# Malware Tolerance

Report 4

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

September 18, 2015

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

# CONTENTS

## List of Figures

## List of Tables

# List of Abbreviations

| | |
|---|---|
| **ABE** | Attribute-based Encryption |
| **APT** | Advanced Persistent Threat |
| **ARM** | Advanced RISC Machine |
| **COM** | Communication Port |
| **DCS** | Distributed Control System |
| **DVD** | Digital Video Disc |
| **HDCP** | High-bandwidth Digital Content Protection |
| **HDMI** | High Definition Multimedia Interface |
| **HIDS** | Host Intrusion Detection System |
| **ICS** | Industrial Control System |
| **I/O** | Input/Output |
| **IP** | Internet Protocol |
| **KP-ABE** | Key-Policy Attribute-based Encryption |
| **mCP-ABE** | mediated Ciphertext-Policy Attribute-based Encryption |
| **MPC** | Multi-Party Computation |
| **mRSA** | mediated RSA |
| **OS** | Operating System |
| **PC** | Personal Computer |
| **PGP** | Pretty Good Privacy |
| **PLC** | Programmable Logic Controller |
| **RISC** | Reduced Instruction Set Computing |
| **RSA** | Rivest-Shamir-Adleman Cryptosystem |
| **SCADA** | Supervisory Control and Data Acquisition |
| **SGX** | Intel Software Guard Extensions |
| **TCP** | Transmission Control Protocol |
| **TEE** | Trusted Execution Environment |
| **TPM** | Trusted Platform Module |

# 1 Introduction

Recent attacks (e.g. Stagefright [7], BadUSB [15], Zeus/ZitMo trojan [11]) 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. [4, 5]

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 [16], and integrated Trusted Execution Environments (TEEs) like the Trusted Platform Module (TPM), Intel Software Guard Extensions (SGX), and 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 (see also Chapter 3.2).

# 2 Scenarios

In order to explore malware tolerance, we focused on a few scenarios which we will explain in the following sections. In brief, the scenarios are (1) protecting assets of a company, (2) malware tolerant password authentication, and (3) malware tolerant messaging.

## 2.1 Protecting Company Assets

Malware tolerance is aimed at more sophisticated threats, which commonly attack more valuable targets like company infrastructures. We assume that parts of the internal network are already in control of an adversary but want to protect the assets of the enterprise – e.g. company secrets, customer data, production lines, and control systems.

We will now introduce two different scenarios in connection with company assets:

1. Protecting Industrial Control Systems (ICSs)
2. Secure data storage

### 2.1.1 Protecting Industrial Control Systems

In this chapter we focus on companies which produce goods (e.g. chemical industry, automotive industry, and food industry) or manage an architecture (public transport companies, water/gas suppliers, and power plants). In other words, this section is about companies that utilise an Industrial Control System (ICS).

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). These systems are difficult to defend because ICSs adopt the standard computer architecture but lack security solutions. [19] For example, ICSs support TCP/IP but do not run a firewall or antivirus.

As Stuxnet [12] shows, adversaries already target such systems with Advanced Persistent Threats (APTs). Hutchins et al. [9] classified these attacks into seven steps:

1. Reconnaissance
2. Weaponisation
3. Delivery
4. Exploitation
5. Installation
6. Command & Control
7. Actions on the objectives

In general, malware tolerance is applicable in steps 4 to 7. The first two steps take place mostly on the attackers machine which we cannot influence. *Delivery* refers to the action of sending the exploit which we could reject but we hardly have any information about the content at that point. The classification of the content will rather happen shortly before step 4. Therefore, we can interfere with steps 4 to 7.

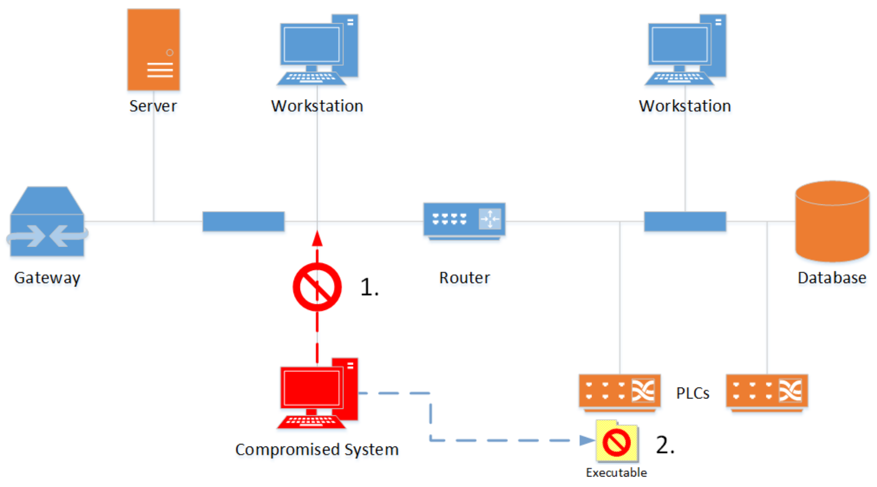
#### Scenario

Since we assumed that some parts of the company network (i.e. some clients and/or servers) are already infected, step 5 took place. The attacker would now try to explore the network and compromise valuable targets like PLCs. Thus, we can interfere with the steps *Command & Control* and *Actions on the objectives* on the network in order to prevent *Exploitation* or later steps on the PLCs.

That means, there are two points at which we could defend the ongoing attack (see Fig. 1). Preferably, the attack should be prevented as early as possible, thus, interaction from the infected machine should be controlled (marked with 1. in Fig. 1).

Figure 1: Assets in the Network of a Company

An adversary will target certain resources in the network of a company (orange) starting from a compromised system (red). We could interfere with (1) *Command & Control* of the compromised system or (2) *Exploitation* of the resources (here: PLCs).



Both interactions from the adversary, i.e. *Command & Control* and *Actions on the objectives*, rely on sending commands to the network. Consequently, our goal is to prevent an adversary with control of the computer from forging or changing commands while enabling an authorised user to still work with the system.

In other words, we need to distinguish between commands from a user and commands from malware.

## Idea

To defend PLCs from attacks, we want to guard them by another device or a proxy, as PLCs usually lack defence mechanisms. Essentially, we move the attack point from the PLC to the proxy with the difference that the proxy is able to defend itself, unlike a PLC. The proxy could be a dedicated device or integrated into a device which is already in place. Two architectures are imaginable: one proxy shielding one (Fig. 2) or multiple (Fig. 3) PLCs behind it. To protect multiple PLC the network needs to be segregated into zones.

Figure 2: PLC-Proxy Setup 1-on-1

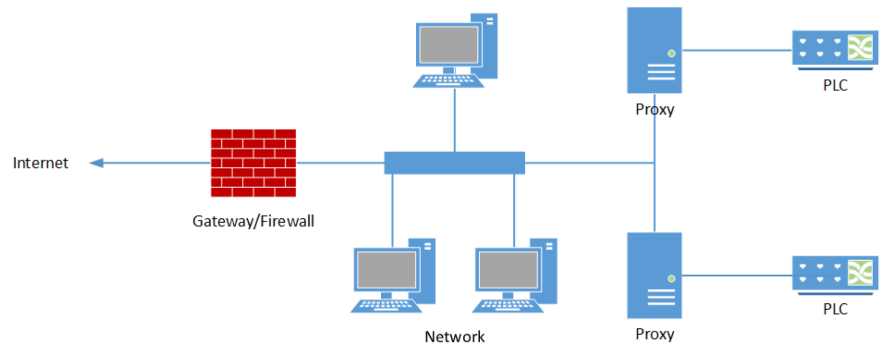
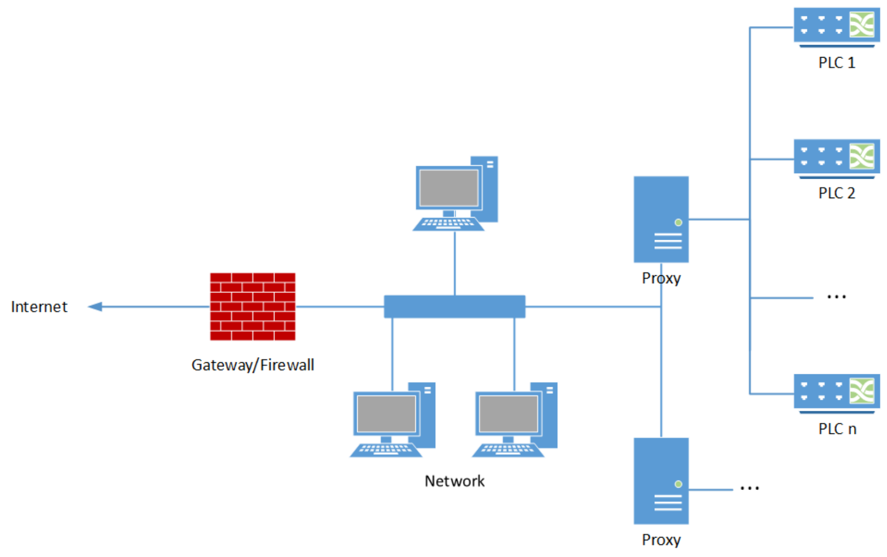


Figure 3: PLC-Proxy Setup 1-on-N



To defend PLCs, the proxy must intercept all commands being sent to the PLCs and check their authenticity. For this purpose, we will cryptographically sign all commands to the PLCs with Smart-Guard (see Chapters 3.1 and 3.1.1).

This might affect the performance of commands but since ICSs usually manage themselves autonomously commands should happen infrequently. Considering for example an automotive manufacturer with PLCs to control production lines. Common commands could be *start conveyor belt* when starting to work and *stop conveyor belt* at the end of a working day. This would even tolerate delays of a few seconds.

It is also possible to generally allow certain commands without authentication at the cost of security for these special commands, e.g. for a status request or an emergency stop for the mentioned conveyor belt. Special care has to be taken to avoid denial of service attacks.

**Improvements: Defending the ICS Proxy**

The proxy architecture will only shift the attack point from the PLC to the proxy. This already improves security because, in contrast to a PLC, the proxy is able to defend itself with e.g. anti-virus, firewall, and Host Intrusion Detection System (HIDS). We would like to improve this architecture to be malware tolerant, too.

Unfortunately, malware tolerance with multiple devices seems impossible: Since a simple PLC does not have any defensive mechanisms, an adversary controlling the channel to this device is in full control over it. That means, all end points to this channel need to be secure. Consequently, malware tolerance is impossible for the channel as one malicious device is enough to compromise the channel and the PLC. Additional devices cannot improve security, in fact they rather lower it. Integrated hardware in the end points can be used to create a one device malware tolerant approach which is weaker than multiple devices.

We will experiment with integrated hardware like TrustZone (see also Chapter 3.2) to harden the proxy architecture. The secure world of TrustZone could encapsulate the network card. Incoming traffic is forwarded to the normal world which verifies signatures of commands. The verification itself could be done truly malware tolerant using e.g. a smart SD-card. However, single point of failure will remain the secure world because it controls the network card and the channel to the PLC. As the informal discussion above shows, improvements to this situation are only possible by adding cryptographic capabilities to the PLC itself.

Additional security could be achieved by testing commands in a sandbox or a shadow honeypot [1] in the normal world after verifying the signature. Instead of preventing the commands of the adversary, this would interfere with the *Exploitation* step on (or shortly before) the PLC.

Such a proxy is also applicable in other scenarios as e.g. smart-homes.

### 2.1.2 Securing Company Data

Information theft is a more general threat to companies compared to security of ICSs. Company secrets, passwords, and customer data must be stored in a secure place even during attacks on network and servers. To be malware tolerant, the storage servers have to provide security despite compromised administrator accounts, infected clients machines, or compromised encryption keys.

Data that is not currently in transit could be encrypted with a distributed key. Decryption would involve Multi-Party Computation (MPC) which distributes trust over several devices such as an access control server of the company, smart-cards, or integrated TEEs in the PCs.

We need to classify access rights for data items into several groups which can be achieved with Key-Policy Attribute-based Encryption (KP-ABE). The idea of KP-ABE is that certain, pre-defined attributes are assigned to each data item and decryption keys are derived from attributes. That means in order to decrypt data the recipient must have these attributes.

The contribution would be to implement Attribute-based Encryption (ABE) in a distributed way. We are looking for a combination of mediated cryptography (e.g. mRSA[2]) and KP-ABE. A possibility could be mediated Ciphertext-Policy Attribute-based Encryption (mCP-ABE) [10].

A scenario is a company network with e.g. customer data and configuration files for their web server. Sales assistants need access to customer data, while web developers should be able to configure the web server. Administrators need to have access to both.

Defining the attributes *sales*, *web*, and *admin*, we could tag the storage like this:

- customer data $\rightarrow$ sales $\vee$ admin
- web server $\rightarrow$ web $\vee$ admin

Keys should be mediated, that means every party only holds a share of the key. The other one is on a key-server or similar. An imaginable approach would be:

- Sales persons and web developers have a share on their working computer while the other half is stored on an access server of the company.
- Administrators have three shares: on their computer, on the access server, and on a smart-card.

For every access, all shares are needed to decrypt the data. An adversary would have to collect every share of a key to attack the corresponding part of the system.

An access policy for the web server could look like the following:

$(\textit{web developer key} \wedge \textit{server key}) \vee (\textit{administrator key} \wedge \textit{server key} \wedge \textit{smart-card key})$

## 2.2 Password Authentication

Password authentication delivers a short secret string to a recipient without enabling replays or revealing the plaintext at any stage. Malware tolerance also considers attacks of the involved components like the keyboard, PC, or potential keyloggers. As a result, keystrokes need to be protected during transit to prevent exfiltration. Outgoing texts need to be encrypted and verified by at least two components. Since verification needs the plaintext, the two verifying components should have limited access to communication channels with external devices (e.g. internet). A modified version of Smart-Guard would be applicable (see Chapter 3.1.3).

### 2.3 Messaging and E-mail

When sending a message to another party, it should be confidential and authenticated. There are many encryption schemes to secure a message during transit over the network. We will look at the security of the end-points, since an attacker is able to retrieve the plaintext of the message when controlling the computers of sender or recipient.

Malware tolerance could improve the security of the end-points by including further components in the computations.

On the sender side, the additional components would enforce encryption of the messages and reject signing spoofed messages. Notice that, since we assume the adversary is already in control of the computer, we cannot prevent him or her from sending (unauthenticated) spoofed messages. However, the recipient is able to identify them easily by their missing or invalid signature.

For the recipient, a malware tolerant approach could prevent decryption of the message if one component disagrees. Moving decryption from untrustworthy components to semi trusted ones would secure confidentiality even further.

## 3 Progress

### 3.1 Smart-Guard

Smart-Guard protects user keystrokes from the input event to a recipient (e.g. a TEE or a third party). Protecting user input in such a way is referred to as trusted input in literature. It is part of trusted I/O and belongs to the domain of trusted execution.

We achieve this by signing and encrypting the keystrokes of a user with a system of three components: a keyboard, a smart-card, and a PC. Our contribution is a protocol that protects against hardware keyloggers and even tolerates attacks of its own components up to a certain point.

In brief, keyboard and smart-card create a shared signature using mRSA [2] and a shared ciphertext with Diffie-Hellman plus a verification step. For example, keyboard and smart-card could produce a shared PGP e-mail.

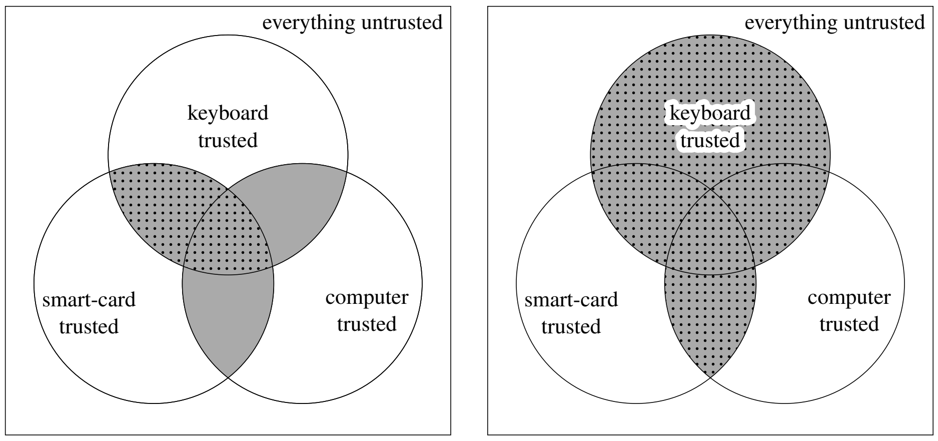
The whole protocol becomes malware tolerant by the fact that keyboard and smart-card both have to agree to create a message while the PC is preventing any of the two devices to connect to the internet directly to leak plaintext. Keyboard and smart-card only communicate via an encrypted channel to remove the PC from the trusted base. This way, we split the system into two zones: one zone with external communication channels (PC) and another zone with plaintext data but without external communication channels (keyboard and smart-card).

Figure 4: Trust Model for Smart-Guard

The "keyboard trusted" circle represents the cases in which the keyboard is trusted, and similarly for the other two circles. Thus, the very centre of the diagram means that all devices are trusted. The intersection between keyboard and computer without smart-card refers to an untrusted smart-card with trusted keyboard and trusted computer.

Grey area: protocol satisfies confidentiality (Fig. 4a) or integrity (Fig. 4b).

Dotted area: protocol defends against hardware keyloggers.



(a) Confidentiality (b) Integrity

We formally verified Smart-Guard using ProVerif, a state-of-the-art protocol verifier. Figure 4a and 4b show the results of our proofs. As we can see, integrity is better protected than confidentiality. The reason is that integrity only requires one honest device, while confidentiality is compromised by one malicious device leaking the plaintext.

Main problem of Smart-Guard is that, since it lacks trusted output, it relies on other techniques to provide this (e.g. trusted output of Zhou et al. [21]). We will introduce possibilities on how to improve or adjust with Smart-Guard in the following chapters.

### 3.1.1 Confidential Output

To improve Smart-Guard, we need to solve the issue of displaying data securely. As Smart-Guard already takes care of integrity, all we need is confidential output – in contrast to (full) trusted output. Even if an attacker modifies the output, the stored input and the produced signature will not change. In addition, a user will immediately notice when the input does not match the output. That is why, output only needs to be confidential.

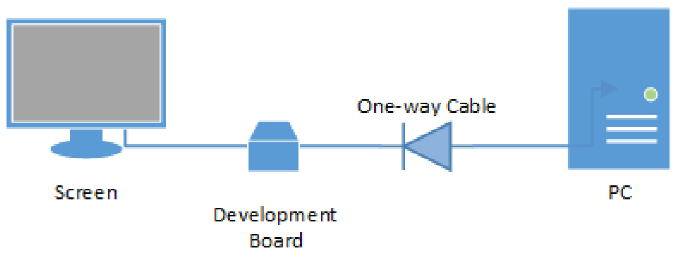
For any form of trusted path, the end points need to hold security credentials. While encryption-capable keyboards exist, encryption-capable screens are not available at the moment. A possibility to improve Smart-Guard is to simulate this encryption-capable screen with a development board (e.g. a raspberry pi) between PC and screen. The optimal solution would be an embedded chip in the screen similar to High-bandwidth Digital Content Protection (HDCP)[1], the encryption of HDMI between DVD-players and televisions.

The setup with a development board could provide trusted output but this is not malware tolerant as malware on the development board compromises confidentiality. To overcome this, we will use a special one-way cable which prevents data exfiltration from the development board. Crafting such a cable can be accomplished by cutting the "read" wire of e.g. a COM cable. Assuming the development board does not have any other interfaces, an adversary cannot return the captured plaintext.

Our architecture is sketched in Fig. 5.

Figure 5: Confidential Output Setup

Our setup for confidential output includes a development board (e.g. raspberry pi) that controls the screen and is connected to the PC via a one-way cable (write-only). This way, the PC can send encrypted output to the screen or rather the development board. The one-way cable prevents an attacker who controls the development board to exfiltrate plaintext.



### 3.1.2 Protecting Commands in Industrial Control Systems

Instead of raising the complexity of Smart-Guard to handle advanced cases, we could also apply it in simpler ways. Since authentication and integrity work better than confidentiality, we could utilise it to protect commands for ICSs.

Commands usually do not require confidentiality but integrity and authentication are important. For Smart-Guard this means we could only execute the shared signature and do not require trusted output or encryption.

We are currently investigating where this is applicable in ICSs.

[1]The HDCP standard is vulnerable to retrieval of the master key. [3]

### 3.1.3 Password Authentication

Another possibility to develop Smart-Guard would be to focus on password inputs. We could suspend trusted output since passwords are not displayed and improve the shared encryption between keyboard and smart-card. Integrity is not needed and could be omitted for performance.

An issue, however, is how to turn it on or off. McCune et al. [14] suggested to use a certain key combination like "@@" to start the service. Another possibility is to use a specific app or a plugin to activate the smart-card and password input. Each time the user wants to authenticate to a service, he or she would launch the authentication program which activates the smart-card and terminates with line-endings.

## 3.2 Trusted Execution

In addition to improving Smart-Guard, we decided to explore TrustZone in more details for various reasons. The vendor model of ARM allows multiple manufacturers to implement ARM architectures which results in different chips being available. Some of the TrustZone implementations leave the cryptographic key unset which enables administrators to set own keys. By that, the software environment is controlled locally instead of trusting a third party. This is different to the TPM chip and presumably SGX where the vendor sets and controls this key. Thus, the ARM architecture is less dependent and enables a more distributed trust model.

We already purchased a *FriendlyARM Mini 6410 SBC board* as this apparently has no key set. [20] We identified three existent TrustZone operating system implementations which we would like to explore:

Table 1: TrustZone Operating Systems

| Operating System | Developer | Code |
|---|---|---|
| Genode [6, 8] | Genode Labs | open source |
| T6 [13] | Li et al. | open source |
| SierraTEE [17, 18] | Sierraware | closed source |

Genode seems to be most promising Operating System (OS) as it is implemented as a microkernel, separating kernel and OS services. The services run unprivileged in the normal world of TrustZone independent from the kernel in the secure world. This separation matches to the idea of distributing trust in malware tolerance. For the ICS proxy the normal world could serve for untrusted tasks like testing commands while the secure world is protecting critical resources as e.g. the network card.

## 4 Timetable

Table 2: 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 [Former: Develop a mechanism to only display needed information] |
| 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 | Improve Smart-Guard, rework paper |
| 2015-11 | Examine TrustZone in depth; goal: Trusted Execution, Multi Party Computation |
| 2016-02 | **Paper 2:** TrustZone for ICS |
| 2016-04-03 | **Report 5** |
| 2016-04 | Possibly: examination of Intel SGX; further use cases of TrustZone |
| 2016-07 | **Paper 3:** TrustZone/Intel SGX |
| 2016-07 | 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 our future work, we will further develop Smart-Guard and experiment with TrustZone. We will concentrate on protecting company assets, password authentication, and messaging.

## References

[1] Kostas G Anagnostakis, Stelios Sidiroglou, Periklis Akritidis, Konstantinos Xinidis, Evangelos P Markatos, and Angelos D Keromytis. Detecting targeted attacks using shadow honeypots. In *Usenix Security*, 2005. URL `https://www.usenix.org/legacyurl/14th-usenix-security-symposium-151-technical-paper-8`. accessed: 2015-02-13.

[2] Dan Boneh, Xuhua Ding, Gene Tsudik, and Chi-Ming Wong. A method for fast revocation of public key certificates and security capabilities. In *USENIX Security Symposium*. Usenix, 2001. URL `http://static.usenix.org/publications/library/proceedings/sec01/full_papers/boneh/boneh.ps`. accessed: 2015-02-17.

[3] Scott Crosby, Ian Goldberg, Robert Johnson, Dawn Song, and David Wagner. A cryptanalysis of the high-bandwidth digital content protection system. In *Security and Privacy in Digital Rights Management*, pages 192–200. Springer, 2001. URL `http://link.springer.com/chapter/10.1007/3-540-47870-1_12`. accessed: 2015-09-14.

[4] Michael Denzel. Malware tolerance. Technical Report 2, University of Birmingham, September 2014.

[5] Michael Denzel. Malware tolerance. Technical Report 3, University of Birmingham, April 2015.

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

[7] 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.

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

[9] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1:80, 2011. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.2845&rep=rep1&type=pdf#page=123. accessed: 2015-08-11.

[10] Luan Ibraimi, Milan Petkovic, Svetla Nikova, Pieter Hartel, and Willem Jonker. Mediated ciphertext-policy attribute-based encryption and its application. In *Information security applications*, pages 309–323. Springer, 2009. URL http://link.springer.com/chapter/10.1007/978-3-642-10838-9_23. accessed: 2015-09-14.

[11] 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.

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

[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] Jonathan M McCune, Adrian Perrig, and Michael K Reiter. Safe passage for passwords and other sensitive data. In *Proceeding of the 16th Annual Network and Distributed System Security Symposium*, February 2009. URL http://www.cs.unc.edu/~reiter/papers/2009/NDSS.pdf. accessed: 2014-11-26.

[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] 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.

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

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

[19] 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.

[20] Johannes Winter. Experimenting with arm trustzone–or: How i met friendly piece of trusted hardware. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 1161–1166. IEEE, 2012. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6296108`. accessed: 2015-02-17.

[21] Zongwei Zhou, Virgil D Gligor, James Newsome, and Jonathan M McCune. Building verifiable trusted path on commodity x86 computers. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 616–630, 2012. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6234440`. accessed: 2014-11-13.

# Appendices

Table 3: 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 |