**Sixth Framework Programme
Information Society Technology**

RE-TRUST

Remote EnTrusting by RUn-time Software authentication

Project Number: **FP6-021186**

Deliverable: **D1.2**

Software-based method - initial architecture

Contents

1	Summary	2
2	Introduction	3
2.1	Document control	3
2.2	Abstract	3
2.3	Requirements	3
3	Initial architecture	5
3.1	Untrusted host (UH)	6
3.2	Program (P)	6
3.2.1	Entrusting agent/monitor	6
3.3	Application Hosts (AH)	7
3.4	Verification server (VS)	7
3.5	The Network	7
4	Untrusted host	8
4.1	Hardware	8
4.2	HW Virtualization	8
4.3	Operating System	9
4.4	Application Virtualization	9
5	Conclusions	10

1 Summary

Project Number:	FP6-021186
Project Title:	RE-TRUST: Remote EnTrusting by RUN-time Software authentication
Deliverable Type:	RP

Deliverable Number:	D1.2
Contractual Date of Delivery:	Oct 2007
Actual Date of Delivery:	Oct 2007
Title of Deliverable:	Software-based method - initial architecture
Workpackage Contributing to the Deliverable:	WP1
Nature of the Deliverable:	Report
Author(s):	Stefano Di Carlo, Jasvir Nagra, Brecht Wyseur
Contributions:	POLITO: Stefano Di Carlo, Alberto Scionti; KUL: Brecht Wyseur, Dries Schellekens; UNITN: Yoram Ofek, Mariano Ceccato; GEMALTO: Jerome D'Annoville
Reviewer(s):	

Abstract:	This documents presents an initial architecture to solve the remote entrusting problem. Besides proposing any type of solution to the problem, it highlights the main components that will be available to compose the final solution.
Keywords:	
Classification:	N/A

Name of Client:	European Commission
Distribution List:	European Commission Project Partners
Authorised by:	Yoram Ofek, University of Trento
Issue:	1.0
Reference:	RE-TRUST Deliverable
Total Number of Pages:	10
Contact Details:	Prof. Yoram OFEK University of Trento Dept. of Information and Communication Technology (DIT) Trento, Italy Phone: +39 0461 88 3954 (office) Fax +39 0461 88 2093 (DIT) E-mail: ofek -at- dit.unitn.it

2 Introduction

2.1 Document control

Issue number	Issue date	Reason of change
1.1	March 25th, 2007	First release
1.2	July 12th, 2007	Formatting according to the new deliverable template
2.0	October 15th, 2007	Document rewritten based on the comments and requests of EU reviewers
2.1	October 21th, 2007	Introduction of the concept of defender, attacker, end-user
2.2	October 21th, 2007	Network application are not any more a strong requirement. A stand alone application can be converted into a network one.
2.3	October 25th, 2007	Introduction of the concept of monitor
2.4	October 26th, 2007	Introduction of main assumption: trusted zone components are completely trusted. The concept of virtualization has been expanded and fixed in some unclear points

2.2 Abstract

From an architectural point of view, the concept of *remote entrusting* addressed by the RE-TRUST project refers to definition of a system in which one or more remote trusted components are able to detect if software running on an untrusted component is executed untampered.

This deliverable suggests a potential architecture for this type of system. The document will mainly focus on the definition and description of the set of components that will be part of the final system highlighting their security trustworthiness, their vulnerabilities and the configuration in which they are connected together.

The presented architecture represents the starting point for the trust model and proposed solutions for software-only (WP2) and software-hardware solutions (WP3).

The deliverable is organized as follows: first of all, some initial requirements will be analyzed (Section 2.3) followed by the definition of the initial architecture (Section 3). Section 4 will then details the Untrusted Host component that represents a key element in the architecture and finally Section 5 concludes the document.

2.3 Requirements

This section analyzes the main requirements and assumptions to consider in the architecture definition.

First of all let us consider the system to model. In order to identify an architecture as general as possible, we will target a general distributed system, i.e., a system composed of a set of different computational components connected trough “*the network*”. The concept of network here must be considered as an abstract concept (i.e. a communication mechanism between the components). It can be then easily mapped on real mechanisms such as the Internet or a local/corporate network.

Defined the type of system to model, we need to consider the type of software applications that will be executed by the computational nodes composing the system and for which we need to guarantee a *trusted execution* (see Section 2.2). A detailed analysis of classes of applications to consider in the framework of the RE-TRUST project was performed in deliverable 1.1 “*Analysis of generic classes of applications*” [1]. The main requirement coming from this analysis is the need of working with so called “*network applications*”, i.e. applications that require or involve

RE-TRUST Deliverable D1.2 - Software-based method - initial architecture

the network during their execution. Typical examples of these applications are network games, DRM, e-services, peer-to-peer networks, etc..

The proposed architecture will be constructed starting from these initial requirements.

3 Initial architecture

Based on the requirements discussed in Section 2.3, Figure 3 introduces our overall architecture. The architecture is defined using a top-down approach. This section will focus on the top level view whereas the following sections will details when needed the characteristics of the different components.

Figure 1: Overall architecture

There are several actors that use the system. The *defender* is the person who deploys the remote entrusting mechanism and seeks to protect some aspects of the system. All trusted relationships described in the remainder of this document are in terms of what the defender trusts. The other two actors in the system are the *end-user* and the *attacker*. The end-user is the user who the defender aims to provide a service to while the attacker attempts to defeat the goal of the defender. A more in-depth description of these actors is provided in D2.1/D3.1 [2, 3].

The complete system is split in two portions denoted as *zones*:

- *Trusted zone*: this zone represents the portion of the system the defender can trust. This zone includes two different classes of components:
 - *Sources of trust (SoT)*: components that can be completely trusted by every component of the system. They represent the only components able to export some type of trust properties to the system;
 - *Users of trust (UoT)*: components that require to use the trust properties of the system. In the context of the remote entrusting scenario these components are the ones requiring to trust the computation performed on a remote component. We are not interested in evaluating the trust properties of these components, but we are interested in understanding the mechanisms to guarantee to these components the required level of trust.
- *Untrusted zone*: this zone includes the components of the system that cannot be trusted (i.e. the defender and more in general the UoTs cannot trust). Components belonging to this zone are untrusted because an attacker may obtain full control on them and possibly gaining access to a property which the component is supposed to prevent. As an example, a smart card may be able to prevent many types of static analysis by preventing access to the code it is executing, however, it is vulnerable to physical tampering.

Solving the remote entrusting problem means finding a mechanism to let a UoT belonging to the trusted zone trusting a software running on a component belonging to the untrusted zone.

From this classification a first strong assumption comes up: *components belonging to the trusted zone are completely trusted*. We are perfectly aware that real servers and hosts can be attacked (e.g., internally with some back doors

left intentionally by some embittered/dishonest employees) but this problem is out of the scope of the project. The project will focus on the untrusted zone.

In the following subsections, the different components composing the overall architecture will be analyzed.

3.1 Untrusted host (UH)

Any computational node (e.g., a personal computer, a mobile handset, etc.) running in the untrusted zone is identified as Untrusted Host (UH). An UH is able to run a software application P but it cannot provide any guarantee that the execution will respect the original specifications of the software (i.e., the execution of an application on an UH is completely untrusted). A more detailed analysis of the UH architecture will be provided in Section 4.

3.2 Program (P)

We denote with P a software application whose execution must be trusted by a UoT. In our reference architecture we consider the situation in which P is running on an UH (see Section 3.1). In Section 2.3 we already mentioned that the main target of our project are network applications. Even if this initial requirement may be considered as a strong limitation, stand-alone applications are not completely cut off from the scope of the project. Protection techniques as the one presented in [13] already demonstrated how standalone applications can be efficiently turned into networked ones making these applications still compatible with the proposed architecture.

3.2.1 Entrusting agent/monitor

In this section we would like to include some additional considerations about the program P. Even if the goal of this document is to provide a general architecture for the implementation of a remote entrusting system not tight to a specific solution a few specific details can be added to the program P.

To entrust P, some protection techniques should be applied to make difficult for the attacker tampering with the application. In the hypothesis of having a set of techniques that allow a perfect obfuscation of the program (assumption for which we are not going to discuss its feasibility now), a user of trust can easily rely on the results provided by P without the need of introducing any additional complex protection schema. In the case this first hypotheses cannot be considered true we will be forced to add some verification functionalities to P with the purpose of monitoring its execution. We denote this extra functionality as *monitor* (M) or *entrusting agent* (see Figure 3.2.1).

Figure 2: Program P Structure

Due to the extended control of the attacker over the UH, even the extended verification functionalities alone will be probably not enough if the verification is performed locally. For this reason the monitor will be probably the main individual interacting with the verification server to guarantee the application integrity. In any case, to keep the architecture as general as possible, the monitor is depicted with a dashed line to clearly state that it is an optional element.

3.3 Application Hosts (AH)

An Application Host (AH) is an instance of a UoT. i.e., an host interacting with the program P executed on the UH. The Application Hosts represent the components of the system requiring a trusted execution of P.

3.4 Verification server (VS)

The Verification Server is an instance of a SoT for the system. It is the node in charge of verifying a set of assertion/properties of the execution of P and enforce the program to run as intended. This component is trusted. More specifically, in the RE-TRUST project we assume that the Verification Server cannot be physically tampered by the attacker. The attacker has no access or visibility to it and no external view to its internal details is possible to an external observer. The attacker neither knows what software the server is running, nor what the underlying hardware and operating system is. We further assume that the only communication with the Verification Server is via its declared protocol.

As such, a well behaved Verification Server is highly resilient to attack.

While the verification server and the application hosts are depicted as two different components they can be physically implemented by a single host on the network.

3.5 The Network

The network is the key element of our architecture. The network represents the link between the components in the untrusted zone (i.e., untrusted hosts) and the components in the trusted zone (i.e., application hosts and verification server). As shown in Figure 3 the network is identified as a component of the untrusted zone. This means that we cannot rely on any trust properties coming from the network. Nevertheless, our idea is to use the network as a main element to build our trust infrastructure.

4 Untrusted host

This section tries to provide a more detailed view on the architecture of the Untrusted Host. As already introduced in Section 3.1 an Untrusted Host is a computational node (e.g., a personal computer, a mobile handset, etc.) running in the untrusted zone. From an architectural point of view an UH can be modeled as a component organized in a set of nested layers as highlighted in Figure 4.

Figure 3: Untrusted host architecture

The following subsections will provide information about the different layers identified in Figure 4. It is important to note that being the UH part of the untrusted zone its components must be considered as completely untrusted

4.1 Hardware

This layer includes the actual hardware of the host (i.e., CPU, memory, disks, network interfaces). To be general, our architecture does not refer to any specific hardware component but tries to consider any type of hardware level. The only hardware requirement we impose is the availability of a network interface (being the network an essential part of our architecture). Even if in a real implementation the characteristics of a given hardware may be known we do not rely on them to build our trust infrastructure.

4.2 HW Virtualization

In computing, virtualization is a broad term that refers to the abstraction of computer resources [4]. In this context we refer to the virtualization of the hardware layer of a computer based system defined by a virtual machine having its own hardware and allowing a guest OS to be run in isolation.

Projects such as XEN [6] and L4 [9] highlight the high interest on these type of systems and in 2005 and 2006, Intel and AMD provided additional hardware to support virtualization.

In addition, while Figure 4 introduces this layer immediately running over the hardware layer we can have the same type of HW virtualization running over the operating system layer. Examples include VMware [7], and Parallels Desktop for Mac [8].

While virtualization is attractive for improving security through virtual machine isolation [5, 10], it opens a whole new Pandora's box of security problems. For this reason our architecture will have to consider the presence of this layer.

Figure 4 depicts the virtualization layer with a dashed line to highlight that it is optional and it is not always present on an untrusted host.

4.3 Operating System

This layer considers the operating system running on the UH. Again to be general we do not consider any specific operating system such as Linux or Windows but we will try to identify solutions as open as possible.

4.4 Application Virtualization

This layer introduces a second type of virtualization, it considers the situation in which an application is running locally, using local resources, within an appropriate virtual machine which is itself an application or an extension of the operating system. This virtual environment acts as a layer between the application and the operating system, and eliminates application conflicts and application-OS conflicts. Examples include the Sun Java Virtual Machine [11], Mono [12], etc.

This approach to virtualization is clearly different from the hardware virtualization since in this case we do not provide any hardware abstraction (e.g., no additional OS can run on this virtual environment) but we simply want to provide a virtual execution environment. Only an arbitrary line separates it from the world of interpreted language

5 Conclusions

This document presented a preliminary architecture proposed for the implementation of a remote entrusting system. We mainly focused on the definition of the components of the architecture trying to highlight their security trustworthiness, their vulnerabilities and the configuration in which they are connected together.

A strong assumption is introduced by this deliverable: components belonging to what we defined as the trusted zone are completely trusted (see Section 3). For this reason we concentrated on the description of the key element of the untrusted zone that is represented by the untrusted host (see Section 4) trying to model its architecture and its main components.

This architecture should be used as a reference to build specific solution for the remote entrusting problem and eventually refined based on the results obtained by this experimentation.

References

- [1] RE-TRUST Project, *D1.1 - Analysis of generic classes of applications*.
- [2] RE-TRUST Project, *D2.1-Trust model and assumption for software-based TR methods*.
- [3] RE-TRUST Project, *D3.1-Trust model and assumption for SW/HW-based TR methods*.
- [4] Virtualization, <http://en.wikipedia.org/wiki/Virtualization>.
- [5] T. Garfinkel and M. Rosenblum, *When virtual is harder than real: Security challenges in virtual machine based computing environments* In Proceedings of the 10th Workshop on Hot Topics in Operating Systems (HotOS-X), May 2005.
- [6] XEN, <http://www.xensource.com>.
- [7] VMWare, <http://www.vmware.com>.
- [8] Parallels desktop for mac, <http://www.parallels.com>.
- [9] L4 , <http://l4ka.org/>.
- [10] OpenTC , <http://www.opentc.net>.
- [11] Java Virtual Machine (JVM) , <http://java.sun.com/>.
- [12] Mono , http://www.mono-project.com/Main_Page.
- [13] X. Zhang and R. Gupta, *Hiding program slices for software security*, International Symposium on Code Generation and Optimization, 2003. CGO 2003. 3-26 March 2003, pp. 325–336.