SoK: Attacks on Industrial Control Logic and Formal Verification-Based Defenses

Ruimin Sun
r.sun@northeastern.edu
Northeastern University

Alejandro Mera
mera.a@husky.neu.edu
Northeastern University

Long Lu
l.lu@northeastern.edu
Northeastern University

David Choffnes
choffnes@ccs.neu.edu
Northeastern University

***Abstract*—Control logic programs play a critical role in industrial control systems. A vulnerable control logic could lead to devastating consequences in the physical processes, as shown in Stuxnet and similar attacks. Over the years, academic and industrial researchers have investigated various fault injection and modification attacks on control logic as well as formal verification-based defenses. Although formal verification techniques have in general improved the quality of control logic programs, we find a significant gap between the academic research and the industry practices in defending against attacks on control logic. Besides, the future research directions remain unclear as to protect control logic from the ever-expanding attack surface partly caused by the increasing needs for interconnectivity.**

This work fills the gap by systematizing the knowledge of control logic modification attacks and the formal verification-based defenses. Our study covers the full chain of developing and deploying control logic programs, from engineering stations to target PLC. The primary goals of the systematization are (1) to explore the evolving technology and security landscape surrounding control logic programs, (2) to investigate newly emerged attack surfaces on PLC systems and the formal verification-based defenses, and (3) to identify the open challenges and needs that existing formal verification based-defenses failed to address. Based on the knowledge systematization, we provide a set of recommendations for both academic researchers and industry practitioners to better focus their work on countering critical and emerging threats.

I. Introduction

Industrial control systems (ICS) are subject to attacks sabotaging the physical processes, as shown in Stuxnet [65], Havex [85], TRITON [64], Black Energy [35], and the German Steel Mill [101]. These attacks cause severe financial losses and devastating consequences to multiple industrial sectors, such as water treatment, chemical processing, and nuclear plants. Control logic modification attacks are among the most critical thanks to their ability to subvert the entire infrastructure controlled by the victim systems. For example, the well-known Stuxnet [65] attack damaged one-fifth of Iran's nuclear centrifuges.

As a principled approach to detect flaws in programs, formal verification on control logic was first presented nearly thirty years ago [117].Some recent work [54], [55] has shown that control logic is amenable to formal verification thanks to the graph-based programming languages and the small program size. Formal verification provides rigorous mathematical proofs at different levels of abstractions.

However, with recent advances in Industrial Internet of Things (IIoT) and expanding attack landscape, existing formal verification techniques become insufficient for countering existing attacks. Control logic can interact with multiple components e.g., engineering software (integrated development environment), supervisory control and data acquisition (SCADA), human-machine interface (HMI) and online services (web). Attacks targeting control logic modification can happen in different stages: during development, transmission, or execution. Formal verification faces challenges in (1) countering the complex industrial networking, with proprietary network protocols [38], [97], [102], [131], (2) overcoming the real-time constraints with limited resources on the programmable logic controller (PLC) [143], and (3) efficiently define explicit and complete properties to avoid evasive defenses [111].

This work aims to systematically investigate control logic modification attacks and formal methods based defenses. We first define the scope of control logic attacks and investigate the existing techniques used in such attacks. To identify the current trends of in recent attacks, we collect related vulnerabilities from two widely used vulnerability databases [19], [20]. The results show extensive possibilities for control logic modification attacks, which inject malicious payload in commands or organization blocks. We found that there has been an exponential increase in the number of related vulnerabilities in recent years. Complex networking and infrastructure, high diversity in vendors and program versions, and emerging technical trends (e.g., cloud-based control) all contributed to the increase.

Second, we investigate the state-of-the-art formal verification-based defenses. Surveying the related papers published in the last thirty years, we found that applying formal verification to control logic has made great progress [142], including automatic formalization of complete control logic languages defined in IEC-61131 [39], and hybrid models that support transition from a continuous part to a discrete part of a system [92]. It is also encouraging to see prototypes of joint formal verification for both PLC and SCADA [114], and user-friendly verification with simulation and visualization. Nevertheless, we discovered multiple security challenges for formal verification based defenses, including synchronization, multitask, and etc.

As a call for solutions to address these challenges, we provide a set of recommendations for vendors, control logic programmers, and security researchers. We recommend PLC vendors to prioritize security with standardized design, so that formal verification can be more targeted and effective.

We recommend users, including control logic developers and operators, and formal verification experts, to adopt existing best practice formal verification techniques, at programming, transmission, and runtime. We recommend security researchers to further investigate formal verification with mixed languages, explicit properties, multitask, and synchronization. We suggestion extension to firmware, runtime, and input protection, through collaborating with other technologies.

A. Systematization Methodology

1) Scope of the paper: This paper focuses on systematization of control logic modification attacks, and formal verification based defenses to ensure control logic integrity. Specifically, we consider the following aspects while performing the systematization.

- **Stage**: This refers to when a control logic attack/defense happens, such as the programming/development stage, the transmitting stage (from an engineering software to a PLC), or the execution stage.
- **Assumption**: This refers to the threat model of an attack/defense, the underlying weaknesses of a reported vulnerability, and other dependencies to perform the attack/defense.
- **Approach**: This refers to the concepts, techniques, tools, and strategies being used in an attack/defense.
- **Industrial sector**: This refers to the domain where a control logic program is designed for or used.
- **Open challenges**: These are the unresolved challenges associated with detecting control logic modification attacks, awaiting future research to address.

2) Paper selection criteria: The literature is selected based on the following criteria. The work (1) investigates control logic modification attacks or formal verification based defenses, (2) is impactful considering its number of citations, or (3) discovers a new direction for future investigation.

B. Contributions

- Systematization of control logic modification attacks and formal verification-based defenses in the last thirty years.
- A framework to collect, filter, and analyze control logic modification vulnerabilities from two notable databases.
- Bridging the gap between industrial and research efforts, with a set of recommendations on the best practices of using formal verification to mitigate control logic modification attacks.
- Pointing out future research directions.

II. Background

This section first defines control logic, then introduces its background, and finally describes its attack surface and formal verification based protection.

A. Control Logic

We define control logic as a program that is executed in a PLC and controls a process taking inputs from sensors and sending outputs to actuators. Control logic code is usually programmed in a remote engineering software, after that is transmitted to a PLC, and finally executes in a PLC. During the runtime, control logic repeatedly executes a three-step scan cycle. First, the *input scan* takes a snapshot of the input and saves it in a data table. Then, the *logic execution* feeds the input to the control logic and executes its instructions. Finally, the *output scan* produces the output based on the execution result.

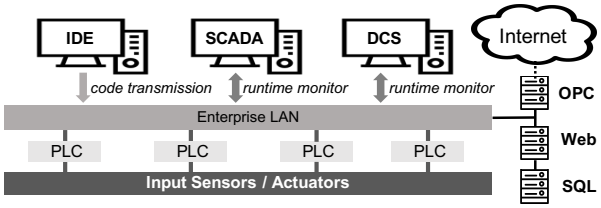

Fig. 1: The interacting components of control Logic.

1) Programming languages: Control logic can be programmed in five languages defined by IEC-61131 [132]: function block diagram (FBD), ladder diagram (LD), structured text (ST), sequential function chart (SFC), and instruction list (IL). FBD and LD are diagram-based languages. Some high-end PLCs also support computer-compatible languages (*e.g.,* C, BASIC, and assembly), special high-level languages (*e.g.,* Siemens GRAPH5 [28]), and boolean logic languages [108]. Manufacturers may choose different schemes for I/O addressing, memory organization, and instruction sets, making it hard for the same code to be compatible across vendors, or even models within the same vendor. This poses challenges for control logic protection.

2) Interacting components: Due to the advances in modern PLCs, control logic can interact with a set of software and services, as Figure 1 shows. For example, engineering software or IDE in a general-purpose computer, e.g. Allen Bradley RSLogix [24], Mitsubishi GX Developer [17], designs and programs the control logic. An HMI, e.g. Rockwell RSView32 [25], provides a user-friendly interface allowing a process personnel to interact with the control logic running in the PLCs . A SCADA [23] supervises the PLC through analyzing the the state of input, outputs and internal variables of the control logic. [1]. Recent advance in cloud computing and IIoT allows control logic to communicate through the Internet, with Web, SQL, Email, or Message Queueing Telemetry Transport (MQTT), and interconnect in a Distributed Control System (DCS).

B. Control Logic Security

Control logic suffers from numerous types of attacks, e.g. denial-of-service attacks, spoofing attacks, and modification attacks [118]. To defend against these attacks, industrial and research efforts have presented and standardized a set of security solutions, such as authorization and access control,

[1]The definition of these components are getting blurry, for example, a PLC may have an HMI embedded as a bundle, and a broader definition of SCADA considers HMI as part of its components.

securing communication (e.g. SSL/TLS, HTTPS, and VPN), and Intrusion Detection System (IDS), and forensic analysis, as used in most IT industries [59], [118], [137]. This work focuses on control logic modification attacks and formal verification based protection, since a large volume of works have been done over the years, while (un)satisfied requirements are still unclear between the industry and the academy [54], [59].

1) Control logic modification attacks: We define such attacks as modifications that can cause control logic to output abnormal and potentially unsafe commands. The modifications can come from injections and manipulations at the stages of source code programming, code transmission, and runtime. In particular, runtime modification can result from the input sensors [110], communication and synchronization between control logic and an interacting component [143], and the PLC memory [129].

2) Formal verification: Formal verification is a method proves or disproves if a program/algorithm meets its specifications or desired properties based on certain form of logic [15]. The specification may contain security requirements and safety requirements. Commonly used mathematical models to do formal verification include finite state machines, labeled transition systems, vector addition systems, Petri nets, timed automata, hybrid automata, process algebra, and formal semantics of programming languages, e.g. operational semantics, denotational semantics, axiomatic semantics, and Hoare logic. In general, there are two types of formalization: model checking and theorem proving [83]. Model checking uses temporal logic to describe specifications, and efficient search methods to check whether the specifications hold for a given system. Theorem proving describes the system with a series of logical formulae. It proves the formulae imply the property via deduction with inference rules provided by the logical system. It usually requires more background knowledge and nontrivial manual efforts.

Existing works concentrate on formal verification at the control logic programming stage. For example, the European Organization for Nuclear Research (CERN) uses PLCverif, a formal verification tool to check critical control logic programs [59].

III. Systematization of Attacks

In this section, we systematize control logic modification attacks from industrial and research efforts. We aim to identify the context of changing technology and threat landscape. We then call out the open challenges and shine light on the improvement of security mechanisms.

A. Industrial Efforts

Industrial efforts are represented by reported Common Vulnerabilities and Exposures (CVE)s. We first extract the CVEs to understand the attack methods, underlying weaknesses, and their impact on different industrial sectors. We then identify the general trends of control logic modification attacks.

The information of control logic vulnerabilities come from several sources: the ICS-CERT [19], the National Vulnerability Database (NVD) [20], and the Exploit Database [21] created by Offensive Security. We developed an analysis framework to automatically crawl and extract useful information, such as Common Weakness Enumeration (CWE)s, the impacted vendors and industrial sectors, and the complexity of the CVEs. With this framework, we extracted useful information and generated reports [2] from year 2008 (i.e. the first control logic related vulnerability was reported) to the end of year 2019. The detailed description of the databases and the framework is listed in the Appendix.

1) Stage: Some vulnerabilities aims at modifying control logic source code, by exploiting the engineering software [4]. Some vulnerabilities leverage Ethernet design flaw and use a specially crafted packet to delete control logic programs [7], [13]. Other vulnerabilities modify control logic at runtime, leveraging firmware and authentication flaws, and triggering PLC fault states to overwrite the control logic [10], [29]. Most of the vulnerabilities leverage the runtime communication of control logic with the other components, using specially crafted packets, or commands to achieve remote code modification [5], [6], [9].

2) Assumptions: The collected control logic vulnerabilities assume some triggering conditions and weaknesses of the programs. To trigger a vulnerability, an attacker will either need direct access to the PLC, or the communication link between a PLC and a supervisory software.

The weaknesses are represented by CWEs, as shows in Figure 7. Half of the CWEs are not novel, such as Improper Restriction of Operations CWE-119, Cross-site Scripting CWE-79, Improper Input Validation CWE-20, in which CWE-119 has the most vulnerabilities associated with nearly 50 records in total. These CWEs are reported almost every year in the last ten years. The other half CWEs are only reported in the last few years, e.g. Stack-based Buffer Overflow CWE-121, Uncontrolled Resource Consumption CWE-400.

Furthermore, some of the CWEs are specifically common to control logic vulnerabilities, e.g. Stack-based Buffer Overflow CWE-121, Uncontrolled Search Path Element CWE-427, Improper Access Control CWE-284. The rest of the CWEs align with the general top 25 CWEs [1] reported.

3) Approaches: Industrial security practitioners have found various types of vulnerabilities in performing control logic modification attacks. Representative examples [3] of control logic vulnerabilities are as follows:

a) Engineering software: CVE-2010-5305 [4] allows an unauthorized user to program and configure the control logic programs in a series of Rockwell controllers. The vulnerability is caused by a potential for exposure of the product's password used to restrict unauthorized access to Rockwell controllers.

b) Ethernet: CVE-2017-12088 [7] leverages a specially crafted packet, with certain bytes in the crash section, to send to a Micrologix controller over the default port. This causes

[2]The databases have some missing points. We only report available ones.

[3]Some vulnerabilities are from a PC, but can potentially modify control logic, according to the ICS advisory.

the PLC to enter a fault state, and clear the existing control logic.

c) Stealthy attacks: CVE-2019-10929 [13] allows an attacker in a Man-in-the-Middle position to modify network traffic exchanged on port 102/tcp to SIMATIC controllers. This is caused by certain properties in the calculation used for integrity protection. If control logic programs are communicating through this port, it is possible to stealthily change the code without being notified.

d) PLC authentication: Some vulnerabilities are due to unauthenticated users having read and/or write permissions on numerous files within the PLC. With these permissions, it is possible to change control logic, insert invalid values, or trigger device faults [29]. CVE-2017-14463 allows an attacker to trigger a fault state to overwrite the control logic data file with null values. CVE-2017-14465 allows an attacker to force any PLC input or output, causing unpredictable activity of the control logic. CVE-2017-14467 allows an attacker to edit rungs online, including addition, deletion, or modification of existing control logic. Additionally, faults and cpu state modification can be triggered if specific control logic is used.

e) PLC firmware: Firmware is usually considered secure because of authentication at boot-time and runtime integrity check. CVE-2017-14468 allows to trigger the PLC to load programs from EEPROM on error. A larger exploit can leverage this ability to flash custom firmware [10]. This is possible because firmware updates are either often publicly available online or can be dumped from memory using cold boot attack [30]. With some off-the-shelf reverse engineering tools, attackers can customize malicious payload and disguise their attacks.

f) OPC standard: CVE-2011-3330 [5] leverages buffer overflow in the device driver of the Open Platform Communications (OPC) Factory Server, to execute arbitrary code via an unspecified system parameter. OPC Factory Server provides real time access to Schneider Electric automation controllers and SCADA connected to Ethernet networks or fieldbuses.

g) SCADA: CVE-2011-5163 [6] leverages buffer overflow in an unspecified third-party component in the batch module of SCADA, and allows local users to execute arbitrary code via a long string in a login sequence. Ironically, we also found multiple vulnerabilities in a SCADA based security manager [18], which has high privilege to access important services and files.

h) Web services: CVE-2017-12739 [8] allows unauthenticated remote attackers to leverage an integrated web server (used for diagnostics purpose) to execute arbitrary code on the affected device. This vulnerability leverages code injection CWE-94. Our study further reveals that around 20% of the reported control logic vulnerabilities are from web services, such as replay attacks via http traffic, injection of arbitrary web script or html via unspecified vectors. Web services may have internet connection, making the control logic vulnerable to worldwide attacks.

i) HMI: CVE-2017-13997 [9] allows an HMI client, e.g. InduSoft Web Studio, to trigger script execution on the server to perform customized calculations or actions. A remote malicious entity could bypass the server authentication and trigger the execution of an arbitrary command.

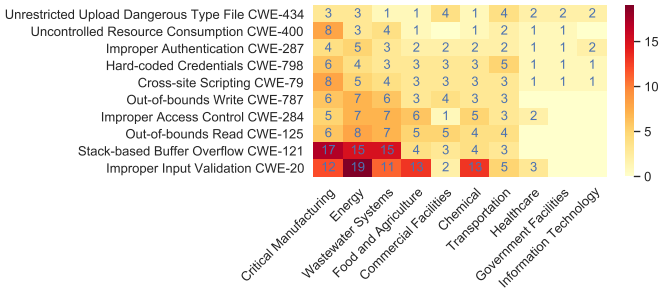

Fig. 2: The relationship between reported common weaknesses and the affected industrial sectors. The notation number denotes the number of CVEs reported.

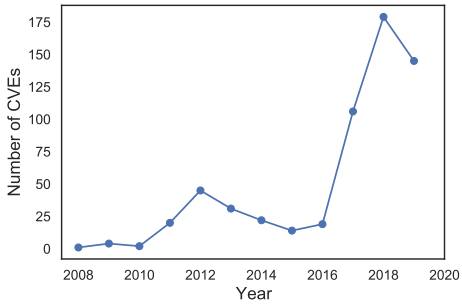

Fig. 3: Yearly reported control logic related CVEs.

j) DCS: CVE-2019-10922 allows an attacker with DCS access to execute arbitrary code, due to due to devices configured without encrypted communication [12]. CVE-2019-10918 allows an attacker with low-privileged user account access to the dcom interface, to leverage a crafted packet to execute arbitrary commands with system privileges [27].

4) Industrial sectors: We found some CWEs almost equally impact different industrial sectors, as shows in Figure 2. These weaknesses are Cross-site Scripting CWE-79, Hard-code Credentials CWE-798, Improper Authentication CWE-287, and Unrestricted Upload Dangerous Type File CWE-434.

On the other hand, we found some weaknesses are specifically targeting the top three industrial sectors, including Stack-based Buffer Overflow CWE-121 and Improper Input Validation CWE-20. We hypothesize that these industrial sectors, e.g. manufacturing, energy, and waste water system, are more motivated to embrace new technologies with complex networking and diverse vendors. A closer look at the CVEs caused by CWE-121 corroborates this, showing the sources of vulnerabilities from InduSoft Web Studio for HMI and SCADA, Proficy Historian, Micrologix Ethernet functionality, programmable controller communication commands, etc. Ironically, one of them is LAquis SCADA Smart Security Manager, which brings vulnerabilities as side products of protection.

5) Open challenges: The reported vulnerabilities pose open challenges to security mechanisms.

a) The exponential increase in the number of reported vulnerabilities: Figure 3 shows the number of control logic related CVEs on a yearly basis. In total, there are 588 different CVEs published. Before 2010, few vulnerabilities were reported each year. Starting from 2011 (e.g. the year after Stuxnet), the number of vulnerabilities increased almost exponentially, apart from a minor drop from 2013 to 2015. In 2018, the number of reported CVEs was more than 175, reaching the maximum of the past years. This demonstrates that, before 2010, control logic related vulnerabilities have long been overlooked. Given the current trend, it remains challenging to effectively reduce the number of new vulnerabilities in the next few years.

b) Expanded attack surfaces from runtime communication: The evolved design of control logic makes it inevitable to communicate with expanded attack surfaces. The vulnerabilities can come from the design flaws within (1) the PLCs, (2) the supervisory components, (3) the embedded services and functionalities, such as profile monitors, security managers, databases, etc [18], and (4) internet or cloud-based hosting and management [8], [11]. It remains an open problem to eliminate the design flaws within these factors.

c) The increased diversity in implementations: The diversity can come from vendors, product versions, and third-parties. For example, there are vendor-specific HMI products, such as InduSoft Web Studio, Advantech WebAccess, Wecon LeviStudioU, and etc. There are third-party implementations of OPC services, such as OPC DataHub. The diversity was brought by the differences in the requirements from hardware and software producers. However, the diversity amplifies the impact of single vulnerability, since one vulnerability can affect multiple vendors and product versions [5], [9]. The diversity also increases the possibility to find new vulnerabilities, due to similar design flaws. It remains challenging to balance the impact of vulnerabilities, and the requirements of diversity.

d) Insufficient adoption of formal verification: We found that half of the weaknesses are not novel—some CWEs are reported almost every year, as figure 7 shows. A lot of the vulnerabilities are due to bad coding examples, such as no authentication, string length check, or unused variables. Such mistakes can be easily prevented by formal verification of the control logic code. We believe current adoption of formal verification is insufficient in the industry.

B. Research Efforts

The research efforts are represented by existing papers presenting control logic modification attacks. To reflect the context of changing technology, these papers are categorized based on the methods to reach the control logic, and the approaches to hide malicious behaviors from detection, as Table I shows.

1) Stage: Existing works focus on modifying control logic during communication [38], [71], [91], [97], [102], [131] and runtime [32], [79], [109]–[111], [130].

TABLE I: Control logic modification research.

Detection to evade	Stage	
	Runtime	**Transmission**
Implicit specification	McLaughlin'11 CaFDI	N/A
Engineering operations	LLB Senthivel'18 HARVEY Lim'17 CLIK	CLIK
Other detections	Abbasi'16 Xiao'16	Yoo'19
None	SABOT PLCInject PLC-Blaster	Beresford'11 PLC-Blaster ICSREF

*Note: Some attacks happen in more than one stages.

2) Assumptions: To perform control logic modification attacks, existing works assume three types of threat models: (1) attackers can directly modify control logic programs in a PLC [109], [111], [130], or (2) attackers can compromise the SCADA network, or engineering station, connected to a PLC and modify the control logic programs [38], [71], [91], [97], [131], or (3) attackers can compromise the input of the PLCs, e.g. sensor measurement, and use a crafted value to deceive the control logic to behave maliciously [32], [79], [110].

Further, some works assume the attackers to have internal knowledge of the ICS, e.g. PLC vendors, software versions, facility layout, etc [109], [110]. As to the impact of the attacks, some works aim at maximizing damages to the physical plant with straightforward attacks [38], [97], [131], e.g. upload random instructions. Other works attempt to be stealthy to bypass detection through searching implicit security properties of the control logic [111], or the engineering operators [102].

3) Approaches: Since the discovery of Stuxnet attacks in 2010 [65], increasing attention has been paid to control logic modification attacks. Initially, researchers held the assumption that automated attacks on control logic are not practical without having a prior knowledge of the target physical process.

a) 2011-2013: McLaughlin [111] eliminated this assumption by proposing a prototype attack that can automatically obtain clues about the process structure and operations. They assumed they have access to the PLCs and can read the code and data memory, to find clues including the process fieldbus IDs of devices in the plant, and the safety interlocks that prevent the process from entering unsafe states. They propose to leverage existing work [68] to find implicit safety properties that can cause unsafe behaviors without alarming the systems, and then generate malicious payloads based on the findings. This attack was designed to target railway interlocking system. On a different setting, Beresford presented similar attacks on control logic using Siemens Simatic S7 [38]. Since the control logic and the engineering station communicate using ISO-TSAP packets in plain text, attackers can intercept and reverse engineer these packets. Attackers can retrieve information such as tag names, data block names, and PLC names. Attackers can also rewrite logic to the controllers based on needs, e.g. replace one tag with the other, use the

tag as the identifier for further payload injection. Beresford further pointed out that Simatic PLCs (IPC line) run x86 Linux, and an inserted payload can create shells with root permissions, at the time of writing. These attacks have the potential to cause critical damages to the physical processes. However, the chance for a successful mapping between the variables and the devices through memory probing is small. Even though a mapping is available, attackers have to further craft the malicious payload to satisfy the implicit properties, making it hard to guarantee an effective and stealthy attack.

To make these attacks more adaptive and scalable, SABOT was implemented in 2012 [109]. SABOT requires some high level descriptions of the physical plant, e.g. "the plant contains two ingredient valves and one drain valve". Such information can be acquired from public channels, and are similar for processes in the same industrial sector. With this information, a behavioral specification can be generated for plant devices, and a list of temporal logic properties can be defined. SABOT assumes access to the PLCs, so it can decompile the control logic bytecode to a logical model. SABOT uses incremental model checking to search for a mapping between a variables within the control logic, and a specified physical device. With this mapping, SABOT can compile a dynamic payload customized for the physical device, and can upload it to the victim PLC to manipulate the device. SABOT makes it realistic to attack specific devices automatically without a pre-compiled payload as in Stuxnet.

b) 2014-2016: Inspired by SABOT, a controller-aware false data injection attack [110], CaFDI, was presented in 2014. CaFDI also assumes access to some high-level information (i.e., the sensors/actuators connected to the PLCs). CaFDI can work when access to the PLCs and HMI servers or the privilege to upload malware to the PLCs are not granted. CaFDI achieves this through manipulating some compromised sensors within the substation. It first constructs a model of the control logic program from its I/O traces, and then searches for a set of inputs that may cause the model to produce the desired malicious behavior. CaFDI calculates the Cartesian product of the safe model and the unsafe model, and recursively search for a path that satisfies the unsafe model in the formalization. To stay stealthy, CaFDI avoids noticeable Boolean input, e.g. a LED indicator. Motivated by the I/O manipulations of CaFDI, Abbasi [32] leveraged the lack of hardware interrupts to the pins, and proposed configuration manipulation attacks and control-flow attacks by exploiting certain pin control operations. Xiao [136] further fine tuned the attacks to evade existing sequence-based fault detection [96]. An attacker could first construct a discrete event model from the collected fault-free I/O traces using non-deterministic autonomous automation with output (NDAAO), then build a word set of NDAAO sequences, and finally search for the undetectable false sequences from the word set to inject into the compromised sensors. These works make it practical to inject malicious payload either through a compromised SCADA network or insecure sensor configurations. It remains challenging to eliminate the dependency on the accurate knowledge of the plant to be successful. It is also challenging to build a search engine that can efficiently explore the space for stealthy malicious payloads, and skip unnecessary input vector trials.

In 2015, Klick [97] presented a tool, PLCInject, with the goal to spread control logic modifications to the network of multiple internet-facing PLCs. PLCInject targets Siemens Simatic PLCs that use the first Organization Block (OB) as the initialization of the system. Through patching this OB, PLCInject can further inject its scanner and proxy. Since Simatic PLCs have UDP based Simple Network Management Protocol (SNMP) enabled by default, the scanner uses SNMP to read the sysDesc object. PLCInject implements a SOCKS 5 proxy in a control logic because it is lightweight and can be supported by all applications. Using the same communication features, Spenneberg [131] implemented a worm, PLC-Blaster, that can spread among the PLCs. The worm spreads by replicating itself and modifying the target PLCs to execute the worm in addition to the already installed user programs. PLC-Blaster is advanced in some anti-detection mechanisms: 1) it uses the anti-replay byte in S7CommPlus protocol to avoid detection, 2) it interrupts execution every few milliseconds to meet the maximum cycle limit and continues during the next cycle, and 3) it stores at a less used OB to prevent overwriting so as to survive restart. PLCInject and PLC-Blaster successfully demonstrate the scalability of control logic modification attacks.

With all the aforementioned tools available, in recent years, attackers aim to counter against human operators at the engineering station, with more sophisticated attacks.

c) 2017-2018: Garcia [71] developed HARVEY, a PLC rootkit at the firmware level that can evade operators viewing the HMI. HARVEY assumes access to the PLC firmware. Compared to the control logic, firmware is less monitored by the operators. Firmware allows interchanging of updated values to and from the LED display on the device, and the HMI. HARVEY fakes sensor input to the control logic to generate adversarial commands, while simulates the legitimate control commands that an operator would expect to see. In this way, HARVEY can maximize the damage to the physical power equipment and cause large-scale failures, without the notice of the operators. Ladder logic bombs (LLB), presented by Govil *et.al* [79], also manipulates sensor readings while attempts to hide from manual inspection of the control logic. LLB is achieved through invoking subroutines, naming an instruction similar to a commonly used one, and storing information on a SD card. It is the first to make the attacks compatible to IEC-61131 defined languages. To make the attacks more impactful, Lim [102] demonstrated common-mode failure attacks targeting an industrial-grade (Tricon) Triple-Modular-Redundant PLC that consists of redundant modules for recovery purpose. These modules are commonly used in nuclear power plant settings. Lim used DLL hijacking to intercept and modify the command-37 packets sent between the engineering station and the PLC, and could cause all the modules to fail.

Besides hiding from engineering operators, research works also presented denial of engineering operation attacks [130]. Attackers could interfere with engineering operations aimed at updating control logic programs, in response to changing requirements. Attackers can manipulate the control logic so that the engineering station cannot process it while the program continues to execute. The goal is to seek cover for actual attacks while the operators try to understand the problem.

d) 2019: To further deceive the engineering operators, Kalle [91] presented CLIK, in 2019. CLIK employs a virtual PLC at the network level to provide a full chain of attacks by exploiting existing and zero-day vulnerabilities, e.g. in the password authentication mechanism, in the PLCs. CLIK modifies the control logic running in a remote target without access to an engineering station nor the PLCs. It can also hide the malicious modifications by providing a captured network traffic of the original (uninfected) control logic. CLIK is more realistic as a remote attack to disrupt physical processes, compared with existing work.

In addition, a recent work presented a reverse engineering attack, ICSREF [93], which can automatically generate malicious payloads, against CODESYS-based control logic. ICSREF operates directly on the native binary, and unlike SABOT [109], it does not require prior knowledge of the ICS. To make the modifications even more stealthy, Yoo presented stealthy transmission of control logic code [139] that can evade from a bump-in-the-wire formal verification. These attacks transfer control logic code as data, then split the data into small fragments (one-byte per packet), and further pad the fragment with large amount of noise data. Formal verification will fail when it cannot even detect the existence of control logic code.

4) Industrial sectors: We found only two papers explicitly mention the applicable industrial sectors for their proposed attacks. McLaughlin [111] designed the attack for railway interlocking system, and Harvey [71] evaluated the attack in a power grid test environment. Within each industrial sector, the layouts of the physical processes are similar [109]. For example, the plant contains two ingredient valves and one drain valve, or when the start button is pressed the valve for ingredient A opens. Such information are accessible in reality and can be used by attackers as prior knowledge to design domain-specific attacks.

The rest of the papers focus on certain products, with the goal to attack control systems among various industrial sectors. The two commonly tested products are Siemens Simatic S7 [38], [131] and Allen Bradley ControlLogix/CompactLogix [71], [79], [128]. These products are widely used in the industry, and offer advanced built-in features exposed with multiple vulnerabilities.

5) Open challenges: The aforementioned attacks pose open problems for security research.

a) Evasive attacks from PLC firmware: PLC firmware provides important interfaces between the control logic and the hardware, e.g. Ethernet module, web services, and is usually considered as secure because of authentication and boot-time and runtime integrity check. However, with a Joint Test Action Group (JTAG) connection to the CompactLogix PLC and recalculation of the checksum, firmware can be hijacked without being noticed [71]. Through firmware hijacking, control logic modifications can evade from detection.

b) Expanded attack surfaces from engineering workstations: With the increasing popularity of DCS, recent research in control logic modification attacks focuses on manipulate engineering operations. Through manipulating control commands sending between the engineering station and the PLC, attackers can generate dynamic and targeted payload to modify control logic programs [71]. These attacks can spread as worms to cause large scale impactful failure [97], and can evade detection of human operators [79]. Given that more vulnerabilities for DCS were reported recently [12], [27], future attacks using engineering workstations can be more complicated.

c) Implicit or incomplete specifications: Research works have shown crafted attacks using the implicit properties [109]–[111]. The difficulties of defining precise and complete specifications lie in that (1) product requirements may change over time thus require update of semantics on inputs and outputs, (2) limited expressiveness can lead to incompleteness while over expressiveness may lead to implicitness, and (3) domain-specific knowledge is usually needed Existing works have investigated automatic specification generation, e.g. PLCspecif [62], to satisfy general security requirements. However, it is still challenging to meet domain specific requirements.

d) Confidentiality and integrity of control logic input: We found multiple works leveraging control logic inputs to perform attacks. Attackers use the input information for two things: (1) monitor the input and extract high-level layout information of the physical processes [109], [110], (2) manipulate the input to some crafted values to deceive the program to produce dangerous output [32], [79]. To the best of our knowledge, there is no product protecting the confidentiality and integrity of control logic inputs.

e) Stealthiness detection: Attacks can be stealthy in several ways. To evade from deep packet inspection, attackers can perform code obfuscation [80], or hide the code in a packet through fragmentation and noise padding. Such a packet contains a small size of a code fragment with substantial padding of noises [139]. Attackers can also place code in data block, given that many PLCs do not enforce data execution prevention [139]. Formal verification fails when it cannot even detect the existence of control logic code.

IV. Formal Verification based Defenses

A. Industrial Efforts

Industrial efforts on formal verification mainly come from (1) standards that provide definitions and guidelines for best practices, and (2) products from different vendors conforming to these standards, or explicitly include formal verification as a built-in security feature.

1) Stage: At the control logic programming stage, IEC-61131 defines programming languages that are widely adopted or referenced by PLC vendors. IEC-61499 extends IEC-61131 by adding events as inputs, and supporting function blocks to interact with other controllers within DCS. These definitions facilitate formal verification of control logic code. At the transmitting stage, and during runtime communication, NIST SP 800-82 [16] provides with an overview of the typical system topology, threats and vulnerabilities, and general guidance on securing PLC, DCS, SCADA, and other control system configurations. At the execution stage, IEC-62443-3 defines formal specifications for system diagnosis, e.g. the exchange and processing of diagnostic information and the control of diagnostic processes.

2) Assumptions: The standards assume that (1) they should be general and applicable to broad areas, (2) they could motivate vendors to conform to them, and (3) the price is affordable by different types of businesses. Vendors assume that standards-compliance indicates secure systems [2], [107].

3) Approaches: Formal verification is achieved through using publicly free software, or purchasing proprietary software, or independent research and development. ITTECH provides paid proprietary software, Logic Error Hunter [22], for automated verification of PLC control logic to achieve reliability and functional safety, in the development stage. PLCverif [55] and Arcade.PLC [122] also provide formal verification of control logic code. They are originally from research projects, and are now open and free to industrial control logic programs.

In addition to the off-the-shelf formal verification tools, some standards provide formal definition for DCS or general distributed systems. IEC-18384 defines a formal ontology for service-oriented architecture (SOA) for cloud computing and distributed platforms. IEC-61588 defines a protocol for DCS with precise synchronization of clocks in the measurement and the control system. IEC-24707 defines a framework for a family of logic languages using formal interpretations of the symbols, allowing disparate computer systems to interchange information using first-order logic-based systems. ISO-8807 defines LOTOS, a formal description on the temporal ordering of observational behavior. In a broad range of applicable areas, IEC-15909 defines a Petri net modeling language, and IEC-13568 defines Z formal specification notation.

4) Industrial sectors: Each industrial sector has specific standards providing safety and security requirements. Such requirements lead to the rules in a specification. For instance, the medical industry uses IEC-60601, the nuclear industry uses IEC-61513, the avionics industry uses DO-254 and DO-178, and the automotive industry uses ISO-26262 [81].

Besides the standards, industry-specific regulations may limit the type of information that can be used by formal verification. For instance, in medical industry, the security rules of HIPPA regulation protect specific health information that is held or transferred in electronic form [31], e.g. from PLC to SCADA.

5) Open challenges: Formal verification of control logic has to satisfy many requirements, from the general security standards, industrial sector specific standards, and the regulations.

- The standards may not be entirely complete, and new versions may add more specifications. Given that IEC-62443 has a total length of more than 800 pages already, it is difficult for products to efficiently comply to the standards.
- The price may not be affordable by small businesses.
- The performance and efficiency of conforming to the standards are usually not addressed.
- Industrial sector specific regulations can limit the information for formal verification to use, and the place formal verification can happen.

B. Research Efforts

1) Stage: Existing works focus on control logic programming stage [37], [39], [56], [75], [112], [114], [117], [133], [143], [145], aiming to detect malicious code, and conform to safety and functionality requirements. The interests of formal verification span from different programming languages, specifications, and verification frameworks, evolving with the technology, standards, and requirements. Fewer works investigate code transmitting stage [67], or runtime execution stage [88], [103].

2) Assumptions: These works usually assume that (1) the behavior of control logic should satisfy general functionality and security requirements of the ICS, (2) the behavior of the physical processes should satisfy domain-specific and manufacture-specific safety requirements, or (3) the communication between control logic and other components should follow protocols correctly and securely.

Specifically, early works assume IEC-61131 defined languages, automating relatively simple operations [116], [117], [125]. Later works assume IEC-61499 defined languages with DCS [43], [48], [133], model-driven development [75], timers [75], [134], [143], multitask control logic [115], and incorporating machine learning [103]. More recent works assume interactions and communications [47], [67], multi-layer verification [114], complete verification frameworks [56], [112], [145].

3) Approaches: The approaches detail the techniques, tools, and platforms for formal verification of control logic in a chronological order. Figure 4 shows the timeline of representative research.

a) 1992-1998: In 1992, Moon [117] presented the first solution to applying formal verification on LD to automatically test the safety and operability of an alarm acknowledge system. This solution models each operation to be checked as an "assertion" in temporal logic and designs a model checker to determine if the system model satisfies the assertions. For example, whether the desired operating sequence of valves and ignitors are satisfied. The checker provides an example for each error violating the safety properties. This work assumes

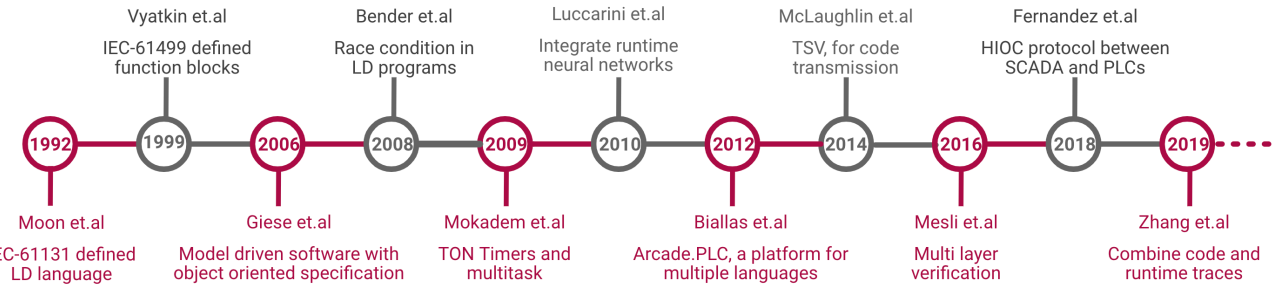

Fig. 4: The timeline of representative formal verification research.

a pure discrete system with all the operations happen sequentially. In 1994, Moon [116] extended this work to multiple alarm systems using relay ladder logic, and further analyzed the performance, e.g. CPU time and memory size of the model checker. Moon pointed out the needs for high-level language to describe assertions, and model templates of timers, counters, file handling, and etc.

Before 1999, formal verification of control logic programs focused on IEC-61131 defined SFC and LD languages [100].

b) 1999-2005: In 1999, with the emergence of IEC-61499, Vyatkin [133] provided preliminary guidelines on the formalism of IEC-61499 defined function blocks in DCS. Vyatkin expressed basic event operations of the function blocks using Net Condition/Event systems (NCES), then developed formal rules to transform function blocks to NCES, and finally implemented distributed algorithms to use these function blocks. To formally verify sophisticated timed-NCES, SESA was used. Vyatkin also outlined further steps towards the full-scale verification of IEC-61499 applications.

Since then, IEC-61499 compliant engineering software had emerged, including FBDK, CORFU, Torero and ISaGRAF [48]. However, IEC-61499 only standardized the execution model of a single function block, rather than function block networks. This resulted in the same application, consisting of multiple function blocks, to behave differently and harmful potentially, when executing in two different IEC-61499 compliant engineering software. To ensure deterministic behaviors for distributed function blocks, Cengic [48] introduced Fuber [48]. Fuber translates an IEC-61499 application to a set of interacting state automata. The automata model synthesizes function block scheduling, based on a given specification. Fuber analyzes the behavior through supervisor verification and synthesis tool Supremica.

c) 2006-2007: In 2006, model-driven software development gained attention in ICS. First, security and safety specifications are defined in object-oriented languages, e.g. Java. Then, control logic code is automatically generated based on a transformation. To formally verify the transformation, Giese [75] leveraged triple graph grammars in the Fujaba Tool Suite to specify the transformation from model to code, and then used the theorem prover Isabelle/HOL to verify the semantic equivalence between the given model and the code. Giese tested the verification based on a specification from an interlocking of railway transfer gate.

d) 2008: In 2008, to detect race conditions in real-time systems, Bender [37] investigated model checking on LD programs [37]. Race condition happens when the system attempts to perform two or more operations at the same time. This can cause security problems. In a LD program, race condition reflects as when under fixed inputs and function block states, one or more outputs keep changing values. Bender first defined timed Petri net semantics for LD programs through an ATL model transformation, and then generated behavioral properties of the LD models using LTL formulae. To detect race condition, Bender used Tina toolkit to check the timed Petri net over the generated properties.

e) 2009: In 2009, formal modeling of timers became pivotal in real-time systems. Given a sequence of stimuli to the control logic, timers decide when to produce the outcomes.It is difficult to verify timers because the input depends on its output from previous cycles. Modeling timers as symbolic function block calls, or separate timed automata, restricts the uses to guard transitions, i.e. each step needs to be associated with a timer [105]. Modeling timers as NCES can only support to start at the beginning of the calculation phase [133]. Modeling timers with theorem system Isabelle/HOL [75] assumes the current value to increase monotonously without reset actions in this process. Wan [134] presented a formalization of Timers ON delay (TON) using the theorem proving system in Coq. Coq is expressive to model TON-timers at different abstract levels for different resolutions. Coq allows to reuse specifications (not directed supported by model checking), and prove parameterized properties. Wan tested the behavior of a timed quiz machine, by describing the behavior of TON-timer as a set of axioms, and assuming a function f to map the start of each scan cycle to its time point. This method ensured consistent execution time for each instruction, consistent TON-timer values within one scan cycle, and avoided loops in one scan cycle.

Mokadem extended formal verification of TON timers to multitask control logic [115]. The multitask control logic is part of the Mechatronic Standard System platform from Bosch Group, consisting of a linear conveyor, and an event-driven detector for the press-fit and present of material. The safety properties depend on accurate reaction time, for example, the conveyor must stop on its way out but not after unloading, and the conveyor stops in less than 5 ms. Mokadem modeled TON timers as network of timed automata, constrained

with atomicity hypotheses concerning program execution. The timed automata was first described in SFC and then translated to LD. Mokadem used UPPAAL to check the reaction time between the detection of a signal and the resulting event.

f) 2010-2011: In 2010, to achieve fault detection and isolation in the physical processes, Luccarini [103] presented a modular architecture using artificial neural networks to aid formal verification of runtime control logic, in a waste water treatment plant. Neural networks were used to extract qualitative patterns from the continuous signals of the water, e.g. pH, dissolved oxygen. These qualitative patterns were then mapped to the control events in the physical processes. Luccarini logged the mapping using XML format, translates it into formal rules, and used SCIFF checker to validate the processes against these rules. This work demonstrated the feasibility of neural network aided formal verification for runtime control logic execution.

g) 2012-2013: Before 2012, formal verification of control logic either required a transformation or was limited to a subset of languages or language constructs [46], [78], [117], [123], [125]. In 2012, Biallas implemented Arcade.PLC [39], a verification platform independent from implementation languages. Arcade.PLC translates different languages into an intermediate representation, and provides a generic interface for low-level analysis. It implements different orthogonal approaches to account for the specific hardware platform and its cyclic scanning mode. It supports static analysis, $\forall$CTL and past-time LTL model checking, using counterexample-guided abstraction refinement. Arcade.PLC can verify programs involving complex control flow and heavy interaction with the environment.

h) 2014-2015: In 2014, Mclaughlin [112] realized the need of a minimal trusted computing base for the verification of safety-critical control logic code. Mclaughlin designed and implemented a bump-in-the-wire formal verification platform, Trusted Safety Verifier (TSV), sitting between the PLC and the engineering software. TSV is superior in that it can analyze the binary directly without using source code, and handle instruction set architectures from different vendors. Since PLCs support features different from normal instruction sets, Instruction List Intermediate Language (ILIL) was designed for binary analysis. TSV first translates controller assembly into ILIL, and then combines symbolic execution and model checking to generate Temporal Execution Graphs (TEG) from ILIL. Z3 theorem prover was used to check path feasibility during symbolic execution, and simplified symbolic variable values during TEG generation. TSV uses NuSMV for model checking of the TEG, and verifies safety properties including bounds on numerical device parameters, e.g., maximum drive velocity and acceleration, and safety interlocks, to make sure physically conflicting events do not occur. In the same year, Zonouz [145] extended the work with a cyber-physical security intrusion detection engine, which identifies malicious control logic code injection.

In 2015, to simplify formal verification for users, including model construction, model reduction and requirement formalization, Darvas [56] designed and implemented PLCverif [56], an open-source framework for control logic formal verification. Before PLCverif, Arcade.PLC is the only publicly available tool, but only supports CTL or LTL defined requirements for model checking. Initially, PLCverif supports the verification of ST code, and can extend to other languages defined in IEC-61131, with new model checkers. Later, re-engineered PLCverif [55] further supported native STL code with formal assertion violation checking, and integrated CBMC model checker for C programs, and Theta framework for abstraction refinement.

i) 2016-2017: In 2016, Mesli [114] first realized the need of multi-layer formal verification of control logic. Mesli proposed joint verification of supervision interfaces (e.g., HMI) and the controller ladder programs. In this work, all the components along the control-command chain are modeled as timed automata, and checked by UPPAAL with a set of safety and usability properties written in CTL. An industrial case showed that errors can be successfully detected in the control programs and the supervision interfaces.

j) 2018: With the increasing need of runtime communication, formal verification of communication protocols became critical. Fernandez [67] first proposed to formally verify the execution of the High Integrity Operator Commands (HIOC) protocol. This protocol dedicates to the communication between PLC and SCADA. Before that, formal verification concentrated on general communication protocols using ProVerif [41], [124]. Fernandez verified each step of the HIOC protocol, with the goal to ensure the critical commands sent from SCADA are properly received by the PLCs, and the requirements of IEC-61508 are satisfied. HIOC was implemented in SCL. Fernandez performed formal verification using PLCverif [56], and found suspicious corner cases fail to match the specification.

k) 2019: Zhang [143] presented VETPLC, the first solution to verify timing-aware safety on runtime event-driven control logic. Before VETPLC, research efforts on static code verification failed to reveal runtime events, while dynamic analyses and symbolic execution failed to handle timing sensitive control logic. VETPLC combines static code and runtime traces to verify safety properties. Zhang generated Timed Event Causality Graphs (TECG) from the control logic code, and mined temporal invariants from runtime data traces to gauge temporal dependencies constrained by machine operations. With the generated TECG, VETPLC performs permutations to create timed event sequences, and applies to execute the control logic. Based on the formalized specifications from domain experts, VETPLC performs runtime safety vetting and finds hidden safety violations in two real-world testbeds.

4) Industrial sectors: Multiple papers have investigated domain-specific tasks and requirements. Some papers used the same or similar verification techniques, but to overcome the challenges in specification inconsistencies and availability. Domain experts usually have less expertise in formal verification, therefore it is challenging to generate accurate

specification [94], [141]. Even through the specification is accurate, sometimes only a certain format is available [119].

In nuclear industry, to bridge the gap between nuclear engineers and formal verification experts, the Korean Nuclear Instrumentation and Control System consortium (KNICS) developed a set of CASE tools [141], for easy verification of FBD programs in nuclear reactor protection systems. CASE integrates model checking and equivalence checking, to verify the timing properties and regulation requirements. In 2017, KNICS proposed Nuclear Development Environment (Nude) 2.0 for formal-method-based software development, verification and safety analysis [94]. Nude 2.0 integrates twenty five CASE tools, which can automatically develop FBD programs from defined specifications, and ensure safety properties to be equivalent to the original specification.

In 2018, the Darlington Nuclear Power Generating Station requested a rigorous verification of the FBD control logic automating the power generation [119] The quality assurance program requires the specification to be written in tabular expressions. Newell [119] presented a method to rigorously translate FBD language to tabular expressions and prove with a mathematical model in PVS theorem proving tool.

5) Open challenges:

a) Customized and inconsistent specification requirements: Requirement differences across industries and vendors increase the effort of formal modeling, and limit scalability. In water treatment area, the functional and safety properties are highly distinct from the control tables specifying the rules for the railway automation industry. Even within railway automation industry, the rules are affected by local regulations. This results in specifications to be expressed in different types of languages using various domain expertise, making it difficult for formal verification practitioners.

b) Lack of verified specification translator: In most of safety-critical systems, e.g. nuclear systems, the software specification properties are first written in natural languages, and then modeled with control logic languages such as FBD or LD. PLC vendors provide tools to translate FBD/LD programs into C programs and executable codes for specific PLCs. However, the nuclear industry has not acknowledged empirically the correctness and safety of a compiler that is capable to translate FBD to C. Similarly, translation to an intermediate language blurred the defined properties. Verification of property translators remains challenging.

c) Accurate and efficient abstraction: Model checking on continuous features (e.g. time) is expensive, while reduction and abstraction methods (e.g. Uppaal) can lose details of the original code and are vulnerable to attacks. Choosing an appropriate discrete abstraction for continuous behavior is still an ad-hoc process. If the chosen abstraction is too coarse, then the analysis may produce infeasible attack scenarios; on the other hand, if the abstraction is unsound, then it may fail to detect actual attacks on the system. Even though researchers have been investigating the construction of suitable abstractions, coming up with a "ground truth" model for the continuous behavior remains challenging.

d) Synchronization with evolved system design: Unsynchronized execution can cause nondeterministic transitions to malfunction or unsafe states. require each state transition in the composite SMV model to accurately correspond to one scan of the PLC program [125]. When interconnected function blocks are executing in an event driven fashion, synchronization extends to the combination of input and output data and signals, and execution mode (e.g. enabled, disabled), etc [43]. Since control logic interacts with other components, synchronization between them is also needed. It is challenging to ensure every state transition among all the components and their functionalities, considering the latest PLC design.

e) Insufficient research on subroutine, interrupts, and multitask: Very few works can support extensive instructions, including time-related instructions, such as timer and counter, and the subroutine and interruption instructions, in a multitask fashion. Present solutions to support these features are targeting small and medium sized PLC programs. When the scale and complexity increases, it becomes challenging to overcome state explosion, without sacrificing accuracy or posing higher demands on the hardware.

f) Runtime formal verification: The challenges include (1) fault injection from the input sensors can only be detected at runtime [110], (2) complex networking with expanded attack landscapes, (3) real-time constraints, (4) synchronization between control logic and other components [143], and (5) limited resources are available on PLC, while powerful supervisory components require secure communication.

g) Trend of cloud-hosted control logic management: The rise of Industry 4.0 facilitates cloud-hosted control logic management, which allows elastic functionality, scalability and multi-tenancy. One application can be partially within control logic, and partially in the cloud. For example, Beckhoff PLCs proposed to integrate voice recognition skills using AWS IoT [77]. Goldschmidt implemented a prototype for a cloud-based software PLC serving as a Control-as-a-Service solution [76]. Such applications will inevitably need control logic to communicate using OPC, Web, SQL, and other services. The confidentiality and integrity of control logic and input/output information will entirely depend on the cloud. Existing research on formal verification-based defenses is insufficient to adapt to cloud-hosted control logic management.

V. Recommendations

We have highlighted the security challenges in defending control logic modification attacks, and formal verification-based protection. Next, we offer recommendations to overcome these challenges.

A. Vendors

We recommend vendors to prioritize security, and standardize existing ad-hoc, vendor-specific implementations.

1) Development toolings: We recommend vendors to prioritize security of the development toolings. Vulnerabilities in these toolings [4] have invalidated user efforts in securing programs. First, we recommend vendors to standardize the

built-in features of the programming environments. Existing vendor-specific features, such as simulation, virtualization, "all inclusive", tend to be ad-hoc, and pose challenges to build effective formal verification solutions. Second, We recommend vendors to provide development toolings that support code formal verification as a default feature. This eases the work of users to prevent common design flaws.

2) Code transmissions: We recommend vendors to pay more attention to secure code transmission, since existing works have shown multiple vulnerabilities and attacks modifying control logic in this stage. We recommend standardization of protocols dedicated to code transmission, including the commonly used Ethernet, RS-232, USB, etc. We also recommend vendors to prepare for emerging internet-facing code transmissions, with standardized security solutions to defend against possible man-in-the-middle, denial-of-service, and other stealthy modification attacks.

3) Runtime communication: We recommend vendors to prioritize security during runtime communication. We recommend vendors to adopt existing formal verification solutions, to implement secure (1) PLC firmware, which embeds Web, SQL, and other services, (2) supervisory components, which send commands and exchange data with control logic, and (3) real-time communication protocols, such as Modbus, EtherCat, Profinet, etc. Some protocols are already standardized by design, for example, EtherCat is standardized in IEC-61158. However, different implementations, especially untrusted third-party implementations made them vulnerable. We recommend standardized implementations to eliminate possible vulnerabilities resulting from uncertain implementations. Faced with internet connected PLCs, we recommend vendors to follow IIoT standards, such as Data Distribution Service (DDS), and latest IIoT guidelines from industrial groups, such as PLCOpen, OPC Foundation, OMAC. In addition, we recommend open source implementation of the standards to improve security.

B. Users

The users include control logic developers, and operators, who are domain engineers, and formal verification experts, who do not have sufficient domain knowledge.

1) Program design and implementation: We recommend developers to program with standard languages, e.g. IEC-61131 or IEC-61499 defined, so that abundant formal verification solutions will be available. We also recommend to avoid using third-party ad-hoc development toolings, which have led to multiple control logic modification attacks. We also recommend control logic developers to explicitly specify parameters during programming, since multiple reported vulnerabilities are due to unspecified system parameters [26], unspecified/unverified length of parameters [14], and other unspecified vectors.

2) Adopting formal verification: We recommend control logic developers to adopt formal verification, such as PLCverif [55], Arcade.PLC [39], ITTECH [22], and etc [122]. These solutions are dedicated to control logic, and have demonstrated to have enhanced code quality [54], [121]. We recommend operators to adopt protocol-specific formal verification, such as Fernandez's work on IEC-61508 satisfied HIOC communication protocol, the AVANTSSAR validation platform [3], [34] aims at large-scale Internet security-sensitive protocols and applications, and other solutions [124] based on ProVerif [41]. We recommend formal verification experts to explicitly specify properties in the specification, and check equivalence between the specification and its previous versions or legacy code. We also recommend formal verification experts to leverage existing well-defined models, and conform to domain-specific specifications, as described in Section IV-A

3) Developing bump-in-the-wire solutions: We recommend users to reduce dependency on vendors in securing code transmission. Specifically, we recommend users to develop own bump-in-the-wire solutions, which sit between the system operators and the PLCs [112], and are independent from the supervisory software and hardware. We recommend users to integrate formal verification to these solutions, by intercepting every piece of transmitted control logic code and checking with pre-defined security properties.

4) Leveraging embedded hypervisors for runtime verification: Within the PLC, only limited memory is available for runtime formal verification. On a SCADA, memory is abundant, but runtime data traces have to transfer on a possibly insecure network. We recommend users to leverage PLC-embedded hypervisors or coprocessors to perform runtime formal verification. Such embedded hypervisors should have shared memory with a PLC, allowing programming blocks to be integrated synchronously or asynchronously, and enabling direct memory access to the inputs and outputs of the PLC [44], [74]. We recommend users to carefully control memory accesses, and dedicate this hypervisor to runtime formal verification.

C. Security Researchers

Next, we provide recommendations for future research directions. In general, we highlight the need of a full-chain protection of control logic.

1) Programming and specification: Based on the fact that, at the programming stage, extensive research on formal verification has been performed, and made fruitful progress, we recommend future research to focus on open subdivision. Specifically, we recommend the investigation of an intermediate representation based on mixed programming languages, such as a function written in IL calling an ST function [59]. The intermediate representation should minimize the loss of useful information, while consider heuristics with the increase of program sizes. We recommend the investigation of formally verified specification translators. Such translators should generalize the requirements from various domains, and support incremental model checking based on domain-specific requirements. We also recommend the investigation of eliminating implicit properties. This should be based on explicit-state model checking that maintains all state transitions. To minimize the search space, researchers should con-

sider context-bounded model checking, which uses abstraction techniques and partial-order reduction algorithms [53], [86].

2) Multitask control logic: The structure of multitask control logic are temporal parallelism and spatial parallelism, based on the hardware characteristics. For temporal paralleled multitask control logic, we recommend researchers to extend investigation on formal verification of TON timers. We hope future research to relax the following assumptions: (1) timer values will increase monotonously without reset actions, (2) executions of the same instruction take the same time, and (3) within each scan cycle, the values of a TON-timer used by several instructions are the same. For spatial paralleled multitask control logic, formal verification is a new and challenging research area. We recommend researchers to investigate formal verification solutions considering task priority and execution interval [73].

3) Detecting stealthy code transmission: Stealthy attacks have shown to hide control logic code during transmission, making bump-in-the-wire solutions, such as TSV [112], fail to detect the existence of code [139]. We suggest security researchers to investigate protocol-specific DPI to detect such stealthy attacks. For commonly used code transmission protocols, researchers should inspect the protocol structure, the sequence of packets, and the expected byte values. Researchers should investigate existing machine learning based de-obfuscation techniques. For example, training SVM algorithms to understand the semantics of packets, and further predict the layout of the code [140].

4) Firmware formal verification: Malicious firmware has demonstrated to modify control logic during runtime. We recommend security researchers to investigate firmware verification before and after execution. This will need the PLC to co-locate with a trusted resource abundant component (e.g. SCADA, embedded hypervisor, etc). First, firmware uploads to a PLC should be captured and formally verified against a known benign version of the firmware executable. After the firmware is loaded, the resource powerful component should monitor its runtime behavior, including I/O access patterns, and network access patterns. These runtime behaviors should further be analyzed, based on pre-defined formal models, to detect possible violation of security properties [113], [138].

5) Control logic synchronization: Attacks have shown to leverage unsynchronized execution to modify control logic, while bypassing formally defined properties [47]. We recommend security researchers to investigate synchronization as a separate problem, and build formal models to represent expanded interacting components and services. For example, researchers should consider synchronizing the combination of input and output data and signals, execution mode, with interconnected control logic, in an event driven fashion. Researchers should also consider runtime communication with SCADA or DCS. We recommend modeling synchronization properties as server/client states, to efficiently represent all the components and alleviate state explosion problem [51].

6) Input confidentiality and integrity: Advance in IIoT enables cloud-hosted SCADA and DCS to monitor control logic, and store the monitored input and output data in the (potentially insecure) cloud. Attacks have shown to use the data either to acquire the high-level layout information of the ICS, or to craft special payload to bypass formal verification [109], [110]. We recommend security researchers to ensure data confidentiality and integrity, through investigation of SGX-enabled SCADA/DCS [49]. Specifically, we recommend researchers to consider assigning a secure enclave for every input source, e.g. a sensor, an actuator, to prevent data leakage or pollution due to a hijacked malicious sensor. In the meantime, a formal verification framework should be able to access the input data from all the sources, and prevent side-channel attacks through profiling memory reads and writes. We also recommend researchers to consider multitask control logic in this scenario, and investigate whether the performance overhead could satisfy the time constraints of a scan cycle.

7) Incorporating machine learning for runtime verification: Formal verification on runtime control logic is insufficiently researched. The major difficulty is specification generation and maintenance, given the expanded control logic interactions, with a complex networking. We recommend security researchers to leverage the power of machine learning, and investigate automatic specification generation, based on flexible threat models, and specification refinement with optimized policies and parameters [63], [95]. Specifically, researchers should first monitor legit runtime traces [52], [66], such as time series based I/O and network accesses. Next, researchers should perform legit trace mutation, to serve as counter examples simulating malicious payload or stealthy traffic. The mutation should consider state estimations to reduce the potential search space [73]. Later, researchers should train recurrent neural networks, e.g. LSTM, using the time series data, and generate specifications based on the neural networks. This investigation will mitigate previous constraints from domain expertise, and fast changing industry requirements.

VI. Conclusion

This paper investigated control logic injection and modification attacks, and formal verification based defense solutions among various industrial sectors. We found the attacks are driven my emerging technologies with imperfect design, for example, the inclusion of web services, SQL and email services exposes control logic to broaden attack surfaces. It will not be surprising to see such attacks in the near future. We found formal verification has been used to defense control logic since nearly thirty years ago, and is continuously evolving with new technologies and industrial standards. Even though such defenses have made progress in improving code quality, we believe device/component-driven (e.g. PLC, HMI, SCADA) protection is not enough. Control logic needs a full chain of protection—from programming in the engineering software, transmitting to a PLC, to executing in a PLC. Formal methods alone, nevertheless, have seen challenges in control logic communication and runtime verification, because of the complex networking, and limited resources in the PLCs. We therefore recommend best practices in leveraging formal

verification-based defenses to mitigate control logic attacks, and urge future works to complement formal methods with other defense technologies, e.g. neural networks, to fulfill a full chain of protection.

REFERENCES

[1] 2019 CWE Top 25 Most Dangerous Software Errors. https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html.

[2] Achieving strong industrial security. https://www.infineon.com/cms/en/product/promopages/iec62443/.

[3] AVISPA - Automated Validation of Internet Security Protocols and Applications. . http://www.avispa-project.org/.

[4] CVE-2010-5305. https://nvd.nist.gov/vuln/detail/CVE-2010-5305.

[5] CVE-2011-3330. https://nvd.nist.gov/vuln/detail/CVE-2011-3330.

[6] CVE-2011-5163 Detail. https://nvd.nist.gov/vuln/detail/CVE-2011-5163.

[7] CVE-2017-12088. https://nvd.nist.gov/vuln/detail/CVE-2017-12088.

[8] CVE-2017-12739. https://nvd.nist.gov/vuln/detail/CVE-2017-12739.

[9] CVE-2017-13997. https://nvd.nist.gov/vuln/detail/CVE-2017-13997.

[10] CVE-2017-14468 Detail. https://nvd.nist.gov/vuln/detail/CVE-2017-14468.

[11] CVE-2018-10619. https://nvd.nist.gov/vuln/detail/CVE-2018-10619.

[12] CVE-2019-10922 Detail. https://nvd.nist.gov/vuln/detail/CVE-2019-10922.

[13] CVE-2019-10929. https://nvd.nist.gov/vuln/detail/CVE-2019-10929.

[14] Delta Electronics Delta Industrial Automation COMMGR. https://www.us-cert.gov/ics/advisories/ICSA-18-172-01.

[15] Formal Verification. https://en.wikipedia.org/wiki/Formal_verification.

[16] Guide to Industrial Control Systems (ICS) Security . https://csrc.nist.gov/publications/detail/sp/800-82/rev-2/final.

[17] GX Developer. https://us.mitsubishielectric.com/fa/en/products/controllers/programmable-controllers-melsec/engineering-software/other-engineering-softwares/gx-developer/gx-developer.

[18] ICS Advisory (ICSA-18-289-01) . https://www.us-cert.gov/ics/advisories/ICSA-18-289-01.

[19] ICS CERT. https://www.us-cert.gov/ics.

[20] National Vulnerability Database. https://nvd.nist.gov/vuln/data-feeds.

[21] Offensive Security. https://www.exploit-db.com.

[22] PLC Logic Error Hunter. http://www.ittech-automation.com.au/downloads.html.

[23] PLC vs. Scada. https://www.onupkeep.com/answers/asset-management/plc-vs-scada/.

[24] RSLogix 5000. https://www.mathworks.com/products/connections/product_detail/rslogix-5000.html.

[25] RSView32. https://www.rockwellautomation.com/rockwellsoftware/products/rsview32.page.

[26] Schneider Electric UnitelWay Buffer Overflow. https://www.us-cert.gov/ics/advisories/ICSA-11-277-01.

[27] Siemens SIMATIC PCS7, WinCC, TIA Portal (Update D). https://www.us-cert.gov/ics/advisories/ICSA-19-134-08.

[28] Simatic S5 PLC. https://en.wikipedia.org/wiki/Simatic_S5_PLC.

[29] TALOS-2017-0443. https://talosintelligence.com/vulnerability_reports/TALOS-2017-0443.

[30] Undocumented Access Feature Exposes Siemens PLCs to Attacks. https://www.securityweek.com/undocumented-access-feature-exposes-siemens-plcs-attacks.

[31] What is HIPAA Compliance? 2019 HIPAA Requirements. https://digitalguardian.com/blog/what-hipaa-compliance.

[32] Ali Abbasi and Majid Hashemi. Ghost in the PLC designing an undetectable programmable logic controller rootkit via pin control attack. *Black Hat Europe*, 2016:1–35, 2016.

[33] Borja Fernandez Adiego, Dániel Darvas, Enrique Blanco Viñuela, Jean-Charles Tournier, Simon Bliudze, Jan Olaf Blech, and Víctor Manuel González Suárez. Applying model checking to industrial-sized PLC programs. 11(6):1400–1410, 2015.

[34] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, P Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, et al. The AVISPA tool for the automated validation of internet security protocols and applications. In *International conference on computer aided verification*, pages 281–285. Springer, 2005.

[35] Michael J Assante. Confirmation of a coordinated attack on the ukrainian power grid. *SANS Industrial Control Systems Security Blog*, 207, 2016.

[36] Nanette Bauer, Sebastian Engell, Ralf Huuck, Sven Lohmann, Ben Lukoschus, Manuel Remelhe, and Olaf Stursberg. Verification of PLC programs given as sequential function charts. In *Integration of software specification techniques for applications in Engineering*, pages 517–540. Springer, 2004.

[37] Darlam Fabio Bender, Benoît Combemale, Xavier Crégut, Jean Marie Farines, Bernard Berthomieu, and François Vernadat. Ladder metamodeling and PLC program validation through time Petri nets. In *European Conference on Model Driven Architecture-Foundations and Applications*, pages 121–136. Springer, 2008.

[38] Dillon Beresford. Exploiting siemens simatic s7 plcs. *Black Hat USA*, 16(2):723–733, 2011.

[39] Sebastian Biallas, Jörg Brauer, and Stefan Kowalewski. Arcade. PLC: A verification platform for programmable logic controllers. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 338–341. IEEE, 2012.

[40] Sidi Ould Biha. A formal semantics of PLC programs in Coq. In *2011 IEEE 35th Annual Computer Software and Applications Conference*, pages 118–127. IEEE, 2011.

[41] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. Proverif 2.00: automatic cryptographic protocol verifier, user manual and tutorial. *Version from*, pages 05–16, updated 2020.

[42] Dimitri Bohlender and Stefan Kowalewski. Compositional verification of PLC software using horn clauses and mode abstraction. *IFAC-PapersOnLine*, 51(7):428–433, 2018.

[43] Marcello Bonfe and Cesare Fantuzzi. Design and verification of mechatronic object-oriented models for industrial control systems. In *EFTA 2003. 2003 IEEE Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No. 03TH8696)*, volume 2, pages 253–260. IEEE, 2003.

[44] Allen Bradley. 1771 Control Coprocessor User Manual. https://literature.rockwellautomation.com/idc/groups/literature/documents/um/1771-um676_-en-p.pdf.

[45] Ed Brinksma and Angelika Mader. Verification and optimization of a PLC control schedule. In *International SPIN Workshop on Model Checking of Software*, pages 73–92. Springer, 2000.

[46] Géraud Canet, Sandrine Couffin, J-J Lesage, Antoine Petit, and Philippe Schnoebelen. Towards the automatic verification of PLC programs written in Instruction List. volume 4, pages 2449–2454. IEEE, 2000.

[47] Henrik Carlsson, Bo Svensson, Fredrik Danielsson, and Bengt Lennartson. Methods for reliable simulation-based PLC code verification. *IEEE Transactions on Industrial Informatics*, 8(2):267–278, 2012.

[48] Goran Cengic, Oscar Ljungkrantz, and Knut Akesson. Formal modeling of function block applications running in IEC 61499 execution runtime. In *2006 IEEE Conference on Emerging Technologies and Factory Automation*, pages 1269–1276. IEEE, 2006.

[49] Gianfranco Cerullo, Giovanni Mazzeo, Gaetano Papale, Luigi Sgaglione, and Rosario Cristaldi. A secure cloud-based scada application: The use case of a water supply network. In *SoMeT*, pages 291–301, 2016.

[50] Simon Chadwick, Phillip James, Markus Roggenbach, and Tom Wetner. Formal Methods for Industrial Interlocking Verification. In *2018 International Conference on Intelligent Rail Transportation (ICIRT)*, pages 1–5. IEEE, 2018.

[51] Minho Chang, Kwan Hee Han, Jong Geun Kwak, Gi Nam Wang, SC Park, SM Bajimaya, and Chang Mok Park. Development of virtual simulator for visual validation of PLC program. In *2006 International Conference on Computational Inteligence for Modelling Control and Automation*, pages 32–32. IEEE, 2006.

[52] Yuqi Chen, Christopher M Poskitt, and Jun Sun. Learning from mutants: Using code mutation to learn and monitor invariants of a cyber-physical system. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 648–660. IEEE, 2018.

[53] Zuxi Chen, Zhongwei Xu, Junwei Du, Meng Mei, and Jing Guo. Efficient encoding for bounded model checking of timed automata. *IEEJ Transactions on Electrical and Electronic Engineering*, 12(5):710–720, 2017.

[54] Stephen Chong, Joshua Guttman, Anupam Datta, Andrew Myers, Benjamin Pierce, Patrick Schaumont, Tim Sherwood, and Nickolai Zeldovich. Report on the NSF workshop on formal methods for security. *arXiv preprint arXiv:1608.00678*, 2016.

[55] D Darvas, E Blanco, and Switzerland V Molnár. PLCverif Re-engineered: An Open Platform for the Formal Analysis of PLC Programs. ICALEPCS.

[56] Dániel Darvas, Enrique Blanco Vinuela, and Borja Fernández Adiego. Plcverif: A tool to verify plc programs based on model checking techniques. 2015.

[57] Dániel Darvas, Enrique Blanco Vinuela, and István Majzik. A formal specification method for PLC-based applications. 2015.

[58] Daniel Darvas, Enrique Blanco Viñuela, and Istvan Majzik. What is special about PLC software model checking? 2018.

[59] Dániel Darvas, István Majzik, and Enrique Blanco Viñuela. Generic representation of PLC programming languages for formal verification. In *Proc. of the 23rd PhD Mini-Symposium*, pages 6–9.

[60] Dániel Darvas, István Majzik, and Enrique Blanco Viñuela. Formal verification of safety PLC based control software. In *International Conference on Integrated Formal Methods*, pages 508–522. Springer, 2016.

[61] Dániel Darvas, István Majzik, and Enrique Blanco Viñuela. PLC program translation for verification purposes. *Periodica Polytechnica Electrical Engineering and Computer Science*, 61(2):151–165, 2017.

[62] Dániel Darvas, Enrique Blanco Viñuela, and István Majzik. PLC code generation based on a formal specification language. In *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, pages 389–396. IEEE, 2016.

[63] Ankush Desai, Tommaso Dreossi, and Sanjit A Seshia. Combining model checking and runtime verification for safe robotics. In *International Conference on Runtime Verification*, pages 172–189. Springer, 2017.

[64] Alessandro Di Pinto, Younes Dragoni, and Andrea Carcano. Triton: The first ics cyber attack on safety instrument systems. In *Proc. Black Hat USA*, pages 1–26, 2018.

[65] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6):29, 2011.

[66] Cheng Feng, Venkata Reddy Palleti, Aditya Mathur, and Deeph Chana. A systematic framework to generate invariants for anomaly detection in industrial control systems. In *NDSS*, 2019.

[67] Borja Fernández Adiego, Enrique Blanco Viñuela, Bhimavarapu Avinashkrishna, Sailaraj Sreekuttan, Riccardo Pedica, Gyula Sallai, Yogesh Gaikwad, Daniel Darvas, Gisik Lee, and Ignacio Prieto Diaz. Applying model checking to critical PLC applications: An ITER case study. *IEEE Transactions on Industrial Informatics*, 2018.

[68] Alessio Ferrari, Gianluca Magnani, Daniele Grasso, and Alessandro Fantechi. Model checking interlocking control tables. In *FORMS/FORMAT 2010*, pages 107–115. Springer, 2011.

[69] Georg Frey and Lothar Litz. Formal methods in PLC programming. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, volume 4, pages 2431–2436. IEEE, 2000.

[70] Joel Galvão, Cedrico Oliveira, Helena Lopes, and Laura Tiainen. Formal verification: Focused on the verification using a plant model. In *International Conference on Innovation, Engineering and Entrepreneurship*, pages 124–131. Springer, 2018.

[71] Luis Garcia, Ferdinand Brasser, Mehmet Hazar Cintuglu, Ahmad-Reza Sadeghi, Osama A Mohammed, and Saman A Zonouz. Hey, My Malware Knows Physics! Attacking PLCs with Physical Model Aware Rootkit. In *NDSS*, 2017.

[72] Luis Garcia, Stefan Mitsch, and André Platzer. HyPLC: Hybrid programmable logic controller program translation for verification. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 47–56, 2019.

[73] Luis Garcia, Stefan Mitsch, and André Platzer. Toward multi-task support and security analyses in plc program translation for verification. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 348–349, 2019.

[74] Luis Garcia, Saman Zonouz, Dong Wei, and Leandro Pfleger De Aguiar. Detecting PLC control corruption via on-device runtime verification. In *2016 Resilience Week (RWS)*, pages 67–72. IEEE, 2016.

[75] Holger Giese, Sabine Glesner, Johannes Leitner, Wilhelm Schäfer, and Robert Wagner. Towards verified model transformations. In *Proc. of the 3rd International Workshop on Model Development, Validation and Verification (MoDeV 2a), Genova, Italy*, pages 78–93. Citeseer, 2006.

[76] Thomas Goldschmidt, Mahesh Kumar Murugaiah, Christian Sonntag, Bastian Schlich, Sebastian Biallas, and Peter Weber. Cloud-based control: A multi-tenant, horizontally scalable soft-plc. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 909–916. IEEE, 2015.

[77] Sven Goldstein. PLC to Cloud: Changing industrial automation with AWS. http://aws-de-media.s3-eu-west-1.amazonaws.com/images/AWS_Summit_Berlin_2017/Presentations/NGA2-7_Beckhoff_Sven_Goldstein_PLC_to_Cloud.pdf.

[78] Vincent Gourcuff, Olivier De Smet, and J-M Faure. Efficient representation for formal verification of PLC programs. In *2006 8th International Workshop on Discrete Event Systems*, pages 182–187. IEEE, 2006.

[79] Naman Govil, Anand Agrawal, and Nils Ole Tippenhauer. On ladder logic bombs in industrial control systems. In *Computer Security*, pages 110–126. Springer, 2017.

[80] Ravish Goyal, Suren Sharma, Savitri Bevinakoppa, and Paul Watters. Obfuscation of stuxnet and flame malware. *Latest Trends in Applied Informatics and Computing*, 150:154, 2012.

[81] Tomás Grimm, Djones Lettnin, and Michael Hübner. A survey on formal verification techniques for safety-critical systems-on-chip. *Electronics*, 7(6):81, 2018.

[82] Muluken Hailesellasie and Syed Rafay Hasan. Intrusion Detection in PLC-Based Industrial Control Systems Using Formal Verification Approach in Conjunction with Graphs. *Journal of Hardware and Systems Security*, 2(1):1–14, 2018.

[83] Joseph Y Halpern and Moshe Y Vardi. Model checking vs. theorem proving: a manifesto. *Artificial intelligence and mathematical theory of computation*, 212:151–176, 1991.

[84] Nannan He, Victor Oke, and Gale Allen. Model-based verification of PLC programs using Simulink design. In *2016 IEEE International Conference on Electro Information Technology (EIT)*, pages 0211–0216. IEEE, 2016.

[85] Daavid Hentunen and Antti Tikkanen. Havex hunts for ics/scada systems. In *F-Secure*. 2014.

[86] Gerard J Holzmann. Explicit-state model checking. In *Handbook of Model Checking*, pages 153–171. Springer, 2018.

[87] Yanhong Huang, Xiangxing Bu, Gang Zhu, Xin Ye, Xiaoran Zhu, and Jianqi Shi. KST: Executable Formal Semantics of IEC 61131-3 Structured Text for Verification. *IEEE Access*, 7:14593–14602, 2019.

[88] Helge Janicke, Andrew Nicholson, Stuart Webber, and Antonio Cau. Runtime-monitoring for industrial control systems. *Electronics*, 4(4):995–1017, 2015.

[89] Fredrik Johansson. Attacking the Manufacturing Execution System: Leveraging a Programmable Logic Controller on the Shop Floor, 2019.

[90] Timothy L Johnson. Improving automation software dependability: A role for formal methods? *Control engineering practice*, 15(11):1403–1415, 2007.

[91] Sushma Kalle, Nehal Ameen, Hyunguk Yoo, and Irfan Ahmed. CLIK on PLCs! Attacking control logic with decompilation and virtual PLC. In *Binary Analysis Research (BAR) Workshop, Network and Distributed System Security Symposium (NDSS)*, 2019.

[92] Eunsuk Kang, Sridhar Adepu, Daniel Jackson, and Aditya P Mathur. Model-based security analysis of a water treatment system. In *2016 IEEE/ACM 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*, pages 22–28. IEEE, 2016.

[93] Anastasis Keliris and Michail Maniatakos. ICSREF: A framework for automated reverse engineering of industrial control systems binaries. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019*. The Internet Society, 2019.

[94] Eui-Sub Kim, Dong-Ah Lee, Sejin Jung, Junbeom Yoo, Jong-Gyun Choi, and Jang-Soo Lee. NuDE 2.0: A Formal Method-based Software Development, Verification and Safety Analysis Environment for Digital I&Cs in NPPs. *Journal of Computing Science and Engineering*, 11(1):9–23, 2017.

[95] Minyoung Kim, Mark-Oliver Stehr, Carolyn Talcott, Nikil Dutt, and Nalini Venkatasubramanian. Combining formal verification with observed system execution behavior to tune system parameters. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 257–273. Springer, 2007.

[96] Stéphane Klein, Lothar Litz, and Jean-Jacques Lesage. Fault detection of discrete event systems using an identification approach. *IFAC Proceedings Volumes*, 38(1):92–97, 2005.

[97] Johannes Klick, Stephan Lau, Daniel Marzin, Jan-Ole Malchow, and Volker Roth. Internet-facing PLCs-a new back orifice. *Blackhat USA*, pages 22–26, 2015.

[98] Sam Kottler, Mehdy Khayamy, Syed Rafay Hasan, and Omar Elkeelany. Formal verification of ladder logic programs using NuSMV. In *SoutheastCon 2017*, pages 1–5. IEEE, 2017.

[99] Egor Vladimirovich Kuzmin, AA Shipov, and Dmitrii Aleksandrovich Ryabukhin. Construction and verification of PLC programs by LTL specification. In *2013 Tools & Methods of Program Analysis*, pages 15–22. IEEE, 2013.

[100] Sandrine Lampérière-Couffin, Olivier Rossi, J-M Roussel, and J-J Lesage. Formal validation of PLC programs: a survey. In *1999 European Control Conference (ECC)*, pages 2170–2175. IEEE, 1999.

[101] Robert M Lee, Michael J Assante, and Tim Conway. German steel mill cyber attack. *Industrial Control Systems*, 30:62, 2014.

[102] Bernard Lim, Daniel Chen, Yongkyu An, Zbigniew Kalbarczyk, and Ravishankar Iyer. Attack induced common-mode failures on plc-based safety system in a nuclear power plant: Practical experience report. In *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 205–210. IEEE, 2017.

[103] Luca Luccarini, Gianni Luigi Bragadin, Gabriele Colombini, Maurizio Mancini, Paola Mello, Marco Montali, and Davide Sottara. Formal verification of wastewater treatment processes using events detected from continuous signals by means of artificial neural networks. Case study: SBR plant. *Environmental Modelling & Software*, 25(5):648–660, 2010.

[104] Christoph Luckeneder and Hermann Kaindl. Systematic top-down design of cyber-physical models with integrated validation and formal verification. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, pages 274–275, 2018.

[105] Dominique L'Her, Philippe Le Parc, and Lionel Marcé. Proving sequential function chart programs using automata. In *International Workshop on Implementing Automata*, pages 149–163. Springer, 1998.

[106] Angelika Mader. A classification of PLC models and applications. In *Discrete Event Systems*, pages 239–246. Springer, 2000.

[107] Monika Maidl, Dirk Kröselberg, Jochen Christ, and Kristian Beckers. A comprehensive framework for security in engineering projects-based on iec 62443. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 42–47. IEEE, 2018.

[108] PLC Manual. Basic Guide to PLCs: PLC Programming. https://www.plcmanual.com/plc-programming.

[109] Stephen McLaughlin and Patrick McDaniel. SABOT: specification-based payload generation for programmable logic controllers. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 439–449, 2012.

[110] Stephen McLaughlin and Saman Zonouz. Controller-aware false data injection against programmable logic controllers. In *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 848–853. IEEE, 2014.

[111] Stephen E McLaughlin. On Dynamic Malware Payloads Aimed at Programmable Logic Controllers. In *HotSec*, 2011.

[112] Stephen E McLaughlin, Saman A Zonouz, Devin J Pohly, and Patrick D McDaniel. A Trusted Safety Verifier for Process Controller Code. In *NDSS*, volume 14, 2014.

[113] Lucille McMinn and Jonathan Butts. A firmware verification tool for programmable logic controllers. In *International Conference on Critical Infrastructure Protection*, pages 59–69. Springer, 2012.

[114] S Mesli-Kesraoui, A Toguyeni, A Bignon, F Oquendo, D Kesraoui, and P Berruet. Formal and joint verification of control programs and supervision interfaces for socio-technical systems components. *IFAC-PapersOnLine*, 49(19):426–431, 2016.

[115] Houda Bel Mokadem, Béatrice Berard, Vincent Gourcuff, Olivier De Smet, and Jean-Marc Roussel. Verification of a timed multitask system with UPPAAL. *IEEE Transactions on Automation Science and Engineering*, 7(4):921–932, 2010.

[116] Il Moon. Modeling programmable logic controllers for logic verification. *IEEE Control Systems Magazine*, 14(2):53–59, 1994.

[117] Il Moon, Gary J Powers, Jerry R Burch, and Edmund M Clarke. Automatic verification of sequential control systems using temporal logic. *AIChE Journal*, 38(1):67–75, 1992.

[118] Amrit Mundra and VC Kumar. Programmable Logic Controllers — Security Threats and Solutions. *Security Application Brief SPRACO8*, 2019.

[119] Josh Newell, Linna Pang, David Tremaine, Alan Wassyng, and Mark Lawford. Translation of IEC 61131-3 function block diagrams to PVS for formal verification with real-time nuclear application. *Journal of Automated Reasoning*, 60(1):63–84, 2018.

[120] Mohamed Niang, Alexandre Philippot, François Gellot, Raphaël Coupat, Bernard Riera, and Sébastien Lefebvre. Formal Verification for Validation of PSEEL's PLC Program. In *ICINCO (1)*, pages 567–574, 2017.

[121] Tolga Ovatman, Atakan Aral, Davut Polat, and Ali Osman Ünver. An overview of model checking practices on verification of PLC software. *Software & Systems Modeling*, 15(4):937–960, 2016.

[122] Antti Pakonen, Teemu Mätäsniemi, Jussi Lahtinen, and Tommi Karhela. A toolset for model checking of PLC software. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–6. IEEE, 2013.

[123] Olivera Pavlovic and Hans-Dieter Ehrich. Model checking PLC software written in function block diagram. In *2010 Third International Conference on Software Testing, Verification and Validation*, pages 439–448. IEEE, 2010.

[124] Maxime Puys, Marie-Laure Potet, and Pascal Lafourcade. Formal analysis of security properties on the opc-ua scada protocol. In *International Conference on Computer Safety, Reliability, and Security*, pages 67–75. Springer, 2016.

[125] Mathias Rausch and Bruce H Krogh. Formal verification of PLC programs. In *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No. 98CH36207)*, volume 1, pages 234–238. IEEE, 1998.

[126] Blake C Rawlings, John M Wassick, and B Erik Ydstie. Application of formal verification and falsification to large-scale chemical plant automation systems. *Computers & Chemical Engineering*, 114:211–220, 2018.

[127] Olivier Rossi and Philippe Schnoebelen. Formal modeling of timed function blocks for the automatic verification of Ladder Diagram programs. In *Proceedings of the 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems (ADPM 2000)*, pages 177–182. Citeseer, 2000.

[128] Carl Schuett, Jonathan Butts, and Stephen Dunlap. An evaluation of modification attacks on programmable logic controllers. *International Journal of Critical Infrastructure Protection*, 7(1):61–68, 2014.

[129] Carl D Schuett. Programmable logic controller modification attacks for use in detection analysis. 2014.

[130] Saranyan Senthivel, Shrey Dhungana, Hyunguk Yoo, Irfan Ahmed, and Vassil Roussev. Denial of engineering operations attacks in industrial control systems. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pages 319–329, 2018.

[131] Ralf Spenneberg, Maik Brüggemann, and Hendrik Schwartke. Plc-blaster: A worm living solely in the plc. *Black Hat Asia, Marina Bay Sands, Singapore*, 2016.

[132] Michael Tiegelkamp and Karl-Heinz John. *IEC 61131-3: Programming industrial automation systems*. Springer, 1995.

[133] Valeriy Vyatkin and H-M Hanisch. A modeling approach for verification of IEC1499 function blocks using net condition/event systems. In *1999 7th IEEE International Conference on Emerging Technologies and Factory Automation. Proceedings ETFA'99 (Cat. No. 99TH8467)*, volume 1, pages 261–270. IEEE, 1999.

[134] Hai Wan, Gang Chen, Xiaoyu Song, and Ming Gu. Formalization and verification of PLC timers in Coq. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, volume 1, pages 315–323. IEEE, 2009.

[135] Rui Wang, Yong Guan, Liming Luo, Xiaoyu Song, and Jie Zhang. Formal modelling of PLC systems by BIP components. In *2013 IEEE 37th Annual Computer Software and Applications Conference*, pages 512–518. IEEE, 2013.

[136] Min Xiao, Jing Wu, Chengnian Long, and Shaoyuan Li. Construction of false sequence attack against PLC based power control system. In *2016 35th Chinese Control Conference (CCC)*, pages 10090–10095. IEEE, 2016.

[137] Zhen XU, Xiaojun ZHOU, Liming WANG, Zelong CHEN, Kai1 CHEN, Zhenbo YAN, Wei ZHANG, and Cong CHEN. Recent Advances in PLC Attack and Protection Technology. *Journal of Cyber Security*.

[138] Huan Yang, Liang Cheng, and Mooi Choo Chuah. Detecting payload attacks on programmable logic controllers (plcs). In *2018 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2018.
[139] Hyunguk Yoo and Irfan Ahmed. Control logic injection attacks on industrial control systems. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 33–48. Springer, 2019.
[140] Hyunguk Yoo, Sushma Kalle, Jared Smith, and Irfan Ahmed. Overshadow plc to detect remote control-logic injection attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 109–132. Springer, 2019.
[141] Junbeom Yoo, Eunkyoung Jee, and Sungdeok Cha. Formal modeling and verification of safety-critical software. *IEEE software*, 26(3):42–49, 2009.
[142] M Bani Younis, Georg Frey, et al. Formalization of existing PLC programs: A survey. In *Proceedings of CESA*, pages 0234–0239, 2003.
[143] Mu Zhang, Chien-Ying Chen, Bin-Chou Kao, Yassine Qamsane, Yuru Shao, Yikai Lin, Elaine Shi, Sibin Mohan, Kira Barton, James Moyne, et al. Towards Automated Safety Vetting of PLC Code in Real-World Plants. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 522–538. IEEE, 2019.
[144] Min Zhou, Fei He, Ming Gu, and Xiaoyu Song. Translation-based model checking for PLC programs. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, volume 1, pages 553–562. IEEE, 2009.
[145] Saman Zonouz, Julian Rrushi, and Stephen McLaughlin. Detecting industrial control malware using automated PLC code analytics. *IEEE Security & Privacy*, 12(6):40–47, 2014.

APPENDIX

A. Vulnerability Databases

This analysis is based on some interesting information, i.e. the reported Common Vulnerabilities and Exposures (CVE)s, their corresponding Common Weakness Enumeration (CWE)s, the impacted vendors and industrial sectors, the complexity of the CVEs, and their public exploits. We obtain the information from several sources: the ICS-CERT [19], the National Vulnerability Database (NVD) [20], and the Exploit Database [21] created by Offensive Security. The ICS-CERT reports timely security issues and vulnerabilities specifically in the industrial control systems. The NVD dataset contains general reports from all types of vulnerabilities, without details of affected industrial sectors and mitigation methods, as does by ICS-CERT. Nevertheless, the NVD dataset is less likely to miss data because of its popularity and has longer history (since 2002), while ICS-CERT provides data since 2010. The Exploit Database is a CVE compliant archive of public exploits, developed by penetration testers and vulnerability researchers.

B. Analysis Framework

We developed an analysis framework to first crawl and download data from the above sources, then extract interesting information on control logic related vulnerabilities, combine the results from all the datasets, and finally generate statistical reports from these information. The extraction combines filtering notable PLC vendors [89], matching general keywords, such as *PLC*, *control logic*, and combining specific keywords such as *HMI* and *remote code execution*, meaning such vulnerability in the HMI can lead to control logic execution in the PLC. The vulnerability may exist in the following places: (1) PLC; (2) upper level components that affect running code on the PLCs (e.g. HMI, SCADA, engineering station); (3) the communication between a PLC and such components or among PLCs. We randomly choose 100 CVEs from our collected datasets. We manually label them as control logic related or not, by going through the description of the vulnerability and checking online documentation of the affected products. We consider these labels as the ground truth. Then we run our automatic extraction framework and record control logic related CVEs. By comparing this record and the ground truth, we obtain the accuracy of our automatic extraction, with false positive rate of 1% and false negative rate of 4%.

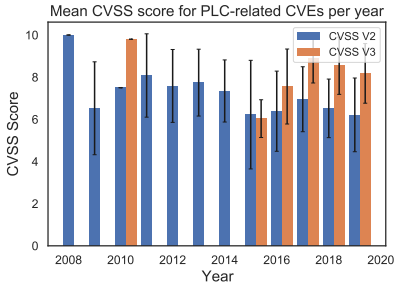

Fig. 5: The complexity of control logic related CVEs, depicted with the mean and the standard deviation of CVSS scores. Some points are missing as they are not reported in the databases.

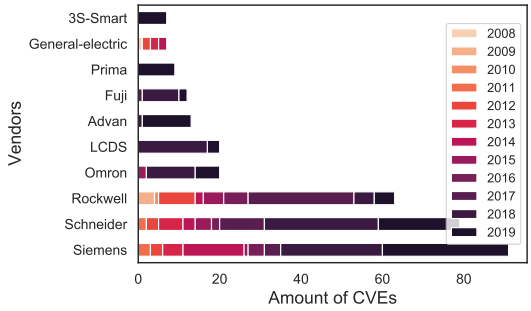

Fig. 6: Notable PLC vendors and number of related control logic vulnerabilities reported per year.

C. Analysis Result

Figure 5 shows the CVSS scores from both version 2 and version 3. For these reported CVEs, we also analyzed the top ten most affected vendors, as Figure 6 shows.

TABLE II: Systematization of PLC formal verification research

Paper	Venue	Goal	Formalization Methods	Application Area
Moon'92 [117]	AICE	Model checking solution	CTL Model Checker	Chemical
Brinksma'00 [45]	*SPIN*	Non-real-time model checking	SPIN, Promela	Chemical
Bauer'04 [36]	*ISSTAE*	Model checking for timed and untimed SFC	Cadence SMV, Uppaal	Chemical
Rawlings'18 [126]	JCCE	Handle combined invariance and reachability	symbolic model checking,	Chemical
Rausch'98 [125]	ACC	Model checking solution	SMV, Petri Net	Manufacturing
Bonfe'03 [43]	ICEFTA	Compare IEC 61131 and IEC 61499	SMV	Manufacturing
Cengic'06 [48]	ICETFA	Runtime environment for IEC 61499	Supremica (automata)	Manufacturing
Carlsson'12 [47]	*TII*	Simulation-based model checking	NuSMV	Manufacturing
Yoo'09 [141]	*Software*	Model checking & equivalence checking	NuSCR, Cadence SMV , VIS	Nuclear
Darvas'15 [57]	ICALEPCS	Complete specification	N/A	Nuclear
Adiego'15 [33]	*TII*	General model checking	nuXmv	nuclear
Darvas'16 [60]	*IFM*	State reduction heuristics	extended PLCverif	Nuclear
Kim'17 [94]	*CSE*	Integrate tools [141] for Nude 2.0	25 Case tools	Nuclear
Newell'18 [119]	JAR	Translate FBD for PVS proving	PVS Theorem proving	Nuclear
Luccarini'10 [103]	EMS	Analysis and validation on formal rules	SCIFF checker	Water treatment
He'16 [84]	EIT	Verify and assess non-timing properties	Simulink PLC Coder	Water treatment
Hailesellasie'18 [82]	HSS	Detect intrusion	BIP, nuXmv, UPPAAL	Water treatment
Pavlovic'10 [123]	*ICSTVV*	State reduction heuristic for FBD	NuSMV	Railway automation
Niang'17 [120]	ICINCO	Optimized safety verification	Uppaal	Railway automation
Fernandez'18 [67]	ICALEPCS	Comply IEC 61508 for critical PLCs	PLCverif	Railway automation
Chadwick'18 [50]	ICIRT	Safety verification for real systems	Theorem Proving, First order logic	Railway automation
Moon'94 [116]	*CSM*	Countermeasures for model checking	temporal, assertions, LD	Other mechatronics
Canet'00 [46]	ICSMC	Instruction list for model checking	Cadence SMV	Other mechatronics
Gourcuff'06 [78]	*DES*	State reduction heuristics	NuSMV	Other mechatronics
Giese'06 [75]	*MoDeVa*	Transformation verification	GROOVE, ISABELLE/HOL, FUJABA	Other mechatronics
Chang'06 [51]	ICCIMCA	Visualize model checking	N/A	Other mechatronics
Wan'09 [134]	*ACSAC*	Theorem proving for TON-timers	N/A	Other mechatronics
Zhou'09 [144]	*ACSAC*	Complete, compact and efficient translation	M Uppaal	Other mechatronics
Mokadem'10 [115]	TASE	Model checking for multitask PLCs	Uppaal (timed automata)	Other mechatronics
Biallas'12 [39]	*ASE*	Verification platform Arcade.PLC	PLCopen	Other mechatronics
Zonouz'14 [145]	*CRS,S&P*	Detect code injection	symbolic execution SMT, TEG	Other mechatronics
Mclaughlin'14 [112]	*NDSS*	Trusted safety verification platform	TEG, symbolic execution	Other Mechatronic
Mesli'16 [114]	HMS	Multiple layer verification	Uppaal	Other mechatronics
Darvas'17 [61]	PPEECS	ST to represent all languages	PLCverif	Other mechatronics
Kottler'17 [98]	SoutheastCon	Detect malfunctions by intruding NuSMV	NuSMV	Other mechatronics
Vyatkin'99 [133]	ICETFA	Verification for IEC 61499	Petri Net, CTL	Generic
Rossi'00 [127]	ICAM	Model checking using LD	Cadence SMV	Generic
Bender'08 [37]	MDAFA	Model checking to detect race condition	Petri net (TPN), Tina tool	Generic
Biha'11 [40]	*ACSAC*	Formal semantics for IL in Coq	Coq with on-delay timers	Generic
Wang'13 [135]	*ACSAC*	Formal semantics in BIP models	BIP	Generic
Kuzmin'13 [99]	*ACCS**	Construct and verify PLC programs	Cadence SMV	Generic
Darvas'16 [59]	PMS	Generic model representation	PLCverif	Generic
Bohlender'18 [42]	*DES*	Mode abstraction for impact computation	SMT	Generic
Galvao'18 [70]	HELIX	Review of plant models in form verification	N/A	Generic
Luckeneder'18 [104]	*SAC*	Adaptive Cruise Control with model checking	N/A	Generic
Huang'19 [87]	Access	Formal semantics of ST applied in C, Java	N/A	Generic
Garcia'19 [72]	ICCPS	Translate DC code to PLC code and vice versa	Theorem prover KeYmaera X	Generic
Zhang'19 [143]	*S&P*	Automated safety vetting of PLC code	BUILDTSEQS on TPTL	Generic
Lamperiere'99 [100]	ECC	Survey SFC and LD validation	N/A	Survey
Frey'00 [69]	ICSMC	survey based on A-F-M	N/A	Survey
Mader'00 [106]	*DES*	Classification criteria of modelling	N/A	Survey
Younis'03 [142]	*CESA*	Survey formalization	N/A	Survey
Johnson'07 [90]	CEP	Survey formalization in software dependability	N/A	Survey
Ovatman'16 [121]	*SSM*	Survey model checking	N/A	Survey
Darvas'18 [58]	ICALEPCS	Survey challenges and solutions N/A	N/A	Survey

*Note: IT venues are labeled *italic*, IT OT hybrid venues, e.g. Automatic Control and Computer Sciences, are labeled *italic**.

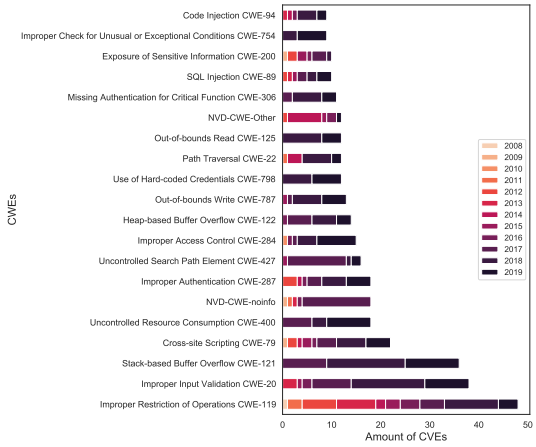

Fig. 7: The type of CWEs and their corresponding number reported per year.