Contents

Contents 1
1 Security 2
1.1 Scope and Goals 2
1.2 Scope 2
1.3 Goals 3
2 Background 3
2.1 Trusted Computing 3

1 SECURITY

An adversary cannot violate the computer system's integrity if all integrity enforcement mechanisms are properly configured and enabled (including mechanisms protecting runtime process integrity §3.3) because the kernel rejects untrusted files from loading to the memory. However, an adversary can run arbitrary software if she gets enough privileges to boot the computer with disabled enforcement mechanisms. We run a set of micro-benchmarks to estimate the vulnerability window size expressed with Equation(1), during which the integrity violation remains undetected

$$t_{vw} = t_{rq} + 2 * (nt_{re} + t_{vp}), \tag{1}$$

t_{vw} is the vulnerability window size, t is the time to read a TPM quote, n is the maximum number of events that can be opened within t_{rq}, t_{re} is the time to read a single event from the IMA log, t_{vp} is the time required by the agent to verify the policy and by a verifier to send, receive, and process the verification request.

1.1 Scope and Goals

Thesis Statement State-of-the-art security technologies, such as trusted execution environments and trusted computing techniques, protect the confidentiality and integrity of high-assurance security systems' execution and data by running them on top of a trustworthy operating system and in strong isolation from other software executing on the computer. However, these technologies were designed for different use cases, operate under various threat models, and offer distinct security guarantees. It is unclear whether their combination gives any security advantages and what possible security, performance, and usability trade-offs must be made

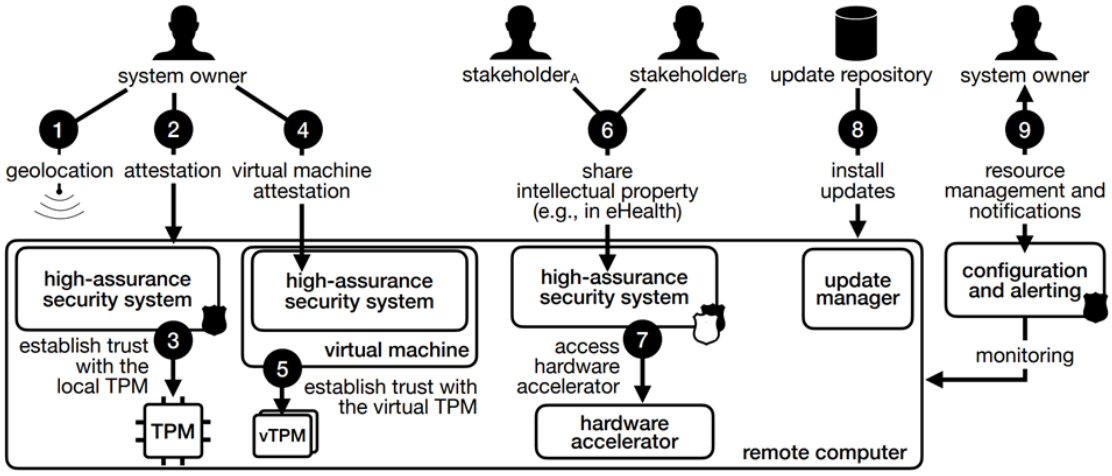

Fig. 1. Research problems addressed in this thesis. The TPM stands for the trusted platform module, a secure element collecting integrity measurements of software executing on a computer. The vTPM stands for a virtual trusted platform module, a TPM emulator collecting virtual machine integrity measurements.

to use them together. This thesis explores how high-assurance security systems can benefit from both trusted computing and trusted execution environments in a secure, practical, and efficient way.

1.2 Scope

In this thesis, I focus on leveraging well-established state-of-the-art security techniques to solve existing problems in domains of remote attestation, secure remote computation, confidential computing, and secure multi-party computation (Figure 1). I start with a fundamental problem of how to verify that a remote computer I interact with is in the expected geographical location, i.e., inside a trusted data center (1). Having a technical assurance that a computer is in the specific data center is the first step in establishing trust with that computer because it allows us to assume that the computer is protected from physical and hardware attacks. Next, I tackle the problem of verifying that this specific computer runs only the expected software in the expected configuration (2). For that, I solve the cuckoo attack problem that prevents from establishing trust with a secure element collecting software integrity measurements (3). After that, I address the issue of how to verify that the operating system running inside a virtual machine executes the expected software in the expected configuration (4). For that, I analyze the existing state-of-the-art approach of virtualizing a security element compliant with the trusted platform module standard [90], and I propose how to improve it (5). After solving the problems 1-5, I show how the proposed mechanisms might be used in practice. I tackle

the problem of how multiple stakeholders could cooperate to perform collaborative computation on remote computers (6) and how they could trade-off between security and performance (7). Specifically, how they might agree on which security mechanisms they want to rely on to protect their workloads while gaining access to hardware accelerators. Finally, I deal with practical issues that limit the usage of integrity enforcement and moni7 1 Introduction toring techniques in production. First, I investigate how to safely install software updates on an integrity-enforced operating system (8), i.e., I look for a solution in which a remote verifier who monitors the integrity of the operating system can ensure that the new integrity state is a result of the trusted update and not of an attack. Second, I check how a system owner could in practice manage a group of resources, i.e., define, configure, and monitor remote computers that differ in terms of running workloads and applied security mechanisms (9).

1.3 Goals

The main goal of this thesis is to build a framework to harden high-assurance security systems. The design goals are:

- Security. The framework should provide strong security guarantees to high-assurance security systems. It should allow individual processes to be run in isolation from privileged software (under the trusted execution environment threat model) and on top of a trustworthy operating system (under the trusted computing threat model).
- Attestation. The system owner should obtain technical assurance that the high-assurance security systems execute in well-defined geographic locations inside an execution environment meeting his security requirements.
- Practicality. The framework must support running legacy systems without requiring source code changes. It is acceptable to instrument source code at the compilation level or run inside virtual machines. It must also support software updates and incur acceptable, low (≤ 10%) performance overhead.
- Usability. The framework must be configurable to individual use cases by allowing users to declaratively state their trust boundaries and make a trade-off between security and performance. It should permit central management (configuration distribution and notification collection) of multiple computing resources.

2 BACKGROUND

High-assurance security systems are deployed as multiple services distributed across multiple computers to ensure high availability, fault tolerance, and resource scalability. As such, their architects, owners, and security officers face the problem of secure remote computation, i.e., how to ensure the confidentiality and integrity of data and code executing on a remote computer? This chapter explores existing techniques that allow users to establish trust with a remote computer, attest to the integrity of software running on such a computer, and protect the integrity and confidentiality of individual applications' code and data against malicious operating systems and administrators. Figure(2) shows existing defense mechanisms used to protect computing systems at different levels.

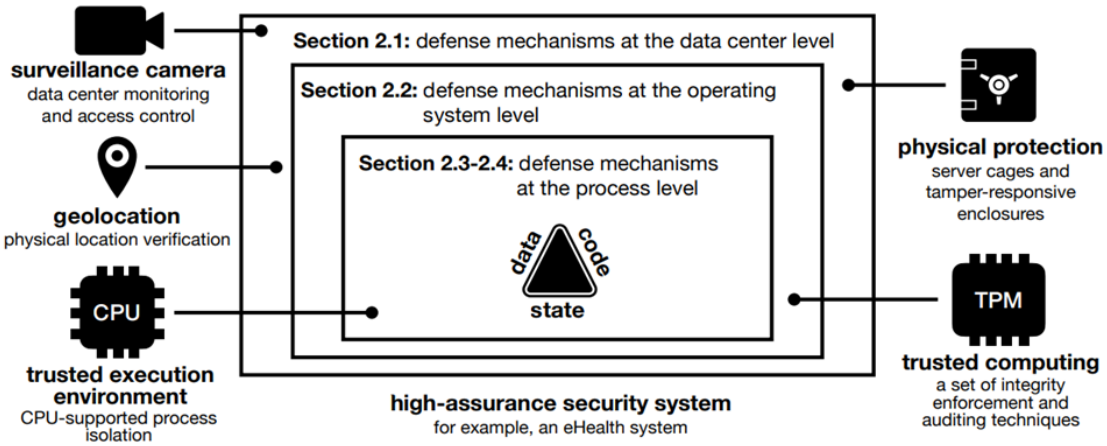

Fig. 2. Overview of mechanisms protecting high-assurance security systems against a wide range of threats

2.1 Trusted Computing

Overview of mechanisms protecting high-assurance security systems against a wide range of threats

(1) remote attestation, i.e., auditing, of what software has executed on the computer
(2) integrity enforcement mechanisms ensuring that only expected software in the expected configuration can execute on the computer.

2.1.1 Security Guarantees. TCTs define how to measure, store, enforce, and report the load-time integrity of firmware and software that has been loaded to the computer's memory since the moment a computer was powered-up. The reporting capability (also referred to as auditing or remote attestation) allows verification that the operating system is in the expected, well-defined state, while the enforcement capability prevents the operating system from moving into an untrusted state by refusing to load an unknown, potentially malicious software to the memory. Crucially, the reporting capability verifies that the enforcement mechanism is enabled and certifies this to a remote entity with the help of the secure element.

Table 1. The VM boot time depending on the TPM.

	MC	TPM	IMA	Boot time
No TPM	-	-	-	9.7 sec (σ = 0.1 sec)
swTPM	-	+	+	14.0 sec (σ = 0.1 sec)
TRIGLAV				
No MC	-	+	+	14.1 sec (σ = 0.1 sec)
With MC	+	+	+	50.8 sec (σ = 0.1 sec)
Fast MC	+	+	+	15.8 sec (estimate)

Table(1) shows how TRIGLAV impacts VM boot times. As a reference, we measure the boot time of a VM without any TPM attached. Then, we run experiments in which a VM has access to different implementations of a software-based TPMs. Except for the reference measurement, the Linux IMA is always turned on. Each VM has access to all available cores and 4 GB of memory. As the guest operating system, we run Ubuntu 18.10, a Linux distribution with a pre-installed tool (systemd-analyze) to calculate system boot times. Shannon's entropy formula:

$$H(X) = -\sum p(x) * \log_2 p(x),$$

$H(X)$ is the entropy; $p(x)$ is the probability of symbol x.

$$hash = (S[i] + d * (hash - S[i - len + 1])) \mod q,$$

where S is the string; d is the alphabet size; q is a prime number.