1 Securing Smart Environments with Authentic Execution

1.1 Introduction

The ultimate goal is to provide confidentiality, integrity and authenticity guarantees of

- the execution of every component of an application
- the communication between two different components
- the interaction between a component and an I/O device

Basing on these security requirements, a framework called Authentic Execution was proposed, to provide strong assurance of the secure execution of a distributed, event-driven application. The Authentic Execution concepts can be applied to a generic TEE; however, the implementation provided only includes support for Sancus, an embedded TEE that extends the TI MSP430 CPU, limiting the applicability of the framework in a real scenario. Therefore, this Master's Thesis aims to remove this limitation by providing the Authentic Execution framework support for SGX, a TEE included in recent Intel processors.

1.2 Background and related work

1.2.1 Trusted Computing

Trusted Computing is based on the following key concepts

- Endorsement Key
- Memory Curtaining
- Secure IO
- Sealed Storage
- Remote Attestation
- Trusted Third Party

1.2.2 Trusted Execution Environments

Sancus consists of

- Infrastructure
  - Nodes
  - Software Provider

  - Software Modules
- Isolation and Memory Access Control
- Remote Attestation
- Secure I/O

Intel SGX consists of

- Isolation
- Enclave Identities
- Local Attestation
- Remote Attestation
- Data Sealing

1.2.3 Smart Farming

Smart Farming Technologies are divided into three main categories

- Data Acquisition Technologies
- Data Analysis and Evaluation Technologies
- Precision Application Technologies

There are various security concerns for SFTs

- Threats to Confidentiality
- Threats to Integrity
- Threats to Availability

1.3 Problem Statement

1.3.1 System Model

The System Model describes a distributed, event-driven application. An event-driven application, is an application that reacts to external inputs, called events. A key component of the application is the Event Manager (EM), which is responsible for receiving and processing external events, as well as executing the logic associated to them. After an event is received, a function (called handler) is executed; handlers can be associated to a specific event both at compile time and at runtime. Then, after the code is executed, the application returns back to a waiting state, until a new event arrives.

The main limitation of this model is that there are no availability guarantees: for instance, if the EMs are controlled by the attacker, they might drop all the

events they receive. Finally, two important aspects need to be emphasised: firstly, that the system is heterogeneous, meaning that the Software Modules (SMs) might be developed for different architectures, according to their functionality. The second significant aspect is that the Deployer is not necessarily the owner of the infrastructure.

1.3.2 Attacker Model

Attackers have the following capabilities

- Software Manipulation (EMs, OS and can add software)
- Communication Network (Sniffing, Modification and MitM)
- Cryptographic (Dolev-Yao model)

Attacks against the hardware are out of scope and side-channel attacks are not considered because although important protection against them is orthogonal and complementary to the methods discussed here.

1.4 Design and Implementation

1.4.1 Extending the Authentic Execution Implementation

The most important extensions are

- Support for Intel SGX
- Many-to-many Relationships (instead of one-to-one connections between input and output)
- Periodic Events (in the EventManager)

1.4.2 Application-level protocol

The protocol consists of the following messages

- Command
- Result
- Connect
- Call
- Remote Output
- Load
- Ping
- RegisterEntrypoint

1.4.3 Authentic Execution in SGX

The Software Modules are written in rust because it provides

- Performance
- Reliability (Memory Safety,...)
- Productivity (Easy to learn)

1.4.4 AuthenticExecution in Sancus

The main changes that were made are the ability to support many-to-many connections and the use of the nonce in the payload to verify whether the message has been processed already or not to avoid availability attacks by replaying messages.

1.4.5 Deploying the System

The reactive tools module is used to easily deploy the system.