A Malware-Tolerant, Self-Healing Industrial Control System Framework

Michael Denzel, Mark Ryan, and Eike Ritter

University of Birmingham, School of Computer Science,
B15 2TT Birmingham, United Kingdom
{m.denzel,m.d.ryan,e.ritter}@cs.bham.ac.uk

Abstract. Industrial Control Systems (ICSs) are computers managing many critical infrastructures like power plants, aeroplanes, production lines, etc. While ICS were specialised hardware circuits without internet connection in former times, they are nowadays commodity computers with network connection, TCP/IP stack, and a full operating system, making them vulnerable to common attacks. The defensive mechanisms, however, are still lacking behind due to the strong requirement for availability of ICSs which prohibits to deploy typical countermeasures like e.g. an anti-virus. New techniques are needed to defend these systems under their distinct prerequisites.

We introduce the concept of a malware-tolerant ICS network architecture which can still operate securely even when some components are entirely compromised by an attacker. This was done by replacing all single point-of-failures with multiple components verifying each other. We provide ProVerif proofs to show the correctness of the network protocol one-by-one assuming each device compromised.

Furthermore, we added a self-healing mechanism based on invariants to the architecture on network as well as system level which will reset failed or compromised systems. To demonstrate system level self-healing, we implemented it on top of FreeRTOS and ARM TrustZone. The network level self-healing was incorporated into the ProVerif proofs by formally verifying the absence of type 1 (falsely identified attacks) and type 2 errors (missed attacks).

Keywords: Malware Tolerance, Self-Healing, Industrial Control System (ICS), Security

1 Introduction

Industrial Control Systems (ICSs) received a lot of media attention with the Stuxnet attack [18]. But there are also more examples like Duqu, Flame, Red October, MiniDuke [31], Gauss, Energetic Bear, Epic Turla [15], and the attack on a German steel mill [4].

ICSs are sensor-actuator networks that control physical systems. The core components are so-called Programmable Logic Controllers (PLCs), which nowadays are essentially commodity computers with specialised software to satisfy the requirement for high availability and real-time operation. Due to these requirements, they cannot run common defensive measures like an anti-virus. Defensive mechanisms have, thus, to be deployed (less effective) elsewhere in the network. Moreover, PLCs have a long lifetime (10-20 years) and are not usually patched to avoid downtime and bricking the devices. [26] A corrupted patch can render a PLC unusable possibly leading to a shutdown of part of the network which is potentially life-threatening. In combination with historic protocols which do not even offer basic authentication (like the Modbus protocol [10]) these systems fall in the hands of attackers as soon as the attacker has network access.

Governmental organisations [13, 26] recommend a strategy called "defence in depth" which tries to deploy defences at every layer of the network. We want to go one step further and, instead of only defending problematic devices, we aim to distribute trust over several independent components in a way that an individual component infected with malware cannot break the security policy. We call this approach *malware-tolerance*. Simply put, we want to remove every single point-of-failure at critical intersections throughout the entire ICS architecture. Our secondary goal is to enable the architecture to automatically repair ordinary and malicious faults, so-called self-healing. With this approach, it is also possible to recover from corrupted or incomplete patches.

Contributions:

- We design the architecture of a malware-tolerant ICS that has no single point-of-failure at critical intersection points and can self-heal failed or (maliciously) misbehaving PLCs.
- We also formally prove the network architecture with state-of-the-art protocol verifier ProVerif[1]. The proofs can be found online[2].
- To achieve our architecture, we also develop a self-healing mechanism which detects incorrect behaviour by verifying invariants, and recovers to a good state. We adjusted FreeRTOS[3] to include our mechanism and released our implementation as open-source[4].

2 Overview

2.1 Traditional Industrial Control System Architecture

Traditional ICSs separate the network into zones which are isolated from each other by firewalls resulting in a layered network with Intrusion Detection System (IDS) at intersection points (defence in depth). The innermost part of an ICS is called *control loop* and consists of a PLC as well as sensors and actuators. This part of the system is the actual cyber-physical system and has hardly any (often no) defensive measures apart from the firewall in front of it due to availability

[1] `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`

[2] `https://github.com/mdenzel/malware-tolerant_ICS_proofs`

[3] `www.freertos.org`

[4] `https://github.com/mdenzel/self-healing_FreeRTOS`

and real-time constraints. The control loop can be at a different location (*field site*) than the control centre. An example for such a control loop would be a temperature control system with e.g. some water tanks which should neither freeze nor boil. The PLC would read the temperature from a sensor and adjust the heating/cooling of the water to maintain a temperature between 0 to $100°C$.

2.2 Assumptions

1. We assume a Dolev-Yao attacker [8] on the network who interacts with software-side technologies. The attacker has no physical access to the facilities and cannot change cabling or remotely introduce electrical signals directly into wires (apart from assumption 2).
2. Additionally, the attacker can choose one[5] device (except the actuator) of which he gains full control − i.e. also access to corresponding cryptographic keys and "software" access to the physical wires connected to the chosen device. If the attacker chooses e.g. a PLC, he has access to the connected sensors, can read their values, but cannot change wiring or sensors. The attacker can manipulate the hardware of a chosen device once (during production) but has no physical access afterwards any more. If the chosen device has network access, the attacker can update software and firmware.
3. We are initially not aware which device the attacker chose.
4. We assume the 2-out-of-3 circuit is hardware-only and in scope of verification.
5. Attacks on cryptography and phishing attacks are out of scope.
6. PLCs and sensors work synchronously or are buffered.

2.3 Proposed Architecture

Our approach is an extension to already existent firewalls, network zones, IDSs etc. and changes the control loop at the field site. Figure 1 displays our infrastructure with the changes being highlighted in red. Our concept adds hardware in form of reset-circuits; data by images and policies; and software in form of a self-healing Real-Time Operating System (RTOS) and the netboot firmware of the reset-chip. Additionally, we leverage existent redundancy of PLCs and a 2-out-of-3 (2oo3) circuit which are already in place in some ICS facilities.

Basis of the malware-tolerant architecture are three diverse PLCs combined with trusted computing. The 2-out-of-3 hardware circuit combines the results of the PLCs and forward them to the actuator. That means none of the PLCs has to be invulnerable to attacks or failures, it is enough if two of the three work. The PLCs must differ in their soft- as well as hardware which we achieve with a special kind of N-variant system and diverse hardware (details in section 4.2).

We also added self-healing functionality that can recover failed and compromised PLCs with (1) a RTOS based on ARM TrustZone that can reset user level tasks and (2) a network protocol and reset-circuits to defend against attacks on system level and on the Trusted Execution Environment (TEE).

[5] We only show the basic case of an attacker compromising one system. Tolerating attacks on multiple systems is more challenging but similarly possible.

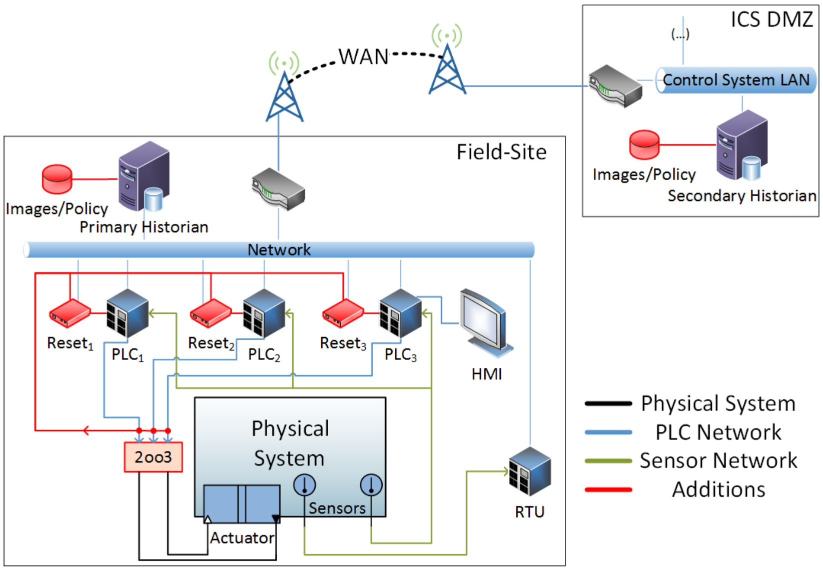

Fig. 1: Proposed Industrial Control System Architecture

Self-Healing RTOS: To demonstrate the RTOS, we created a proof-of-concept implementation based on ARM TrustZone and the FreeRTOS operating system which we ported to ARM TrustZone to protect critical functionality like scheduling and interrupts. ARM TrustZone, is a TEE which consists of two separated environments: the *secure world* and the *normal world*. While the secure world has full access to the system, the normal world is restricted in its capabilities. The switch between the two worlds is handled by the so-called *monitor*. TrustZone chips usually come with the TrustZone Interrupt Controller (TZIC) and functionality to manage memory, i.e. a TrustZone-aware Memory Management Unit, routines to forbid Direct Memory Access, and so on. We refer to the ARM documentation [2] for more details.

Figure 2 shows the control flow of our TrustZone-aware RTOS. Periodically, the TZIC will generate a timer interrupt (1.) which is setup as an Fast Interrupt (FIQ) trapping into monitor mode (2.). The monitor will save the context and jump to the interrupt handler (3.) which will, for timer interrupts, call the FreeRTOS scheduler (4.). After scheduler (5.) and interrupt handler (6.) return, the next *task* is determined. At this point, the monitor will invoke a detection routine (7.). To reveal faults or malicious behaviour of certain tasks, the detection routine checks various system variables and external values (e.g. sensor values) against invariants which are stored in form of a policy, cryptographically signed, on at least two servers. These invariants are implicitly given by the setpoints the operator of the system placed. For our water tank example, the operator could e.g. set the temperature t to 0 to $100°C$, forbid heating for $t > 50°C$, and forbid cooling for $t < 50°C$. If the temperature is below or above this range

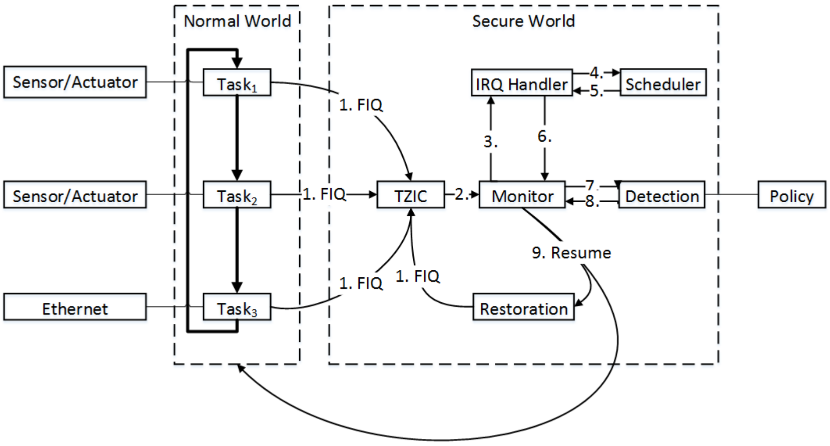

Fig. 2: TrustZone-aware Real-Time Operating System

and the task does not enable the actuator, our task is faulty. The result of the detection routine is returned to the monitor (8.) which then (9.) either runs the task or dispatches a restoration routine if the task was misbehaving. The restoration routine only runs during the time-slice of the misbehaving task which ensures availability of the rest of the system including other tasks and operating system functionality. The restoration terminates the task and loads an image of the original task from a protected memory region inside the secure world. Lastly, the task is added to the scheduler again. The critical steps are run inside the TrustZone secure world to protect them from manipulation.

To avoid unnecessary resets due to false positives, we created a specification-based detection technique. These have a lower rate of false alarms than non-specification-based techniques but might miss some attacks. [24] If the presented online self-healing mechanism fails, the network level self-healing approach (see following paragraph) will restore the particular PLC but at the cost of a restart.

Reset-Circuits and Network Protocol: Our reset-circuits consist of a network boot chip (e.g. iPXE[6]) and a logical circuit to control resets (Fig. 3). A low frequency clock signal restricts resets to a certain interval. Optionally, the inputs to the circuit (label 1. in Fig. 3) can be replaced with flipflops to enable synchronising the PLCs. Circuits for PLC_2 and PLC_3 can be similarly derived.

Since network-based detection indicates that system level self-healing (taking place beforehand) failed, we intentionally clear the state to recover from the attack. We re-initialise state either by discovery or by requesting it from the other PLCs. In our temperature maintenance example, discovering the temperature and adjusting the actuator is straight forward. For more complex scenarios, the reset PLC would request the state from the other two PLCs and compare it.

[6] www.ipxe.org

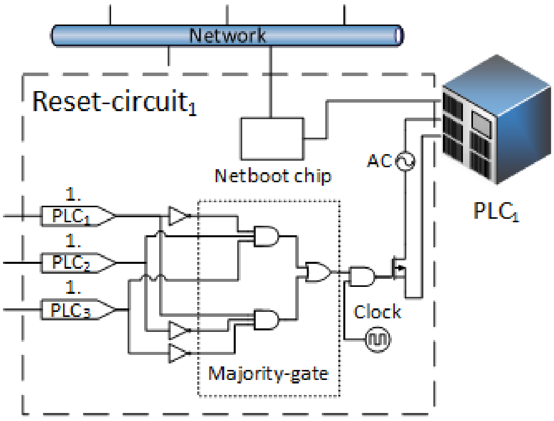

Fig. 3: Reset-circuit for PLC_1

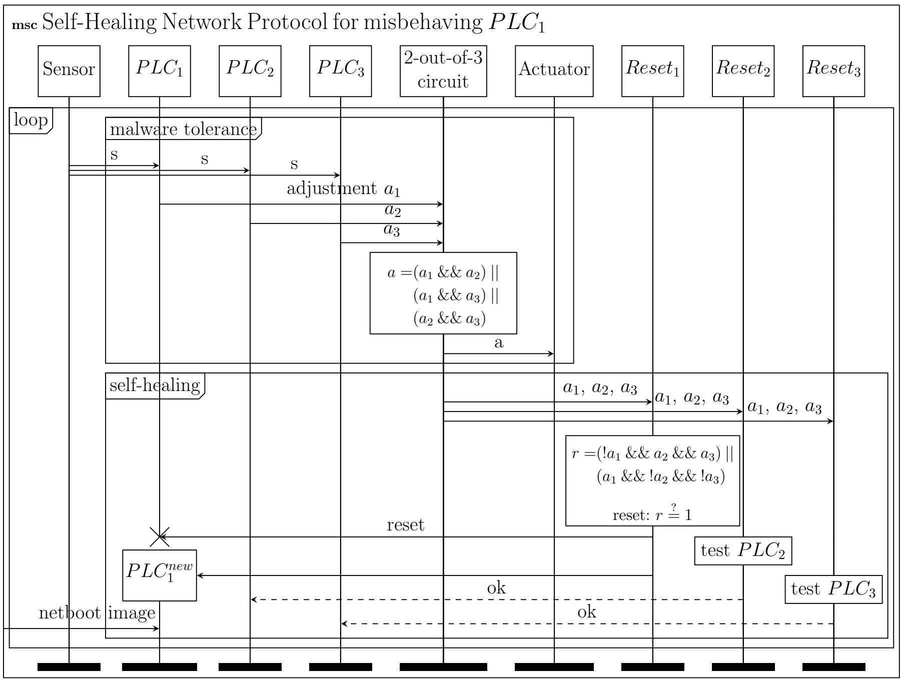

Fig. 4: Malware-tolerant, self-healing protocol

The full message sequence chart of our malware-tolerant, self-healing network protocol is presented in Fig. 4: Every PLC reads the current sensor value s (for simplicity we drew only one sensor but multiple sensors are possible) and computes the adjustment a_i of the actuator. This is sent to the 2-out-of-3 circuit which forwards the end result a to the actuator. Parallel, each reset-circuit receives the three response values of the PLCs, each checks if a reset for its corresponding PLC should happen ($r \overset{?}{=} 1$), and resets it if this is the case. Rebooting PLCs load the netboot image from the network. The figure displays a reset of PLC_1 as an example.

3 Security Analysis and Results

3.1 ProVerif Proofs

To give evidence of the security features of our architecture, we utilised ProVerif – a state-of-the-art protocol verifier – to test our network protocol. We modelled the protocol as shown in Fig. 4 for various configurations (see Table 1) where we grant the attacker control over different sets of devices.

In order to reason about malware-tolerant systems, we analysed the system based on multiple Trusted Computing Bases (TCBs). A TCB is the minimum set of honest components needed to secure the system; multiple such sets can exist. A system is malware-tolerant if there are at least two disjoint TCBs that provide the same property. Each independent component of the system – i.e. mostly entire devices as e.g. the CPU depends on the computer and is not an independent component – can be part of multiple TCBs.

With this notation, we can reason about a system based on TCBs. The system is secure if any TCB_i is secure. E.g. if device d_1 and d_2 are part of TCB_1 and device d_2 and d_3 are part of TCB_2, then the system is secure if $(d_1 \wedge d_2) \vee (d_2 \wedge d_3)$ is secure. As we can see here, d_2 is part of all TCBs and is, thus, the single point-of-failure. The system is not malware-tolerant because there are no disjoint TCBs (TCB_1 and TCB_2 overlap).

The TCB model of our proposed ICS is shown in expression 1. R_i stands for the reset-circuit of PLC number i. Since R_i controls PLC_i, we have to consider (R_i, PLC_i) pairs as they are the smallest subset of *independent* components. Our system is malware tolerant, because no (R_i, PLC_i) pair is part of all TCBs.

$$\begin{aligned} TCB_1 &= \{(R_1, PLC_1), (R_2, PLC_2)\} \, . \\ TCB_2 &= \{(R_1, PLC_1), (R_3, PLC_3)\} \, . \\ TCB_3 &= \{(R_2, PLC_2), (R_3, PLC_3)\} \, . \end{aligned} \tag{1}$$

We formally verified our architecture by testing five properties of our protocol (Fig. 4) with ProVerif (results shown in Table 1):

1./2. $1^{st}/2^{nd}$ iteration: As ProVerif cannot verify loops, we modelled two iterations of the protocol. These two iterations are sufficient, since computations are independent from each other and resets only affect the next loop iteration. For each iteration, we tested if the actuator received the correct value.

3./4. Self-Healing: The self-healing column in Table 1 consists of two proofs; the absence of type I errors and the absence of type II errors.

  - Type I error (false positive): Regarding our protocol, a false positive is the case where we reset an honest PLC. In ProVerif this has to be expressed as: For all *reset* events of PLC_i, PLC_i misbehaved.
    At first, this seems to not prove that *if PLC_i misbehaved, a reset happens* but in combination with the type II error and knowing that reset is a binary event (it can either happen or not happen), we prove the property.
  - Type II error (false negative): False negative refers to the case where a misbehaving PLC is not reset (missed attack). In ProVerif: For all *not_reset* events of PLC_i, PLC_i behaved correctly.
    Again, it seems that *if PLC_i is honest, no reset happens* is not proven. This is applicable, similar to before, by the absence of type I errors.

5. End reached: We tested if the protocol runs through. This is done in ProVerif by detecting the deliberate leak of a secret value at the end of the protocol.

№	Compromised Devices	1^{st} Iteration	2^{nd} Iteration	Self-Healing		End reached
		(1.)	(2.)	(3.)	(4.)	(5.)
1	None	✓	✓	✓	✓	✓
2	PLC_1	✓	✓	✓	✓	✓
3	PLC_2	✓	✓	✓	✓	✓
4	PLC_3	✓	✓	✓	✓	✓
5	2oo3					✓
6	R_1	✓	✓			✓
7	R_2	✓	✓			✓
8	R_3	✓	✓			✓
9	PLC_1, R_1	✓	✓			✓
10	PLC_2, R_2	✓	✓			✓
11	PLC_3, R_3	✓	✓			✓
12	PLC_1, R_2	✓				✓
13	PLC_1, PLC_2					✓
14	PLC_1, 2oo3					✓
15	2oo3, R_1					✓
16	R_1, R_2	✓				✓
17	PLC_{1-3}					✓
18	PLC_{1-2}, 2oo3					✓
19	PLC_1, 2oo3, R_1					✓
20	2oo3, R_{1-2}					✓
21	PLC_1, R_{1-2}	✓				✓
22	PLC_1, R_{2-3}	✓				✓
23	R_{1-3}	✓				✓
24	All					✓

Table 1: ProVerif Results

The proofs (Table 1) show that the physical system, i.e. the actuator, is supplied with correct values for the cases where the adversary controls one device (cases 2–4 and 6–8) or one $(PLC_i, Reset_i)$-pair (cases 9–11). Self-healing works for one compromised PLC but functional reset-circuits (cases 2–4). Also, everything works if there is no attack (case 1). We tested more hypothetical cases as sanity checks, e.g. the case where the attacker can physically change the 2-out-of-3 circuit (case 5) which is a validation of our assumption. The expected result is that the protocol fails since we hand the asset to the attacker from the very beginning. Case 24 is a special sanity check where we give the adversary control over literally everything. Both cases (case 5 and 24) fail as predicted.

3.2 Evaluation of Self-Healing FreeRTOS

Since we are not aware of any Common Vulnerabilities and Exposures (CVE) for FreeRTOS, we could not test any real-world attacks against our extended FreeRTOS operating system (a broader analysis of attacks follows in section 4.1).

To test the system level self-healing capability of our proof-of-concept implementation, we introduced a buffer overflow in our Input/Output (I/O) driver using the vulnerable C-function *strcpy*. We exploited this vulnerability by overflowing the buffer and overwriting the settings in the PLC. We chose this attack as it is the most common vulnerability in C and is similar to a range of attacks, e.g. format string attacks and return-oriented-programming.

Our simplified detection routine checks that the settings are within an accepted range and otherwise triggers restoration. The results show that the task and the temperature driver are reset to their original if the maximum temperature is changed to values outside the range (for more details see code of adjusted FreeRTOS implementation).

3.3 Performance Analysis of TrustZone

We conducted a performance analysis of the TrustZone world switch on a Free-Scale i.MX53 Quick Start Board with 1 GB DDR3 SDRAM running a 1 GHz ARM Cortex-A8. For this, we measured the time of 1531 task switches on a system running four tasks and FreeRTOS. The measurements start from the timer interrupt until restoring the context of the next task. This experiment was executed twice, once with and once without TrustZone. To measure the time accurately, we read the CCNT-register which stores the cycle count. The overhead

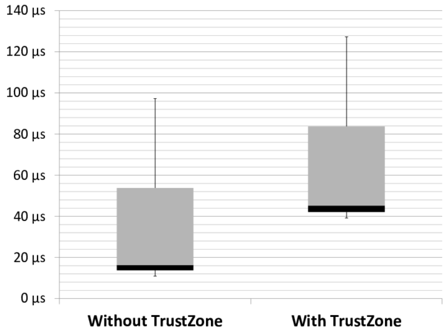

Fig. 5: Box-and-whisker diagram of time for task switches

Value	Without TrustZone	With TrustZone
Maximum	(97 ± 1) µs	(126 ± 1) µs
75%-Quantile	(54 ± 1) µs	(84 ± 1) µs
Mean	(16 ± 1) µs	(45 ± 1) µs
25%-Quantile	(14 ± 1) µs	(42 ± 1) µs
Minimum	(11 ± 1) µs	(39 ± 1) µs

Table 2: Data values of the Box-and-whisker diagram in Fig. 5

of our timing function calls was 0.9 µs, allowing us an accuracy of microseconds. The average overhead of a TrustZone task switch was 29 µs. Figure 5 presents the overhead as box-and-whisker diagram and the values are listed in Table 2.

A TrustZone task switch in comparison to a non-TrustZone one is equal to 3.6 *malloc* system calls (average time for *malloc* on our system: 8 µs) overhead; in other words memory management overhead is comparable to TrustZone.

4 Discussion

4.1 Attacks

We examine how our architecture behaves in different attack classes.

- Attacks that change invariants: Let us assume an adversary compromises a PLC but changes some invariants – e.g. he overflows a buffer and inserts a new task but the policy states that there are only N tasks. The system level self-healing will immediately reset the tasks, removing the malicious task. This type of defence was demonstrated in a simple fashion by the buffer overflow above (section 3.2).
- Stealthy attacks (Advanced Persistent Threats (APTs), backdoors, rootkits, trojans etc.): Suppose an attacker manages to deploy a stealthy rootkit on a PLC without being detected. He can now manipulate the PLC as he likes, but the other two PLCs continue to function correctly. If the rogue PLC affects outputs, its reset-circuit will notice this and reinstall the PLC, thus removing the rootkit. If the rootkit resides in the normal world, it will be removed even earlier by the system level self-healing.
- Firmware/hardware attacks: Suppose the attacker deploys a firmware or hardware rootkit in a PLC or in a reset-circuit, then software-based self-healing becomes impossible. However, if the other two PLCs remain operational, then manipulations of outputs are still detected and outvoted by those other two PLCs (through the 2-out-of-3 circuit).
- Attacks on network protocol flaws (e.g. Modbus): Since all PLCs presumably have to support the same protocols, exploits targeting the protocol itself (in contrast to its implementation) are not prevented. To defend against these attacks, one has to modify the protocol standard which is beyond our scope.
- Attacks on the policy/administrator account: We rely on the information in the (cryptographically signed) policy. If the account in charge of it is compromised, the adversary can change the policy freely. To prevent this, trusted input techniques as in [33] should be utilised.
- Denial of Service (DoS): Since we deliberately grant the attacker full control over some devices (see assumption 2 section 2.2), he already has the ability to turn these devices off, but we aim at not enabling further DoS attacks.

4.2 Diversity of PLCs

It is crucial for our architecture that PLCs are diverse in their software as well as hardware. To achieve this, we suggest to use different reset-chips and different CPU architectures, e.g. ARM TrustZone, Intel Software Guard Extensions

(SGX), and a PowerPC with a TPM chip. Since the architectures are different, an adversary has to craft different exploits, however, he can still use the same exploit idea to attack the software of all PLCs.

N-version programming was shown to be ineffective against malicious attacks [5, 23] as people make correlated mistakes. Hence, we suggest to use a form of artificial N-variant systems where N systems are crafted such that they are distinct by design. [32] Cox et al. [7] used address space partitioning and instruction set tagging to create different programs that cannot be compromised with the same exploit. Salamat et al. [22] proposed to invert the stack and demonstrated this by a special compiler. There is also a compiler which splits stack into data and control structures. [17]

By utilising N-variant system techniques, we can artificially create distinct software, that cannot be compromised with the same exploit idea – e.g. a RTOS with the stack growing downwards, one with the stack growing upwards, and one with separate data and control stack cannot all be exploited with the same stack-based attack. If we additionally require each netboot image to be diverse from the last one, similar to [25], an attacker would need to learn the properties of the new image to compromise it. That means, it is considerably harder for an attacker to compromise two PLCs at the same time.

4.3 Implications

The practical implications of our architecture for the real world are that an attacker would have to find twice the amount of vulnerabilities for an ICS since he has to compromise two different devices (e.g. an ARM TrustZone and an Intel SGX PLC). Hence, our system would double the cost for the attacker.

The cost for the defender will not double. Considering that a lot of companies already have redundant PLCs, the hardware cost for the company would roughly stay the same. Diverse software versions can be created similar to the artificial N-variant systems. As the special compilers used to generate these variants [7, 22] demonstrate, software can be automatically diversified except for architecture dependent code. In our proof-of-concept implementation based on FreeRTOS 7.8% is platform specific code (780 lines Assembly; 9956 lines C-code)[7].

Another advantage is that network-based self-healing prevents bricking PLCs. If a PLC is partially flashed with an image and crashes, it would automatically reboot triggering the netboot chip to reinstall the image. Thus, patching of the PLCs can now be done conveniently by central image servers with signed images.

5 Related Work and Comparison

Industrial Control Systems: The ICS architecture was already analysed in general: Virvilis et al. [31] studied a variety of APT attacks in depth and their countermeasures. They suggest proper patch management, network segregation,

[7] measured with the CLOC Linux tool

white-listing of outgoing connections, filtering dynamic content execution, and employing trusted computing. Fitzek et al. combined ARM TrustZone with an ICS but relied on a single TCB. [9] Another new TEE is Intel SGX [12] but we are not aware of any studies considering APTs or ICSs.

On the defence side, there is the term *secure control*, i.e. controlling a cyber-physical system while preventing attacks. [6] Also governmental organisations already approached the topic [13, 26] focusing on thoroughly enhancing every layer of the architecture, so-called *Defence in Depth*. Coexisting, there is also a *Defence in Breadth* which is not clearly defined but is described as the use of multiple instances of a security technology within a security layer. [20] Commonly, intrusion detection and tolerance systems [16] and firewalls are deployed to prevent or limit damage through attackers. Intrusion tolerance draws its ideas from the field of fault-tolerance [30, 1] which focuses on safe operation of a system by using redundancy. Totel et al. [27] proposed to use multiple diverse off-the-shelf devices in combination with an IDS proxy to detect attacks. They demonstrated this on the example of a webserver. However, their proxy is a single point-of-failure.

Self-Healing Systems: The field of self-healing is not well established, especially considering security. Ghosh et al. [11] gave a detailed overview of existing techniques. The closest ones to our technique are Finite State Automaton (FSA) approaches [28, 14] which model the systems as an FSA and rejuvenate it when invalid states are reached. Our self-healing technique based on invariants is more efficient than FSA as we do not actively keep track of the state. Since invariants can be declared as regular expressions and every FSA can be translated into a regular expression [19], our technique is as representative as FSA.

Bessani et al. [3] use proactive-reactive rejuvenation to restore intrusion-tolerant Crutial Information Switches (a firewall device) throughout the network. It is an example of a hybrid distributed system [29]. While their system targets the firewall in front of critical devices like PLCs, our approach is aimed at PLCs directly. Platania et al. [21] proposed a rejuvenation architecture similar to ours. Instead of self-healing upon detected misbehaviour, they proactively rejuvenate PLCs periodically; each on its own to ensure availability of the whole system.

Periodic system resets can defend against attacks before visible effects occur – independently of any detection algorithm – however, they impose an overhead on the system even though the system is mostly in a valid state. In contrast to Bessani et al. and Platania et al., we removed single point-of-failures[8] and kept the system running as long as possible through system level reactive measurements making it more applicable to scenarios were availability is a major concern.

6 Conclusion

We presented a malware-tolerant Industrial Control System architecture without single point-of-failures at critical intersection points. We achieve this by relying

[8] The wormhole devices of Bessani et al. storing the cryptographic key and the rejuvenation device of Platania et al. are single point-of-failures.

on diverse, redundant PLCs and a 2-out-of-3 circuit. The infrastructure can push an attacker out of any single PLC using its offline self-healing abilities on the network level. By also employing online self-healing at system level, we maintain high availability during basic failures or simple attacks. To prove our claims, we utilised ProVerif, a state-of-the-art protocol verifier, and implemented proof-of-concept self-healing capabilities on top of FreeRTOS and ARM TrustZone. Proofs as well as RTOS implementation are open-source.

References

1. Amir, Y., Coan, B., Kirsch, J., Lane, J.: Prime: Byzantine replication under attack. Dependable and Secure Computing, IEEE Transactions on 8(4), 564–577 (2011)
2. ARM: Building a secure system using trustzone technology. Tech. rep., ARM (April 2009), `http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf`
3. Bessani, A.N., Sousa, P., Correia, M., Neves, N.F., Verissimo, P.: The crutial way of critical infrastructure protection. IEEE Security & Privacy 6(6), 44–51 (2008)
4. BSI: Die lage der it-sicherheit in deutschland 2014. Tech. rep., Bundesamt fÃijr Sicherheit in der Informationstechnik (2014), `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2014.pdf`
5. Cai, X., Lyu, M.R., Vouk, M.A.: An experimental evaluation on reliability features of N-version programming. In: Int. Symp. on Software Reliability Engineering (ISSRE'05). IEEE (2005)
6. Cardenas, A.A., Amin, S., Sastry, S.: Secure control: Towards survivable cyber-physical systems. In: Int. Conf. on Distributed Computing Systems. IEEE (2008)
7. Cox, B., Evans, D., Filipi, A., Rowanhill, J., Hu, W., Davidson, J., Knight, J., Nguyen-Tuong, A., Hiser, J.: N-variant systems: A secretless framework for security through diversity. In: Usenix Security. vol. 6, pp. 105–120 (2006)
8. Dolev, D., Yao, A.C.: On the security of public key protocols. Transactions on Information Theory, IEEE 29(2), 198–208 (1983)
9. Fitzek, A., Achleitner, F., Winter, J., Hein, D.: The andix research os-arm trustzone meets industrial control systems security. In: Int. Conf. on Industrial Informatics (INDIN) (2015)
10. Fovino, I., Carcano, A., Masera, M., Trombetta, A.: Design and implementation of a secure modbus protocol. In: Conf. on Critical Infrastructure Protection (2009)
11. Ghosh, D., Sharman, R., Rao, H.R., Upadhyaya, S.: Self-healing systems - survey and synthesis. Decision Support Systems 42(4), 2164–2185 (2007)
12. Hoekstra, M., Lal, R., Pappachan, P., Phegade, V., Del Cuvillo, J.: Using innovative instructions to create trustworthy software solutions. In: Workshop on Hardware and Architectural Support for Security and Privacy, HASP. (2013)
13. Homeland Security: Recommended practice: Improving industrial control systems cybersecurity with defense-in-depth strategies. Tech. rep., U.S. Homeland Security (October 2009), `https://ics-cert.us-cert.gov/sites/default/files/recommended_practices/Defense_in_Depth_Oct09.pdf`
14. Hong, Y., Chen, D., Li, L., Trivedi, K.S.: Closed loop design for software rejuvenation. In: Workshop on Self-Healing, Adaptive, and Self-Managed Systems (2002)
15. Kaspersky Lab: Empowering industrial cyber security. Tech. rep., Kaspersky Lab (2015), `http://media.kaspersky.com/en/business-security/Empowering%20Industrial%20Cyber%20Security_web.pdf`

16. Kuang, L., Zulkernine, M.: An intrusion-tolerant mechanism for intrusion detection systems. In: Availability, Reliability and Security. pp. 319–326. IEEE (2008)
17. Kugler, C., Müller, T.: Scads separated control- and data-stacks. In: Int. Conf. on Security and Privacy in Communication Systems. Springer (2014)
18. Langner, R.: Stuxnet: Dissecting a cyberwarfare weapon. Security & Privacy, IEEE 9, 49–51 (Nov 2011)
19. Meduna, A., Vrabel, L., Zemek, P.: Converting finite automata to regular expressions. online (2012), `http://www.fit.vutbr.cz/~izemek/grants.php.cs?file=%2Fproj%2F589%2FPresentations%2FPB05-Converting-FAs-To-REs.pdf&id=589`
20. Paillet, D.: Defending against cyber threats to building management systems. online (FM Magazine) (April 2016), `https://www.fmmagazine.com.au/sectors/defending-against-cyber-threats-to-building-management-systems/`
21. Platania, M., Obenshain, D., Tantillo, T., Sharma, R., Amir, Y.: Towards a practical survivable intrusion tolerant replication system. In: Reliable Distributed Systems (SRDS), 2014 IEEE 33rd Int. Symp. on [1], pp. 242–252
22. Salamat, B., Gal, A., Jackson, T., Manivannan, K., Wagner, G., Franz, M.: Multivariant program execution: Using multi-core systems to defuse buffer-overflow vulnerabilities. In: Complex, Intelligent and Software Intensive Systems (CISIS), 2008.
23. Salewski, F., Wilking, D., Kowalewski, S.: The effect of diverse hardware platforms on N-version programming in embedded systems-an empirical evaluation. In: Workshop on Dependable Embedded Sytems (WDES). Citeseer (2006)
24. Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., Zhou, S.: Specification-based anomaly detection: a new approach for detecting network intrusions. In: Conf. on Computer and Communications security. ACM (2002)
25. Sousa, P., Bessani, A.N., Correia, M., Neves, N.F., Verissimo, P.: Highly available intrusion-tolerant services with proactive-reactive recovery. IEEE Transactions on Parallel and Distributed Systems (2010)
26. Stouffer, K., Lightman, S., Pillitteri, V., Abrams, M., Hahn, A.: Guide to Industrial Control Systems (ICS) security. NIST Special Publication, (May 2014), `http://www.gocs.com.de/pages/fachberichte/archiv/164-sp800_82_r2_draft.pdf`
27. Totel, E., Majorczyk, F., Mé, L.: COTS diversity based intrusion detection and application to web servers. In: Int. Workshop on Recent Advances in Intrusion Detection. Springer (2005)
28. Tu, H.y.: Comparisons of self-healing fault-tolerant computing schemes. In: World Congress on Engineering and Computer Science (2010)
29. Verissimo, P.E.: Travelling through wormholes: a new look at distributed systems models. ACM SIGACT News (2006), `http://dl.acm.org/citation.cfm?id=1122497`
30. Veronese, G.S., Correia, M., Bessani, A.N., Lung, L.C., Verissimo, P.: Efficient byzantine fault-tolerance. IEEE Transactions on Computers 62(1), 16–30 (2013)
31. Virvilis, N., Gritzalis, D., Apostolopoulos, T.: Trusted computing vs. advanced persistent threats: Can a defender win this game? In: 10th Int. Conf. on Autonomic and Trusted Computing (ATC). pp. 396–403. IEEE (2013)
32. Weatherwax, E., Knight, J., Nguyen-Tuong, A.: A model of secretless security in N-variant systems. In: Workshop on Compiler and Architectural Techniques for Application Reliability and Security (CATARS-2) (2009)
33. Zhou, Z., Gligor, V.D., Newsome, J., McCune, J.M.: Building verifiable trusted path on commodity x86 computers. In: Symp. on Security and Privacy (SP). pp. 616–630. IEEE (2012)