electronics

MDPI

Article

Anomaly Detection Based on Temporal Behavior Monitoring in Programmable Logic Controllers

Seungjae Han [1], Keonyong Lee [2], Seongje Cho [3] and Moonju Park [4,*]

[1] Department of Computer Science & Engineering, Dankook University, Yongin 16890, Korea; sjhan93@dankook.ac.kr
[2] Department of Applied Computer Engineering, Dankook University, Yongin 16890, Korea; lky9620@dankook.ac.kr
[3] Department of Software Science, Dankook University, Yongin 16890, Korea; sjcho@dankook.ac.kr
[4] Department of Computer Science & Engineering, Incheon National University, Incheon 22012, Korea
* Correspondence: mpark@inu.ac.kr

Abstract: As Programmable Logic Controllers (PLCs) are increasingly connected and integrated into the industrial Internet of things, cybersecurity threats to PLCs are also increasing. Adversaries can perform a denial of service (DoS) attack based on the transmission of a large number of network packets, and a control-logic injection attack through sophisticated packet transmission. We propose an approach to detecting and defending against attacks that exploit security vulnerabilities in a PLC system. In order to protect against indiscriminate packet transmission attacks that exploit uncontrolled resource consumption vulnerabilities, an abnormal temporal behavior detection method is proposed that monitors the CPU usage of tasks. If a temporal anomaly is detected, the proposed approach tries to detect control-flow anomalies by examining the sequences of function calls, then detects stack-based buffer overflow attacks. The proposed method is implemented in a water tank control system for evaluation purposes. The experimental results show that the proposed method can improve the security of the system by detecting anomalies in temporal behavior with little system overhead.

Keywords: programmable logic controller; anomaly detection; embedded system; industrial control system

Citation: Han, S.; Lee, K.; Cho, S.; Park, M. Anomaly Detection Based on Temporal Behavior Monitoring in Programmable Logic Controllers. *Electronics* **2021**, *10*, 1218. https://doi.org/10.3390/electronics10101218

Academic Editor: George Angelos Papadopoulos

Received: 24 April 2021
Accepted: 18 May 2021
Published: 20 May 2021

1. Introduction

An Industrial Control System (ICS) consists of Supervisory Control and Data Acquisition (SCADA) systems, a Distributed Control System (DCS), and Programmable Logic Controllers (PLCs). Although ICSs are widely used in industries for critical infrastructures such as electric power generation, chemical manufacturing, oil refinery and water treatment, increasing numbers of security vulnerabilities are introduced to ICSs as the majority of them become connected directly or indirectly to the Internet. Vulnerabilities in ICSs have been found 438 times in 2019 [1], which is a 70% increase from 2018 [2]. Cyberattacks on ICSs may cause serious devastation in critical infrastructures, and PLCs are very important in defending ICSs against the cyberattacks as PLCs are at the end of the control sequence, monitoring and controlling the physical system [3,4].

As PLCs are widely used in industries closely related to civil life, they are becoming prime targets for cyberattacks. Code injection attacks to PLCs have been made targeting a nuclear reactor [5] and engineering software [6]. Additionally, a Denial-of-Service (DoS) attack using vulnerabilities in SCADA protocol to block the communication to PLCs was reported [7]. Attacks to PLCs could cause errors in a physical system by maliciously controlling the I/O signal of PLCs [8].

Attackers aim to compromise the confidentiality, integrity, or availability of PLCs by gaining unauthorized access to networks and programs of a PLC or exploiting vulnerabili-

ties of the software in PLCs. In ICS, the most important security property is the availability to which one of the biggest threats is the Denial-of-Service (DoS) attack that exhaust system resources [9]. However, a traditional Intrusion Detection System (IDS) or an anti-virus program cannot be used for protection of PLCs due to the characteristics of the PLC as an embedded system with limited resources [10,11]. Moreover, the ModBus protocol that is used for communication between PLCs and engineering workstation (EWS), sensors, or actuators is not designed to provide any security features [12].

A DoS attack on a PLC disturbs normal processing of the PLC by disabling its capability to communicate with other ICS components or execute control-logic programs [13]. The availability is affected by the indiscriminate transmission of traffic data that is typical of a DoS attack. DoS attacks can cause packet losses, longer latency in the network traffic of the control system, missed deadlines of critical tasks, and finally, overall disruption of the system.

In this paper, a method for detecting a behavioral anomaly in task scheduling is presented to stop the DoS attacks by blocking the overrun of the misbehaving task. Data reception to a PLC triggers the execution of tasks related to the data, which consequently increases the execution rate or the amount of execution time in a given time interval. Therefore, the proposed method monitors tasks' CPU utilization periodically to report an abnormally large amount of increase in CPU usage of tasks. Once an anomaly in the temporal behavior of a task is detected, it is checked as to whether the control flow of the task is manipulated by an attacker. In case the control-flow alteration is detected, the proposed method reports that the integrity of the PLC software is compromised. Otherwise, it suspends the communication service of the PLC to protect the system from the DoS attack.

This paper is organized as follows. Section 2 overviews the related works. Section 3 describes the anomaly detection based on temporal behavior monitoring, and the software integrity examination by checking the control flow using shadow stack information. In Section 4, experimental results with an actual water control system are presented. Finally, Section 5 concludes this work.

2. Related Works

If an attacker can allocate limited system resources such as memory, file system storage, database connection pool entries, and CPU without proper control and management of the resources, then the attacker could cause a denial of service by consuming all system resources [14]. One of the most important security exploits of PLCs is a DoS attack by sending communication packets indiscriminately from a flawed EWS connected to the PLCs, where the EWS is a high-end computing platform that is used for the development, control, and management of PLCs.

Ylmaz et al. implemented a ping flood attack on PLC devices using the *hping* program that assembles and analyzes TCP/IP packets [9]. In the experiments, the response time in the communication of the attacked PLC device is greatly increased to 1212 ms from 3 ms. Moreover, the ping flood attack made the PLC device uncontrollable by disabling all control and monitoring functionalities of the EWS. Niedermaier et al. studied the SYN flooding attack to multiple PLCs by disrupting the PLC cycle time [15]. The operating cycle of PLC consists of four steps: input scan for detecting the state of input devices connected to the PLC, program scan that processes the program logic, output scan for energizing or de-energizing the outputs, and housekeeping that includes communication and internal diagnostics. These steps continuously take place in a loop repeatedly. In [15], it was reported that the interrupt request for processing network traffic could be doubled due to the SYN flooding attack, which increased the cycle time that affected the response time of the PLC, consequently affecting the stability of the system. Long et al. tried a DoS attack on a Network-Based Control System (NBCS) and analyzed delay jitter and packet losses due to the DoS attack by establishing a queueing model [16]. A simulation study of an attack on a SCADA system can be found in [17], which reported the packet drop

rate and the packet delays due to a distributed DoS (DDoS) attack on the SCADA system. In [18], it was shown to be possible that DDoS attacks on the industrial Internet of things (IIOT) can cause discontinuation of production lines in a factory.

Due to the limited computing power of the PLC, little work can be found on the protection of PLC from such attacks. Xiao et al. presented an anomaly detection method using power consumption information of a PLC in [10]. Their method used a neural–network-based method for detection and required hardware modification to gather the power consumption information. Because existing anomaly detection methods based on pattern matching or neural networks require high computing power that cannot be provided by PLCs, defense mechanism inside a PLC is yet to be developed.

Exploiting memory vulnerabilities is another possible attack on PLCs [19]. The attack is made by manipulating control flows of the PLC software in two ways: the code injection attack that maliciously modifies the control logic in a target PLC, and the code reuse attack that re-organizes the existing binary codes in the target PLC to exploit vulnerabilities, thus hides the malicious logic of the attacker. Control-logic injection attacks have been studied as code injection/reuse attacks to PLCs [3,5,6,8,19,20]. A typical control-logic injection attack aims to modify the existing control logic in a PLC by a man-in-the-middle attack intercepting the communication between the ESW and the PLC [5,6,21,22]. An example of a control-logic injection attack was Stuxnet, which infected the STEP 7 engineering software in order to insert malicious control logic into the Siemens S7-300 PLC in a nuclear plant [19]. Senthivel et al. reported three control-logic injection attacks that disturb uploads/downloads of PLC control logics from/to the ESW called Denial of Engineering Operations [6].

3. Anomaly Detection Method Based on Temporal Behavior

3.1. Detecting CPU Usage Anomaly

PLC tasks run at a pre-determined frequency, i.e., periodically, with a fixed interval. A PLC task usually runs only once in one period, but its execution time may vary depending upon the current runtime environment. For example, if more data packets than expected are queued in the communication buffer, the execution time of the task responsible for communication could be longer than usual. In many cases, PLC tasks are very cautiously designed so that their execution time does not vary too much. It is assumed in design time that the runtime environment of PLC tasks is well controlled, so the task's period and its execution time are known a priori. However, if the PLC is under attack by a malicious adversary, the design-time assumption could not hold. A DoS attack of indiscriminate packet transmission could cause the overrun of PLC tasks, which would affect the execution time of the tasks. To detect such an attack, we present a method to monitor the CPU usage of tasks. If the CPU usage of a task is out of bounds that have been set in design time, it can be regarded as a behavioral anomaly of the task.

To monitor and track the CPU usage of tasks, we need to record and summarize every execution time of each task in a period. For this purpose, we check the execution time of a task in the context switch routine. In PLC systems, a context switch is triggered when a task voluntarily yields the CPU or it goes to the wait status, or when an interrupt occurs. In short, the current task's execution time can be measured by reading a hardware timer at each context switch. In case a context switch occurs via an interruption, it is necessary to consider the overhead of handling the interrupt. In our approach, the overhead due to the interrupt service routine (ISR) is accounted for in the execution time of the task that is incurred by the ISR. For example, if network packets generate an interruption that induces the execution of the communication task, the execution times of the corresponding ISR and the recovery time from the interrupt state are added to the execution time of the communication task. Figure 1 illustrates the calculation of the execution time with the interrupt overhead considered. In the user data area of a task, three values are stored: the number of context switches (Context Switch Counter in Figure 1), the task's execution time in this period, and the task's total execution time so far.

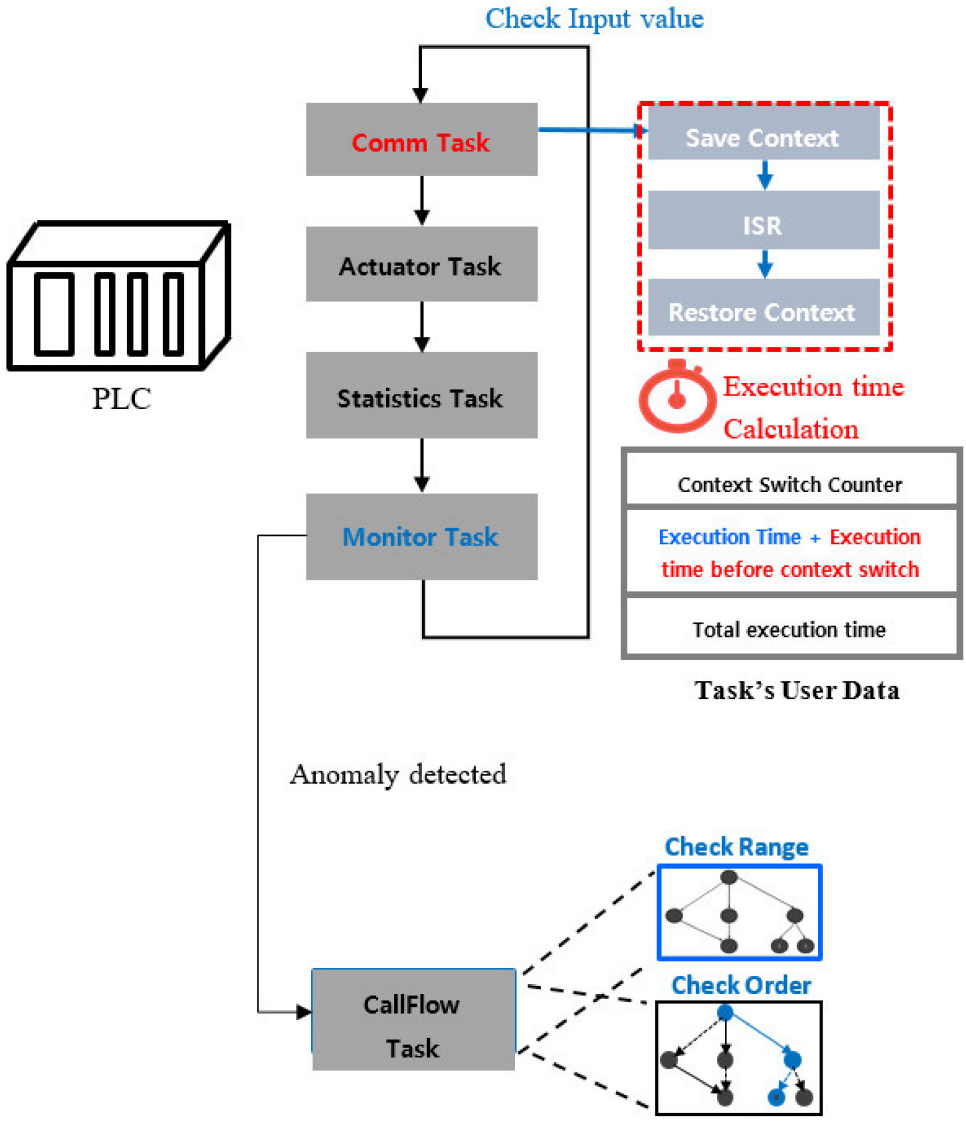


Figure 1. Measurement of task execution time and control-flow anomaly detection.

The measured execution times of tasks are tracked by a statistics task. Each PLC task has its period, so the CPU usage is calculated by (Task execution time)/(Task period). It is important that the *Statistics* task must not affect the schedulability of the PLC tasks. For this reason, the period of the *Statistics* task is determined as a common multiple of the tasks' periods. For example, if the *Comm* task and the *Actuator* task in Figure 1 have a period of 40 ms and 50 ms, respectively, the *Statistics* task's period can be a multiple of 200 ms. In this way, the execution of the *Statistics* task is harmonized with PLC tasks so that it does not interfere with the execution of the PLC tasks. Let us further assume that the worst-case execution time of the *Comm* task is 10 ms and that of the *Actuator* task is 20 ms, which makes their CPU usage 45% totally. If it takes 10 ms for the tracking job in the *Statistics* task, it requires 5% of the CPU usage, which can be accepted, for the system is not overloaded. In a real control system, the periods are not selected arbitrarily, rather determined to be harmonic for the performance and the stability [23]. Harmonic periods pairwise divide each other for any two periods; therefore, the least common multiple of the periods is equal to the longest period among tasks. Because a shorter period gets higher priority, the *Statistics* task has the lowest priority as not to interfere with other tasks; nevertheless,

its execution in a period can be finished no later than the task with the longest execution period.

With the execution time tracked by the *Statistics* task, the *Monitor* task detects when a task exceeds its pre-specified CPU utilization and determines whether the task behavior is anomalous. Because the CPU utilization of a task can fluctuate somewhat due to hardware interrupts or network jitters, two values α and β are used as configuration values. We can specify the additional utilization the system could bear (α), and how many times (β) a task should violate the utilization bound before it is determined as anomalous behavior. If a task utilizes α more than its pre-specified CPU utilization β times, the *Monitor* task alarms an anomaly detection to the administrator and discontinues the communication in which the task is involved.

3.2. Detecting Control-Flow Anomaly

If a temporal anomaly is detected, we need to find out whether it is caused by resource-exhausting requests such as DoS attacks, or by modifying control flows of PLC logic such as code injection attacks. To this end, we generate a call graph that represents all valid control flows of a task at compile-time and add a shadow stack to each task to store return addresses of every function call. Additionally, the *CallFlow* task that maintains the call graphs of tasks is introduced into the PLC system to check any problems in the sequence of function calls; it searches the shadow stacks for addresses out of normal range or out of normal order. The conceptual operation of the *CallFlow* task is shown in Figure 1.

The call graphs are generated at compile-time by analyzing the compiled machine codes using the information of the function address and its size. Each function is given an index, which becomes a node in a call graph. An edge in the call graph is a call from one function to another. The call graph also has the information of the start address and the size of functions, so we can find which function is executed and compare the current function call trace in the shadow stack with the call graph. The call graphs are stored in a secured memory area in PLC that can be accessed by the *CallFlow* task.

A shadow stack is used for protecting the return address information from a buffer overflow attack. It is a secondary, separate stack added to each task. A function call stores its return address to both the original task stack and the shadow stack. Usually, other security mechanisms use the shadow stack to detect the corruption in the task stack. However, in our scheme, the shadow stack is used to detect malicious control flow changes by an adversary. If a behavioral anomaly is detected, then the *CallFlow* task is triggered to check any anomaly by comparing the call trace in the shadow stack and the call graph of tasks. It checks whether the address is in the callable range of the functions appearing in the call graph, then whether the call sequence is a subgraph of the call graph. If an address is not matched with a node in the graph or the call sequence is not a subgraph of the call graph, it is considered a control-flow anomaly.

3.3. Implementation

We have implemented the proposed anomaly detection scheme on the uC/OS-II operating system that is a widely used real-time operating system for PLCs. All task information is stored in the Task Control Block (TCB) in the uC/OS-II. To monitor the execution time of tasks, we utilize the *Extension Pointer* in the TCB. The Extension Pointer can be used for including additional user data by storing the address to the data. The extension of TCB for temporal anomaly detection is illustrated in Figure 2.

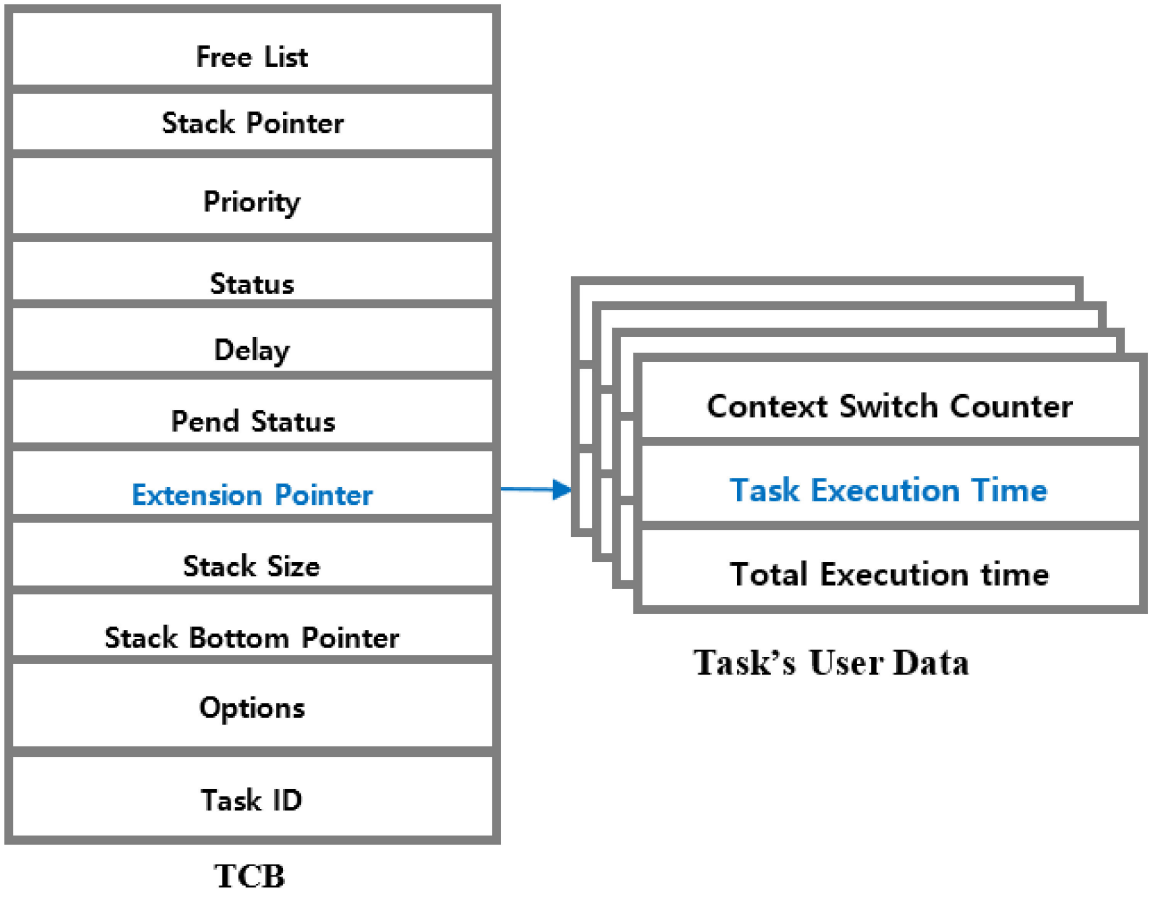


Figure 2. Task Control Block (TCB) extension for CPU usage monitoring.

A shadow stack is created at runtime. However, because the CPU does not have a register for the shadow stack, the information for the shadow stack is maintained in the TCB. We have modified the TCB to include the stack top, bottom, and size information as shown in Figure 3.

Call graphs are created by analyzing the linker map file containing addresses and sizes of functions. The function name, start address, and length are extracted from the linker map file, and the function is indexed in order of its appearance in the call graph. For example, if the *Comm* task calls *BSP_MOTOR* function first, it has the index of 1 in the call graph of the task. The call graph is stored in an array data structure in the flash memory. Figure 4 shows the actual call graph stored in the persistent memory of the PLC. The "Normal Sequence" in Figure 5 is a valid call sequence 1→2→5→6. The start address and the size of the functions are used for checking the range check, and the sequence information is used for checking function call sequences.

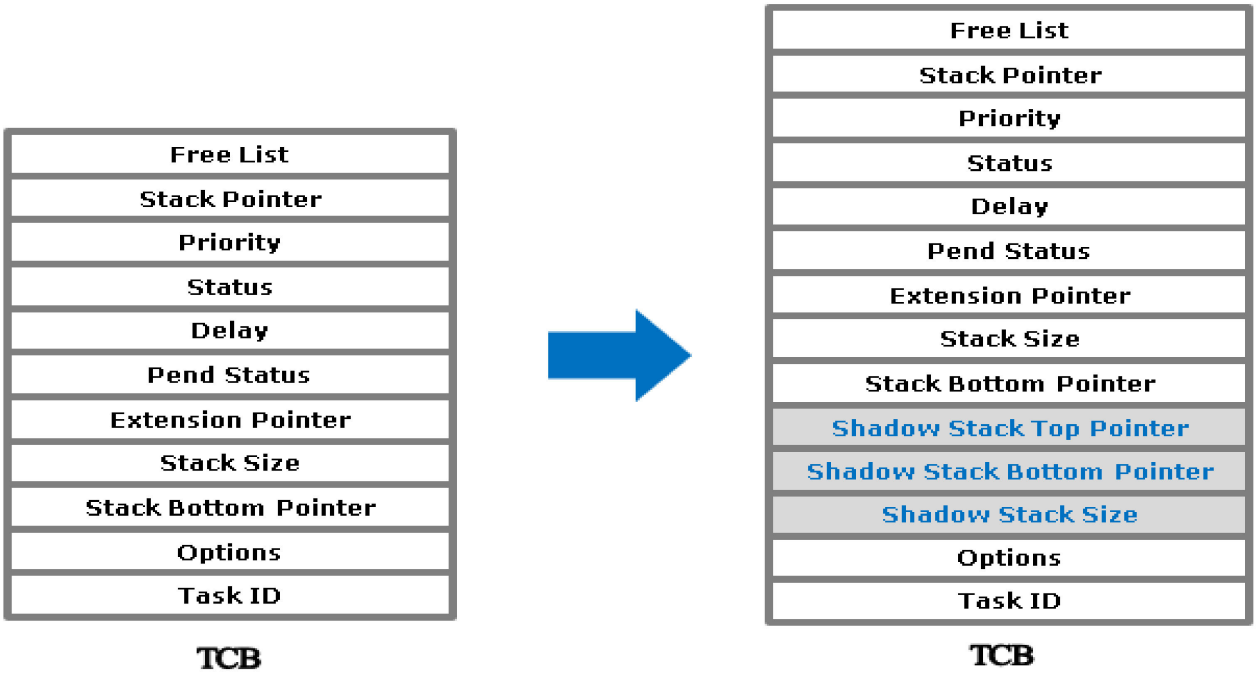


Figure 3. Task Control Block (TCB) with shadow stack information.

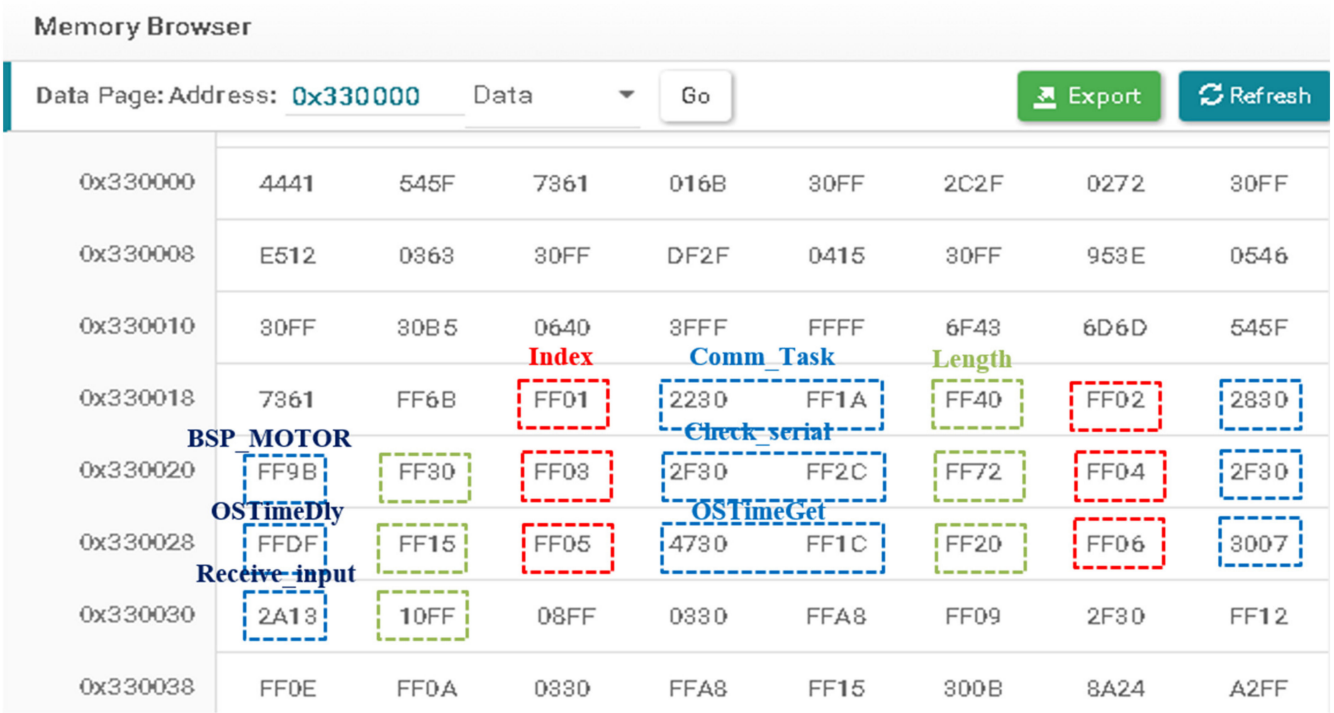


Figure 4. A call graph stored in memory.

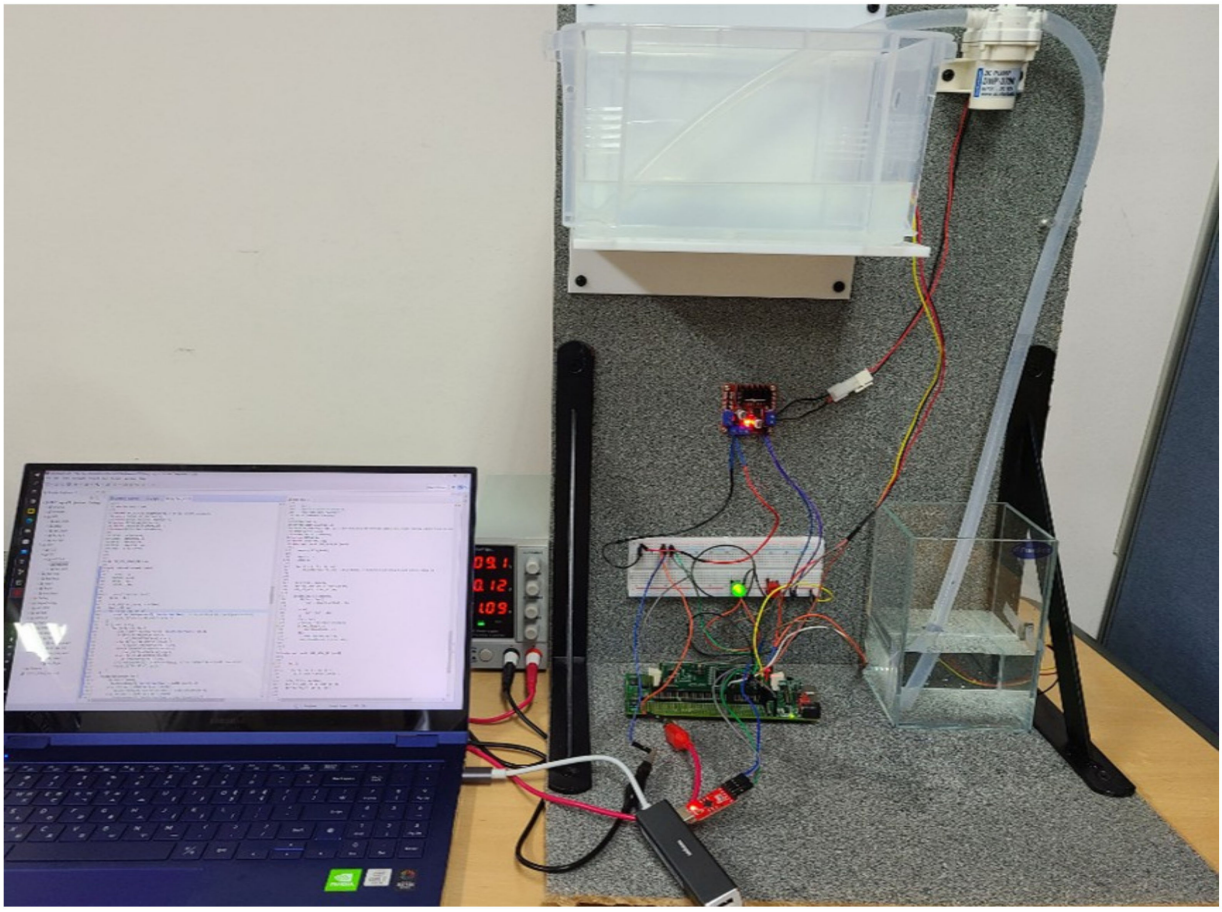


Figure 5. Water tank testbed.

4. Experimental Results

We have implemented the proposed method in the PLC controlling the water level of a water tank that can be adapted to various water treatment systems such as drain system, water supply system, and wastewater treatment system. Figure 5 shows the implemented testbed for the experiments. The laptop and the embedded board with PLC logics communicate with UART (Universal Asynchronous Receiver/Transmitter). To make the experiments as close as a real PLC, we configured the hardware based on the POSAFE-Q [24] that is a PLC system used for controlling a nuclear reactor. Table 1 summarizes the hardware configuration of the testbed. The PLC system used in the experiment has the same MCU, memory, and the same operating system as the real one.

Table 1. Hardware specifications.

Component	Specification
PLC hardware	TMDSDOCK28335 MCU: TMS320 (32-bit @150 MHz) Memory: 512KB
Operating system	uC/OS-II
Water level sensor	FIT0212
UART	CP2102 (300~1 Mbps)
Water pump	DWP-370N
Motor driver	L298 (2 A, 5 V~35 V)

Figure 6 illustrates how the testbed is organized and works. The PLC scans the sensor to detect the water level. If it is higher than the threshold level, the PLC commands the motor drive module to operate the water pump to transfer water from the lower tank to the higher tank. The temporal anomaly detection software has been implemented to the PLC, and it sends an alarm message to the user at ESW when an anomaly is detected.

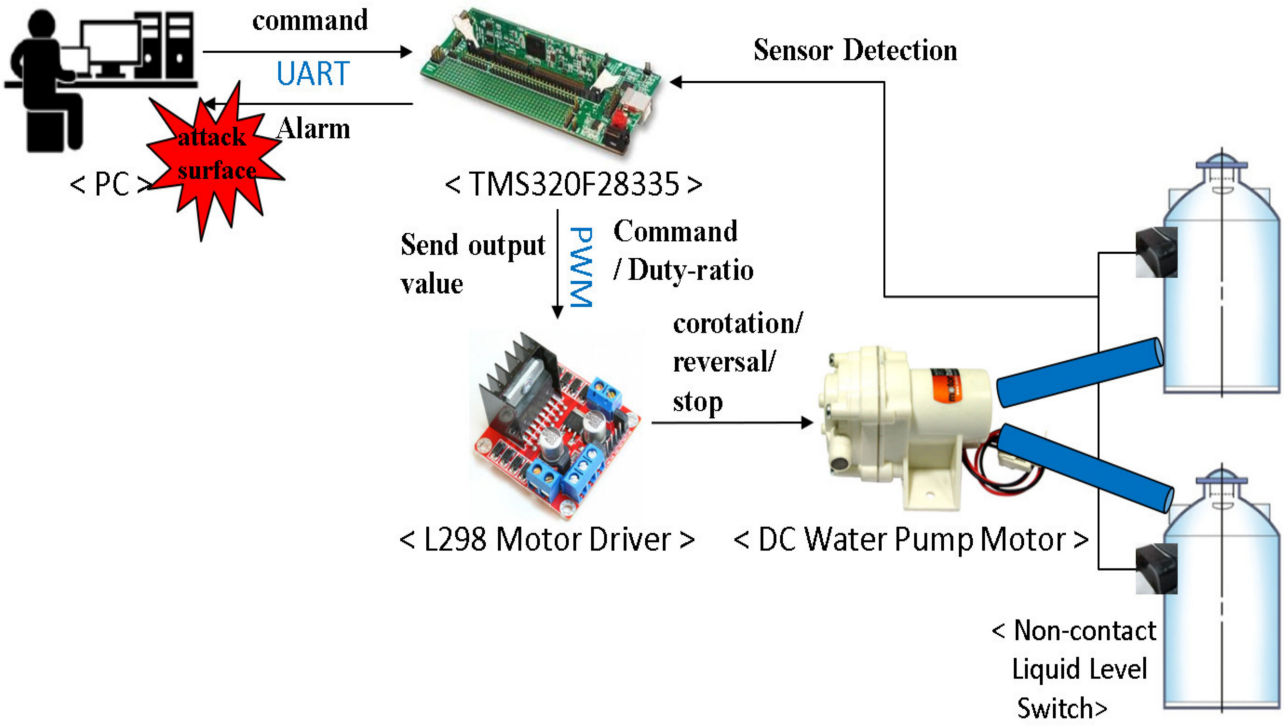


Figure 6. Configuration of the testbed.

Tasks in the system are shown in Table 2. The *Comm* and *Control* tasks perform PLC operating cycles. Both tasks run every 40 ms periodically, utilizing the processor in total by 37.5%. The *Statistics* task keeps track of the CPU utilization of *Comm* and *Control* tasks, for which it requires 20% of the CPU utilization. To detect changes in temporal behavior of the PLC tasks, the *Monitor* task is executed at every two consecutive runs of the *Statistics* task. Temporal anomaly detection in this testbed system requires 28.75% of CPU utilization including the overhead of periodic monitoring CPU utilization, but the schedulability of the system is not affected because the execution period of the *Statistic* and the *Monitor* tasks are harmonically arranged with the PLC tasks, and the total utilization is only 66.25%. The *CallFlow* task is executed only when a behavioral anomaly is detected; thus, the task is invoked sporadically.

Table 2. Tasks in the PLC.

Task	Priority	Execution Time	Period	Description
Comm	1	5 ms	40 ms	communication with EWS and devices
Control	2	10 ms	40 ms	motor control logic
Statistics	3	8 ms	40 ms	tracking CPU utilization
Monitor	4	7 ms	80 ms	detecting behavioral anomaly
CallFlow	4	4 ms	-	detecting control-flow anomaly

To assess the system overhead imposed by monitoring tasks' execution time, we have measured the time required to perform context switching in the operating system. Figure 7 compares the context switching time before and after adding the CPU usage monitor, where *Y*-axis represents the time for context switching in milliseconds, and *X*-axis the number of context switches. Before monitoring the execution time, the context switching time (CTX_TIME2) was about 0.024 ms in the steady state. The context switching time, including the monitoring overhead (CTX_TIME), is slightly increased to about 0.028 ms, or a 0.004 ms increase from the original value. Although it is about a 16.7% rise in the context switching overhead, it does not affect the execution of tasks because the overhead is negligible in terms of the CPU utilization, which is about 0.02% for two context switches in a 40 ms period.

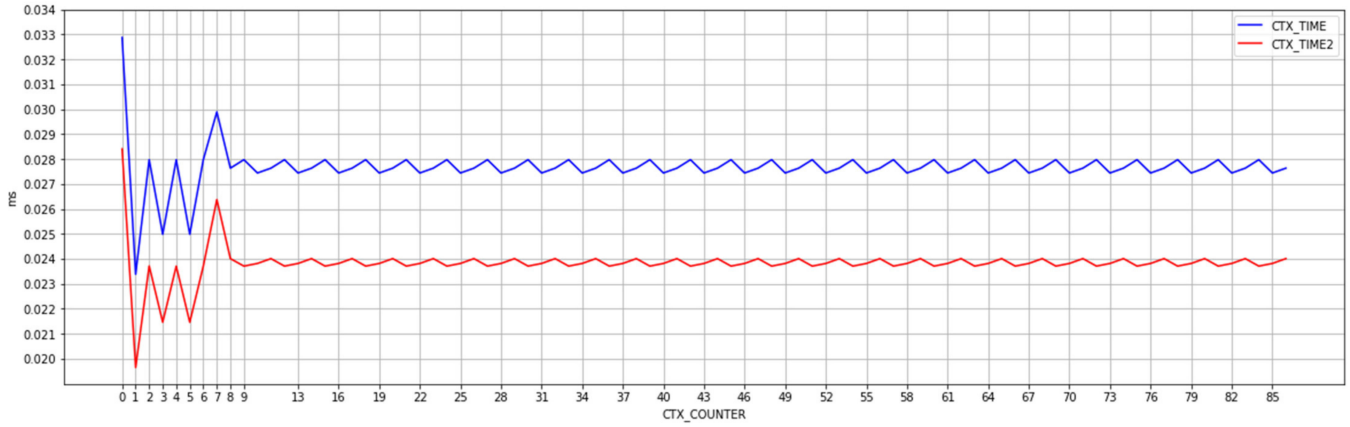
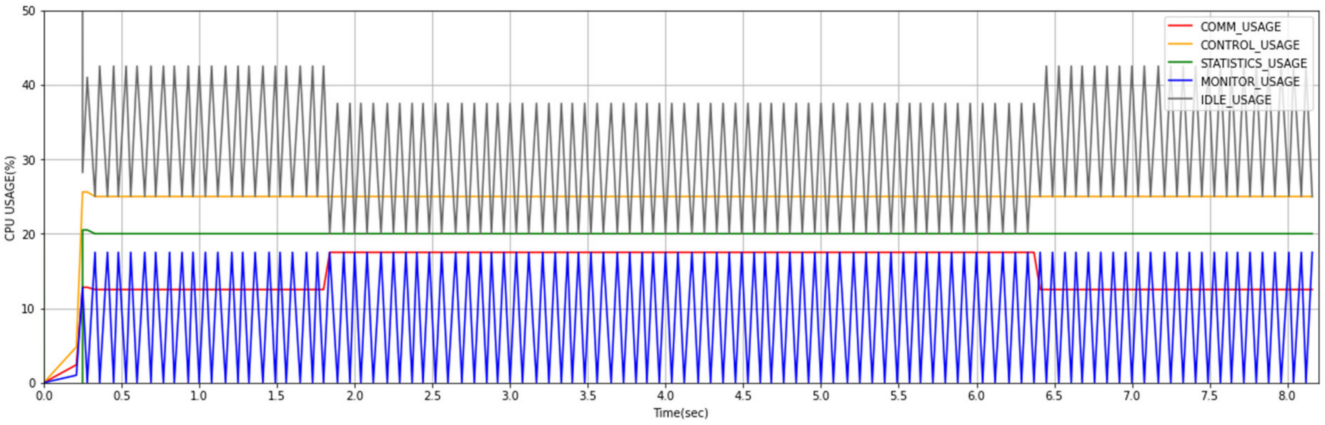


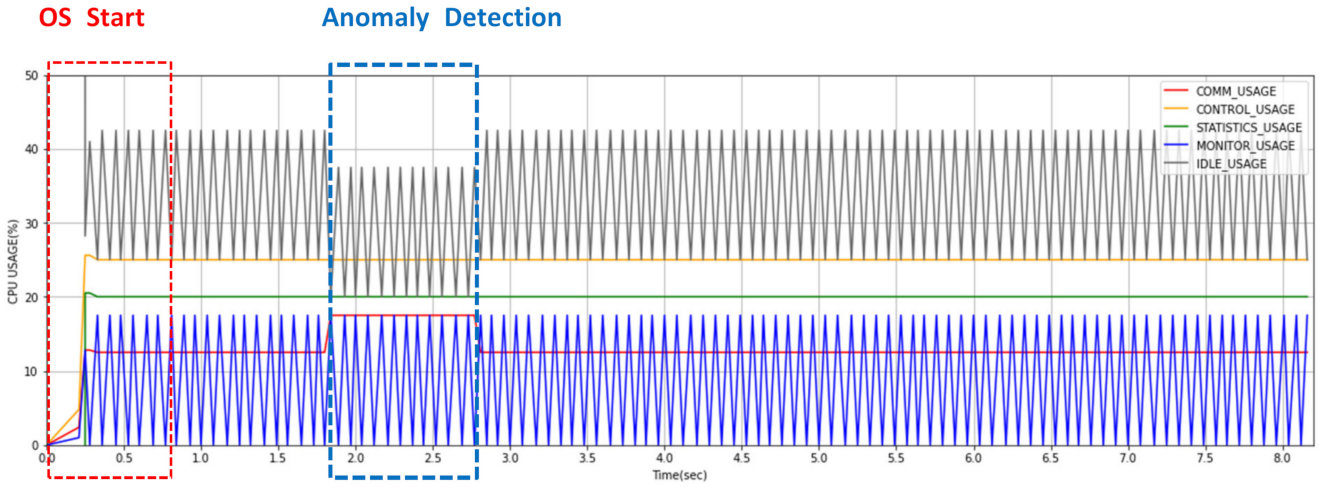
Figure 7. Context switching overhead.

First, we demonstrated a DoS attack by indiscriminate packet transmission by using boofuzz, which supports serial fuzzing [25]. Sending data packets every 0.01 s caused PLC communication down, which subsequently led to the shutdown of the motor driver. Figure 8a shows that the CPU usage of the *Comm* task had risen from 12.5% at its stable state to over 20% when the system was attacked. This effect of the DoS attack resulted in the system being overloaded, consequently failing to operate properly.

In the next experiment, the proposed anomaly detection method was deployed on the PLC. The *Monitor* task seeks any changes in CPU usage, and if the change is above $\alpha = 5\%$ for $\beta = 10$ periods of time, it reports the behavior as an anomaly. Once an anomaly is detected, an alert is alarmed and the ESW can block the packet transmission by disconnect the problematic communication. In Figure 8b, we can see that the proposed method detected the DoS attack of indiscriminate packet transmission around time 18–28 s and stabilized the system by preventing a further attack.



(a) CPU usage change due to a DoS attack



(b) Anomaly detection and prevention of further attacks

Figure 8. Temporal anomaly detection and recovery.

5. Conclusions

Cyberattacks on ICSs may cause serious damage in safety-critical systems that are closely related to the safety of society and people. Thus, it is important to protect PLCs from malicious attacks, for PLCs are at the border between the computerized control system and the physical system. In this work, we have proposed an anomaly detection scheme based on the temporal behavior of tasks. In the operating system, a monitor tracks the CPU usage of tasks by measuring the execution time of tasks. Because the PLC control tasks are real-time in nature, they run periodically with a predefined frequency. If the monitor detects a task overruns, i.e., the CPU usage of the task exceeds the allowable threshold, it sends an alert to the engineering workstation. Moreover, it triggers the control-flow checking task to examine if the task's control flow is corrupted using the shadow stack and the pre-generated call graph.

We have implemented the proposed scheme into uC/OS-II on PLC hardware in a water tank control system. In experiments of the DoS attack on the implemented system, we can see that the proposed mechanism can detect massive packet transmission attacks and can be used to avoid resource exhaustion that can cause system failure.

For future work, we need to optimize the performance of anomaly detection algorithms. Although we can make the anomaly detection tasks execute without interfering

with the PLC tasks by adjusting the execution period, that period could become longer if the PLC tasks' utilization becomes higher, which results in delayed detection of anomalies. By developing a more efficient calculation method in CPU usage monitoring, the proposed method can be adapted to a wider range of PLCs.

Author Contributions: Data curation, S.H.; Software, S.H.; Validation, K.L.; Writing—original draft, M.P.; Writing—review and editing, S.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Incheon National University Research Fund in 2020.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dragos, 2019 Year in Review ICS Vulnerabilities. Available online: https://www.dragos.com/reviw/2019-ics-year-in-review-ics-vulnerabilities/ (accessed on 5 April 2021).
2. Positive Technologies, ICS Vulnerabilities: 2018 in Review. Available online: https://www.ptsecurity.com/ww-en/analytics/ics-vulnerabilities-2019/ (accessed on 5 April 2021).
3. Yoo, H.; Irfan, A. Control logic injection attacks on industrial control systems. In *IFIP International Conference on ICT Systems Security and Privacy Protection*; Springer: Cham, Germany, 2019.
4. Shin, H.K.; Lee, W.; Yun, J.H.; Kim, H. Implementation of programmable CPS testbed for anomaly detection. In Proceedings of the 12th USENIX Workshop on Cyber Security Experimentation and Test, Santa Clara, CA, USA, 12 August 2019.
5. Falliere, N.; Murchu, L.O.; Chien, E. W32.Stuxnet dossier: White paper. *Secur. Response* **2011**, *5*, 29.
6. Senthivel, S.; Dhungana, S.; Yoo, H.; Ahmed, I.; Roussev, V. Denial of engineering operations attacks in industrial control systems. In Proceedings of the ACM Conference on Data and Application Security and Privacy, Tempe, AZ, USA, 19–21 March 2018.
7. Sayegh, N.; Chehab, A.; Elhajj, I.H.; Kayssi, A. Internal security attacks on SCADA systems. In Proceedings of the 3rd International Conference on Communications and Information Technology, Beirut, Lebanon, 19–21 June 2013.
8. Abbasi, A. Ghost in the PLC: Stealth on-the-fly manipulation of programmable logic controllers' I/O. In Proceedings of the Black Hat EU, London, UK, 1–4 November 2016.
9. Ylmaz, E.N.; Ciylan, B.; Gönen, S.; Sindiren, E.; Karacayılmaz, G. Cyber security in industrial control systems: Analysis of DoS attacks against PLCs and the insider effect. In Proceedings of the 6th International Istanbul Smart Grids and Cities Congress and Fair, Istanbul, Turkey, 25–26 April 2018; pp. 81–85.
10. Xiao, Y.; Xu, W.; Jia, Z.; Qi, D. NIPAD: A non-invasive power-based anomaly detection scheme for programmable logic controllers. *Front. Inf. Technol. Electron. Eng.* **2017**, *18*, 519–534. [CrossRef]
11. Shahzad, A.; Lee, M.; Lee, Y.-K.; Kim, S.; Xiong, N.; Choi, J.-Y.; Cho, Y. Real time ModBus transmissions and cryptography security designs and enhancements of protocol sensitive information. *Symmetry* **2015**, *7*, 1176–1210. [CrossRef]
12. Bhatia, S.; Kush, N.; Djamaludin, C.; Akande, J.; Foo, E. Practical ModBus flooding attack and detection. In Proceedings of the 12th Austrian Information Security Conference, Auckland, New Zealand, 20–23 January 2014; Volume 149, pp. 57–65.
13. Ahmed, I.; Obermeier, S.; Sudhakaran, S.; Roussev, V. Programmable logic controller forensics. *IEEE Secur. Priv.* **2017**, *15*, 18–24. [CrossRef]
14. MITRE. CWE-400: Uncontrolled Resource Consumption. Available online: https://cwe.mitre.org/data/definitions/400.html (accessed on 5 April 2021).
15. Niedermaier, M.; Malchow, J.O.; Fischer, F.; Marzin, D.; Merli, D.; Roth, V.; von Bodisco, A. You snooze, you lose: Measuring PLC cycle times under attacks. In Proceedings of the 12th USENIX Workshop on Offensive Technologies, Baltimore, MD, USA, 13–14 August 2018.
16. Long, M.; Wu, C.-H.; Hung, J.Y. Denial of service attacks on network-based control systems: Impact and mitigation. *IEEE Trans. Ind. Inform.* **2005**, *1*, 85–96. [CrossRef]
17. Markovic-Petrovic, J.D.; Stojanovic, M.D. Analysis of SCADA system vulnerabilities to DDoS attacks. In Proceedings of the 11th international conference on telecommunications in modern satellite, cable and broadcasting services, Nis, Serbia, 16–19 October 2013; pp. 591–594.
18. Horak, T.; Strelec, P.; Huraj, L.; Tanuska, P.; Vaclavova, A.; Kebisek, M. The vulnerability of the production line using Industrial IoTs systems under DDoS attack. *Electronics* **2021**, *10*, 381. [CrossRef]
19. Kalle, S.; Ameen, N.; Yoo, H.; Ahmed, I. CLIK on PLCs! attacking control logic with decompilation and virtual PLC. In Proceedings of the Workshop on Binary Analysis, San Diego, CA, USA, 24 February 2019.
20. Saranyan, S.; Ahmed, I.; Roussev, V. SCADA network forensics of the PCCC protocol. *Digit. Investig.* **2017**, *22*, S57–S65.
21. Biham, E.; Bitan, S.; Carmel, A.; Dankner, A.; Malin, U.; Wool, A. Rogue7: Rogue Engineering-Station attacks on S7 Simatic PLCs. In Proceedings of the Black Hat USA 2019, Las Vegas, NV, USA, 3–8 August 2019.
22. Jeong, E.; Park, J.; Oh, I.; Kim, M.; Yim, K. Analysis on account hijacking and remote DoS vulnerability in the CODESYS-based PLC runtime. In Proceedings of the International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Lodz, Poland, 1–3 July 2020; pp. 457–467.

23. Mohaqeqi, M.; Nasri, M.; Xu, Y.; Cervin, A.; Årzén, K.-E. Optimal harmonic period assignment: Complexity results and approximation algorithms. *Real-Time Syst.* **2018**, *54*, 830–860. [CrossRef]
24. Kwon, K.-C.; Lee, M.-S. Technical review on the localized digital instrumentation and control systems. *Nucl. Eng. Technol.* **2019**, *41*, 447–454. [CrossRef]
25. Boofuzz: Network Protocol Fuzzing for Humans. Available online: https://github.com/jtpereyda/boofuzz (accessed on 5 April 2021).