Malware Tolerance

Report 5

Michael Denzel <m.denzel@cs.bham.ac.uk>

April 5, 2016

Supervisor: Prof. Mark Ryan
Thesis group: Dr. Flavio Garcia,
Dr. David Parker (RSMG Member)

CONTENTS

List of Tables

List of Abbreviations

ARM	Advanced RISC Machine
DCS	Distributed Control System
FSM	Finite State Machine
GPL	GNU Public License
ICS	Industrial Control System
IT	Information Technology
N/A	Not Available
NIST	National Institute of Standards and Technology
OEM	Original Equipment Manufacturer
OS	Operating System
PLC	Programmable Logic Controller
RISC	Reduced Instruction Set Computing
SCADA	Supervisory Control and Data Acquisition
SGX	Intel Software Guard Extensions
TEE	Trusted Execution Environment
TPM	Trusted Platform Module
USB	Universal Serial Bus

1 Introduction

Recent attacks (e.g. Stagefright [4], BadUSB [15], Zeus/ZitMo trojan [7]) show that it is challenging to defend a system and even more difficult to guarantee the honesty of a system.

We propose a new technique, *malware tolerance*, to deal with sophisticated attacks. Our idea is that a system of multiple components can distribute trust over these components in such a way that the individual component cannot meaningfully tamper with the resources.

The parts of a malware tolerant system should be as independent as possible to avoid an attacker gaining control over multiple components at the same time. Thus, separate devices of different type are more suitable than integrated hardware.

The components should be able to hold security credentials and have to have some basic protection, but, since one compromised part does not render the whole system vulnerable, they do not need to be ultimately trustworthy. In this sense, the components only need to be "semi" trusted.

We already identified a few semi trusted devices that are suited for malware tolerance: Smart-cards, security tokens as used in online banking, a special USB-stick [18], and integrated Trusted Execution Environments (TEEs) like the Trusted Platform Module (TPM), Intel Software Guard Extensions (SGX), and ARM TrustZone.

Since SGX was not available until lately, we focused on ARM TrustZone. TrustZone splits the system into two worlds, the *normal world* and the *secure world*. This separation of worlds matches the idea of malware tolerance and makes TrustZone a suitable technology.

2 Background: Industrial Control Systems

For now, we focus on companies which utilise an Industrial Control System (ICS). Those companies produce goods (e.g. chemical industry, automotive industry, and food industry) or manage an architecture (public transport companies, water/gas suppliers, and power plants). This management is controlled by ICSs.

Industrial Control Systems are computers that supervise other computers, machines, and manufacturing. Usually, a special kind of computer, a so-called Programmable Logic Controller (PLC), is controlling network cells or machines. PLCs can be organised in a centralised or distributed manner. The former is called Supervisory Control and Data Acquisition (SCADA) architecture while the latter is named Distributed Control System (DCS).

While the system architecture of ICSs changed from special purpose computers to commodity computers, the tasks and objectives remained distinct. The National Institute of Standards and Technology (NIST) [21] and the US Homeland Security [6] described the key differences between traditional IT architectures and ICS (see Table 1).

Table 1: Comparison IT System to ICS (adapted from [6])

Category	IT System	ICS
Objective	Confidentiality/integrity	Availability
Performance	Maximum throughput	Responsiveness/real-time operation
Antivirus and mobile code	Very common, easily deployed and updated	Performance impact, legacy problems
Patch management	Remote and automated	Very long runway to install patches, Original Equipment Manufacturer (OEM) specific patches
Lifetime	2-3 years, multiple vendors	10-20 years, same vendor
Testing and audit	Modern methods	Has to be tuned to system, fragile equipment might break
Change management	Regular and scheduled	Strategic scheduling, non trivial due to impact
Asset classification	Common practice	Only performed when obligated
Incident response	Easily developed and deployed	Uncommon beyond system resumption
Physical security	Poor (office systems) to excellent (critical operations systems)	Excellent (operations centres)
Secure systems development	Part of development process	Not part of process
Security compliance	Limited regulatory oversight	Specific regulatory guidance

The first five properties of Table 1 (above dashed line) affect security solutions the most.

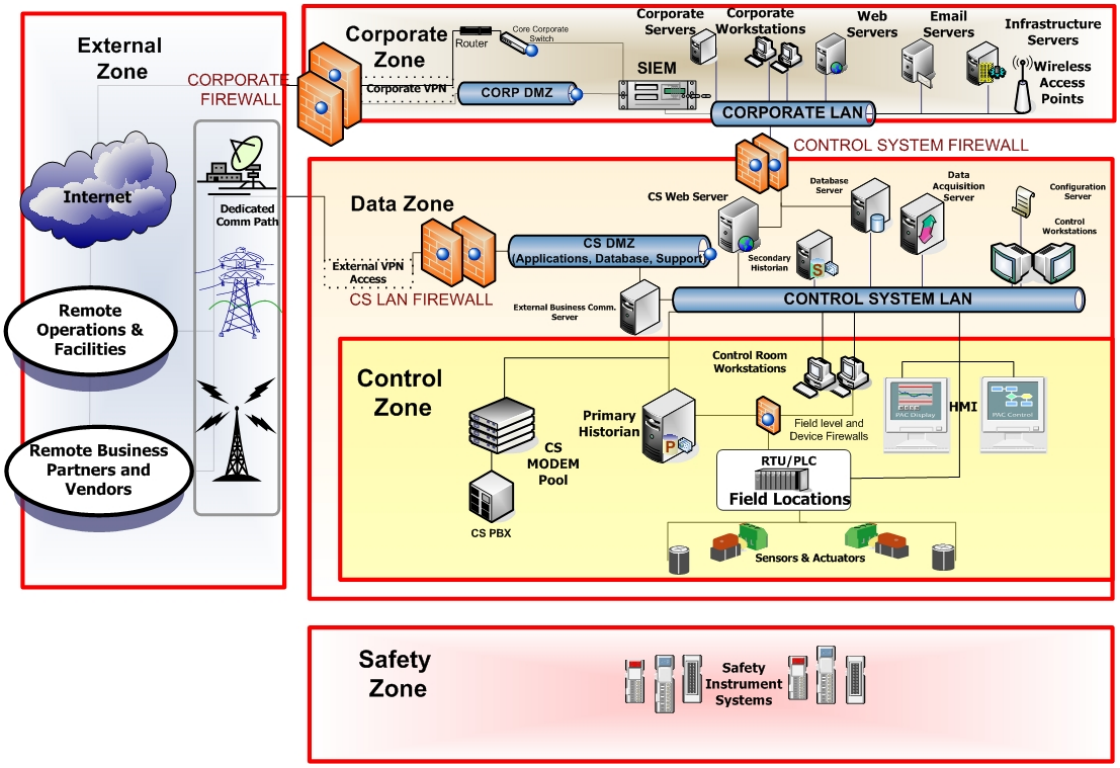
Common defensive solutions like anti-virus and other host-based intrusion detection systems are not applicable because they would affect availability and real-time operation. ICSs are rarely patched and run long-term (10-20 years) as they are e.g. deployed at inaccessible locations[1] and changes sometimes require re-certification [10] of the whole infrastructure.

These systems are difficult to defend because ICSs adopt the standard computer architecture but lack any form of protection exposing a large attack surface. [21] Stuxnet, Duqu, Flame, Gauss, Energetic Bear, and Epic Turla [8] are real-world examples of such attacks and demonstrate how dangerous intrusions of ICSs are.

NIST [21] and US Homeland Security [6] recommended to divide the company network into multiple layered zones (see also Fig. 1) which shield ICSs from the outside world. Additionally, they recommend to air-gap the most critical systems in the so-called safety zone.

[1] e.g. oil pipeline in the Atlantic Ocean

Figure 1: Recommended Network Structure to Defend ICS [21, 6]



However, e.g. Stuxnet [11] managed to overcome air-gaps by spreading via USB sticks. Installing more and more security solutions in front of ICSs rather mitigates the problem. A more effective approach would be to take up improving ICSs itself.

As current defensive measurements are not applicable, new security techniques are needed to protect these specialised systems. In particular, we need solutions that maintain high availability of the system and are suitable for the real-time requirements of ICSs.

In the next chapters, we will present our idea of a next generation ICS which automatically heals itself from ordinary or malicious faults.

3 Next Generation Industrial Control System

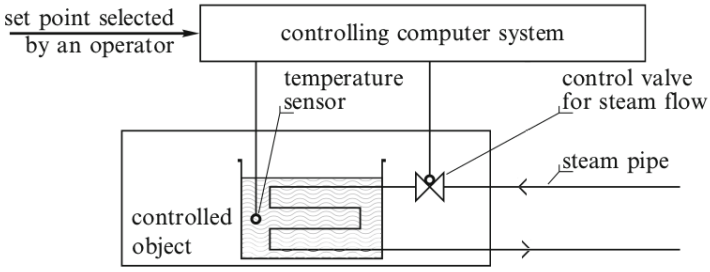
3.1 Idea

Our idea is to restore an ICS at run-time to a known good state after an attack or a fault. For this, we leverage state-of-the-art security features of modern CPU's, namely Intel SGX or ARM TrustZone. These so-called Trusted Execution Environments provide multiple isolated computational units (TrustZone only provides two). We use these hardware features to split the system into two components: an outer layer running the ICS tasks and an inner layer which restores the outer layer in case of faults or attacks. As a result, we need to provide a detection routine, a real-time operating system scheduler, and self-healing functionality (Fig. 2b).

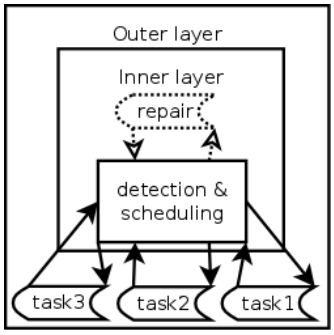
Detection: The detection routine will compare the current state of the system – i.e. sensor and actuator values – to a behaviour model. This comparison can identify invalid states or invalid transitions regarding our model and then trigger the self-healing process.

The model consists of several finite-state machines which are defined per task and characterised by the data of sensors, actuators, and a policy (e.g. a maximum temperature). Finite-state machines have two benefits: first, they are deterministic and fit well with real-time operation systems. Second, we can generate them offline since the behaviours of ICS are well-defined compared to commodity computers.

Distributed Detection: When the system consists of several redundant ICSs, detection could also be applied in a distributed manner. Every device with



(a) Control loop [9]



(b) Proposed architecture

Figure 2: Example Scenario

access to particular sensors and actuators can verify the current state of that particular sub-system and identify a misbehaving component. We can extend this principle to automatically use sane components and tolerate attacks of some of them.

Self-healing: When the detection routine identified a (potentially malicious) fault, the scheduler will replace the faulty task with an online restoration routine. To maintain availability of the other tasks, this routine will only run during the time slice of the faulty task. For restoration, we will load a verified image of the task from read-only memory, verify all patches that took place since the initial image, and apply them.

Our procedure will automatically recover from ordinary faults (safety) and attacks like e.g. virus infections of binaries (security). Since the initial vulnerability, through which the infection happened, is likely unpatched it is crucial to signal this to the administrators.

First considerations: We identified a few challenges for this self-healing architecture: attacks on the restoration routine, patching the devices, and the accuracy of the detection algorithm are key issues that need to be considered.

While the centralised self-healing mechanism prevents destruction through outer layer updates, updates to scheduler or self-healing routine could still render the device unusable. Distributed detection can already reveal malicious or faulty patches to the restoration routine but we need a mechanism to enforce restoration of certain devices if the majority of devices agree. Such a distributed self-healing system would automatically recover from malicious or faulty updates – even to the inner layer.

3.2 Example

Consider a control loop with an ICS which heats liquid. The system is attached to a temperature sensor and a valve to adjust the heating (Fig. 2a). Let us further assume, the system runs three tasks:

$task_1$: a temperature control task with a policy for minimum and maximum temperature,

$task_2$: a task sending the temperature data to a historian server,

$task_3$: and a task to handle user input from the control centre (e.g. ssh or similar).

Detection for $task_1$: We can create a graph of valid states and transitions from the policy, the original behaviour of $task_1$, and the positions of the valve, i.e. *open* or *closed*. States outside of this model, like $(temp < min \;\&\&\; temp > max)$, and invalid transitions, e.g. $(temp > max;\ valve\ closed)$ to $(temp > max;\ valve\ open)$, are considered failures and activate the self-healing routine for $task_1$. Note that during restoration $task_2$ and $task_3$ would operate as normal.

3.3 Improvements

3.3.1 Detection Routine

A first improvement of the Finite State Machine (FSM) would be a timed FSM similar to UPPAAL [12]. This way, transition states which take longer than expected can be identified. E.g. the ICS should only be a very short time in state $(temp < min \;\&\&\; valve\ closed)$ and quickly open the valve in order to reach adequate temperatures. The time intervals of the timed FSM would be equivalent to the sampling rate of the sensors of the ICS.

Another possibility are regular expressions as FSM

can be converted into regular expressions. [14] To speed up the simulation, we could create such regular expressions for sensor and actuator values of the ICS and compare recent sensor readings to the regular expression. This way, no look up table is needed and code could be generated directly.

Further improvement could be through statistical methods like non-linear regression analysis. We want to predict the next sensor value depending on the last e.g. 5 ones. This way, manipulated sensor values can be detected. Correct sensor values must be mapped to output values (i.e. actuator values) which is possible with linear interpolation. We would also like to explore additional estimation techniques such as the Gauss-Newton algorithm.

3.3.2 Patch Management

A distributed self-healing system could also enable a certain type of patch management. Patches could be applied on one of the N devices first and after a testing phase implemented on all devices. In case of faults, the self-healing routine would restore the state before the update instead of the initial state.

Updates and patches need to be authenticated and hardened against rollback attacks and similar.

3.4 Implementation: Self-Healing Trusted Execution Environment

We examined a couple of Operating System (OS) which seemed to be suitable as a TEE for a self-healing ICS (see Table 2). All of the examined OS are micro-kernels or virtualisation OS as monolithic OS, bibliography OS, and exo-kernels seem to be less popular for TEEs. This is understandable considering that monoliths are fairly large and exo-kernels and bibliography OS grant programs direct hardware access which is unacceptable from a security perspective.

The micro-kernel architecture matches the architecture of TrustZone since it separates the system into two worlds: kernel and OS services. The services could run unprivileged in the normal world of TrustZone independent from the kernel in the secure world. This separation also matches to the idea of distributing trust in malware tolerance.

We ported *FreeRTOS*, a compact real-time micro-kernel OS, together with the basic C-library *newlib* to ARM. Our test hardware consists of a *FriendlyARM Mini6410* and a *FreeScale i.MX53*.

Table 2: Considered Operating Systems

Operating System	Developer	License	Architecture	TrustZone
Genode [1, 5]	Genode Labs	GPL	microkernel	✓
Andix [2, 3]	Fitzek et al.	GPL	virtualisation OS	✓
Qubes [17]	Rutkowska et al.	GPL	virtualisation OS	-
FreeRTOS [16]	Real Time Engineers Ltd.	GPL	microkernel	∼[1]
T6 [13]	Li et al.	N/A	N/A	✓
SierraTEE [19, 20]	Sierraware	closed[2]	virtualisation OS	✓

[1] We are currently porting FreeRTOS to TrustZone on the *FriendlyARM Mini6410* and *FreeScale i.MX53*.
[2] "The source code is no longer available for download." [19]

4 Timetable

Table 3: Milestones

2014-03-14	Start
2014-05-19	**Report 1**
2014-06	Literature review
2014-09-20	**Report 2**
2014-10	Explored scenario “storage” under two assumptions
2014-12	Idea: double encryption
2015-01	Researched formal representation of distributed systems, Idea: Trusted input (“sign-what-you-type”)
2015-02/-03	**Report 3:** Thesis proposal
2015-05	Expanded trusted input idea, verification with ProVerif
2015-06	**Paper 1:** Smart-Guard
2015-07	Presentation at CryptoForma Workshop (CSF 2015)
2015-08	Researched TrustZone (papers, OS, *FriendlyARM* board)
2015-09-20	**Report 4**
2015-10	Researched Multi-Party Computation/Industrial Control Systems
2015-12	Reworked Smart-Guard paper
2016-01	Examined TrustZone of *FriendlyARM Mini6410*
2016-02	Researched self-healing systems; adjusted *FreeRTOS* to run on *FriendlyARM* board
2016-03/-04	Improved scenario of Smart-Guard; adjusted *FreeRTOS* to run on *FreeScale i.MX53*
2016-04-03	**Report 5**
2016-05/-06	**Paper 2:** A self-healing ICS using TrustZone
2016-06	Malicious fault detection/Trusted Execution Operating Systems
2016-08	(Paper 3: Trusted Execution/Fault detection)
2016-08	Begin writing the dissertation
2016-10-02	**Report 6**
2017-03-13	Planned submission
2017-04-09	Report 7
2017-10-08	Report 8
2018-03-13	Submission deadline

5 Conclusion

In summary, we will focus on self-healing Industrial Control Systems on the basis of ARM TrustZone. Future work will be on improvements to the fault detection with special emphasis on maliciously induced faults and patch management.

References

[1] Norman Feske and Christian Helmuth. An exploration of arm trustzone technology. Online, 2014. URL `http://genode.org/documentation/articles/trustzone`. accessed: 2015-02-25.

[2] Andreas Fitzek. Development of an arm trustzone aware operating system andix os. Master's thesis, Graz University of Technology, April 2014. URL https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=77938. accessed: 2015-12-07.

[3] Andreas Fitzek, Florian Achleitner, Johannes Winter, and Daniel Hein. The andix research os-arm trustzone meets industrial control systems security. In *IEEE International Conference on Industrial Informatics (INDIN)* Fitzek [2]. URL http://www.arrowhead.eu/wp-content/uploads/2013/03/TUG_Fitzek_INDIN2015.pdf. accessed: 2015-11-18.

[4] Thomas Fox-Brewster. Stagefright: It only takes one text to hack 950 million android phones. Online, July 2015. URL http://www.forbes.com/sites/thomasbrewster/2015/07/27/android-text-attacks/. accessed: 2015-08-17.

[5] Genode Labs. Genode operating system framework. Online, 2015. URL http://genode.org. accessed: 2015-08-27.

[6] Homeland Security. Recommended practice: Improving industrial control systems cybersecurity with defense-in-depth strategies. Technical report, U.S. Homeland Security, October 2009. URL https://ics-cert.us-cert.gov/sites/default/files/recommended_practices/Defense_in_Depth_Oct09.pdf. accessed: 2015-12-10.

[7] Kaspersky. Teamwork: How the zitmo trojan bypasses online banking security. Online, October 2011. URL http://www.kaspersky.com/about/news/virus/2011/Teamwork_How_the_ZitMo_Trojan_Bypasses_Online_Banking_Security. accessed: 2015-02-16.

[8] Kaspersky Lab. Empowering industrial cyber security. Technical report, Kaspersky Lab, 2015. URL http://media.kaspersky.com/en/business-security/Empowering%20Industrial%20Cyber%20Security_web.pdf. accessed: 2016-02-10.

[9] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*, volume 2. Springer Science & Business Media, 2011.

[10] Maryna Krotofil and Dieter Gollmann. Industrial control systems security: What is happening? In *11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 670–675. IEEE, 2013. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6622964. accessed: 2015-12-01.

[11] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *Security & Privacy, IEEE*, 9:49–51, Nov 2011. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5772960. accessed: 2014-08-06.

[12] Kim Larsen, Wang Yi, and Paul Pettersson. UPPAAL. online, Jun 2015. URL http://www.uppaal.org/. accessed: 2016-03-14.

[13] Wenhao Li, Yubin Xia, Liang Liang, Mingyang Ma, Zhichao Hua, Jinyu Gu, Jinchen Han, and Haibo Li. Trustkernel. Online, 2013. URL https://www.trustkernel.com/. accessed: 2015-08-27.

[14] Alexander Meduna, Lukas Vrabel, and Petr Zemek. Converting finite automata to regular expressions. online, 2012. URL http://www.fit.vutbr.cz/~izemek/grants.php.cs?file=%2Fproj%2F589%2FPresentations%2FPB05-Converting-FAs-To-REs.pdf&id=589. accessed: 2016-03-14.

[15] Karsten Nohl and Jakob Lehl. BadUSB–on accessories that turn evil. Online (Black Hat 2014), 2014. URL https://srlabs.de/blog/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf. accessed: 2015-02-09.

[16] Real Time Engineers Ltd. Freertos. online, 2016. URL http://www.freertos.org/. accessed: 2016-03-08.

[17] Joanna Rutkowska, Marek Marczykowski, and Wojciech Porczyk. Welcome to the qubes os project. Online, October 2015. URL https://www.qubes-os.org/. accessed: 2015-10-12.

[18] Seal One AG. Seal one usb - offering simplicity and elegance. Online, 2013. URL http://www.seal-one.com/products-list.en-UK.html. accessed: 2014-08-21.

[19] Sierraware. Open virtualization's sierravisor and sierratee. Online, 2013. URL http://www.openvirtualization.org/. accessed: 2015-08-27.

[20] Sierraware. Sierratee for arm trustzone. Online, 2015. URL http://www.sierraware.com/open-source-ARM-TrustZone.html. accessed: 2015-08-27.

[21] Keith Stouffer, Joe Falco, and Karen Scarfone. Guide to Industrial Control Systems (ICS) security. *NIST Special Publication*, 800(82):1–155, June 2011. URL http://www.gocs.com.de/pages/fachberichte/archiv/164-sp800_82_r2_draft.pdf. accessed: 2015-08-21.

Appendices

Table 4: Attended Workshops

Date	Workshop/Conference	Location
7th May 2014	Google Hack Day (Certificate Transparency)	Google London
19th-23rd May 2014	Academic Writing Seminar	University of Birmingham
27th-28th May 2014	CryptoForma 2014	University of York
14th-18th Jul 2014	Enterprise Summer School	University of Birmingham
23rd-24th Jul 2014	Publishing Academic Journals, Editing your writing	University of Birmingham
22nd Oct 2014	Time Management	University of Birmingham
Oct 2014 - Jan 2015	Research Skills Seminar	University of Birmingham
18th Nov 2014	Speed Reading	University of Birmingham
24th Nov 2014	Note Taking	University of Birmingham
2014	Cryptography 1	University of Stanford (Coursera)
2014 - 2015	Writing in the Sciences	University of Stanford (Coursera)
14th-15th Jan 2015	CryptoForma 2015	University of Kent
13th July 2015	CryptoForma Workshop at CSF 2015	University of Verona
31th Aug-5th Sept 2015	FOSAD Summer School 2015	University Residential Centre of Bertinoro
29th Jan 2016	RITICS meeting	Imperial College London