Wireless Personal Communications
https://doi.org/10.1007/s11277-021-09055-1

Zero-Trust Principles for Legacy Components

12 Rules for Legacy Devices: An Antidote to Chaos

Geir M. Køien[1]

Accepted: 16 August 2021

Abstract
In this paper we briefly outline as set of rules for integration of legacy devices into a modern industrial control system. These rules are fairly simple, and are mostly derived from "Zero Trust" principles. These rules aim to be pragmatic, and cost-effectiveness trumps completeness.

Keywords Legacy devices · Industrial control systems · Zero trust

1 Introduction

1.1 Background

This paper is mainly concerned with how to enhance security for legacy components in Industrial Control Systems (ICS). The main security problems with ICS architectures are outlined in [1]. The legacy components may include sensors, actuators and controllers. These will often need to be retained, while the overall control system is being digitalized and perhaps migrated to a cloud solution.

The "Zero Trust" (ZT) paradigm provides most of the inspiration for how to integrate legacy components in a safer and more robust way [2]. We shall return to how ZT is used as a foundational principle for the legacy component integration.

1.2 A Scenario

The problem we will attempt to alleviate is found in many ICS system. One typical scenario can be that of waste water/spill water handling by municipal authorities. Specifically, how to avoid local flooding by excessive water levels in the drainage system. In such case, one typically has a control system located centrally at the municipality. The system will have many distributed flow sensors, etc., that report in the current water level and the current flow. There will also be a number of valves and motor, that may

✉ Geir M. Køien
geir.koien@usn.no

[1] University of South-Eastern Norway, Campus Vestfold, Horten, Norway

be used to control the flow patterns and may be used to redirect excessive water. The motors and valves may or may not be remotely activated, and they may or may not be automatically handled by the control system.

Whatever the case may be, the "remote" part of the system will be geographically distributed. It may enjoy some level of casing and it may be located at more-or-less inaccessible locations, but generally speaking there will be few security measures as such to protect these devices. The devices will typically be communicating over low-bandwidth long-range radio networks.

1.3 The Problem

In ICS, the problem is that Legacy Devices (LDs) often have little or no security. At the same time, these devices often needs to be retained since replacing them tends to be a complicated and complex undertaking. The devices tend to have limited flexibility and very often there is no way to upgrade or update the devices to conform to modern standards. Thus, we have a situation where these devices may have:

- Outdated software (sw), firmware (fw) and hardware (hw)
- Limited connectivity
- No, or inadequate, means for updating the sw/fw
- No, or inadequate, communications security
- No, or inadequate, means to conduct access control

The Legacy Devices (LDs) are potentially vulnerable to a whole range of security threats. For the general case, this will include issues such as:

- Platform integrity cannot be guaranteed (sw/fw, incl. configuration data)
- *Tampering* data-at-rest integrity cannot be assured
- *Tampering* data-in-transit integrity cannot be assured
- *Disclosure* data-at-rest confidentiality cannot be assured
- *Disclosure* data-in-transit confidentiality cannot be assured
- *Authentication* Inability to authenticate entities (device access)
- *Authorization* Inability to verify legitimate device access

1.4 Assumptions

For the scope of the present paper, we have the following assumption:

- *Replacement* It is deemed infeasible to replace the LDs
- *Updates* The LDs cannot be fixed by updates (will remain "legacy")
- *Connectivity* The LDs will communicate with an up-to-date core system
- *Bandwidth needs* The LDs will have modest bandwidth needs
- *Trust-level* The LD belongs to its own trust domain (untrusted)
- *Protection* No security protection can be assumed (unprotected)

1.5 Scope

Only the high-level aspects will be covered. We will thus not point to specific requirements for data confidentiality algorithms or mandate specific standards to be used during the integration of LDs into a modern ICS.

That is, the provided architecture description should properly be viewed as an example (see Fig. 1). The generic rules will apply to other configurations too, even if one may have to adapt the concrete aspects.

The 12 rules that we propose are not written in stone. There could be good reasons not to adhere to all of them. It is, however, important that one can assure oneself that those reasons are sound and valid.

2 Threat Landscape and Threat Modeling

2.1 Threat Landscape

Many institutions and companies publish annual threat landscape reports. These reports address many different topics, ranging from natural disasters, economic affairs, military conflicts, etc. For our purpose, the threat landscapes we are most interested in concern digital systems and industrial control system specifically. The European Union Agency for Cybersecurity (ENISA) annually publishes a threat landscape report. That is, from 2020, the ENISA threat landscape (ETL) report has grown into a set of publications. The designated ETL entry point is the year-in-review report [3]. The ETL has a very strong standing in the EU system, and has international significance. ENISA also publishes many topical threat reports.

For our purpose, the ENISA report, and other similar efforts, does not quite capture the problems that ICS faces. That is, while the "generic" cybersecurity reports surely apply to an ICS, there are aspects that are left out or inadequately handled. However, during the last few years, there has been more attention on the ICS aspects. In particular, we want to highlight the white paper from MITRE, "MITRE ATT&CK for Industrial Control Systems: Design and Philosophy" [1]. MITRE is a not-for-profit organization, and has a strong standing in cybersecurity in general. The MITRE ATT&CK model is in common use in the whole cybersecurity sector. The MITRE ATT&CK

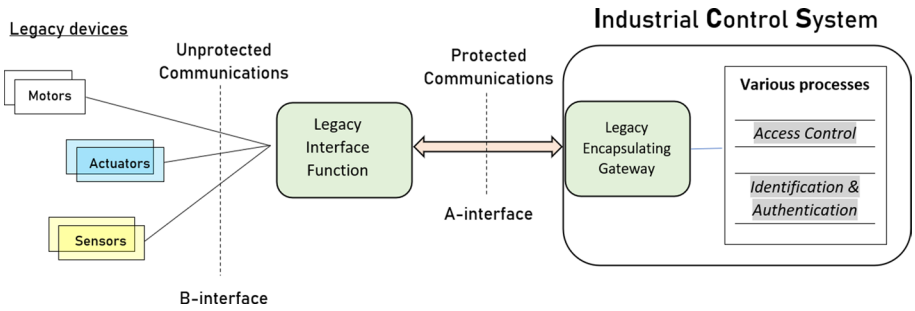

Fig. 1 The legacy component architecture (example)

model for ICS is not a threat landscape report, but rather a model of the ICS security and a way to highlight the specific threats and attack-types ("kill chains") that affect the ICS sector.

Dragos is a company that specializes in cybersecurity for the ICS sector. They published a threat landscape report for the ICS sector for 2020 [4]. Among the major problems reported are the lack of updating of software/firmware and the "remote access risk in ICS". We should note that the update problem reported also covers more modern components, where the system owner for one reason or another do not implement the updates. Normally, this is rooted in lacking operational routines, but in an ICS context it can be quite difficult and time consuming to carry out software updating.

The Industrial Control System (ICS)/Operational Technology (OT) environment and the product update cycles found in modern ICT system do not match well. The Agile development methods used in web-centric solutions typically have a "continuous delivery" roll-out model for new functionality. In an ICS/OT setting, one tend to have an "if it isn't broken, don't fix it" mentality.

2.2 Threat Model

A threat model of the specific target system should be devised. The threat model should be regularly revised.

For our purpose, we specifically mandate that the legacy system parts be modeled. Threat modeling is a cost-effective way of finding flaws, errors and omissions in the system design. Threat modeling as a method will not provide any guarantees or proofs, but it is a pragmatic and often efficient method of uncovering many future problems. We propose using an established threat modeling standard. There are several options available, and one of the prolific ones are the STRIDE method [5].

Typically, the design of a threat model go through four stages:

1. Describe the target system architecture (the model)
2. Find potential threats and vulnerabilities
3. Handle the threats (mitigate, remove, transfer, ...)
4. Validate the model (1/2) and the devised responses (3).

The "Validate" stage typically also entails assigning a "best-before" date on the model. There is normally focus on entities (users or processes) that initiate activity, on data flows and data storage. Obviously, there is also focus on the assets in the model.

The "asset focus" afford focus on:

– Things that have value to the system owner
– Things that one believes can have value for an attacker
– Steps that lead to the assets

We shall not go further into threat modeling, or the STRIDE approach, here. Threat modeling, by it nature, is predominantly a design-phase activity. However, it is not restricted to what kind of design one is attempting, and it is therefore also suitable for designs that deal with integration of legacy system components.

3 Architectural Aspects

3.1 The Legacy System Components

The example Legacy Component Architecture (LCA) is outlined in Fig. 1. In additions to this, the system architecture must support an identifier scheme and entity authentication functions. There must also be a security policy, which details the authorizations, and an associated access control scheme. The identification, authentication and authorization should be aligned with the rest of the ICS system. The LCA consists of the following elements and interfaces:

- *Legacy devices (LD)* No security properties are assumed or expected of these devices.
- *Legacy Interface Function (LIF)* A LIF handles input/output towards the LDs. It includes a data validation function (DVF), which verifies input data.
- *Legacy Encapsulating Gateway (LEG)* A LEG is integrated into the ICS. It can interface towards one or more LIFs. The LEG handles identification and authentication for access towards the LIF. It also performs access control functions.
- *The A-interface (bi-directional)* The A-interface is between a LEG and a LIF. All communications over shall be fully protected. The A-interface supports multiple logical channels.
- *The B-interface (bi-directional)* The B-interface is between a LIF and the LDs. The communications over the B-interface may be protected, but generally it will be unprotected. The B-interface shall preferably support individual channels for each LD.

The LEG and LIFs are hosted on modern embedded platforms. They will have an up-to-date operating system (OS) and a full feature set of security functions available. This can include log forwarding (including alarms and warning), firewall functionality, intrusion detection/prevention systems, access control functions, cryptographic function (authentication, encryption and data integrity assurance). It may also include device attestation functionality, which enables remote assurance as to the security state of the distributed device. A typical example of a LEG/LIF platform may be an industrial embedded device running a security hardened Linux OS.

We cannot make any assumptions on the respective locations of the LEG and the LIF. This is very much in line with the central tenets of the Zero Trust paradigme, in that one must not make any assumptions with regard to location. The LIFs and LDs are likely to be located in a physically secluded location. Thus, there *may* be some measure of protection in effect. We recommend that the LCA threat model explicitly evaluate this. Any trust assumed should be made explicit and expressed as conditional.

3.2 Legacy Device Encapsulation

Exposure of the B-interface will be a liability. The obvious remedy is to avoid needing to expose the B-interface, which is doable if the LDs are physically adjacent (co-located) with a LIF. The LIFs are not supposed to be expensive devices, and it could be realized on a fairly simple embedded ARM-based controller running an embedded Linux OS. Even a unit like the lowly Raspberry PI Zero (W) platform would suffice. It should be economically feasible for many cases to have one LIF per LD. The LIF/LD combined unit should

be encased to have some physical protection too. LIF/LD integration is the recommended solution.

3.3 Plane Separation

The communications in the LCA is over three logical planes. Each will support one or more logical channels. These planes are:

- *Management Plane (MP)* It covers communications over the A-interface (ICS/LEG ⇆ LIF).
- *Control Plane (CP)* It covers communications between the ICS/LEG and individual LDs, with a LIF as a mediator (ICS/LEG ⇆ LIF ⇆ LD).
- *Data Plane (DP)* It covers communications between the ICS/LEG and individual LDs, with a LIF as a mediator (ICS/LEG ⇆ LIF ⇆ LD).

3.3.1 The Management Plane

The MP channel is dedicated to management of the LIF. Thus, all communications with the LIF as an end-point, is conducted over the MP. The MP shall be logically separated from other communications. The MP traffic will include administration of the LIF, including security updating of the LIF, administration of firewall functionality, of intrusion detetcion/prevention systems and of the data validation function. Note that there needs to be a specific DVF for each individual LD. Additionally, forwarding of log information, warnings, alarms, etc. from the LIF to the LEG/ICS shall be supported.

Note The LEG will need similar administrative functionality as does the LIF. However, we propose that this is handled similarly to that of other (modern) ICS devices/servers. Thus, it is outside the "legacy" scope of this paper.

3.3.2 The Control Plane

The CP channels are for administration of the LDs. There will be a separate CP channel for each LD. The communications for carrying out CP commands will originate within the ICS. The commands are mediated and forwarded by the designated LEG and LIF, and will cover setup and configuration handling of the LDs, etc. Only the CP channel shall be used for making changes to the LD software/firmware and configuration data. This permits stricter access control on these functions. Immediate status reports and responses from the LDs shall be reported on the CP channel.

3.3.3 Data Plane

These channels are for communications that logically pass through the LIF-LEG. That is, for LD data and commands that are carried out during normal operations.

3.4 Logical Channel Instances

Each separate access session from an ICS process towards an LD should result in a separate logical channel being established. This channel will be associated with a session context.

The logical channel instances shall be identifiable through a *channel instance identifier (CII)*. The creation of a CII is subject to authentication and authorization procedures. The creating of the CII should be logged, together with a time stamp and identification of the ICS process and the target LD. Subsequent use of the CII should be subject to logging.

4 Zero-Trust as a Foundational Principle

4.1 Never Trust, Always Verify

These days, there is a lot of focus on the so-called Zero Trust (ZT) concept. The concept is not about zero trust per se, but rather about *unwarranted* trust. That is, there should be no "blind" or unspoken/assumed trust. In particular, there should be no trust assumed due to the location of a node or entity. All claims **must** be corroborated. We will hold Zero Trust as a foundational principle, with its ethos of "Never trust, always verify".

There is another central aspect of the ZT philosophy that also needs to be internalized, and that is the assertion that the system will be compromised. This is not a question of *if*, but *when* and *how often*. That is, system compromise (breach) is seen as an invariant property of the system being operational. This has the consequence that one must plan for how to detect attacks, how to recover and how to resume business.

4.2 Zero Trust Proponents

The National Institute of Standards and Technology (NIST) is an agency of the United States Department of Commerce. Its mission is to promote innovation and industrial competitiveness. Despite its non-regulatory status, NIST does produce a series of recommendations and "special publication". These are not standards per se, but they nevertheless have a strong standing and it is often considered essential to be compliant with NIST recommendations. Concerning ZT, NIST have published a special publication on its Zero Trust Architecture (ZTA) [2]. We shall generally adhere to the NIST ZTA tenets. The ZT paradigm is officially endorsed not only by NIST, but also by US National Security Agency (NSA). The brief publication "'Embracing a Zero Trust Security Model" [6] confirms this (it is also a very readable high-level exposition of the Zero Trust concepts).

4.3 Main ZT Concepts

Conventional network security has often focused solely on perimeter defenses. This approach has its limitations. There is an implied assumption that all the objects and components on the inside of the perimeter have the same trust level. Furthermore, that all those inside the perimeter can enjoy mutual trust in each other. Thus, if an intruder succeeds in penetrating the network, the intruder will often face few obstacles to compromising other objects and components at will. For large organizations, or for complex networks with many nodes, the conventional network security perimeter defence paradigm is untenable and inadequate. A ZT architecture addresses this problem by focusing on protecting resources, not network perimeters. The trustworthiness is no longer associated with location (inside a network perimeter or not).

The NIST defined ZTA places a lot of significance on identifying work flows and data flows. This includes a focus on assets and on threats to the assets. The basics include:

– Entity identification
– Entity authentication
– Access control (authorization)
– Separate and segment the resources
– Logging and log handling (monitoring)

The "logging" is of course associated with the belief that there will be breaches. Then one needs ample detection capabilities, and means to investigate the details of the breach.

There is also another principle that may come into play here, and that is the principle of "Defense-in-Depth". The underlying assumption is that there will be attacks and that multiple/segmented defenses can afford extra protection. The US Department of Homeland Security has published recommendations concerning this principle for industrial control systems [7].

4.4 Real-World Adoption of the Zero-Trust Principles

There are several companies that now design, implement and operate their networks according to the ZT principles. There is the BeyondCorp implementation by Google [8] and Netflix has developed the Location Independent Security Approach (LISA) architecture [9].

Both BeyondCorp and LISA removes trust from the network location and perimeter, and replaces it with trust in authenticated users and "healthy" devices. The principles are simple enough that large-scale adoption should be possible for many organizations. Of course, the Google and Netflix example networks are office networks rather than ICS networks. But, it demonstrates that ZT designs are possible and practical in the real world.

5 The 12 Rules

The 12 rules are not the ultimate answer to all security problems for legacy components. But, we argue, they do provide a reasonable antidote to many of the problems one may face.

5.1 Threat Awareness

Threat awareness is essential for responsible deployment of legacy devices. This includes an up-to-date awareness of the current threat landscape. Threat modeling is, as has been mentioned, an efficient way to uncover many of the important problems. As a design tool, it also helps the designers, implementers and users to better understand the system.

We strongly advocate LD encapsulation (combined LIF/LD) for all LDs. However, this is not always an attainable goal, and we may thus have an exposed B-interface. If so, it is essential that the B-interface be scrutinized in the threat model.

Rule 1 ***Threat Model Your System (regularly).*** A revision of the threat model should be made when the system changes, the assets change, the users change, etc. Annual inspection of the model is a minimum requirement. There should be a follow-up plan for the identified threats.

5.2 Identification and Authentication

To prevent masquerade (impersonation) is essential. Thus, all access to/from the legacy component must be verified to come from a well-defined entity. To this end, all entities and elements must have a globally unique identifier. It must be possible to authenticate all the defined identifiers.

Rule 2 ***Identify and Authenticate All Device Accesses.*** All device accesses must be identified and authenticated.

If one cannot adhere strictly to this rule, then one must compensate with extended logging or other suitable measures. But, missing authentication is a significant hint that one needs to redesign this part of the system. All A-interface communications should be authenticated irrespective o other shortcomings in the system.

5.3 Authorization and Access Control

It is not sufficient for the LEG/LIF to have verified the identity of the entities that wants to communicate with the LDs. The accesses must also be authorized. That is, the LEG must perform access control checks on all operations towards the LD. To do this, the LEG must have access to the security policy rules for LD access. These rules should be explicit about which ICS entities (processes/devices) that are allow to read and/or write data to the LDs.

There must be separate access rules for CP and DP access. Additionally, there must also be policy rules for MP accesses to the LIF.

Rule 3 ***Device Accesses Must Be Authorized and Verified.*** Device accesses shall be subject to access control according to security policies.

5.4 Strict and Enforced Channel Separation

An important ZT concept is that of separation and segmentation. We therefore require that access to/from LDs be separated. For our purpose, we require that each CII be cryptographically separated. To this end, a separate security context will need to be set-up, and it will contain the associated security key material, etc.

Rule 4 ***Channel Separation Must Be Enforced.*** The logical channels shall be cryptographically separated.

5.5 Event Logging and Handling

In the ZT philosophy, it is a given that there will be breaches. With this as a premise, one must plan how to detect breaches.

For the LEG and the LIF, it is important to set up logging of all significant events. This would minimally include all access attempts and all attempts to communicate with the LDs. On a generic level this would include logging of:

- All authentication events
- All access events (involving the CIIs)
- All anomalous events
- All ordinary non-legacy access attempts

The LEG and LIF will also have access to much more information that one may want to log. This could include firewall events, data validation event, meta-information concerning session instances, etc. The logs should not be stored on the LIF or the LEG, but forwarded to a central log handling function. To be able to detect ongoing intrusions, the logs will have to be actively processed. This would include anomaly detection capabilities and establishment of thresholds for warnings and alarms. The LCA logs should be handled centrally and seen in context with the overall log handling in the ICS.

Rule 5 ***There Must Be Mandatory Log Capture and Log Analytics.*** There must be extensive log capture capabilities. The collected log must be forwarded and analyzed.

5.6 Data Validation and Data Fuzzing Tests

Data validation is a complex problem, and it is directly dependent on the data producer and data consumer functions. Inadequate data input validation is a persistent source of vulnerabilities in digital systems.

5.6.1 The Buffer Overflow Problem

The *buffer overflow* problem belongs to a class of vulnerabilities in which the input buffer capacity of the receiving function is exceeded. Many of buffer overflow vulnerabilities are directly related the C programming language. The semi-legendary 1996 paper on "Smashing The Stack For Fun And Profit" [10] provides an early account of the problem. An overview and taxonomy of the problem can be found in [11, 12].

5.6.2 The Code Injection problem

This is a class of problems where the input function fails to differentiate between data and code. The now decades old (but still applicable) SQL injection attack technique is perhaps the most well-known case [13], but it is by no means the only one. The basics of the attack is to confuse the input function to take data input and handle it as a code/command. This is done by confusing the function at one level to accept the input as data, while still managing to trigger the "data" to be interpreted at code at another level.

5.6.3 Type and Range Checking

The ideas behind type systems and range verification of data are old, and have their root in mathematics. For our purpose, the history is at least as old as that of type systems in computer languages. The typical implementation language for systems and low-level programming has been the C language. The C language has a rather weak type system, and the extensive use of pointers makes it even weaker. Suffice to say, the C language is not a good language if one wants to enforce a type system.

5.6.4 Design-by-Contract

Meyer, when designing the Eiffel language (early 1990ies), wanted to have a systematic approach to verifying input/output data to functions [14]. The paradigm, now known as *design-by-contract*, requires the input values to be associated with type and range. This makes it feasible for the compiler to perform statical checks, and the compiler could also generate the appropriate dynamic checks. The Eiffel design-by-contract has many more features, but suffice to say that *type-safety* is also a security measure. Type-checking as an explicit security measure is further investigated in [15].

5.6.5 The Data Validation Function

For our purpose, we want the data validation to be done on the communications path (on the message protocol as it were). The Data Validation Function (DVF) resides on the LIF, and the validation should be carried out for both direction (data to/from the LDs). For such a scheme to work, the DVF will have to know what data to expect. That includes knowledge about data encoding, data type information and permissible ranges (if known).

5.6.6 The Virtue of Explicitness

The requirement for explicitness is also as a principle in the well-known paper on prudent engineering principles for cryptographic protocols [16]. Explicitness is not only relevant to security protocols, but to (signalling) protocols in general. It was taken very seriously by the International Telecommunication Union (ITU) when they designed the Abstract Syntax Notation 1 (ASN.1) scheme [17]. ASN.1 encodes data according to a `(Type,Length,Value)` tuple. Although the format appears somewhat archaic by today's standards, the TLV principle is still sound.

5.6.7 Mandatory Data Validation

There will need to be a separate DVF for each plane and for each individual LD. There will doubtless be cases where it will be difficult to properly and completely describe the data exchange format. The DVF should log anomalous cases (exceptions), and it should be possible to configure the DVF to block input, issue warnings or raise alarms.

Rule 6 ***There Must Be Mandatory Input Data Validation.*** As a principle, input data should always be explicitly validated.

5.6.8 Fuzzing Tests

Fuzzing is a term for an automated software testing technique that involves providing invalid, erroneous, unexpected, or garbled (random) data as inputs to a computer

program. The program is then monitored for how it handles the invalid data. Fuzzing is also extensively used for testing communications protocols.

5.6.9 Fuzzing UNIX Utilities

The term "fuzz" originates with Miller, and the 1990 paper "'An Empirical Study of the Reliability of UNIX Utilities" [18]. The paper was mainly about reliability, but it also highlighted the relationship to security:

> Second, one of the bugs that we found was caused by the same programming practice that provided one of the security holes to the Internet worm (the 'gets finger' bug). We have found additional bugs that might indicate future security holes.

The reference to the "Internet worm" was to the so-called Morris worm of November 1988 (Spafford gives an account of the Morris worm in [19]).

5.6.10 Fuzzing Today

Fuzzing remains to be a remarkably effective testing technique. Miller, with Zhang and Heymann, reflects on this in the 30th anniversary follow-up paper "The Relevance of Classic Fuzz Testing: Have We Solved This One?" [20]. Sadly, the authors concludes that we haven't solved the problem.

In "Fuzzing: Challenges and Reflections" [21], the author takes a security view on fuzzing as a technique to uncover software vulnerabilities. The authors call for increased awareness of the problem:

> We believe awareness and education, in the small and in the large, are of paramount importance.

In the paper "SeqFuzzer: An Industrial Protocol Fuzzing Framework from a Deep Learning Perspective" [22], the authors investigates use of fuzzing towards an industrial control protocol. Their approach was to use machine learning techniques to do "smart" fuzzing on a stateful protocol. This is labor saving, as tailored fuzzing does require handling a lot of context data. The authors report that their SeqFuzzer, when applied to Ethernet for Control Automation Technology (EtherCAT) devices successfully detected several security vulnerabilities.

Rule 7 ***There Must Be Input Data Validation Testing.***

The input data validation functions must be thoroughly tested. Ordinary function testing techniques shall be used, but these are often insufficient. Use of fuzzing tests are therefore required.

5.7 Data Integrity Protection

Data integrity is the protection of data against illicit and unauthorized modifications. In a ZT system, one must assume that all unprotected data will be manipulated (at some stage). Only by proving ample protection can one avoid this. It applies both to data-at-rest and

data-in-transit. In practice, this invariably means that the data must be cryptographically protected. The use of message authentication codes (MACs) or designated signature functions, is the ordinary way of doing this.

Rule 8 ***There Must Be Data Integrity Protection.*** Date integrity is essential and must be enforced.

The chosen cryptographic protection methods must be regularly verified to be adequate. Only standardized and industrial strength algorithms and protocols should be used. One must also ensure that the implementation is up-to-date.

Under no circumstance should one use cryptographic algorithms or security protocols developed in-house. It is exceptionally hard to design cryptographic algorithm, and only after the algorithm be subjected to prologued study and investigations by the best cryptanalysts can one entertain any hope of having a sound design. We here refer to the so-called Schneier's law [23]:

> Any person can invent a security system so clever that she or he can't think of how to break it.

5.8 Data Confidentiality Protection

Data confidentiality is the protection of data against illicit and unauthorized diclosure. In a ZT system, one must assume that all unprotected data will be eavesdropped on (at some stage). Only by proving ample protection can one avoid this. It applies both to data-at-rest and data-in-transit. In practice, this invariably means that the data must be cryptographically protected. That is, data confidentiality may not always be required. There are cases where it does not matter if an adversary sees the data. However, there are many cases were it surely matters. Unless one is absolutely confident that one can distinguish these cases with 100% certainty, one should always choose to confidentiality protect the data. This is particularly true for data-in-transit. We also note that for modern cryptographic methods and security protocols, the additional cost of having data confidentiality together with data integrity is small. For our purpose, the LEG and LIFs will have no problem supporting data confidentiality for all connections.

Rule 9 ***There Must Be Data Confidentiality Protection.*** Date confidentiality is most likely important. It should be enforced.

5.9 Platform Integrity

The platform integrity of the LEGs and the LIFs is of utmost importance. These platforms should therefore be security hardened to the extent possible. This is particularly important should the devices be located in exposed locations. The operating system should also be a hardened version of the OS, meaning that all unnecessary interfaces and functions be removed before the OS is put into production. The security hardening should also be applied to the applications software.

– Use a hardened hardware platform (if feasible)

  - Avoid outdated hw platforms
  - Select a platform with strong cryptographic support
- Use a hardened (embedded) OS (a current version)
  - Remove unneeded programs and functions
  - Remove support for unneeded protocols and ports
  - Support additional security functionality (firewalls, logging, etc.)
  - Support security updates and functional updates
- Use hardened and security tested applications
  - Update the applications when necessary
- Support for remote attestation and intrusion detection
  - Support for remote attestation is highly desirable
  - Support for intrusion detection capabilities are important

Rule 10 ***There Must Be Platform Integrity Protection.*** The platform should be security hardened. The state of the platform must be regularly asserted.

5.10 Security Testing

The system components should be subjected to exhaustive security tests. This includes normals security testing, but it should also include the aforementioned fuzz testing and penetration tests (pentest). These tests should be executed regularly. Ideally, the pentests should be run by an external party, but in large organisations it may also be run by a designated "Red Team".

Rule 11 ***Security Must Be Tested Exhaustively.*** Security test exhaustive. A security report shall be written, and follow-up plans be made. Identified issues must be addressed.

5.11 Legacy Expiry

Whatever else happens, the passage of time will mean that the legacy devices will continually be getting older and more out-of-date. There may be problems with replacing broken equipment, and the cost of maintaining an old infrastructure could become quite significant. The cost here comes both in actual monetary cost, but also in missed flexibility and inability to adapt to new opportunities. From a security point of view, we are firm believers in the adage "Attacks always get better; they never get worse." (often attributed to Bruce Schneier). In a ZT context, this means that one should plan for replacing the legacy devices. There should be periodic assessments of *when* the legacy devices are to be retired.

Rule 12 ***Plan to Replace the Legacy Devices.*** The LDs will not get better with time. Replacement should be planned.

6 Discussion

6.1 Twelve Rules?

The interested reader may wonder if 12 rules really is an antidote to chaos for legacy devices. Well, there has been at least one other publication which advocates a 12 rule system to avoid chaos [24]. The book and the context is totally unrelated, but attests to the usefulness of manageable set of rules. The specific restriction to 12 rules is of course arbitrary, but 12 is a manageable number and allows scope for insights while avoid overloading the readers with too many rules. Restraint is a virtue, and brevity makes for better focus and clarity too.

As for the rules themselves, we can safely say that they are rooted in a best practices tradition. They may even to some extent be seen as self-evident, but we still find utility in expressing them explicitly. Of course, we unabashedly consider explicitness to be a virtue for system designs.

6.2 Real-world Concerns

The goal of expressing these 12 rules are at the same time both modest and far reaching. That is, we do not expect to solve all problems with these 12 rules. For one, it is going to be difficult to implement the rules on many legacy systems. The LEG and LIF may be new and modern, but the LDs are not. To integrate the LDs into our proposed architecture will therefore take considerable effort. There is of course also the issue of the B-interface, which will remain "legacy".

The far reaching part has to do with the fact that as we move towards increased digitalization in ICS systems, the repercussions of bad security is vulnerabilities in systems we crucially depend on. Then, even modest gains may count for something.

6.3 Limitations of the Approach

As is evident from the architecture, see Fig. 1, complete communications protection can only be had if one has LIF/LD encapsulation. However, it is quite likely that one cannot always have this. It is also quite likely that one cannot implement all of the rules, and that one will have to have unprotected devices in the system. If full compliance is infeasible, we still advocate partial implementation of the rules. Every small measure may count.

7 Summary and Concluding Remarks

In this paper we have identified 12 rules that aim to improve the security of legacy devices in an industrial control system setting. The rules are derived with Zero Trust as a foundational principle. That means that trust and trustworthiness will not be taken for granted.

We have rules that increase security awareness, rules that dictate identity corroboration and verification of access rights. We also have a rule to enforce separation (segmentation). There are also rules concerning event/incident detection capabilities.

A large class of errors and vulnerabilities in software is associated with input data handling. This has been a source of serious vulnerabilities, which have been exploited in attacks. To prevent and mitigate this, we have rules concerning input data validation and verification of the validation function itself (which, after all, is a piece of software).

The Zero Trust paradigm also includes integrity and confidentiality protection. Here we have rules that address data integrity and data confidentiality, both for data-at-rest (stored) and for data-in-transit (communications). We also have a rule that pertains to platform integrity.

In the proposed architecture, the B-interface is an unprotected communications paths. This is an obvious weakness, but also a concession to real-world realities. However, as described, the proper way to address this in a legacy context is to encapsulate the LD, and have a solution where each LD is directly coupled to a LIF. The combined and encased LIF/LD will properly address the communications security problem.

The intruder has the advantage that he/she only needs to find one single vulnerability to be able to compromise our system. As defenders, we have no such luxury. As a rule, we need to identity and address *all* the issues. That is not attainable, but we have to make a serious effort to find as many flaws as possible, because every one of them may potentially be used by the intruder. Apart from having a sound design and impeccable implementation, the next best thing we can do is to test our system with utmost scrutiny. The security tests should be as comprehensive and penetrating as possible (within budgets, time constraints, etc.) This will not totally prevent the intruder from finding a vulnerability, but it will improve the odds considerably in the defenders favor.

The final rules is stating the plainly obvious. One should plan to deprecate and replace the legacy devices.

The 12 Rules and Zero Trust Wisdom

The Zero Trust Ethos and Tenets

We have designated these as the Zero Trust ethos and tenets:

- Never Trust, Always Verify
- No Trust in Location
- The System Will Be Breached

The Explicitness Virtue

We consider **explicitness** a primary virtue in systems design.

- Never assume or take for granted
- Be explicit about anything that matters ...
- ... (and be silent about things that do not matter)

The 12 Rules

1. Threat Model Your System (regularly).
2. Identify and Authenticate All Device Accesses.

3. Device Accesses Must Be Authorized and Verified.
4. Channel Separation Must Be Enforced.
5. There Must Be Mandatory Log Capture and Log Analytics.
6. There Must Be Mandatory Input Data Validation.
7. There Must Be Input Data Validation Testing.
8. There Must Be Data Integrity Protection.
9. There Must Be Data Confidentiality Protection.
10. There Must Be Platform Integrity Protection.
11. Security Must Be Tested Exhaustively.
12. Plan to Replace the Legacy Devices.

Funding Open access funding provided by University Of South-Eastern Norway.

References

1. Alexander, Otis, Belisle, Misha, & Steele, Jacob. (2020). *MITRE ATT&CK® for industrial control systems: Design and philosophy* (p. 03). MITRE: White paper.
2. Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero Trust Architecture. NIST Special Publication 800-207, NIST, 08 (2020).
3. ENISA. (2020). *Enisa threat landscape: The year in review*. ENISA: ETL.
4. DRAGOS. ICS Cybersecurity; Year in review (2020). White paper, DRAGOS, (2020).
5. Shostack A (2014) Threat modeling: Designing for security. Wiley
6. NSA. (2021). Embracing a zero trust security model (v1.0). Cybersecurity. *Information, 02*.
7. Industrial Control Systems Cyber Emergency Response Team (ICS-CERT. Recommended Practice: Improving Industrial Control System Cybersecurity with Defense-in-Depth Strategies. Department of Homeland. (2016). *Security, 09*,
8. Rory Ward and Betsy Beyer. (2014). BeyondCorp: a new approach to enterprise security. *; login: the magazine of USENIX & SAGE, 39*(6), 6–11.
9. Zimmer, Bryan, & LISA: A practical zero trust architecture. In Enigma, . (2018). *Enigma 2018* (p. 2018). CA: Santa Clara.
10. One, Aleph. (1996). Smashing the stack for fun and profit. *Phrack magazine, 7*(49), 14–16.
11. Lhee, Kyung-Suk., & Chapin, Steve J. (2003). Buffer overflow and format string overflow vulnerabilities. *Software-Practice and Experience, 33*(5), 423–460.
12. Bishop, Matt, Engle, Sophie, Howard, Damien, & Whalen, Sean. (2012). A taxonomy of buffer overflow characteristics. *IEEE Transactions on Dependable and Secure Computing, 9*(3), 305–317.
13. Halfond William G, Viegas Jeremy, Orso Alessandro, et al (2006) A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE international symposium on secure software engineering*, volume 1, pages 13–15. IEEE.
14. Meyer, Bertrand. (1992). Applying design by contract. *Computer, 25*(10), 40–51.
15. Volpano Dennis, and Smith Geoffrey (1997) A type-based approach to program security. In Michel Bidoit and Max Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development*, pages 607–621, Berlin, Heidelberg. Springer Berlin Heidelberg. ISBN 978-3-540-68517-3.
16. Abadi, Martín, & Needham, Roger. (1996). Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering, 22*(1), 6–15.

17. ITU-T (2015). Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation. Recommendation X.680, ITU-T, 08.
18. Miller, Barton P., Fredriksen, Louis, & So, Bryan. (1990). An empirical study of the reliability of unix utilities. *Communications of the ACM, 33*(12), 32–44.
19. Eugene, H. (1989). The internet worm program: An analysis. *ACM SIGCOMM Computer Communication Review, 19*(1), 17–57.
20. Miller, Barton, Zhang, Mengxiao, & Heymann, Elisa. (2020). The relevance of classic fuzz testing: Have we solved this one? *IEEE Transactions on Software Engineering*
21. Bhme, Marcel, & Cadar, Cristian. (2020). *and Abhik Roychoudhury*. Fuzzing: Challenges and reflections. IEEE Software.
22. Zhao, Hui, Li, Zhihui, Wei, Hansheng, Shi, Jianqi, & Huang, Yanhong. (2019). Seqfuzzer: An industrial protocol fuzzing framework from a deep learning perspective. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 59–67. IEEE.
23. Doctorow, Cory. (2004). *Microsoft research drm talk* (p. 17). Microsoft Research Group, Redmond, WA: Transcript.
24. Peterson, Jordan B. (2018). 12 rules for life: An antidote to chaos. *Random House Canada*.

Geir M. Køien received his PhD from Aalborg University, on access secuirty for mobile systems. He has also worked for many years in industry, including LM Ericsson Norway and Telenor R&D. During these years he worked extensively with mobile systems and with security and privacy. He has also worked with the Norwegian Defence Research Establishment and with Norwegian Communications Authority on various security and communications related projects. Currently, he is a professor with the University of South-Eastern Norway (USN).