To me, to you: Towards Secure PLC Programming through a Community-Driven Open-Source Initiative

Richard Derbyshire
Orange Cyberdefense
United Kingdom
ric.derbyshire@orangecyberdefense.com

Sam Maesschalck
Lancaster University
United Kingdom
s.maesschalck@lancaster.ac.uk

Alexander Staves
Lancaster University
United Kingdom
a.staves@lancaster.ac.uk

Benjamin Green
Lancaster University
United Kingdom
b.green2@lancaster.ac.uk

David Hutchison
Lancaster University
United Kingdom
d.hutchison@lancaster.ac.uk

***Abstract*—For decades, industrial control systems (ICS) have experienced an increasing frequency of cyber attacks, which in turn have increased in sophistication. Consequently, secure programmable logic controller (PLC) programming practices are becoming crucial as more adversaries are attaining the capability to gain a foothold in the ICS environment and directly attack the physical process through exploiting vulnerable PLC code. Existing programming practices involve the frequent use of vendor-provided, proprietary library functions, which cannot be viewed or edited, inhibiting the incorporation of secure PLC programming practices. This work begins by exploring the viability of open-source PLC functions as an alternative because of their open nature and potential for broader adoption of secure PLC programming practices. However, when analysed, the selected open-source PLC functions are found to contain the same vulnerabilities as those provided by the vendor. In response, a conceptual framework for a community-driven initiative is proposed that would acquire open-source PLC functions and their supporting documentation, review them for vulnerabilities and apply secure PLC coding practices, and finally disseminate the newly-secured, open-source PLC functions for wider use.**

***Index Terms*—ICS, cyber security, PLC programming practices, open source**

1. Introduction

Industrial control systems (ICS) form the backbone of modern industry, managing essential operations in sectors such as manufacturing, energy, water, and transportation [5]. As a subset of the broader operational technology (OT), these systems were traditionally isolated from the broader information technology (IT) environment, and enjoyed a certain degree of immunity from the cyber threats that targeted their IT counterparts. However, in recent years, a convergence of IT and OT environments has evolved, driven by the need for greater efficiency, automation, and data-driven decision making. This integration has led to a dramatic increase in the interconnection of ICS devices, exposing them to a whole new world of cyber threats and vulnerabilities [21].

The IT/OT convergence has seen ICS devices become targeted by an increasing number and variety of adversaries, as opposed to the insider threats and more advanced (but less frequent) adversaries to which they have become accustomed [6], [18]. In response, many IT cyber security practices and controls have been re-appropriated into the ICS environment, most notably network segregation. However, ICS devices were developed primarily for functionality and reliability [10], with little consideration given to the potential risks associated with external access. As a result, many ICS components lack the necessary security features to protect against modern cyber threats, such as secure PLC authentication protocols [3], and even when they are available they are often not used [19]. This issue is exacerbated by the fact that ICS devices often have long lifecycles, with some systems remaining in operation for decades, leading to long vulnerability exposure times [7], [17]. This means that many ICS devices currently in use were designed and implemented well before the contemporary cyber threat landscape took shape, leaving them ill-equipped to deal with modern attacks.

Programmable logic controllers (PLCs) are the embedded devices at the heart of ICS and the industrial process. They are responsible for the monitoring and control of the physical environments in which they reside, automating the reading of sensors and actuation of pumps, valves, or even robots according to their programming. As PLCs form an integral part of the industrial process, it is essential that the code they run to monitor and control the physical environment is implemented securely; this is done by observing secure PLC programming practices [23]. By adhering to established programming standards, implementing secure programming practices, and following thorough testing and validation processes, organisations can minimise the likelihood of vulnerabilities being introduced into their ICS and physical process. However, many organisations rely on PLC code in the form of vendor-provided library functions [15], which are proprietary, unable to be edited, and demonstrably vulnerable [12], [16], meaning that many elements of secure PLC

programming practices cannot be exercised when they are sorely needed.

This paper first explores secure PLC programming practices and their importance, before considering the use of open-source functions as an alternative to vendor-provided library functions. By using existing tooling [16] to scan a PLC's code for vulnerabilities, it is found that such open-source functions are as vulnerable as the vendor-provided library functions, but with one clear distinction; unlike the proprietary, vendor-provided library functions, open-source functions' code may be viewed and edited, affording the incorporation of secure PLC programming practices. This leads to a proposed framework of a community-driven hub for secure, open-source PLC code, which tests and verifies that the code is free of known vulnerabilities and conforms to both secure PLC programming practices and known best practice of standards and guidelines.

The rest of the paper is structured as follows. Section 2 discusses the vulnerabilities and attacks that may arise due to insecure PLC programming practices and initiatives to prevent them. Section 3 presents a brief background of vendor-provided, proprietary library functions and open-source alternatives. Section 4 provides an analysis of a selection of open-source functions to discover whether they are vulnerable. Section 5 proposes a conceptual framework for a community-driven initiative to create secure open-source functions. Section 6 concludes the work and discusses future work that may be necessary.

2. Related Work

There has been limited research on secure programming practices for PLCs, on the vulnerabilities that result from insecure code, and on the consequent attacks on those vulnerabilites. This section presents the prominent work that has investigated this issue, as well as the vulnerabilities and attacks that may arise from them.

Serhane et al. [24] show that inadequately structured and designed ladder logic code can increase the risks of vulnerabilities, even when adhering to company standards and recommendations. These risks are exacerbated when code is not written by professional, experienced individuals. Serhane et al. assert that company-oriented standards often prioritise system functionality, optimisation, and safety over addressing security threats and vulnerabilities, thus potentially creating back doors for hackers or dormant threats within PLC programs. They have also seen that vulnerabilities in ladder logic code arise from various poor programming practices. For example, the use of duplicated instructions, such as output enable (OTE), counters, timers, and jump-to-subroutine (JSR), can lead to undesirable results. Duplicated instructions may produce unintended fluctuating values for operands, complicating debugging efforts. Some PLCs may not permit duplication of specific outputs, while others might allow it. This all goes to show that poor programming practices can lead to significantly more vulnerable systems.

Valentine and Farkas [25] categorise programming errors and intentional attacks on PLCs into major and minor threats. Major threats include duplicate objects, logic errors, syntax errors, programming standard violations, and unused objects. Minor threats comprise scope and linkage errors and hidden jumpers. For example, they state a missing coil to trigger contacts would be a potential security issue. In the correct use case, a normally open contact triggers an alarm, sending an alert message to the operator. A malicious individual could, however, write code that omits the required coil, making the code compile without errors. This programming error might remain undetected until the alarm needs to be triggered. To address this issue, the authors propose converting PLC code into a database and querying it against a predefined set of security rules. For example, a rule would state that there must always be exactly one coil associated with at least one contact of the same name. During code validation, violations would be flagged. This presents a suitable way to enforce secure programming practices within PLC programming.

Focusing on the intersection of PLC programming practices and memory management, Green et al. [12] introduce the technique 'Process Comprehension at a Distance' (PCaaD). PCaaD takes advantage of the fact that many PLC programmers use vendor-provided library functions [15] to distinguish what functions are running on the PLC, thus providing an adversary a greater level of process comprehension [13]. This is done by reading those functions' data blocks (areas of memory dedicated to storing variables) at a byte-code level and matching each function's unique signature of byte-code length and offsets of memory which contain only 0 values. By using a similar technique but instead writing to the PLC's memory, Green et al. also demonstrate the possibility of manipulating variables and therefore the physical process. The areas of memory that contain 0 values are determined to be benign and could be used by adversaries to store their own data, resulting in PLC memory being used as a conduit for command and control.

Building on some of the concepts of PCaaD [12], Maesschalck et al. [16] show that many of the vendor-provided library functions contain a vulnerability that allows adversaries to change the operation of the PLC. The authors present a scanning tool to identify bytes, and therefore variables, in the target PLC's memory that are susceptible to being manipulated over the network. The potential implications of this vulnerability are outlined using an example attack scenario, whereby an adversary may prohibit the PLC from sending an alert email to an engineer by preventing the COUNT_UP function from triggering the pulse to send it. Maesschalck et al. analysed 10 vendor-provided library functions, all of which were found to contain a large number of vulnerable bytes. This was tested against immediate reads to check the value had been written to memory, and delayed reads to check the value persisted in memory without being overwritten. It was discovered that default variables, i.e. those which are left unchanged after importing the vendor-provided library function, are the most vulnerable. However, direct variables (hard coded variables) and variables in the global variable block (an area of memory accessible to multiple functions) may also be vulnerable in certain circumstances though less likely to be vulnerable than their default counterparts.

In contrast to the vulnerabilities and attacks discussed thus far, Ponnada and Fluchs [23] present their top 20 list of secure PLC coding (programming) practices. Despite its name, the list extends further than just programming

practices to more holistic controls such as disabling unused ports and protocols and monitoring uptime. Each item on the list is expanded on to include guidance, example implementation, and a discussion as to why it is beneficial for security, reliability, and maintenance, as well as including a list of references to resources such as MITRE ATT&CK for ICS [20] and ISA 62443 [22]. Where the list items do include programming practices, the guidance provides helpful examples in the form of diagrams representing ladder logic as one of the more common languages for PLC programming.

3. PLC Programming Practices

Current PLC programming practices in industry environments rely on vendor-provided library functions [15]. However, these functions are proprietary and their underlying code cannot be viewed or edited without circumventing digital rights management (e.g. Siemens Know-How Protection, as depicted in Figure 1). This opacity can result in challenges when incorporating secure PLC programming practices, and thus requires a significant amount of trust in the vendor. As mentioned in Section 2, previous research investigating a PLC memory vulnerability [16] identified numerous vulnerable areas in the memory related to Siemens-provided library functions. All of the functions investigated contained at least some vulnerable bytes either after direct or delayed read. Even smaller, commonly used functions, such as COUNT_UP, contain vulnerabilities such that their variables may be overwritten. This means that once an adversary has access to the ICS network, they are able to manipulate variables within the PLC and, in turn, to affect the physical process.

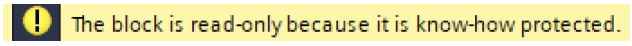

Figure 1. Siemens Know-How Protection error when trying to view library function code

Across the vast domain of cyber security, open source has frequently been a wellspring of concepts and tooling, such as Snort [4] and OpenVAS [14]. Within ICS, open-source initiatives have focused more on device level functionality as can be seen with OpenPLC [2]. OpenPLC is an open-source software alternative to physical systems and is capable of emulating different PLCs. The operation of these PLCs is similar to that of physical PLCs, and they can be programmed in several languages including Function Block Diagram, Ladder Logic, and Structured Text. The tool also allows users to share their projects through a community forum, thus further building a collaborative open-source community. This initiative goes part way to a more open-source approach to PLC programming, but OpenPLC code is not compatible with all other PLCs [1]. However, since vendor-provided, proprietary library functions cannot be viewed or edited, and therefore cannot reliably adhere to secure PLC programming practices without vendor intervention, open-source functions provide a potentially useful alternative.

4. Open-Source PLC Functions

There are many open-source functions available online, with one of the more significant projects being the Siemens Open Library [8] for Siemens PLCs. This library contains multiple functions and function blocks to be used by engineers for motor control, valve control, and other various operations. Two of these function blocks were selected to be evaluated for vulnerabilities according to the scanner proposed by Maesschalck et al. [16], to discover whether open-source functions are inherently more secure. While this is not an in-depth study to comprehensively identify vulnerabilities across a significant sample of open-source functions, it serves to form an understanding of whether they already adhere to secure PLC programming practices.

The function blocks chosen were IO_AnalogInput and Interlock. These functions were deployed on an Siemens S7-300 series PLC as per the supporting documentation and scanned from a system connected to the ICS network of an established testbed [11]. These function blocks are part of the Siemens Open Library V13 SP1 Update 8, which is the latest version supported by the version of TIA Portal used in this evaluation.

Function	Bytes	Direct	Delayed
IO_AnalogInput	112	48	44
Interlock	5	4	4
TOTAL	117	52	48

TABLE 1. SIEMENS OPEN LIBRARY VULNERABLE BYTES

4.1. Analysis

The two open-source blocks that were analysed have a combined total size of 117 bytes, with IO_AnalogInput comprising 112 bytes and Interlock comprising 5 bytes. The results of the experiment can be found in Table 1, whereby the first column is the number of bytes in the function and the following columns reflect vulnerable bytes found per type of read after writing to memory. A confirmed direct read means that the byte was written into memory successfully and that byte can at least be immediately manipulated. However, a confirmed delayed read means that the byte written to memory had persisted for 5 seconds, which means that byte can be persistently manipulated.

From the analysis, it can be seen that for IO_AnalogInput, 43% of its bytes were immediately manipulable and 39% of its bytes were persistently manipulable. This means that should an adversary write to this function during an attack, 39% of the bytes in memory would persist, enabling the potential for a higher impact attack. Although a much smaller function, Interlock was approximately 80% vulnerable for both direct and delayed reads. However, the memory for the Interlock function is used predominantly for Boolean variables which use single bits. Therefore, in two of the bytes there exists an individual bit in each that is not vulnerable. This shows that open-source functions unfortunately contain the type of vulnerability described by Maesschalck et al. [16], though fortunately not to the same extent as the vendor-provided functions.

While the analysis was being set up, a particular variable of interest was noticed in the IO_AnalogInput function called rlnLowLowDeadband. In this context, a deadband is responsible for verifying whether a value is

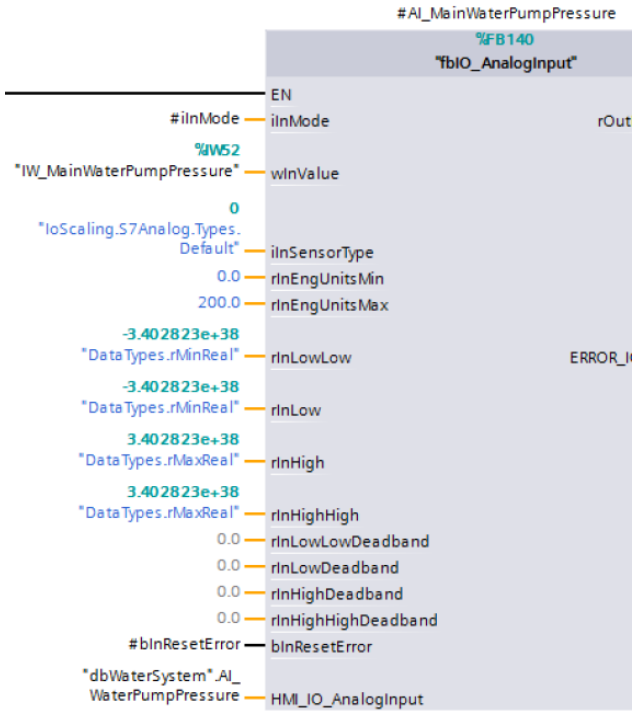

Figure 2. IO_AnalogInput function block configuration guidance from supporting documentation

rInLowLowDeadband	Real	30.0	0.0
rInLowDeadband	Real	34.0	0.0

Figure 3. rInLowLowDeadband variable in the data block of IO_AnalogInput

```
Byte 30 is vulnerable.
Byte 31 is vulnerable.
Byte 32 is vulnerable.
Byte 33 is vulnerable.
```

Figure 4. Vulnerability scanner output of the IO_AnalogInput data block focused on rInLowLowDeadband's offset

at or exceeds a particular threshold for a certain period of time before raising an alert. This is to ensure that, for example, if water is being disturbed in a tank, that disturbance repeatedly exceeding a threshold does not also repeatedly raise an alert, thus causing alert fatigue. The recommended guidance in the documentation is to leave this variable at '0' to disable the deadband, and as depicted in Figure 2, the function comes with rInLowLowDeadband set as '0' by default. However, this may lead unsuspecting engineers into a false sense of security, as this default variable could be persistently overwritten to enable the deadband at an abnormally high time period, stifling any alerts and allowing for dangerous conditions. Paying close attention to the rInLowLowDeadband variable of IO_AnalogInput during the analysis, the variable's offset and length were recorded, viz. Figure 3. As can be seen, rInLowLowDeadband is at offset byte 30 and is 4 bytes long. As expected, the vulnerability scan reveals that an adversary would be able to write to these 4 bytes with persistence, as depicted by the tool output in Figure 4.

Upon further review of the Siemens Open Library documentation [9], extending to functions that were not scanned in the analysis, this specific type of configuration vulnerability was identified across many other functions in this library, including the proportional integral derivative (PID) function. This highlights that not only must the PLC code itself adhere to secure PLC programming practices, but also the supporting documentation that provides the code's implementation guidance.

To reiterate, this analysis has served as just one proof of concept example of open-source functions being vulnerable, which could be fixed by the community either via modification of the code, the supporting documentation, or both. It is not intended to be a comprehensive analysis of vulnerabilities in open-source PLC code; nor is it meant to be a criticism of the Siemens Open Library's admirable efforts to provide open-source PLC code.

5. Framework

Although open-source functions can be viewed and edited, which allows them to be adapted to secure PLC programming practices, an organisation would have to invest significantly more time to incorporate such processes when compared to simply importing the vendor-provided functions. Therefore, this section presents a conceptual framework for a community-driven initiative that would allow for organisations and their engineers to collaborate on acquiring, securing, and disseminating open-source functions. The framework is depicted in Figure 5 and the rest of this section will describe the Code and Documentation Acquisition and Processing, Community Action, and Secure PLC Code processes along with their subprocesses.

5.1. Code and documentation acquisition and processing

The first process is intended to be where the functions are uploaded, scanned for malware, and provide information to the Community Action process.

5.1.1. Scan documentation. As a matter of security, the first sub-process suggested by the framework is to scan any acquired documentation for traditional malware. This is to reduce the risk of the whole operation becoming compromised given that the intention of the eventual secure PLC functions is to be trusted by all manner of industrial processes, potentially including CNI.

5.1.2. Scan code. As well as scanning the acquired documentation, the framework also suggests scanning acquired functions for traditional malware. This is perhaps even more important than the documentation to ensure that the eventual secure PLC functions are not used as part of a supply chain attack.

5.1.3. Profile code. Within the profile code sub-process, the vendor and version of the code are detected. This is for categorisation so it can be understood for which devices the open-source functions are intended.

5.1.4. Generate PDF. By potentially using vendor-provided configuration software, metadata about the function can be extracted and printed as a PDF which may provide additional useful information to the community.

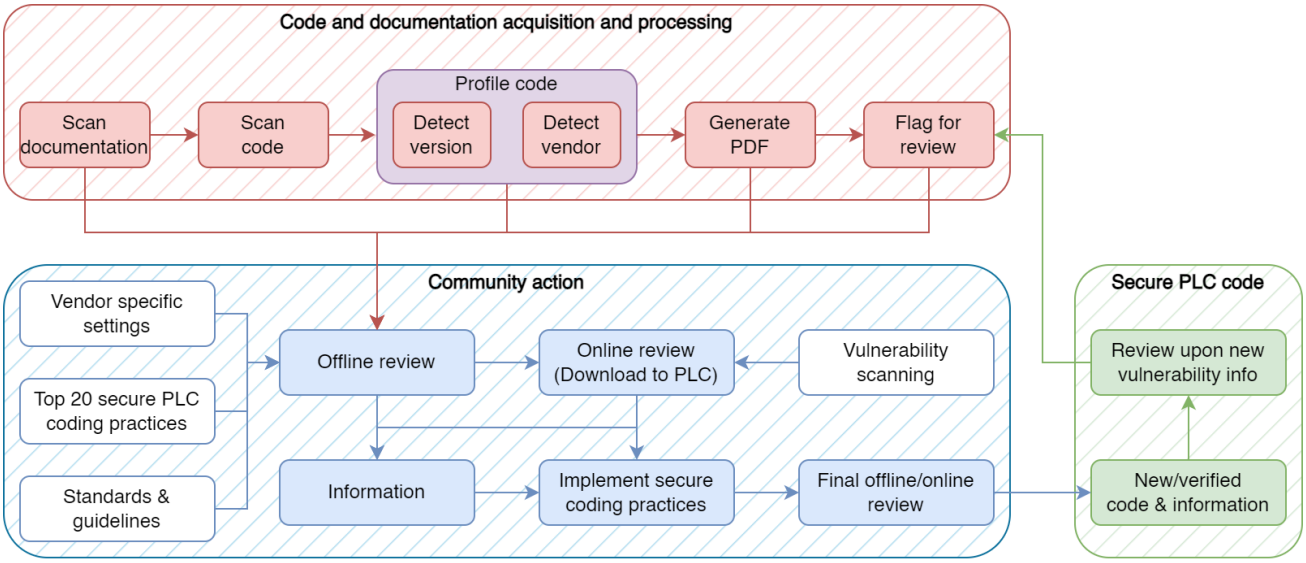

Figure 5. Framework for community secured open-source library functions

5.1.5. Flag for review. Once the function has been successfully acquired thus far, it can then be flagged for review and passed to the community for processing.

5.2. Community action

The community action process is where the open-source functions are checked for vulnerabilities via offline and online reviews.

5.2.1. Offline review. The offline review sub-process recommends that reviewers manually analyse the functions' code, using resources such as recommended vendor-specific settings and practices, Ponnada and Fluchs' Top 20 Secure PLC Coding Practices [23], and relevant standards and guidelines. During the offline review, the framework also suggests that the acquired documentation be reviewed such that no configuration guidelines recommend insecure practices.

5.2.2. Online review. The online review sub-process recommends that reviewers download the function to a PLC and use PLC-focussed vulnerability scanners such as the one proposed by Maesschalck et al. [16].

5.2.3. Information. The offline and online reviews should both provide detailed information and a history of each function to understand what vulnerabilities it contained and what has been done to improve its security.

5.2.4. Implement secure coding practices. This sub-process is where the function will take all output from the offline and online reviews, as well as any historical information, and implement secure PLC programming practices. Should there be any vulnerabilities which cannot be removed due to functionality constraints, a warning should be provided within its documentation, perhaps even with suggestions of how to implement the security features should the eventual user choose to do so.

5.2.5. Final offline/online review. Before being released publicly, the function should pass through one final round of offline and online reviews for quality and security assurance. This can be expedited by focusing on the function's historical information.

5.3. Secure PLC code

The final process is to release and monitor the newly-secured functions.

5.3.1. New/verified code & information. The functions should be released along with historical information of what vulnerabilities it initially had, how they were secured, and any remaining vulnerabilities left to preserve functionality. This release should also include supporting documentation of how to use the function safely and securely.

5.3.2. Review upon new vulnerability info. Finally, the released functions should have a record of release date and a warning upon becoming a certain age with no review, particularly if any of the related historical information about previous vulnerabilities, vendors, or versions are relevant in newly-released vulnerability information. Should the function reach a certain age or be related to a newly-released vulnerability, it should be flagged for review such that it can be patched.

6. Conclusion

In the face of the IT/OT convergence and an ever increasing number of adversaries targeting ICS, this work has explored whether an open source approach may contain the answer to secure PLC programming practices. At present, current PLC programming practices frequently involve the use of vendor-provided, proprietary library functions that are unable to be edited or even viewed, which makes incorporating secure PLC programming practices a challenge at best. Conversely, open-source functions may

be both viewed and edited. In light of this, an analysis was conducted on selected open-source functions to discover whether they contain the same vulnerabilities as their vendor-provided counterparts; unfortunately they do, albeit in fewer numbers. Moreover, the open-source functions' supporting documentation suggest unsafe practices that lead to further vulnerability. Although the existing open-source functions and documentation contain vulnerabilities, they still offer the promise of incorporating secure PLC programming practices. Therefore, a conceptual framework for a community-driven, open-source initiative was proposed. Within this framework, open-source PLC functions and their supporting documentation are acquired and checked for vulnerabilities via multiple methods by a community, creating an audit trail of evidence to document their journey, before finally being disseminated for wider use. The intention is that this framework would be adopted by organisations and engineers that own and operate ICS. In this way they could integrate secure PLC functions into their infrastructure without the time-intensive overhead of adapting all of their PLC code to ensure secure programming practices, and more importantly without leaving their PLC code insecure in the first place.

This paper presents a proposition to improve the cyber security of PLC programming in ICS; the proposed framework has not yet been implemented. Further work would invite discussion and refinements to the framework, before potentially building a consortium of interested parties as stakeholders in order to bring the framework to fruition.

References

[1] Thiago Alves, Rishabh Das, Aaron Werth, and Thomas Morris. Virtualization of scada testbeds for cybersecurity research: A modular approach. *Computers & Security*, 77:531–546, 2018.

[2] Thiago Rodrigues Alves, Mario Buratto, Flavio Mauricio De Souza, and Thelma Virginia Rodrigues. Openplc: An open source alternative to automation. In *IEEE Global Humanitarian Technology Conference (GHTC 2014)*, pages 585–589. IEEE, 2014.

[3] Adeen Ayub, Hyunguk Yoo, and Irfan Ahmed. Empirical study of plc authentication protocols in industrial control systems. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 383–397. IEEE, 2021.

[4] Cisco. Snort - Network Intrusion Detection & Prevention System. https://www.snort.org/, 2023.

[5] CPNI. Critical National Infrastructure. https://www.cpni.gov.uk/critical-national-infrastructure-0, 2020.

[6] Richard Derbyshire, Benjamin Green, Daniel Prince, Andreas Mauthe, and David Hutchison. An analysis of cyber security attack taxonomies. In *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 153–161. IEEE, 2018.

[7] Debabrata Dey, Atanu Lahiri, and Guoying Zhang. Optimal policies for security patch management. *INFORMS Journal on Computing*, 27(3):462–477, 2015.

[8] DMC Inc. Siemens Open Library. https://openplclibrary.com/.

[9] DMC Inc. Siemens Open Library: Detailed Blocks Overview. https://openplclibrary.com/wp-content/uploads/2017/02/Siemens-Open-Library-V13SP1-V1.4.zip, 2016.

[10] Graham Clifford Goodwin, Stefan F Graebe, Mario E Salgado, et al. *Control system design*, volume 240. Prentice Hall Upper Saddle River, 2001.

[11] Benjamin Green, Ric Derbyshire, William Knowles, James Boorman, Pierre Ciholas, Daniel Prince, and David Hutchison. Ics testbed tetris: Practical building blocks towards a cyber security resource. In *The 13th USENIX Workshop on Cyber Security Experimentation and Test (CSET'20)*, 2020.

[12] Benjamin Green, Richard Derbyshire, Marina Krotofil, William Knowles, Daniel Prince, and Neeraj Suri. Pcaad: Towards automated determination and exploitation of industrial systems. *Computers & Security*, 110:102424, 2021.

[13] Benjamin Green, Marina Krotofil, and Ali Abbasi. On the significance of process comprehension for conducting targeted ics attacks. In *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and PrivaCy*, pages 57–67, 2017.

[14] Greenbone. OpenVAS - Open Vulnerability Assessment Scanner. https://openvas.org/, 2023.

[15] Oscar Ljungkrantz and Knut Akesson. A study of industrial logic control programming using library components. In *2007 IEEE International Conference on Automation Science and Engineering*, pages 117–122, New Jersey, 2007. IEEE.

[16] Sam Maesschalck, Alexander Staves, Richard Derbyshire, Benjamin Green, and David Hutchison. Walking under the ladder logic: PLC-VBS: a PLC control logic vulnerability scanning tool. *Computers & Security*, 127:103116, 2023.

[17] Angelos K Marnerides, Vasileios Giotsas, and Troy Mursch. Identifying infected energy systems in the wild. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, pages 263–267, 2019.

[18] Thomas Miller, Alexander Staves, Sam Maesschalck, Miriam Sturdee, and Benjamin Green. Looking back to look forward: Lessons learnt from cyber-attacks on industrial control systems. *International Journal of Critical Infrastructure Protection*, 35:100464, 2021.

[19] Ariana Mirian, Zane Ma, David Adrian, Matthew Tischer, Thasphon Chuenchujit, Tim Yardley, Robin Berthier, Joshua Mason, Zakir Durumeric, J. Alex Halderman, and Michael Bailey. An Internet-wide view of ICS devices. *2016 14th Annual Conference on Privacy, Security and Trust, PST 2016*, pages 96–103, 2016.

[20] MITRE. ATT&CK for ICS. https://attack.mitre.org/matrices/ics/, 2023.

[21] Glenn Murray, Michael N Johnstone, and Craig Valli. The convergence of it and ot in critical infrastructure. 2017.

[22] International Society of Automation. ISA/IEC Series of Standards. https://www.isa.org/standards-and-publications/isa-standards/isa-iec-62443-series-of-standards, 2023.

[23] Vivek Ponnada and Sarah Fluchs. Top 20 Secure PLC Coding Practices. https://plc-security.com/index.html, 2021.

[24] Abraham Serhane, Mohamad Raad, Raad Raad, and Willy Susilo. Plc code-level vulnerabilities. In *2018 International Conference on Computer and Applications (ICCA)*, pages 348–352. IEEE, 2018.

[25] Sidney Valentine and Csilla Farkas. Software security: Application-level vulnerabilities in scada systems. In *2011 IEEE International Conference on Information Reuse & Integration*, pages 498–499. IEEE, 2011.