facilitated [PLC] Logic/Code Optimization Module (MLLCOM)

This section introduces the problem space. Section II presents background information and discusses the operating environment and the state of the challenge. Section III delineates the referenced ARE challenge and presents some experimental findings derived from scrutinizing a particular ICS architectural stack module, which centers upon edge PLCs engaged in inter-PLC communications, via 5G URLLC links; it also posits a prospective pathway for effectuating a viable ARECM. Section IV concludes with some observations, puts forth envisioned future work, and the acknowledgements close the paper.

II. BACKGROUND INFORMATION

Fundamentally, ICS are systems that interconnect, monitor, and control physical processes within industrial settings [6]. A plethora of sectors (e.g., energy, manufacturing, etc.) rely upon ICS for their ongoing operations. Supervisory Control and Data Acquisition (SCADA) systems are an example of ICS, and these also constitute CI/Strategic Infrastructure (SI) (a.k.a., CI/SI).

 ISBN: 978-1-61208-893-8

These CI/SI have been heavily scrutinized for security vulnerabilities, and communications is, among others, an affected area.

A. Operating Environment

With regards to the current operating environment, communications/connectivity has become a backbone of the Industrial Internet of Things (IIoT), wherein devices are interconnected so as to collect, exchange, analyze, and actuate upon data. A commonly used term that captures this paradigm is Machine to Machine (M2M) communications, and in the 5G, Beyond 5G (B5G), and 6G communications context, the envisioned service paradigm is that of massive Machine-Type Communications (mMTC) and URLLC.

As IIoT has advanced, such as within the energy and manufacturing sectors, the attack surface area for communications/connectivity has increased. This has been demonstrated by the SHODAN Internet of Things (IoT) search engine, which returns publicly accessible information regarding IoT devices (e.g., sensitive information related to internet-connected ICS devices) [3]. Many of the SHODAN-illuminated devices do not yet have the firmware updates to mitigate against the Common Vulnerabilities and Exposures (CVE) delineated by the NVD and/or U.S. Computer Emergency Response Team (CERT)- Cybersecurity and Infrastructure Security Agency (CISA) portals, and this incongruity remains an ongoing issue.

The digital transformation advances being effectuated by IIoT are encompassed within what is referred to as Industry 4.0. By way of example, "Smart Grid," a subset of Industry 4.0, is defined by the National Institute of Standards and Technology (NIST) as "modernizing the electric power grid so that it incorporates information technology to deliver electricity efficiently, reliably, sustainably, and securely… a modernized grid enables all participants to benefit from the new introduction of new technologies, from distributed resources to *advanced communications and controls*." "Smart Manufacturing," another subset of Industry 4.0, is defined by NIST as being "fully-integrated, collaborative manufacturing systems that respond in real time to meet changing demands and conditions in the factory, in the supply network, and in customer needs;" roughly speaking, this translates to the fact that, "in the factories of the future, *smart communications* will become increasingly critical in all aspects of the operation," and a smart factory involves physical production processes being combined with digital technology (i.e., *control*) [7].

For both of these industrial subsets of Industry 4.0, communications is paramount, and a key counterpoised element is the PLC. Among other tasks, the PLC acquires data from sensory machines/devices, applies certain logic/mathematical functions, and outputs computationally-derived values (to establish thresholds, etc). Within both the Smart Grid and Smart Manufacturing sectors, while SCADA systems supervise, the PLCs perform the actual operations; they are typically installed on the machines/devices they control. In the spirit of the communications/connectivity envisioned under Industry 4.0/mMTC/M2M, etc, increasingly, PLCs are engaging in inter-PLC communications. Accordingly, interoperability specifications are addressed by reference architectures, such as the Industrial Internet Reference Architecture (IIRA) of the Industrial Internet Consortium (IIC), Reference Architectural Model of Industry 4.0 (RAMI 4.0), and others; IIRA adheres to International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC)/Institute of Electrical and Electronics Engineers (IEEE) 42010:2011 "Systems and Software Engineering – Architecture Description," and RAMI 4.0 showcases various standards, such as IEC 62264 (a standard built upon the American National Standards Institute or ANSI/International Society of Automation or ISA-95 to facilitate information flow across Enterprise Resource Planning or ERP, Manufacturing Execution System or MES, and SCADA systems).

As the PLC is a principal controller for Industry 4.0, it has become a key target for cyber attackers. The ICS section of the CERT-CISA portal, as of 25 July 2021, lists 1730 advisories (69 pages of 25 advisories per page plus 5 advisories on page 70) [8]. While prior thinking held that the PLC was not subject to attack, as it was, theoretically, fully isolated from the publicly-facing external network, case studies, such as Stuxnet, have demonstrated the potential speciousness of this notion [9]. Common communications congestion attack vectors, in the form of Denial-of-Service (DoS) and Distributed Denial of Service (DDoS), are well-known. More recent studies have shown that degradation of ICS can readily be effectuated by communications degradation in the form of delay and/or loss of data packets. Among other methods, PLC output can be mutated and re-written to the PLC (this delay/loss of data has been achieved within the communications channel of ICS, such as from Phasor Measurement Units or PMUs to Phasor Data Concentrators or PDCs [10]).

To conduct PLC exploitation (e.g., malware) analysis, it is necessary to examine the PLC binary. By way of background, hopefully, the involved PLC subscribes to the standards, as delineated by IEC 61131-3, which pertain to PLC architectures, programming languages, data types, variable attributes, etc. If so, the PLC logic/code is usually developed via an IEC 61131-3-compliant Integrated Development Environment (IDE) and then compiled into a PLC binary via some compiler. The resultant PLC binary logic/code then, in effect, controls the involved PLC. The reverse engineering of this PLC binary is not straightforward, as the Tactics, Techniques, and Procedures (TTPs), via available tools/frameworks, do not directly translate between the Operational Technology (OT) arena (wherein the PLC resides) and IT arena [11]. For example, in the OT arena, there are a plethora of proprietary compilers used in generating PLC binaries, and axiomatically, these PLC binaries may not be readily accessible to commonly used IT tools (e.g., Interactive Disassembler or IDA, IDA Pro). If the PLC binary is indeed IEC 61131-3 compliant via one of the major platforms for ICS (e.g., CODESYS), then the reverse engineering process is more straightforward; however, in many situations, this is not the case.

 ISBN: 978-1-61208-893-8

To address this complexity, the notion of ARE has long been discussed [12]. Indeed, the notion of reverse engineering has become a cornerstone of software supply chain verification/integrity, particularly given the recent surge in issued directives, such as the "Improving the Nation's Cybersecurity" (Executive Order 14028, which was issued on 12 May 2021 and proceeded to direct NIST to enhance software supply chain security guidelines). The ability to uncover software supply chain vulnerabilities is essential for enhancing cyber resiliency, and ARE has been shown to effectively contribute by not only discerning vulnerabilities, but also facilitating the re-engineering of legacy software to supplant deprecated components and/or streamline for better performance. In fact, reverse engineering (and its examination and review of the design/components/build) is often used to redesign (as well as aid in source code recovery and binary code reuse [13]), enhance the involved system/product, and facilitate innovation; it is also often coupled with forward engineering, which aims to innovate and develop a new system/product. The amalgam of reverse/forward engineering is more advantageous than just forward engineering, which (in isolation) can lead to recalls/callbacks if actuated without the benefit of a Janusian perspective (i.e., leveraging lessons learned, project retrospectives, after action reviews, etc). Hence, the state of the challenge now resides in successfully counterpoising between the two (reverse/forward engineering).

B. *State of the Challenge*

The open challenge of ARE centers upon the point that while it can indeed accelerate the forensic work of a cyber defender, it also represents a potential exploitation point/accelerant for a cyber attacker. Architecting an ARE cyber module (in a reverse/forward engineering fashion), which favors the defender, has been an elusive, non-trivial feat. However, the literature does present several contributions to this area, and specifically, this paper posits a ML-facilitated "Pre-'ret'" Discernment Module (MLPRDM), which shows some promise; in particular, the MLPRDM focuses upon recognizing the set of legitimate instruction calls prior to the return or "ret" (or "Pre-'ret'") instruction contained within the subroutines of the PLC Program. Legitimate instruction calls proceed accordingly while MLPRDM-recognized illegitimate instruction calls may experience intervention (time-permitting and if practical). The MLPRDM gleans patterns from the work of the MLLCOM and is the key engine for the MLDM. The MLPRDM is delineated within the context of the MDMM Amalgam, as shown in Figure 2 below. The MLPRDM is situated in the "Detect" section and straddles the PLCPEC dotted box (which is located within the ASD dotted box) and the ARECM dotted box.

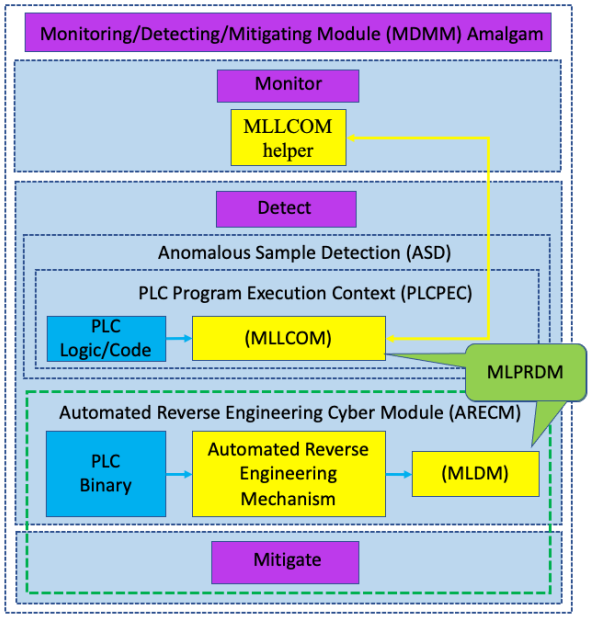

Figure 2. MLPRDM depicted within the context of the MDMM Amalgam

To clarify the value-added proposition of the MLPRDM, some background information is required. Broadly speaking, there are three methods for PLC exploitation: Firmware Modification Attacks (FMA), Control-flow Attacks (CFA), and Configuration Manipulation Attacks (CMA); these attack vector categories had been delineated at the BlackHat European Conference in 2016, and combinatorial attacks involving an amalgam, from among these categories (i.e., FMA, CFA, and CMA), are particularly potent; others classify PLC exploitation by security defects: firmware security defects, program security defects, and operation security defects [14]. Operation security defects can be further sub-divided into: (1) attack on protocol defects (e.g., since most communications protocols are not encrypted, packets can be captured and/or the data store of the registers can be read, and replay attacks [legitimate data is repeated and/or delayed], etc. can then be effectuated), (2) tampering attack at the Input/Output (I/O) interface (e.g., since modifying the I/O pin configuration does not necessarily issue an alarm, a tampering attack can be covertly effectuated), (3a) injection attack to affect the program flow control instructions or operational control flow (e.g., as operational control flow is dictated by PLC code blocks, intermediate code instrumentation and/or malicious code execution can exploit this facet), and (3b) return-oriented attack to affect the operational control flow (e.g., by leveraging exploits, wherein legitimate instructions are overwritten, malicious instructions can be indirectly executed [14]. The latter sub-divisions (3a, 3b) are the focus of MLPRDM.

C. *Formal Verification of PLC Logic/Code*

There are numerous pathways to undertake verification of PLC Programs. One pathway involves formal verification of PLC Programs; this is typically achieved by translating

 ISBN: 978-1-61208-893-8

the PLC Program into a formal model, which in turn can serve as input into a model checker (e.g., SMV Symbolic Model Checker, NuSMV [a re-implementation and extension of SMV], UPPAAL [a portmanteau of Uppsala University and Aalborg University], etc.) [15][16]. Yet, the macro vantage point might not necessarily discern the more micro matters. For example, various software security studies have declared that simple micro-errors (intentional and/or unintentional) can readily disrupt the availability and integrity of a target PLC [17]. To aggravate this matter, in many cases where errors do exist, the code will still compile and run on the PLC; hence, no warning is necessarily issued. Suffice it to say, mitigation of these PLC errors or code defects (e.g., buffer overflows/ overruns, stack overruns, etc) is difficult, as these defects are difficult to discern, and fuzz testing (a.k.a., fuzzing) has met with limited success; furthermore, as the PLC often has numerous constituent programming languages, the requirements for the fuzzing schema tend to be quite elaborate so as to undertake the challenge of the associated semantic complexity, and the efficacy of even state-of-the-art fuzzers has been sub-optimal [18][19].

D. *General Detection within the PLC Program*

The notion of installing a general detection module aboard the PLC has, to date, met with limited appeal; due to the already limited computational resources onboard the PLC, installing an additional program (with its additional computational load requirements) aboard the already resource-constrained PLC, so as to examine the PLC Program, has met with various heuristical challenges. For example, for real-time operations, the task of detection has a lower priority than that of a control task, particularly given the mission-critical nature of a PLC [20]. This de-prioritization sets the stage for detection misses, so the potential efficacy is already in question.

E. *Anomalous Sample Detection within the PLC Program*

The challenge then becomes one of designing a specialized detection module, which has both minimal impact on the computational load of the already resource-constrained PLC as well as a minimal footprint given the higher priority control tasks at hand. It turns out that this approach vector is somewhat feasible in the form of Anomalous Sample Detection (ASD), which demonstrates some promise with regards to code-injection and Return-Oriented Attacks (ROAs).

As a generalization, a code-injection attack (a.k.a., Remote Code Execution or RCE) refers to the exploitation of code defect or bug (e.g., buffer overflow/overrun, dangling pointer, etc) that processes the externally injected malicious code and alters the course of instruction execution (i.e., operational control flow). In the case of a buffer overflow/overrun, the legitimate return address is overwritten, and the operational control flow is diverted to the location specified by the new return address. In contrast, a ROA does not inject malicious code; rather, it attains control of the call stack and leverages the internally resident pre-existing code. ROAs are further sub-divided into Return-to-Libc (Ret2Libc) attacks (which causes the PLC Program to jump to some code block, such as that for various functions — system(), execve(), etc. — within the standard library, say, for the C programming language or libc, which is already loaded into memory) and Return-Oriented Programming (ROP) attacks (which manipulate the call stack to indirectly execute specific instructions or groups of instructions immediately prior to the "ret" instruction contained within the subroutines of the PLC Program). This is shown in Figure 3 below.

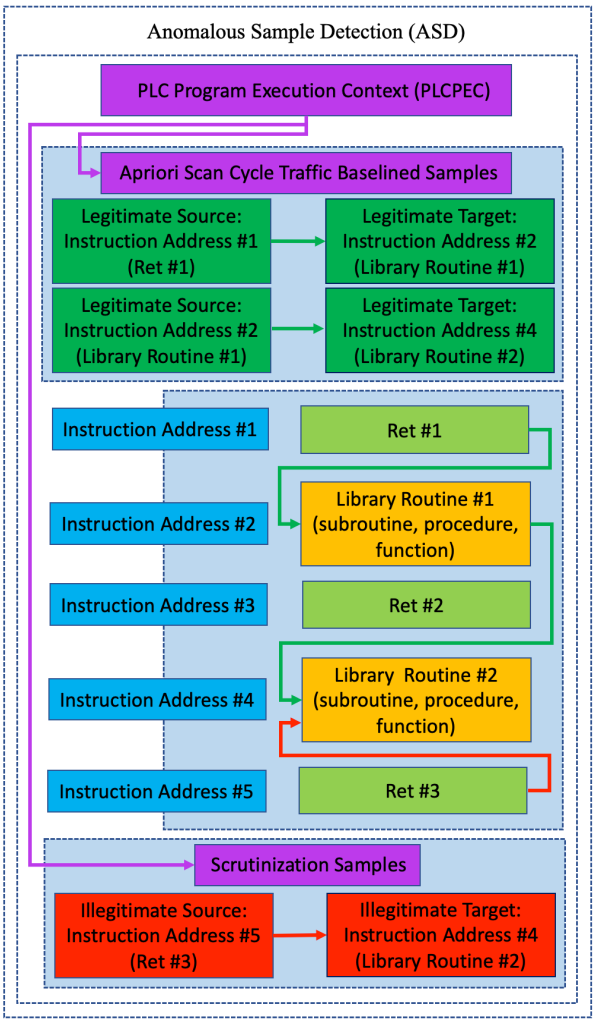

Figure 3. PLC Program Execution Context (PLCPEC) in the context of Anomalous Sample Detection (ASD)

Regardless of the attack vector (e.g., code-injection attack, ROA), the anomalous aspect can be discerned when comparing the examined *scrutinization samples* (a.k.a., performance samples) against *apriori baselined samples*. For example, a Translation Lookaside Buffer (TLB) or Instruction Translation Lookaside Buffer (ITLB) "miss" could indicate a code injection attack (wherein the operational control flow is transferred to the injected

 ISBN: 978-1-61208-893-8

malicious code, and hence, the "miss" for legitimate code). As another example, an examination of the legitimate callers of the various library routines (a code block — subroutine, procedure, function, etc — used for recurring tasks) should be instructive as to whether an illegitimate caller executed a ROA. As anomalies can indeed be discerned, regardless of the attack vector utilized (e.g., code-injection attack, ROA, etc), via these scrutinization samples, the described approach is loosely considered to be operation security defect agnostic (bounded within the scope of either code injection attacks or ROAs). This ASD approach correlates events with the ongoing PLC Program Execution Context (PLCPEC). This PLCPEC is in the form of baselined instruction addresses and callers of library routines, such as shown in Figure 3 above. It can be ascertained, via examining Figures 2 and 3, that the ASD approach is predicated upon the MLPRDM (shown in Figure 2), and this particular module is further explained in the experimentation section.

III. EXPERIMENTATION FINDINGS

The experimentation involved a particular ICS architectural stack module, which centers upon [edge] PLCs focused upon inter-PLC communications, via 5G URLLC links, within an ICS context. As should be axiomatic, a key aspect of the 5G/B5G/6G ecosystem is that hardware is principally supplanted with software so that future upgrades will be software-centric. However, this increased utilization of Software-Defined Networking (SDN) within the network core also expands the attack surface opportunities [21]. For this particular case, the 5G-related PLCs were the targets. Given the comparable nature of the involved PLCs, only one PLC was examined.

In accordance with the acknowledged deterministic behavior of the underlying engineering software for the PLC Program (e.g., the same set of request messages are utilized), a specific heuristic was utilized; given the scan cycle traffic of prior sessions, a pattern among the request messages could be ascertained; this particular heuristic is supported within the literature [22]. Accordingly, to operationalize the posited ASD approach, several facets were baselined apriori by pre-processing the PLC binary and recording the following: (1) legitimate callers of the library, (2) legitimate callers for each library routine, and (3) legitimate callers for various consecutive call patterns for the library routines. Furthermore, the instructions occurring prior to any "ret" instruction were recorded, and a relative weighting was assigned to each instruction depending upon their distance to their associated "ret."

These operationalization actions spawned other complexities. For example, while using dedicated processor commands, such as Branch Trace Store (BTS) can be quite effective for recording (e.g., memorializing the last executed branches), the computational overhead can also be quite high; there are also comparable processor commands (e.g., Last Branch Record or LBR, Architecture Event Trace or AET, etc.). For these cases, a substantive portion of the overhead can be attributed to the large number of *performance samples* (derived from the Performance Monitoring Unit or PerMU of the involved processor), which include various control transfer events (e.g., calls [to subroutines], returns or "rets" [wherein control flow continues with the instruction following the call], exceptions/interrupts [wherein unexpected events disrupt instruction execution/control flow]) and their associated control transfer information (e.g., source address, target address, and various other properties, etc.) [23].

The standard recordings of PerMUs can include TLB/ITLB misses, cache misses [wherein the processor must retrieve the data from main memory], branch prediction misses or branch mispredictions [a misprediction in the next instruction to process], etc. Typically, the computational overhead associated with tabulating these events is compounded by the ensuing interrupts (spawned when the pre-defined memory is saturated). However, Precise Event Base Sampling (PEBS) or comparable approaches can process these recordings of events into pre-defined memory sectors, such as when the event occurrences exceed a particular threshold, rather than spawn an interrupt. Hence, the "tighter" the threshold, the more quickly and more likely it is for, let us say, TLB/ITLB *synthetic misses* (i.e., indicative of a code-injection attack) to be detected. However, to decrease the likelihood of false positives, it is necessary to decrease the likelihood of *naturalistic misses*, and this is achieved via the PLC logic/code [performance] optimization performed by the MLLCOM (e.g., an enhancement of code layout, such as by re-positioning code blocks within a procedure to decrease branch misses). After all, if the control flow frequently traverses several distinct and disparate pathways throughout the code region, it is more likely to experience misses.

Experimentation has shown that the use of certain algorithmic approaches (e.g., Code Tiling-like), particularly for code layout optimization, achieves better performance (e.g., call frequency grouping) within the allotted time span than other algorithms (e.g., Pettis-Hansen); the algorithms are known to have $O(n*(n+n^2))=O(n^3)$ as the worst-case asymptotic complexity, but the difference in approach — the utilization of various approximations — underpin the performance differences [24]. Moreover, with the MLLCOM engaging in code layout optimization and gleaning the apriori consecutive call patterns, it is able to facilitate a reduction in the number of false positives by better distinguishing between legitimate and illegitimate control flow behavior. Preliminary experimentation with MLLCOM has also noted that its profiling (via its MLLCOM helper) at the Monitor level and trace processing at the Detect level well serves to facilitate better optimizing basic blocks within a procedure (i.e., procedure splitting) via the notion of hot blocks (executed frequently) and cold blocks (i.e., executed infrequently). This helps to decrease branch misses.

Overall, these types of optimizations, among others, are central to the discernment equation. Due to "real-time execution deadlines" and Quality of Service (QoS) stipulations, "compilers for PLC binaries typically only undertake very conservative optimizations, if any" [13]. Yet,

 ISBN: 978-1-61208-893-8

there are several opportunities to optimize PLC binaries. As previously discussed, the paradigm of sub-optimal instruction locality can adversely impact the PLC Program performance, via, by way of example, TLB/ITLB misses, which can induce memory stalls (cycles for which the processor is stalled while awaiting memory access) [24]. However, a paradigm of optimized instruction locality (e.g., pre-positioning callers in close proximity to their callees) can dramatically improve performance [24][25].

From an architectural perspective, the overall MDMM amalgam is able to monitor/detect units of work that are in conformance with the PLC scan cycle. By way of background, non-PLC languages (and their associated binaries) typically adhere to sequential units of work as part of their execution model. In contrast, PLC binaries adhere to an execution model that conforms to the continuously executing scan cycle. Due to the continuous nature of the execution scan cycle, dynamic analyses of the PLC binary is non-trivial. The MDMM architecture lends to overcoming this challenge, via the positioning of its various constituent modules. In particular, the ARECM is nicely operationalized within the ASD of the MDMM by way of the interplay between the MLDM and MLLCOM, via the MLPRDM, such as previously shown in Figure 2 and summarized in Figure 4 below. With the enhanced context from the MLLCOM (given the optimization work) and the insights from the MLDM (e.g., comparison of the current scan cycle traffic with apriori scan cycle traffic) serving as accelerants for the ARECM, the MDMM architecture and underpinning MLPRDM provide enhanced discernment.

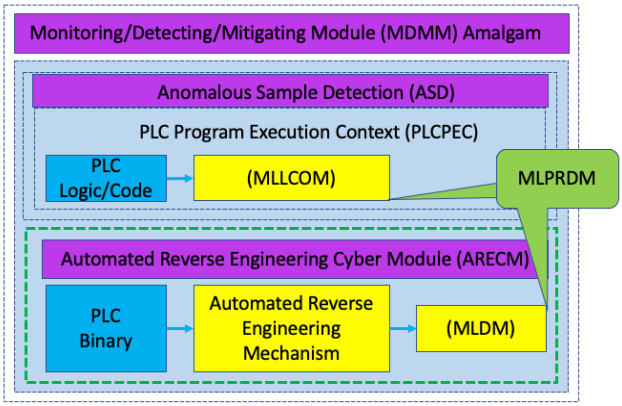

Figure 4. Automated Reverse Engineering Cyber Module (ARECM) in the context of the Monitoring/Detecting/Mitigating Module (MDMM) Amalgam's Anomalous Sample Detection (ASD)

As previously shown in Figures 2, 3, and 4, the methodological approach centers upon the fact that the MLPRDM discerns the set of legitimate instruction calls prior to the return or "ret" (or "Pre-'ret'") instruction contained within the subroutines of the PLC Program. Legitimate instruction calls are permitted to proceed while MLPRDM-recognized illegitimate instruction calls may experience intervention (time-permitting and if practical). The MLPRDM discerns patterns from the work of the MLLCOM and is, in effect, the key engine for the MDMM, which directly addresses the ARE open issue/challenge.

IV. CONCLUSION

While ICS were originally designed to operate in isolated environments, the convergence of OT and IT have increased the attack surface area within this ecosystem, particularly for key devices, such as PLCs. Several attack vectors, within the rubric of Denial of Engineering Operations (DEO), have emerged to target, among other devices, PLCs [26]. Unfortunately, the cyber defense and resiliency capabilities in the OT sector greatly differ from that of the IT sector, such that the TTPs, which can be brought to bear in support of PLCs, have been limited thus far. Furthermore, the PLCs operating within the OT environment have had a high availability onus (of controlling real-world physical processes), but are often limited by their resource-constrained, legacy (and possibly proprietary) environs.

It has been shown that PLCs have been vulnerable to DEO attacks, wherein the legitimate instruction in a control logic is replaced with noise data (e.g., a sequence of 0xFF bytes) to cause the PLC to malfunction, and/or the PLC may have had legitimate instructions relaced with malicious instructions. In either case, the operational control flow has been compromised. Two synergistic pathways for mitigation are evident. First, detection is ideal; to the degree that this can be achieved with enough time to take mitigation action, then it would be ideal to effectuate an active defense (e.g., ironically, a man-in-the-middle counter-attack) by intercepting the malicious traffic and supplanting the malicious code prior to that code being executed by the PLC Program. Whether or not this detection/active defense can be achieved, a forensic analysis to comprehend the extent of the control flow manipulation is required. Consequently, second, ARE is required; to the degree that it can occur in real-time for a robust diagnosis of the attack, such that active countermeasures can be readily deployed, then the intent for which MLPRDM was designed would be operationalized.

Key to its success, the MLPDRM goals of profiling and trace processing are critical; its actions will, among other effects, minimize the number of naturally occurring ITLB misses so that synthetically occurring ITLB misses will be more illuminated. The posited MLPRDM, set amidst the MDMM architecture, requires further quantitative benchmarking, but the initial architectural techniques (for a lower overhead, minimally intrusive approach vector, and more accurate monitoring/detection paradigm) and preliminary experimental results show promise. The significance of this potential is that the approach addresses the Industry 4.0 cyber-physical security open issue/challenge surrounding ARE for PLCs; in essence, the posited ARE cyber module, which leverages the various described ML facilities to aid in the ARE task, is less prone to be utilized as an attack accelerant. The author hopes to focus on substantive quantitative benchmarking as part of future work, as the preliminary results are quite promising.

 ISBN: 978-1-61208-893-8

ACKNOWLEDGMENT

This research is supported by the Decision Engineering Analysis Laboratory (DEAL), an Underwatch initiative. This is part of a VT white paper series on 5G-enabled defense applications, via proxy use cases, to help inform Project Enabler.

REFERENCES

[1] R. Awad, S. Beztchi, J. Smith, B. Lyles, and S. Prowell, "Tools, Techniques, and Methodologies: A Survey of Digital Forensics for SCADA Systems," Annual Computer Security Applications Conference, 2018, pp. 1-8.

[2] T. Wu and J. Nurse, "Exploring the Use of PLC Debugging Tools For Digital Forensic Investigations on SCADA Systems," The Journal of Digital Forensics, Security and Law, vol. 10, 2015, pp. 79-96, https://doi.org/10.15394/jdfsl.2015.1213

[3] S. Chan, "Prototype Orchestration Framework as a High Exposure Dimension Cyber Defense Accelerant Amidst Ever-Increasing Cycles of Adaptation by Attackers: A Modified Deep Belief Network Accelerated by a Stacked Generative Adversarial Network for Enhanced Event Correlation," The Third Conference on Cyber-Technologies and Cyber-Systems (CYBER 2018) IARIA, 2018, pp. 28-38, ISSN: 2519-8599, ISBN: 978-1-61208-683-5.

[4] S. Becker, C. Wiesen, and N. Albartus, "An Exploratory Study of Hardware Reverse Engineering – Technical and Cognitive Processes," Proceedings of the Sixteenth Symposium on Usable Privacy and Security, 2020, pp. 1-17.

[5] M. Fybrbiak et al., "HAL – The Missing Piece of the Puzzle for Hardware Reverse Engineering, Trojan Detection and Insertion," IEEE Transactions on Dependable and Secure Computing, 2018, pp. 498-510.

[6] "Industrial Control Systems Security," Accessed on: Aug 27, 2021. [Online]. Available: https://wp.nyu.edu/momalab/industrial-control-systems-security/

[7] S. Bagchi, "Smart Communication: Factory of the Future – Critical Connections," Accessed on: Aug 27, 2021. [Online]. Available: https://www.automation.com/en-us/articles/2015-2/smart-communication-factory-of-the-future-critical

[8] "ICS-CERT Advisories," Accessed on: Aug 27, 2021. [Online]. Available: "https://us-cert.cisa.gov/ics/advisories?items_per_page=25&page=0

[9] S. Gallagher, "Vulnerable industrial controls directly connected to Internet? Why not?," Accessed on: Jul 23, 2021. [Online]. Available: https://arstechnica.com/information-technology/2018/01/the-internet-of-omg-vulnerable-factory-and-power-grid-controls-on-internet/

[10] Y. Wang et al., "Access Control Attacks on PLC Vulnerabilities," Journal of Computer and Commpunications, Vol. 6, pp. 311-325, 2018, doi: 10.4236/jcc.2018.611028.

[11] "2020 Gartner OT Security Best Practices," Accessed on: Jul 23, 2021. [Online]. Available: https://www.armis.com/analyst-reports/2020-gartner-ot-security-best-practices/

[12] S. Zonouz, J. Rrushi, and S. McLaughlin, "Detecting Industrial Control Malware Using Automated PLC Code Analytics," IEEE Security & Privacy, vol. 12, pp. 40-47, 2014, doi: 10.1109/MSP.2014.113.

[13] A. Keliris and M. Maniatakos, "ICSREF: A Framework for Automated Reverse Engineering of Industrial Control Systems Binaries," 2018, pp. 1-15, doi: 10.14722/ndss.2019.23271.

[14] H. Wu, Y. Geng, K Liu, and Wenwen Liu, "Research on Programmable Logic Controller Security," IOP Conf Series: Materials Science and Engineering, vol. 569, pp. 1-13, 2019, doi: 10.1088/1757-899X/569/4/042031.

[15] J. Bengtsson, K. Larsen, F. Larson, P. Pettersson, and W. Yi, "UP-PAAL – a Tool Suite for Automatic Verification of Real-Time Systems," Proc Workshop Hybrid Systems III: Verification and Control, vol. 1066, 1996, pp. 232-243.

[16] V. Gourcuff, O. Smet, and J. Faure, "Efficient Representation for Formal Verification of PLC Programs," 2006 8th International Workshop on Discrete Event Systems, 2006, pp. 182-187, doi: 10.1109/WODES.2006.1678428.

[17] S. Valentine and C. Farkas, "Software security: Application-level vulnerabilities in SCADA systems," 2011 IEEE International Conference on Information Reuse & Integration, 2011, pp. 498-499, doi: 10.1109/IRI.2011.6009603.

[18] C. Lemieux and K. Sen, "Fairfuzz: A Targeted Mutation Strategy for Increasing Greybox Fuzz Testing Coverage," 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018, pp. 475-485.

[19] L. Simon and A. Verma, "Improving Fuzzing through Controlled Compilation," 2020 IEEE European Symposium on Security and Privacy (EuroS&P), 2020, pp. 34-52, doi: 10.1109/EuroSP48549.2020.00011.

[20] S. Kottler, M. Khayamy, S. Hasan, and O. Elkeelany, "Formal Verification of Ladder Logic Programs using NuSMV," IEEE Southeastcon, 2017, pp. 1–5, doi: 10.1109/SECON.2017.7925390.

[21] K. Fysarakis et al., "A Reactive Security Framework for operational wind parks using Service Function Chaining,"2017 IEEE Symposium on Computers and Communications (ISCC), 2017, pp. 663-668, doi: 10.1109/ISCC.2017.8024604.

[22] S. Qasim, J. Lopez, and I. Ahmed, "Automated Reconstruction of Control Logic for Programmable Logic Controller Forensics," Springer International Publishing, 2018, pp. 1-21.

[23] L. Yuan, W. Xing, H. Chen, B. Zang, "Security Breaches as PMU Deviation: Detecting and Identifying Securing Attacks Using Performance Counters," The 2nd ACM SIGOPS Asia-Pacific Workshop on Systems (APSys), 2011, pp. 1-6, doi: 10.1145/2103799.2103807.

[24] X. Huang, B. Lewis, K. McKinley, "Dynamic Code Management: Improving Whole Program Code Locality in Managed Runtimes," Proc. of the Intl. Conf. on Virtual Execution Environments (VEE), 2006, pp. 1-11, doi: 10.1145/1134760.1134779.

[25] J. Chen and B. Leupen, "Improving instruction locality with just-in-time code layout," Proceedings of the USENIX Windows NT Workshop, 1997, pp. 25-32.

[26] S. Senthivel, S. Dhungana, H. Yoo, I. Ahmed, and V. Roussev, "Denial of Engineering Operations Attacks in Industrial Control Systems," Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, 2018, pp. 319-329, https://doi.org/10.1145/3176258.3176319.

 ISBN: 978-1-61208-893-8