POLITECNICO DI TORINO
Repository ISTITUZIONALE

Trust and integrity in distributed systems

Original
Trust and integrity in distributed systems / Su, Tao. - (2017).

Availability:
This version is available at: 11583/2676918 since: 2017-07-20T13:08:11Z

Publisher:
Politecnico di Torino

Published
DOI:10.6092/polito/porto/2676918

Terms of use:
openAccess

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

04 August 2020

POLITECNICO DI TORINO

SCUOLA DI DOTTORATO

Dottorato in Ingegneria Informatica e dei Sistemi – XXVIII ciclo

Tesi di Dottorato

Trust and Integrity in Distributed Systems

Tao Su

Tutore
Prof. Antonio Lioy

Coordinatore del corso di dottorato
Prof. Matteo Sonza Reorda

May 2017

Acknowledgements

First and foremost, I would like to give my special gratitude to my supervisor Prof. Antonio Lioy, for his important suggestions and support throughout my entire Ph.D study. I am extremely grateful for the great efforts he put into guiding and helping me. It was extremely useful for my Ph.D study as well as future career.

Meanwhile, I would also like to thank Dr. Andrea Atzeni, for his support and encouragement in my research work. In addition, I would like to express my great appreciation to all my colleagues, Roberto Sassu for showing me the way to trusted computing, Paolo Smiraglia for helping me solving various technical problems, Dr. Christian Pitscheider, Dr. Daniele Canavese and Dr. Fulvio Valenza for helping me solving a lot of bureaucratic problems, and also the other group members. I would also like to express my gratitude to all the people I met during my Ph.D study at the Politecnico di Torino. It was a great pleasure to work with these nice people, and I had a great time working in the lab during this period of time.

Finally, very special thanks to my family and my beloved wife Dr. Xin Xiao: their support and encouragement are always the fuel of my journey.

Summary

In the last decades, we have witnessed an exploding growth of the Internet. The massive adoption of distributed systems on the Internet allows users to offload their computing intensive work to remote servers, e.g., cloud. In this context, distributed systems are pervasively used in a number of difference scenarios, such as web-based services that receive and process data, cloud nodes where company data and processes are executed, and softwarised networks that process packets. In these systems, all the computing entities need to trust each other and co-operate in order to work properly.

While the communication channels can be well protected by protocols like TLS or IPsec, the problem lies in the expected behaviour of the remote computing platforms, because they are not under the direct control of end users and do not offer any guarantee that they will behave as agreed. For example, the remote party may use non-legitimate services for its own convenience (e.g., illegally storing received data and routed packets), or the remote system may misbehave due to an attack (e.g., changing deployed services). This is especially important because most of these computing entities need to expose interfaces towards the Internet, which makes them easier to be attacked. Hence, software-based security solutions alone are insufficient to deal with the current scenario of distributed systems. They must be coupled with stronger means such as hardware-assisted protection.

In order to allow the nodes in distributed system to trust each other, their integrity must be presented and assessed to predict their behaviour. The remote attestation technique of trusted computing was proposed to specifically deal with the integrity issue of remote entities, e.g., whether the platform is compromised with bootkit attacks or cracked kernel and services. This technique relies on a hardware chip called *Trusted Platform Module* (TPM), which is available in most business class laptops, desktops and servers. The TPM plays as the hardware root of trust, which provides a special set of capabilities that allows a physical platform to present its integrity state.

With a TPM equipped in the motherboard, the remote attestation is the procedure that a physical node provides hardware-based proof of the software components loaded in this platform, which can be evaluated by other entities to conclude its integrity state. Thanks to the hardware TPM, the remote attestation procedure is resistant to software attacks. However, even though the availability of this chip is high, its actual usage is low.

The major reason is that trusted computing has very little flexibility, since its goal is to provide strong integrity guarantees. For instance, remote attestation result is positive if and only if the software components loaded in the platform are expected and loaded in a specific order, which limits its applicability in real-world scenarios. For such reasons, this technique is especially hard to be applied on software services running in application layer, that are loaded in random order and constantly updated. Because of this, current remote attestation techniques provide incomplete solution. They only focus on the boot phase of physical platforms but not on the services, not to mention the services running in virtual instances.

This work first proposes a new remote attestation framework with the capability of presenting and evaluating the integrity state not only of the boot phase of physical platforms but also of software services at load time, e.g., whether the software is legitimate or not. The framework allows users to know and understand the integrity state of the whole life cycle of the services they are interacting with, thus the users can make informed decision whether to send their data or trust the received results.

Second, based on the remote attestation framework this thesis proposes a method to bind the identity of secure channel endpoint to a specific physical platform and its integrity state. Secure channels are extensively adopted in distributed systems to protect data transmitted from one platform to another. However, they do not convey any information about the integrity state of the platform or the service that generates and receives this data, which leaves ample space for various attacks. With the binding of the secure channel endpoint and the hardware TPM, users are protected from relay attacks (with hardware-based identity) and malicious or cracked platform and software (with remote attestation).

Third, with the help of the remote attestation framework, this thesis introduces a new method to include the integrity state of software services running in virtual containers in the evidence generated by the hardware TPM. This solution is especially important for softwarised network environments. Softwarised network was proposed to provide dynamic and flexible network deployment which is an ever complex task nowadays. Its main idea is to switch hardware appliances to softwarised network functions running inside virtual instances, that are full-fledged computational systems and accessible from the Internet, thus their integrity is at stake. Unfortunately, currently remote attestation work is not able to provide hardware-based integrity evidence for software services running inside virtual instances, because the direct link between the internal of virtual instances and hardware root of trust is missing. With the solution proposed in this thesis, the integrity state of the softwarised network functions running in virtual containers can be presented and evaluated with hardware-based evidence, implying the integrity of the whole softwarised network.

The proposed remote attestation framework, trusted channel and trusted softwarised network are implemented in separate working prototypes. Their performance was evaluated and proved to be excellent, allowing them to be applied in real-world scenarios. Moreover, the implementation also exposes various APIs to simplify future integration with different management platforms, such as OpenStack and OpenMANO.

Contents

Summary iii

List of Figures viii

List of Tables x

1 Introduction 1

2 Background of Trusted Computing 9

2.1 Trusted Platform 9

2.2 Protected Capabilities 10

2.2.1 Roots of Trust 10

2.3 Integrity Measurement 12

2.3.1 Trusted Computing Base 13

2.4 Integrity Reporting 14

2.4.1 Key Hierarchy 14

2.4.2 Configuration-based Remote Attestation 17

2.5 Virtualisation 19

2.5.1 Hypervisor-based Virtualisation 19

2.5.2 Operating-System-Level Virtualisation 21

3 Remote Attestation Framework 23

3.1 State of the Art and The Way Forward 24

3.1.1 Contribution 32

3.2 Requirement Analysis 32

3.2.1 Security Requirements 32

3.2.2 Functional Requirements 33

3.2.3 Possible Attacks . . . 33
3.3 General Architecture . . . 34
3.3.1 Attesting Platform . . . 34
3.3.2 Integrity Verifier . . . 35
3.4 Remote Attestation Workflow . . . 36
3.4.1 Registration Phase . . . 37
3.4.2 Remote Attestation Phase . . . 38
3.5 Details of the Framework . . . 38
3.5.1 Trusted Boot . . . 39
3.5.2 Service Load-time Integrity Measurement . . . 40
3.5.3 Integrity Report . . . 41
3.5.4 Analysis Customisation . . . 45
3.6 Application of Remote Attestation Framework in Network Policy Validation Scenario . . . 51
3.6.1 Motivations of A Trusted Network Policy Validator . . . 51
3.6.2 Contribution . . . 52
3.6.3 Architecture . . . 52
3.7 Discussion . . . 55

4 Trusted Channel 57
4.1 State of the Art and The Way Forward . . . 58
4.1.1 Contribution . . . 61
4.2 Requirement Analysis . . . 62
4.2.1 Security Requirements . . . 62
4.2.2 Functional Requirements . . . 62
4.2.3 Possible Attacks . . . 63
4.3 Trusted Channel Architecture . . . 63
4.4 Creating Trusted Channel . . . 65
4.4.1 Extension to IPsec Authentication . . . 65
4.4.2 Extension to Remote Attestation Verifier . . . 68
4.5 Discussion . . . 71

5 Trusted Network 73
5.1 Softwarised Network . . . 73
5.2 Security and Trust in Softwarised Networks . . . 75
5.2.1 Contribution . . . 82
5.3 Requirement Analysis . . . 83
5.3.1 Security Requirements . . . 83
5.3.2 Functional Requirements . . . 84
5.4 Remote Attestation in Lightweight Virtualisation Environments . . . 84
5.4.1 General Architecture . . . 85
5.4.2 Extension of Linux IMA . . . 86
5.4.3 Extension of Remote Attestation Framework . . . 87
5.4.4 Extension of IMA Verification Procedure . . . 88
5.5 Discussions . . . 89

6 Implementation Details and Performance 91
6.1 Remote Attestation Framework . . . 91
6.1.1 OpenAttestation SDK . . . 92
6.1.2 Enhancements to OpenAttestation . . . 93
6.1.3 Performance Evaluation . . . 100
6.2 Trusted Channel . . . 106
6.2.1 strongSwan . . . 106
6.2.2 Extension of strongSwan . . . 107
6.2.3 Extension of verifier . . . 108
6.2.4 Performance Evaluation . . . 109
6.3 Trusted Networks . . . 111
6.3.1 Docker . . . 112
6.3.2 Enabling Remote Attestation in Docker containers . . . 113
6.3.3 Performance Evaluation . . . 115

7 Conclusion 124

Acronyms 126

Bibliography 128

List of Figures

1.1 Abstract description of our solution. 3
1.2 A TTP verifier attests remote servers for users. 4

2.1 TPM 1.2 Component Architecture. 11
2.2 An example of trusted boot. 13
2.3 Example of TPM managed key hierarchy. 15
2.4 Configuration-based attestation. 17
2.5 Type I and type II hypervisors. 20

3.1 General architecture of model-based behavioural attestation. 26
3.2 General architecture of binary-based attestation. 28
3.3 IMA measure extend operations. 28
3.4 Overall remote attestation framework architecture. 34
3.5 Registration phase for PrivacyCA and the first attesting platform. . . 37
3.6 Remote attestation process. 39
3.7 Diagram of remote attestation process. 39
3.8 Example of IMA measures in ASCII format log file. 41
3.9 QuoteData example. 42
3.10 Example of IMA measurements in an integrity report. 43
3.11 Partial integrity report verification. 44
3.12 Execution policy of IMA. 47
3.13 Example of L3 trust level with downgraded *NetworkManager*. 49
3.14 FilesToPackages column family. 50
3.15 PackageHistory column family. 50
3.16 Policy validation workflow and the involved components. 52
3.17 IMA measures of *iptables* and *sshd* and their initial configuration files. 54

4.1 General architecture of trusted channel. 64

4.2 Trusted channel server attestation steps with and without a trusted third party. 66

4.3 Extension to IKEv2 protocol with RSA-sig authentication. 67

4.4 Shadow server attack without binding server certificate to its hardware root of trust. 69

5.1 SDN architecture. 74

5.2 vTPM implementation architecture in the Xen hypervisor. 78

5.3 Overall architecture of Docker attestation system. 85

5.4 Example of the extended measures (from the IMA ASCII log file). . . 87

5.5 The extended part of an integrity report. 88

6.1 Example of an IMA measure in an OAT integrity report. 94

6.2 A remote attestation request calling two analysis types. 94

6.3 Registering load-time as a new analysis type. 95

6.4 Extend operation with partial integrity report. 95

6.5 Fedora update system. 99

6.6 Example mail from CentOS-announce mailing list. 100

6.7 Actual remote attestation process in the developed framework. 103

6.8 The number of the IMA measures in a typical web server. 105

6.9 Number of IMA measures in a day. 106

6.10 The configuration file of oat_attest plugin in strongSwan. 107

6.11 An remote attestation request example to attest IPsec server. 107

6.12 Remote attestation result from strongSwan log. 108

6.13 The SECURED application with remote attestation enabled. 109

6.14 The image hierarchy in Docker. 112

6.15 A remote attestation request example to attest Docker container host. 115

6.16 Starting a Docker container with simple task. 116

6.17 The error given when the maximum number of active Docker containers is reached. 116

6.18 Time to start a container with and without IMA and RA. 119

6.19 Time to stop a container with and without IMA and RA. 120

6.20 Time to remove a container with and without IMA and RA. 121

List of Tables

6.1 Number of operations and performance index in three configurations. 102

6.2 Average elapsed and processing time for each analysis type. 102

6.3 Performance improvements with partial integrity reports. 104

6.4 Performance difference between attestation plugin disabled and enabled. 111

6.5 Time (in seconds) to start a container with and without IMA and RA, and the difference computed between the average values. 118

6.6 Time (in seconds) to stop a container with and without IMA and RA, and the difference computed between the average values. 118

6.7 Time (in seconds) to remove a container with and without IMA and RA, and the difference computed between the average values. 119

6.8 Number of operations and performance index under three settings in Docker environment. 121

6.9 Average time (in seconds) required to complete a remote attestation request with different number of active containers. 122

Chapter 1

Introduction

Distributed systems, that require multiple entities to co-operate on the same task in order to generate results, are very popular on the Internet. With the help of virtualisation technology (especially in cloud computing environment), distributed systems have changed the way how IT services are designed and deployed, with greatly improved flexibility, increased availability and reduced costs. More recently, network infrastructures are quickly evolving from a hardware-based switch-only layer to a full-fledged computational system able to perform several general tasks, thanks to the advent of two new architectures, namely *Software Defined Networking* (SDN) [1, 2] and *Network Functions Virtualisation* (NFV) [3, 4].

Unfortunately, these innovations come with a penalty since security is negatively affected by these new scenarios. For example, in cloud computing environment, as data and services are no more executed on platforms owned and managed by final users, several threats materialise: from direct access of the data to altered operating system or services, from direct attacks to the hypervisor to cross attacks between services of different tenants executed on the same node [5, 6, 7]. As a matter of fact, studies show that the complexity and the number of attacks on the Internet have enhanced significantly in the last several years, and the costs of defending against attacks are going up, while the costs of conducting attacks are going down [8, 9].

Solutions to these threats have been proposed and discussed by several bodies, such the *Cloud Security Alliance* (CSA)[1], *National Institute of Standards and Technology* (NIST)[2] and *European Telecommunications Standards Institute* (ETSI)[3]. However, most solutions heavily rely on software controls and hence are reliable only if the correct execution of this software is guaranteed, i.e. the software is not manipulated or changed by attackers. In other words, we need to *trust* the control software for its correct behaviour. This means that the most basic problem for security in a

[1] https://cloudsecurityalliance.org/

[2] https://www.nist.gov/

[3] http://www.etsi.org/

distributed system is the ability to trust its execution environment, therefore it is important to guarantee service integrity in softwarised environments.

The definition of trust is much wider than integrity. Like in human society relation, a user may trust some companies because of their good reputation. For instance, services of Google are used by billions of people around the world everyday, even though there have been several successful hacks [10]. On the other hand, integrity has a narrower definition. A platform with integrity does not necessarily mean that a user can safely trust the platform, but that its behaviour may be predictable given some assumptions, e.g., a platform only runs authentic software may comply to its expected behaviour if the software implementations have no vulnerabilities.

Unfortunately, because of the complexity of distributed systems and the hostile environment of the Internet, traditional security that relies solely on software countermeasures becomes insufficient, as it may be easily circumvented by exploiting vulnerabilities or less strict system configurations. Thus it must be coupled with stronger means, such as hardware-assisted protection.

Under this concern, the *Trusted Computing Group* (TCG) [11], a non-profit organisation composed of giant players in each ICT field, released a set of specifications for a new technology to deal with the trust problem. This new technology covers a broad set of scenarios and devices, which is commonly referred to as *Trusted Computing* (TC). Its aim is to provide hardware-assisted protection for sensitive credentials and mechanisms to understand and check the behaviour of remote platforms with a specially designed hardware chip called *Trusted Platform Module* (TPM). Among all the possibilities currently available, TC is a convincing and practical choice for three reasons: from the financial point of view, its standard specifications are available without fees or charges; on the other hand, the essential building block, i.e. TPM, is cheap and available in millions of business class devices including servers, desktops and laptops [12], third, the implementation quality of the chip is good. In the threat model proposed by TCG, physical attacks are not meant to be prevented. However, even if physical access is allowed, until now there is only one successful attack [13] to the TPM device (i.e. TPM 1.2) which is most widely deployed. And this reverse engineering attack requires physical access to the chip itself as well as a high-end electron microscope that can manipulate tiny needles less than a micron across, injecting conductors and insulators to rearrange the chip's circuits. Even with this powerful equipment, there is no guarantee that the attack can succeed, breaking the electrical circuits in the TPM chip may disable it, making it useless.

Taking these advantages of the TPM, this thesis exploits and improves the remote attestation technique of TC technology in distributed systems, which allows a computing platform to provide hardware-based authentic evidence about its integrity state to other entities over the network (Figure 1.1). The contribution of this thesis is threefold. First, we propose a remote attestation framework which is capable of attesting not only the integrity state of physical platforms but also the load time integrity of software services. As an example, we adopted this framework into

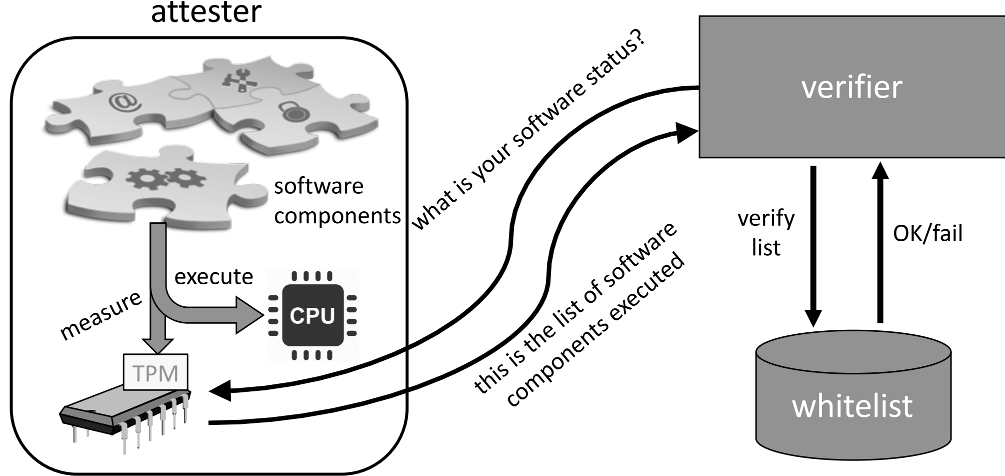

Figure 1.1. Abstract description of our solution.

a formal and trusted network security policy validator. Second, we show our proposal to combine secure channels with remote attestation in order to create trusted channels, where the secure channel endpoint identity is bound to a specific hardware node and its integrity state. Third, we offer our idea of trusted softwarised networks, in which softwarised network functions running inside virtual containers are attested as if they are running in physical platforms, implying the correct behaviour of the whole softwarised network.

After detailing our proposals of adopting remote attestation in distributed systems to prove integrity among different entities, we show our implementations and evaluate their performance, demonstrating the feasibility and the capability of our proposals in real-world applications.

The rest of this Chapter gives an overview of this thesis. Such overview should be interpreted as a high level description of the problems addressed and the solutions proposed. In order to keep the discussion short and simple, many details are intentionally omitted, and only the necessary information is provided.

Remote Attestation Framework

Distributed systems are evolving as the usage of the Internet grows. Their task is becoming much more complex and more computing entities are involved, making their capability to store and process data elastic and approximately limitless. However, security of distributed systems is weakened in this new paradigm, since end users do not have direct control of the device hosting the services. They have to blindly trust remote party to deploy the legitimate services with correct configurations as agreed. Moreover, the hostile nature of the Internet makes distributed systems to be put at stake. They are continuously under remote attacks from the Internet, and even though a great amount of security tools are available and deployed, system

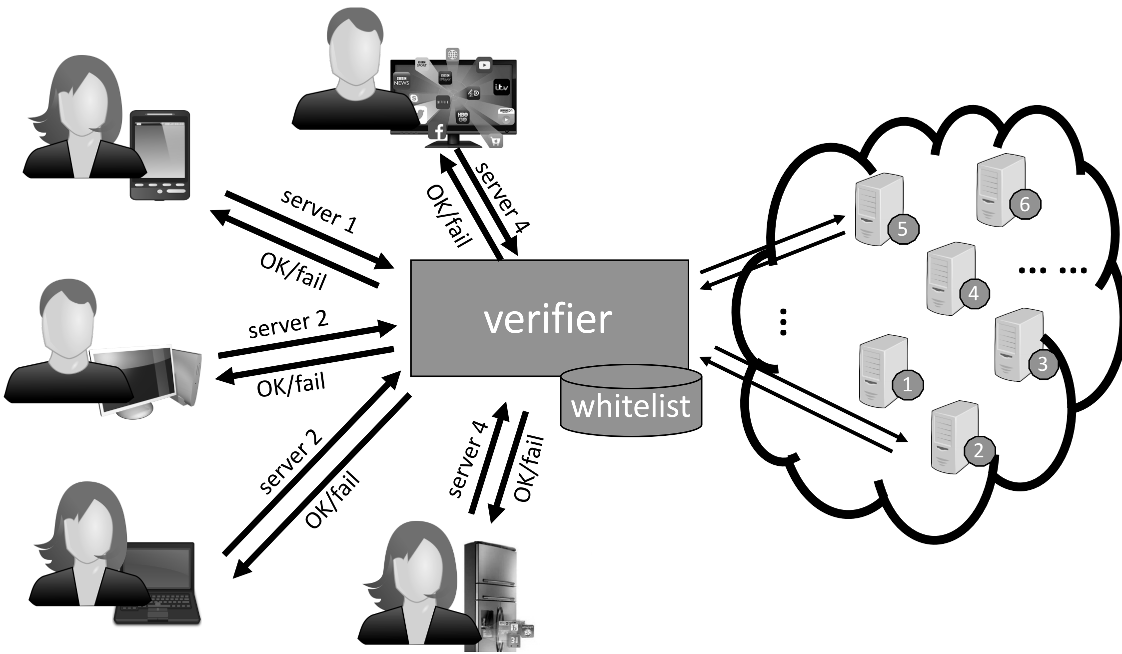

Figure 1.2. A TTP verifier attests remote servers for users.

breaching events are happening frequently [10, 14]. From the system administrator point of view, the number of physical platforms to be run in distributed systems is huge, e.g., hundreds or thousands physical platforms in a cloud scenario. Managing and monitoring such large amount of machines is a daunting work, not to mention the software services running in them. Most of the times, the system administrator is not able to notice that the system is compromised due to attacks or wrong configurations until a massive destruction has already occurred.

In order to tackle these problems, a remote attestation framework is crucial to monitor the integrity of the distributed systems, including not only the integrity of the platforms where the services are deployed but also the services themselves that directly interact with end users. End users may have two approaches to attest the integrity of remote servers. Either they can evaluate the integrity evidence of the servers directly in their terminals or they can offload the attestation work to a *Trusted Third Party* (TTP). However, evaluating the integrity evidence needs a great amount of resources and knowledge, especially if the software services are under concern, i.e. the whitelist is very large and difficult to be created and managed. Even if the whitelist is available, the introduced remote attestation workload may significantly degrade the users experience in their terminals. And a user terminal may not have enough resources to evaluate the integrity evidence by itself, considering the variety of user terminals, e.g., smartphones and *Internet of Things* (IoT) devices. On the other hand, servers may not be willing to share their integrity state to a random entity, which may expose privacy information (e.g., what software services are running and which versions). For such reasons, in this work, we propose an extensible remote attestation framework with a TTP as the verifier to check the

integrity state of servers in distributed systems (Figure 1.2).

In our framework, the integrity of servers is composed of two parts, the boot phase of a physical platform and the load time of all software services running on it. In other words, in the first step, the server must be booted into a trusted state, i.e. all components loaded during the boot phase are known and they are loaded in a specific order. Afterwards, when services are loaded, their associated configurations and executables loaded in the server kernel must be uniquely identified and known by the verifier. In the end, the verifier continuously attests the servers to monitor their integrity state, e.g., whether an unwanted script has been loaded or a service configuration has been changed.

Trusted Policy Verification Framework: in order to show the applicability and the necessity of our remote attestation framework in distributed systems, we present a real use case. We propose a trusted policy verification framework, which validates network policy enforcement, by checking the network status and possible causes in case of misconfiguration and remote attacks with the help of the remote attestation framework.

Network security is a crucial aspect for administrators due to increasing network size and number of functions and controls (e.g., firewall, parental controls). Errors in configuring security controls may result in serious security breaches and vulnerabilities (e.g., blocking legitimate traffic or allowing unwanted traffic) that must be detected and addressed. Moreover, to detect and avoid the situation that the service is correctly deployed but it is cracked by remote attacks, especially for complex distributed scenarios, remote attestation of service integrity is mandatory.

The remote attestation framework in this scenario continuously monitors the integrity state of the network services and their hosts deployed in the policy validation framework. If the remote attestation fails, the verifier immediately alerts the system administrator, specifying the causes. Otherwise if the attestation result is positive but the policy validation result is negative, an analyser is triggered to perform policy analysis.

Trusted Channel

In distributed systems, data transmitted can be well protected with *secure channels*, e.g., TLS [15] or IPsec [16], for data integrity, authenticity and confidentiality. These protocols guarantee the identification of the data source and ensure that no interference happens in the communication. However, they do not give any information about the legitimacy and the integrity of the platforms which the data are originated from (possible data fraudulence) and sent to (possible data leakage). To overcome this issue, secure channels should be combined with remote attestation, which is called *Trusted Channel*.

This combination serves two purposes. The first one is to indicate whether the platform and the service generating the data are compromised and how the data is

handled. However, the usefulness of this point alone is debatable because an attacker may easily act as *Man In The Middle* (MITM) that forwards the remote attestation request to a benign platform and uses the integrity evidence of the legitimate platform as its own, so it is able to pretend to be innocent while it is actually acting maliciously, this is known as *relay attack*. Thus, we mention the second purpose of integrating secure channel with remote attestation, it is to bind the identity of secure channel endpoint to the identity of the hardware chip, i.e. the TPM, which is unique for each physical node. In this way, an end user is able to know that they are interacting with a benign platform based on its hardware identity.

In this work we propose a solution for establishing trusted channels in a client-server model with the help of the aforementioned remote attestation framework. For the sake of user privacy, in this solution only the server is attested in a way that a user may have knowledge about its integrity state before establishing the secure channel.

A live demo of the prototype of proposed trusted channel and remote attestation framework was shown in the ETSI NFV-SEC meeting in Dublin, February 16-19, 2016, and was well accepted and appreciated.

Trusted Network Infrastructure

Network infrastructure is quickly evolving from a hardware-based switch-only layer to softwarised environment in which switching packets being just one feature. This evolution is permitted by the advent of two new architectures, namely *Software Defined Networking* (SDN) [2] and *Network Functions Virtualisation* (NFV) [3].

SDN is a particular approach to provide virtualised traffic routing and unified network flow management across hardware and software-based networking components. The principal design of SDN is to virtualise the existing control and data planes by moving the control part away from all network elements to a centralised node in the network, known as the *SDN controller*.

NFV proposes to virtualise several classes of network node functions into generic building blocks (i.e. virtual instances running on commodity hardware) to be connected for creating various network services. NFV typically exploits SDN to create custom overlay networks connecting the various network functions and in turn SDN can use NFV to host its controller and elements.

The usage of SDN and NFV networks introduces new network abstractions and high-level primitives, but it creates a trust gap for administrators as they cannot easily assess the correctness of enforced device configurations, especially SDN and NFV networks heavily rely on software modules running in distributed nodes. Due to errors or attacks, the software modules may act differently from their expected behaviour. Therefore we need a technique to know and understand if only the expected software modules are loaded and if they are configured correctly.

Network infrastructure has two properties which we must take into concern in our proposal. First, network infrastructure has no privacy issue, because it should

be owned and used by a single entity, such as an *Internet Service Provider* (ISP). Second, network elements (i.e. computing nodes that run network functions) in general are less powerful than commodity servers. For this reason, the used virtualisation technology should be as lightweight as possible and the pressure of remote attestation should be alleviated by offloading the computational complexity to the third party verifier.

Since a software-only solution would be prone to a wide range of remote attacks, our design uses a hardware-based trusted device inside the network element. This trusted device should be generally immutable and used as a basis for trust, which is leveraged by the verifier to attest the network element and its behaviour. Same as previous solutions, in this one we continue our choice of the TPM, and use it as the root of trust.

With the help of the aforementioned root of trust, our goal is to check the integrity state of the boot phase of the network elements and load time of the softwarised network function modules and their configurations. Since most of these software modules are running inside virtual instances, we propose a new method to attest software integrity running in virtual container with hardware-based evidence as if they are running in physical platforms.

A live demo of the prototype of this proposed solution was shown in the ETSI NFV-SEC meeting in Bilbao, February 21-24, 2017, and was well accepted and appreciated.

Foundation

The following publications and pre-prints form the foundation of this thesis:

T.Su, A.Lioy, N.Barresi, "**Trusted Computing Technology and Proposals for Resolving Cloud Computing Security Problems**", in the book "Cloud Computing Security: Foundations and Challenges" edited by J.R.Vacca, CRC Press, pp. 345-358. [17]

T.Su, A.Lioy, N.Barresi, "A practical approach of building trusted computing compliant infrastructures", in preparation.

A.Filograna, P.Smiraglia, C.Gilsanz, S.Krco, A.Medela, T.Su, "**Cloudification of Public Services in Smart Cities the CLIPS project**", ISCC-2016: IEEE Symposium on Computers and Communication, Messina (Italy), 27-30 June, 2016, pp. 153-158. [18]

R.Bonafiglia, F.Ciaccia, A.Lioy, M.Nemirovsky, F.Risso, T.Su, "**Offloading personal security applications to a secure and trusted network node**", Netsoft-2015: 1st IEEE Conf. on Network Softwarization, London (UK), April 13-17, 2015, pp. 1-2. [19]

L.Jacquin, A.Lioy, D.R.Lopez, A.L.Shaw, T.Su, "**The trust problem in modern network infrastructures**", in the book "Cyber Security and Privacy" edited by F.Cleary, M.Felici, Springer, pp. 116-127. [20]

A.Lioy, T.Su, D.R.Lopez, A.Pastor, A.L.Shaw, H.Attak, "Trust in SDN/NFV environments", in the book "Guide to Security in SDN and NFV - Challenges, Opportunities, and Applications" edited by S.Y.Zhu, S.Scott-Hayward, R.Hill, L.Jacquin, Springer, in press.

T.Su, A.Lioy, A.Atzeni, M.Mezzalama, "Practical integrity verification for the Docker lightweight virtualization environment", submitted to Computers & Security, 2017.

F.Valenza, T.Su, S.Spinoso, A.Lioy, R.Sisto, M.Vallini, "**A formal approach for network security policy validation**", in Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications, March 2017, pp. 79–100. [21]

Organisation

The structure is organised as follows:

Chapter 2 provides an overview of TC, by introducing this technology and its main building blocks, especially the details of remote attestation technique. It shows how remote attestation is used to convey authentic integrity evidence of physical platforms with the ability to resist remote attacks.

Chapter 3 describes our proposal of a remote attestation framework which is able to check the integrity state of both the boot phase of physical platforms and the load time of software services running in them. Then it talks about a specific use case, showing that the remote attestation is mandatory in network security policy validation scenario in order to generate validation results with strong integrity guarantee.

Chapter 4 describes our proposal of *trusted channels*, where secure channel endpoint is bound to its integrity state and the physical platform which generates its integrity evidence, in order to avoid relay attacks.

Chapter 5 presents our proposal of trusted softwarised networks, where the integrity state of software network functions running in virtual instances and their host are attested, thus implying the expected behaviour of the softwarised network.

Chapter 6 presents our implementation details, including the remote attestation framework, trusted channels and virtual container attestation system. Besides implementation details, this Chapter also gives experiment results. Such results prove the proposed solutions are feasible in real-world use cases not only from theoretical but also practical points of view.

Chapter 7 concludes this work, which also discusses current limitations of this work and points out future research directions.

Chapter 2

Background of Trusted Computing

Software-based security solutions are becoming insufficient in the current hostile environment of the Internet, as software means alone can be circumvented by exploiting vulnerabilities or less strict configurations, thus they cannot provide enough security guarantees against various attacks in distributed systems. More importantly, they lack strong integrity guarantee to provide authentic evidence about the behaviour of the remote parties involved in computation. In other words, in distributed systems, computational entities need to trust each other ambiguously and blindly with only software-based solutions.

Trusted Computing was introduced in the 1990s to deal with the issue of platform trustworthiness with the rapid increasing of the usage of the Internet. Indeed, it provides hardware protection for sensitive credentials and mechanisms for presenting the behaviour of remote platforms [22].

Apart from TC, *virtualisation* technology is another essential building block of this work. It is commonly adopted in distributed systems to maximise the utility of the underlying physical resources. Thus, in order to cover the integrity state of all software services running in distributed systems, it is important to guarantee the integrity state of not only the host where the virtual instances are running, but also the services running inside these virtual instances.

This Chapter provides an extensive overview of TC and virtualisation main functionalities, in order to help readers understanding its current limitations and our motivation. More importantly, it shows why remote attestation is incompetent to be directly applied in distributed systems and the needs of improvement.

2.1 Trusted Platform

> *Trust* is the expectation that a device will behave in a particular manner for a specific purpose [23].

According to the TCG definition, in order to be a *Trusted Platform* (TP), the

platform must provide three basic features: *protected capabilities*, *integrity measurement* and *integrity reporting* [23]. With these features, a platform has enough basic functionalities to convince a remote entity that it is trustworthy.

2.2 Protected Capabilities

Protected capabilities work in collaboration with *shielded locations*, which are special regions of the platform where it is safe to store and operate on sensitive data. For security reasons, shielded locations must be isolated from the rest of the system in order to reduce the attack surface. On the other hand, protected capabilities are the commands that have exclusive permissions to operate on the shielded locations. These two functions are essential to other features, e.g., integrity measurement and integrity reporting.

2.2.1 Roots of Trust

In order to trust the operations of protected capabilities, the TCG defines three *Root of Trust* (RoT), i.e. components meant to be trusted because their misbehaviour may not be detected [24]:

- *Root of Trust for Measurements* (RTM) implements an engine capable of making inherently reliable integrity measurements and it is also the root of the chain of *transitive trust* (one component gives trust guarantee to another);
- *Root of Trust for Storage* (RTS) securely holds the integrity measurements (or a summary and sequence of those values) and protects data and cryptographic keys used by the TP that are held in external storage (non-shielded storage);
- *Root of Trust for Reporting* (RTR) is capable of reliably reporting to external entities the measurements held by the RTS.

A complete set of RoT encompasses the minimal combination of hardware and software elements that a remote entity needs to trust in order to validate the entire platform. It is recommended by the TCG specifications to use hardware device in combination with software components to create a strong unforgeable identity and provide safer storage of evidence. Therefore the main components of the RoTs are: (*i*) a specialised hardware component to store the identities and measurements away from the software access, (*ii*) an initial isolated component that is able to measure the first non-trusted software, which will be then trusted to measure the next stage software.

In the solution proposed by TCG, *Trusted Platform Module* (TPM) is the core building block to implement the aforementioned functions. To be more specific, it provides shielded locations to store the platform integrity measurements, simple cryptographic functions to create unique keys and to sign data stored inside itself.

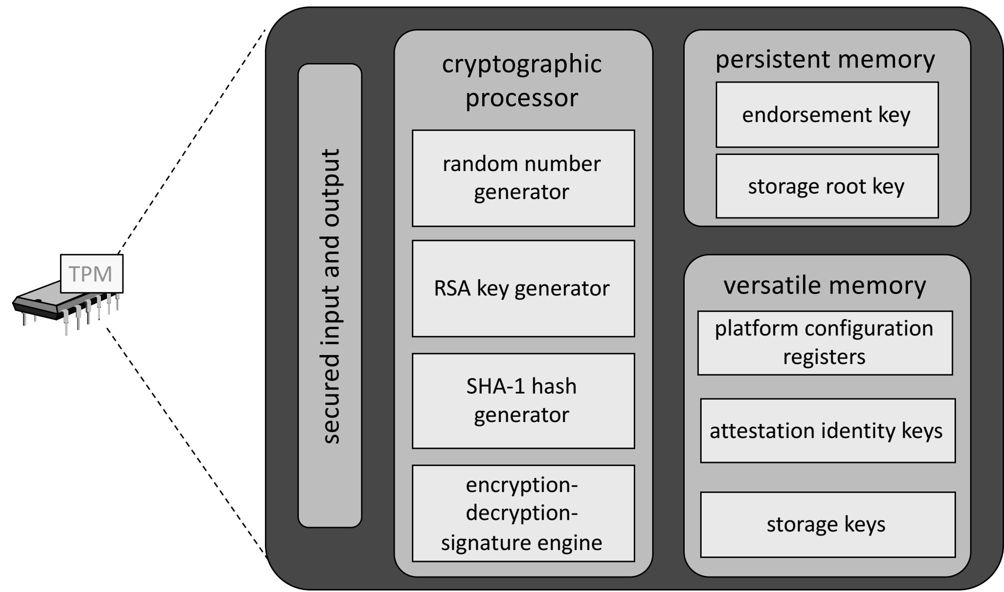

Figure 2.1. TPM 1.2 Component Architecture.

By design, the shielded locations to securely store integrity measurements are seated in the internal memory of a TPM, which are called *Platform Configuration Registers* (PCRs). Each TPM should have a minimum number of 24 PCRs (numbered from 0 to 23) with each of them has 20 B storage (Figure 2.1).

The protected capabilities allow free read access to the PCRs, but direct writing is prevented. These PCRs act as accumulators: when the value of a register is updated, the new value depends on both the new data and its old value, to guarantee that it is impossible to forge the value of a PCR once initialised. Moreover, most PCRs (PCR0 through PCR15) have persistent values until the whole platform is reset (e.g., rebooted).

The PCR values are updated by a operation called `extend` and it works as follows:

$$\text{PCR}_{\text{new}} = \text{SHA-1}(\text{PCR}_{\text{old}} \mid\mid \text{data}) \tag{2.1}$$

where PCR_{old} is the value present in the register before the extend operation, $||$ is the concatenation operator, and data is the new data to be inserted. This approach brings two benefits. First, it allows for an unlimited number of data to be captured in a single PCR, since the size of the values is always the same and it retains a verifiable ordered chain of all the previous data to be inserted. Second, it is computationally infeasible for an attacker to calculate two different hashes that will match the same resulting value of a PCR extend operation[1]. Thus, even if a system is compromised,

[1]Currently, the weakness of SHA-1 is showing as there is known SHA-1 collision found [25]. With the updated TPM version (i.e. TPM 2.0), the hash algorithm is upgraded to SHA-256 and MD5. However in this thesis, we are still referring to the old TPM version which only supports SHA-1.

an attacker cannot forge PCR values in his favour.

Since there are correlations between old and new PCR values, a PCR somehow contains the history of all extended data. However, the nature of the extend operation makes computationally impossible to recover the list of stored values backwards from the current content of a PCR. That is the reason why logging each integrity measure is strongly recommended, even if not compulsory, in order to precisely identify the compromised component and the time of its compromission.

2.3 Integrity Measurement

The integrity of a platform is defined as a set of metrics that identifies the loaded software components, such as *Basic Input/Output System* (BIOS), operating system kernel and their configurations. For each component, a fingerprint acts as a unique identifier and as proof of no modification when it is loaded; in other words, components are "`measured`" by computing the digests of their content. Subsequently, the measurement process uses protected capabilities to "cumulatively" store these platform component measures into the PCRs.

At each instant, the TP has a *trust boundary*, which is the set of its trusted components. Such trust boundary can be extended if a trusted component gives a trustworthy description of (i.e. it measures) another component and extend the measure to a designated PCR before loading it. The result is that the trust boundary is extended from the first to the second entity. This process can be iterated: the second entity can give a trustworthy description of a third one, and so on. In practice, the platform creates a chain of trust where each component is measured by its previous one. This iterative process is called *transitive trust* and it is used to provide a trustworthy description of the whole platform.

Integrity measurement is the operation used to create the transitive trust of the platform: each component in the system startup process measures the to-be-loaded component before transferring the control of the platform and these measures are stored into the PCRs. This forms the basis for other functionalities such as *remote attestation* (Section 2.4).

The RTM should be the first component activated when the system is booted, it is required to be able to measure the subsequent components and extend their measures into the designated PCRs in the platform's TPM. A passive chip as the TPM cannot provide the RTM by itself, since it cannot actively measure the BIOS or other components to create the chain of transitive trust. Typically, in a commodity platform, RTM can be implemented either by the first software module of a computer system executed when the latter is switched on (i.e. a small portion of the BIOS) or directly on-chip by processors of new generations (e.g., Intel processors equipped with *Trusted eXecution Technology* (TXT) [26]). The set of operations and instructions performed in this phase are called *Core Root of Trust for Measurement* (CRTM). On the contrary, the RTS and the RTR can be implemented directly using

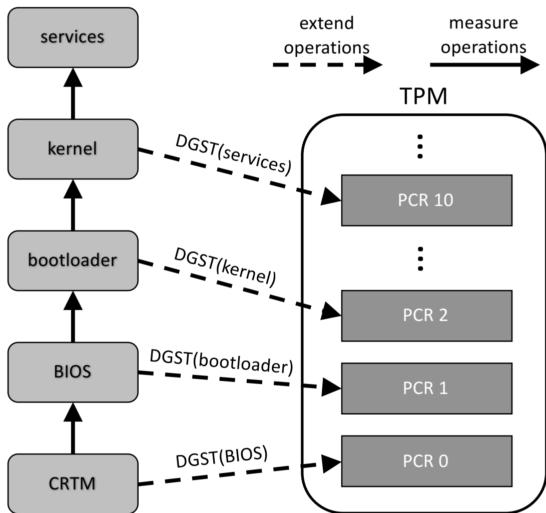

Figure 2.2. An example of trusted boot.

the TPM, as its internal functions supports cryptographic operations, to securely maintain the integrity measures and to report them.

2.3.1 Trusted Computing Base

When a computing platform is powered on, the first component loaded is BIOS. After the BIOS is measured by the RTM and its measure stored into a designated PCR, it takes full control of the platform and starts to measure the bootloader and puts its digest into another PCR. Afterwards, when this operation finishes, the control is passed from the BIOS to the bootloader. Then the bootloader does the same thing to the operating system kernel, it computes the digest of the latter and any parameters or additional code before loading it. Specifically in Unix systems, it is required to take concern of options passed to the system kernel and the initial RAM disks, which may change the behaviour of the kernel. These steps create the transitive trust chain of the platform as described in Section 2.3.

Since each component in the boot process is measured and their digests are extended into the corresponding PCRs, the PCR values are capable of reflecting the content of loaded components. This procedure is called *trusted boot* or *measured boot* (Figure 2.2), which is able to prove the components loaded in the boot phase of a platform are known and they are loaded in a specific order. Unlike *Secure Boot* [27], which checks the content of loaded components against the signatures signed by their manufactures, in trusted boot, there is no limitation of the component that can be loaded in the platform. The only constraint is that the components must be measured and the measures must be extended into the corresponding PCR before they are loaded. The verification task of the loaded components is irrelevant to the platform, but delegated to another entity.

At this point, the operating system kernel can start measuring other loaded services. It may compute the digests of the service code, the configuration files,

the command line options and any data that may influence the service's behaviour. In this way, a *Trusted Computing Base* (TCB) is created from the RoTs to the operating system application layer.

2.4 Integrity Reporting

Integrity reporting (often called remote attestation instead) is the process that a computing platform reports its own integrity state to an external party. The basic idea behind integrity measurement and integrity reporting is that a platform is allowed to enter any state, even untrusted ones, but it cannot lie about its state.

In the previous part of this Chapter, we have shown how integrity is measured in a trusted platform, however, the integrity measurement is not able to be used if it is not authenticated. Therefore TPM provides hardware means to protect and manage its keys.

2.4.1 Key Hierarchy

Protecting and managing of keys using hardware means is mandatory since these keys are used for other security critical operations, such as authentication in remote attestation and encryption/decryption in content sealing.

The keys used by the TPM can be roughly grouped into two categories: authentication keys (also called signing keys) and storage keys (also called encryption keys). A signing key is a general purpose key used for authentication, while a storage key is used to encrypt data or other keys (e.g., wrapping keys). For this reason, each TPM provides two non-migrateable asymmetric keys, *Endorsement Key* (EK) and *Storage Root Key* (SRK), that their private parts never leave the TPM.

Authentication Keys

EK is a special authentication key, which is part of the RTR. This key is generated and injected into the TPM when the chip is manufactured, and it is unique for each TPM chip. Moreover, manufacturers of the TPM should provide each EK a certificate (EK certificate) signed by their keys in order to state that the certified EK belonging to a genuine TPM which follows the specifications defined by TCG. The reason to have a unique and unchangeable EK for each TPM is twofold, (*i*) a TPM cannot forge its or other TPM's EK, (*ii*) a rogue TPM can be detected.

However, this feature may cause a privacy issue if the EK is directly used for authentication, that all transactions can be linked to a specific TPM (i.e. traceability). In order to eliminate the traceability issue, the EK is never used directly for signing nor encrypting. Its purpose is purely to decrypt owner authorisation data when the TPM's ownership is taken and associate the TPM to another type of signing key called *Attestation Identity Keys* (AIKs). In simple words, AIK is a

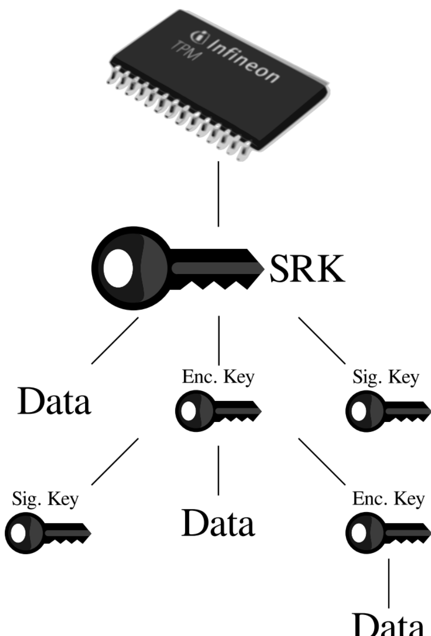

Figure 2.3. Example of TPM managed key hierarchy.

variant of EK, whose private part is also never exposed outside the TPM but can be generated by the TPM itself. In the case that the private part of an AIK needs to be stored outside of the TPM, e.g., there is not enough storage in the TPM persistent memory, the private part of the AIK needs to be encrypted using the SRK. A TPM can only have one EK but it can have multiple AIKs. It is exclusively used to sign data originated by the TPM (e.g., PCR values) during remote attestation instead of EK. Actually, the TCG specification suggests to have a different AIK for each remote attestation session to avoid the traceability.

In order to prove the AIK comes from a genuine TPM, it needs a certificate issued by a *Certificate Authority* (CA). Its role is to assess the trustworthiness of the TPM through its EK. Actually, the CA should issue a certificate for a AIK only if the request comes from a genuine TPM. It should checks the EK certificate was issued by a legitimate TPM manufacture and states that this EK belongs to a genuine TPM which will behave correctly.

The steps need to be done are follows:

1. the CA checks the EK certificate issued by a legitimate manufacture;
2. the CA checks the certificate request comes from this very EK;
3. the CA issues the AIK certificate encrypted with the EK's public key (which is available in the EK certificate);

The last step guarantees that only the TPM made the AIK certificate request can decrypt this blob and retrieve the AIK certificate correctly. If a rogue TPM

sends a certificate request for its AIK, the CA will detect this because the public part of its EK can be identified as rogue. Or if a malicious platform resends the AIK certificate request generated by another platform, it cannot decrypt the blob because the EK of the other platform is known.

In the previous scheme, the CA can still trace a specific TPM, since it has knowledge of all AIK certificates bound to the EK of this TPM. Thus the CA has special requirement regarding to privacy. For this reason, it is often called *Privacy Certificate Authority* (PrivacyCA)[2].

If full anonymity is required, TPM 1.2 supports *Direct Anonymous Attestation* (DAA) protocol [28], which allows a TPM authenticates its integrity evidence without being linked to other remote attestation transactions with the help of proof of knowledge. Generally speaking, DAA can be regarded as a group signature without revocable anonymity but with a mechanism to detect rogue members (i.e. rogue TPMs). More precisely, Camenisch-Lysyanskay (CL) signature scheme [29, 30] is used to issue certificates on a membership public key. Then, to authenticate as a valid group member (i.e. a valid TPM), a TPM proves that it possesses a certificate of the public key for which it knows a secret key. This scheme is provably secure in the random oracle model where the unforgeability of certificates holds under the strong RSA assumption and privacy and anonymity is guaranteed under the decisional Diffie-Hellman assumption in a finite field.

Storage Keys

On the other hand, each TPM provides a *Storage Root Key* (SRK) which is part of the RTS and it is a self generated key by the TPM, i.e. the TPM ensures that its generation and cryptographic operations are executed internally of the chip in a secure manner and its private part never leaves the chip. Immediately after the TPM ownership is taken, that a password is set to ensure only the user (usually the system administrator) knowing the password can operate on this TPM (e.g., enable, disable or clear the TPM through command line remotely), the SRK is generated and available.

At this point, the TPM can securely store other keys by encrypting them with the SRK or their parent keys. This operation is called *wrapping key*. To be more specific, if a key (regardless of its purpose) is generated and inserted into the key hierarchy, it is wrapped with the SRK. Later when another key is created, it may chose to be wrapped with the SRK or an existing storage key (Figure 2.3). Each time a key needs to be used, it must be operated inside the TPM. In order to correctly load a key into the chip, its parent (wrapping key) must be loaded. Since the SRK is the root of the wrapping keys, it must be loaded first, then the other wrapping keys.

[2] More details in Section 3.4, where the PrivacyCA approach is followed in our remote attestation framework.

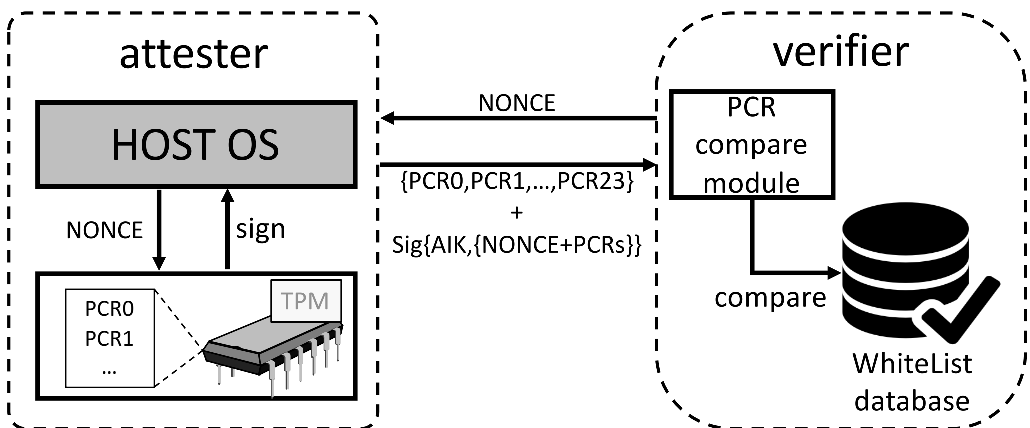

Figure 2.4. Configuration-based attestation.

These keys can be both migrateable or non-migrateable. Non-migrateable keys can only be used by the TPM which generates them, and cannot be known or used by other entities. While migrateable keys can be exchanged between TPM chips in a secure manner. In practice, in a TPM, AIK, EK and SRK are always non-migrateable keys while other keys may be non-migrateable as well as migrateable depending on their usage.

Exploiting the TPM's key management capability, the TPM provides a protected storage for sensitive data with hardware means. This feature is mainly offered by two functionalities, *binding* and *sealing*. Binding is to bind some data to a specific TPM. This happens when non-migrateable keys are used to encrypt data. This guarantees the data can be only decrypted by a particular TPM. Sealing enhances the function of binding, such that the data encrypted are not only bound to a specific platform with a particular TPM, but they can be decrypted only when the platform is in a particular integrity state. Indeed, sealing associates the encrypted data with a specific set of PCR values and a symmetric key. After the data is sealed, it is only possible to be decrypted when the PCR values are the same as those specified during the sealing. Because of this feature, sealing is some times referred as local attestation.

2.4.2 Configuration-based Remote Attestation

In general, in remote attestation process, attesting platform (hereafter called attester for simplicity) needs to transmit the evidences of its integrity state to an external entity (hereafter called verifier or challenger) in an authenticated manner.

The most popular evidence is the TCG's *integrity report* (IR) [31], which comprises the values stored in the PCRs and a digital signature of them computed with an AIK. It is used in a wide range of applications, e.g., *Trusted Network Communication* architecture [32] and *Trusted Compute pools* [33]. As mentioned in Section 2.4.1, AIKs are alias of EK and only used to sign the data structures stored inside a TPM

(e.g., PCR values), and the private parts of these keys are never exposed, thus guarantees the authenticity and integrity of the integrity reports. To be more specific, the operation to get signed PCR values from a TPM is called `Quote`.

This solution is simple from both the verifier's and the attester's points of view. The verifier wishing to validate the platform configurations of the attester sends a remote attestation request specifying an AIK for generating the digital signature, the set of PCRs to quote, and a nonce to ensure freshness of the digital signature. After the TPM receives the remote attestation request, it validates the authorisation to use the AIK with the password set in TPM taking ownership phase, fills in a structure that with the set of PCRs to be quoted and generate a digital signature on the filled in structure with the specific AIK. Then the host operating system returns the digital signature to the verifier. The verifier, after receiving the digital signature, validate the integrity of the PCR values received using the public portion of the AIK. Then if the authenticated PCR values (i.e. platform configurations) are marked as trusted in the whitelist database, then the verifier assesses the attester's trustworthiness.

Since this solution relies on platform configurations (stored inside PCRs) to assess platform's trustworthiness, it is generally referred as *configuration-based attestation* (Figure 2.4).

Configuration-based attestation is simple and sufficient to guarantee that the platform has some specific components and configurations loaded in the system (e.g., in trusted boot scenario) once deployed correctly. However, it has three severe drawbacks.

First of all, because of the limited resource provided by the hardware TPM chip, this solution can only covers limited amount of components loaded in the system. It may happen that even some key components of a platform are measured and extended into the designated PCRs, the system is still compromised because a cracked active component which is checked, and the verifier is not able to detect the attack. Hence, this method needs to be extended to continuously monitor and check the integrity of the whole system at runtime.

Second, the assessment is not flexible and lacks useful information. Using digests for measuring loaded components and their configuration files implies that any change, even different configuration options, may cause a totally different measure. For this reason, integrity measurement does not directly reflect the platform's trustworthiness, since it may happen that a component is updated legitimately but its measure changes.

Third, the nature of extend operation makes the final PCR values depend not only on the components loaded in the platform, but also on the order of loading operations, i.e. `extend(A,B)` $\neq$ `extend(B,A)`. This drawback leads to configuration-based attestation extremely difficult to be managed and maintained, especially when runtime measurement is involved. For instance, on the one hand, the attester needs to be very careful and persistent about the order of the components loaded during the usage, on the other, the verifier needs to update its whitelist of trusted PCR

values every time a new component of the attester is added or an existing one is modified.

2.5 Virtualisation

Virtualisation technology also plays a central role in distributed systems. It can optimise the utilisation of hardware platforms and simplify system management, leading to improved flexibility, availability and reduced cost. Nowadays, it is very popular given their application not only to server consolidation and to cloud computing but also to network softwarisation, as in the case of SDN and NFV technologies.

In practice, the virtualisation system presents to guest operating systems an abstraction of the underlying hardware. The guests, or virtual instances, can then run in parallel sharing the physical hardware (e.g., CPU, memory and harddisk).

Remote attestation is a well-known technique to assess the integrity of a physical platform, but not so well with virtualised instances hosted on top of hypervisor, either hypervisor-based virtualisation environment (such as KVM [34] and XEN [35]) and it is simply not available for operating system level virtualisation environment (such as Docker [36]).

2.5.1 Hypervisor-based Virtualisation

Theoretically, hypervisor-based virtualisation can be broadly grouped into full virtualisation and para-virtualisation.

Full virtualisation provides to guests a complete abstraction of a physical system. Thus the guest is executed as it is running on a real physical platform. In this case, once the guest requests a privileged instruction (e.g., I/O operations) which could not be executed in virtualisation environment, the hypervisor replaces it with the corresponding emulated instruction. The main advantage of this approach is that any software can be run in guest without modification, but it imposes a performance penalty as privileged instructions need to be rewritten.

Paravirtualisation, on the other hand, requires the guest code to be modified, so that no privileged instruction is called directly, but only its virtual version. The virtual privileged instructions called by guests are executed by the hypervisor, which is also called *hypercalls*. This approach can achieve much better performance than full virtualisation, as it is close to a native system.

From the architecture point of view, there are two types of hypervisor, type I and type II (Figure 2.5). Type I hypervisors, that are also called native or bare-metal hypervisors, run directly on the host's hardware to control the hardware and to manage guest operating systems. Type II or hosted hypervisors, run on a conventional operating system just as other computer programs do, which means a guest operating system runs as a process on the host.

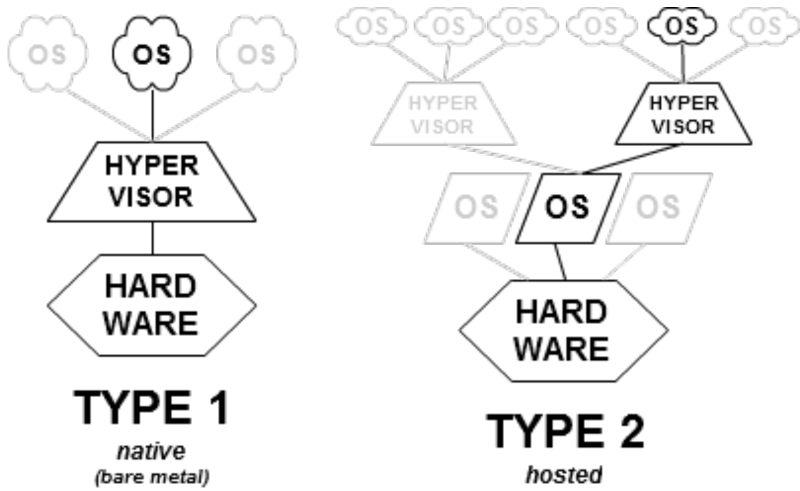

Figure 2.5. Type I and type II hypervisors.

The hypervisor's virtualisation technique defines how its guests can access the shared resources from the hypervisor, and its type defines how the hypervisor accesses the hardware resources. For instance, Xen[3] is a type I hypervisor which adopts paravirtualisation technique, i.e. the hypervisor is deployed on bare metal and the guests need to be modified in order to run on this hypervisor. On the contrary, Oracle VirtualBox[4] is a type II hypervisor which adopts full virtualisation, i.e. the hypervisor is running in the host of the physical platform and the guests running on this hypervisor does not require modification. However, in order to provide higher performance for its guests, VirtualBox also starts supporting paravirtualisation from version 5.0.

From the security point of view, virtual machines are vulnerable to attacks from their hypervisor, since the hypervisor is in charge of managing all their requests to the hardware resources. For instance, a compromised hypervisor may maliciously change the result of a request from a virtual machine, thus its behaviour, without being detected by the virtual machine. On the other hand, the hypervisor is vulnerable to a special type of attack, called *hyperjacking*, where a hacker takes malicious control over its hypervisor from a compromised virtual machine [37]. This issue is especially relevant to provide a secure execution environment for distributed systems.

As a matter of fact, since the hypervisor is loaded as a component in the platform, it can be a target of the attestation process directly. Depending on its type, it can be attested either as a component loaded at boot (type I hypervisor) or as a service running in the host system (type II hypervisor). But this solution is not effective for virtual machines, because the link amongst the services running in VMs and the hardware TPM is in general broken by the virtualisation layer.

[3] https://www.xenproject.org/

[4] https://www.virtualbox.org/

Even if the link is available, as in KVM that implements a pass-through driver to allow the internal of the VM to interact with the hardware TPM [38], the number of virtual machines running on a single platform is significantly higher than the resources provided by the TPM, which makes the chip unable to provide authentic evidence for all the virtual machines. In particular, the secure storage provided by the TPM has very limited size: although it is sufficient to store the integrity measurement of a single operating system, it cannot store the integrity measurement of tens of operating systems coming from tens of virtual machines.

The aforementioned problem is the obstacle to use remote attestation in a hypervisor-based virtualisation environment. Previous works addressed this issue either by introducing a software entity to simulate the TPM functionality [39, 40], which however cannot provide the same strong guarantees provided by an hardware trust anchor, or by modifying the hypervisor to monitor the internal behaviour of the VMs [41, 42], which brings additional performance loss and still misses the direct link to the hardware anchor (more details in Chapter 5).

So, remote attestation for hypervisor-based virtualised environments is still difficult, incomplete, and with bad performance.

2.5.2 Operating-System-Level Virtualisation

Virtual containers represent an operating system level virtualisation technique, sometimes called *paenevirtualisation* [43] or container virtualisation, where the kernel of an operating system allows for multiple isolated user-space instances.

Same as hypervisor-based virtualisation, operating-system-level virtualisation is used in virtual hosting environments, where it is useful for securely allocating finite hardware resources amongst a large number of mutually-distrusting users. It can also be used for consolidating server hardware by moving services on separate hosts into containers on one server.

Operating-system-level virtualisation usually imposes little to no overhead, because programs in virtual partitions use the normal system call interface of the kernel and do not need to be subjected to emulation or be run in an intermediate virtual machine, which removes the overhead introduced by the hypervisor. This feature makes them smaller, agiler and faster to be launched than hypervisor-based virtual machines, taking just seconds to be launched and stopped, while start/stop operations of VMs usually take tens of seconds [44].

Moreover, some container virtualisation implementations (e.g., Docker [36]) provide file-level copy-on-write (CoW) mechanisms. More commonly, a standard file system is shared between partitions, and those partitions that change the files automatically create their own copies, coordinated by a storage driver. This is easier to back up, more space-efficient and simpler to cache.

However, operating-system-level virtualisation is not as flexible as other virtualisation approaches since it cannot host a guest operating system different from the

host one, or a different guest kernel. For example, with Linux, different distributions are fine, but other operating systems such as Windows cannot be hosted.

Since operating system level virtualisation technology is popular just recently, there is little work addressing remote attestation of virtual containers. As described in Chapter 1, in this thesis, we are going to address this limitation and propose our solution to attest the internal of virtual containers based on hardware-based integrity evidence.

Chapter 3

Remote Attestation Framework

Distributed systems nowadays are widely deployed. Their components are located on computer networks, and they communicate and coordinate their actions by passing messages, in order to achieve a common goal. Since the number of involved entities and how they interact are dynamic, the structure of these systems is more flexible. The flexibility allows for much improved extensibility and availability, providing theoretically unlimited computing power and storage. However, these benefits come with a cost, since security is negatively affected by this new scenario, user data and applications are no longer processed and executed on physical platforms directly owned and managed by end users, but on machines seated remotely.

As a real life example, in March 2008, an attack was announced against Hannaford Brothers, a large American grocery store chain. Even though Hannaford's payment systems were designed not to store customer payment details and to adhere to compliance standards of the credit card companies, changes to their service software led to large disclosures. The attacker succeeded in loading unauthorised code to retain the credit card information for each transaction occurring at a store and periodically transmitting the information to a third party. As a consequence, over 4.2 million credit and debit cards were compromised and at least two thousand fraudulent transactions have been identified as results [45].

Thus, for the secure provision of digital services over the Internet, endpoint integrity is vital. To avoid such issues, evidence of distributed system endpoint integrity has to be provided in a secure and reliable manner, to enable peers to evaluate each other's "trustworthiness".

In this Chapter, we propose a remote attestation framework, which is fully compliant with TCG specifications. The details of the implementation and performance evaluation are presented in Chapter 6. This framework attests not only the integrity state of the boot phase of remote platforms, but also the integrity state of services running in them. We also propose intermediate integrity level by identifying if the software running in attester are up-to-date or have known security and functional bugs. Attestation requests are issued periodically in this framework such that compromission can be discovered in a timely fashion and the whole life cycle of remote platforms and services is monitored.

3.1 State of the Art and The Way Forward

Since the beginning of trusted computing, remote attestation has been an important and uttermost useful feature. In this section, we first list the current available remote attestation work from both theoretical and practical points of view. Then we discuss their limitations and show the improvements of our framework against the previous ones.

As described in Chapter 2, configuration-based attestation is simple from the deployment point of view and is the de-facto standard used in real-world applications. However, it is a management nightmare, since the attestation results depend on the order of the measured components that are extended into PCRs, thus making configuration-based attestation impossible to cover runtime software modules. Moreover, every time a component is updated, the golden PCR values need to be changed too. In addition, showing the PCR values to external entities means exposing the platform's configurations, which may be advantageous for attackers.

In literature, researchers proposed various methods to tackle these issues, e.g., introducing additional components specifically to maintain PCR values regarding to component updates, or extending other information instead of digests of loaded components. These solutions can be generalised as different representations of the system based on its security property or its model behaviour.

Property-based Attestation

Property-based Attestation, proposed by Sadeghi et al. [46] and later generalised by Poritz [47], attests if a platform possesses certain security properties instead of raw configurations. These two solutions encourage practitioners to take a closer look at what property a platform fulfils instead of its configuration. In these solutions, properties are defined as a quantity which describes an aspect of the behaviour regarding to certain security requirements, e.g., a platform has confidentiality if in its configuration no information leakage is possible.

The major difficulty in these solutions is to map software and hardware configurations to the security property, i.e. to generate a *Property Profiles*. Though difficult, yet possible, the authors of [46] suggested to use *proof-carrying code* [48, 49] to determine a set of specific properties provided by a set of specific configurations. If a Trusted Thrid Party (TTP) is involved, the TTP can transform platform configurations to properties and vice versa, either with *property certificates* (certify the correctness of the underlying configurations to certain property) or with *security property language* (produce security statements) [50, 51]. Then a platform or an application claiming certain property can download the appropriate certificate from the TTP after proving its configurations, e.g., via conventional configuration-based remote attestation mechanism or off-line human inspection. Afterwards, the property certificate should be extended into a PCR slot which binds the property certificate to this platform. Then, when the platform receives a remote attestation request, it shows to the challenger that it has a valid property certificate and the

digest of the certificate is present in its PCR without disclosing any other sensitive information. Moreover, platform component updates do not require new property certificate, and the attestation result only depends on the corresponding property certificate and if the digest of the certificate is in its PCR.

Implementation of property-based attestation is a hard task. It should be activated in an early phase when a platform is booted, so that the property certificate can cover as many components as possible in the platform.

Under this concern, Kühn et al. [52] and Korthaus et al. [53] tried to enhance the boot loader to support both property-based attestation and sealing with existing TCG-enabled hardware and software. A *Property Certificate Authority* (Property CA) is introduced to issue property certificates in an extended X509v3 format. The enhanced boot loader is in charge of translating the measures of the components loaded in the boot phase (e.g., operating system kernel) into properties with the help of property certificates, and extending the digest of these certificates instead of the components into designated PCRs. Thus it allows property-based attestation and sealing of unmodified operating systems. Moreover, applications running on top of this operating system can use existing tools (e.g., TCG Trusted Software Stack, TrouSerS [54]) to seal data into TPM based on property defined in the property certificate, i.e. whether the digest of the property certificate present in physical PCR is correct or not. Still, this solution relies on binary measures to identify the components loaded before the enhanced boot loader.

Property-based attestation preserves attester's privacy by preventing the leakage of any information not pertinent to the relevant security issue. It can simplify the management efforts regarding to component updates. However, the most challenging task is to determine meaningful properties and map them to specific platform configurations. By considering the origin of this problem, a formal definition of property is still missing. More importantly, the platform is still considered as a whole, thus making it impossible to differentiate with runtime services, e.g., it may happen that the deployed service is compromised due to runtime attacks but the underlying platform is intact, and the verifier does not know which part of the platform is compromised.

Model-based Behavioural Attestation

An alternative solution was proposed by Li et al. [55] and later generalised by Alam et al. [56], which is called *Model-based Behavioural Attestation*. In this approach, the system behaviour is generally defined as *quad-tuples* with a set of subjects, a set of executable programs, a set of behaviour relevant inputs and a set of behaviour outputs.

After the trusted computing base of a platform is measured with a conventional configuration-based attestation mechanism, an additional component is introduced to collect system behaviour measures or monitor the attester's behaviour (e.g., as part of *Linux Security Module* [57]) and extend these measures into PCRs to prove

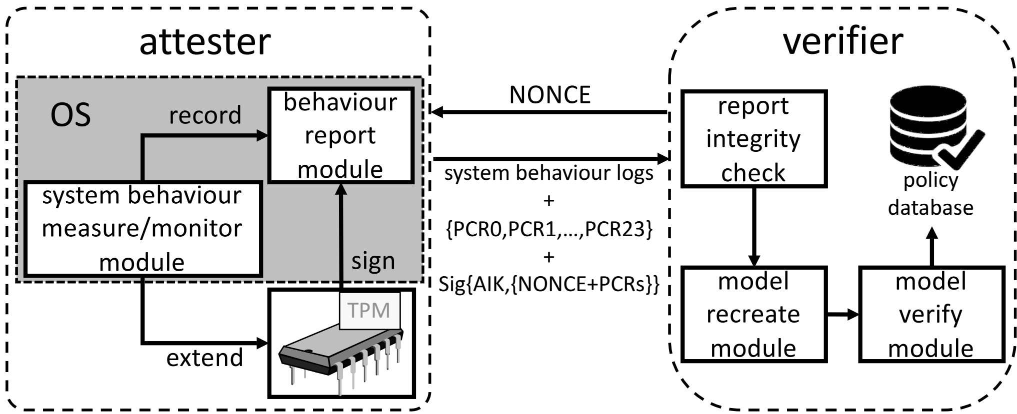

Figure 3.1. General architecture of model-based behavioural attestation.

their integrity. Thus when a remote attestation request is received, a report module (e.g., as part of operating system) prepares a report with enough information to model the system behaviour, e.g., system measures and the quote data from the TPM to ensure the integrity of the measures. Then the verifier first checks the report's integrity and generates the system behaviour model. Subsequently, the model is confronted against a policy to decide if the attester's behaviour is legitimate or not (Figure 3.1).

Under the assumption that a secure kernel enforces process separation, Gu et al. proposed a solution modelling a specific program [58]. It uses static analysis to create a *System Dependence Graph* (represents the data dependences and the control dependences among procedures of a program) and a *Program Dependence Graph* (represents the language level dependences of a program), that are used as a benchmark for attestation. On the attester side, an attestation agent is running in the kernel space using the kernel services such as process management and file system management to monitor the execution of the program of interest. Every operation performed by the program, e.g., file access operation, socket operation, other program invocation or inter-process communication are recorded and extended into a free PCR. In the end, the recorded operations and quote data from TPM are sent back to the verifier for evaluation.

Usage CONtrol (UCON) model [59, 60] is a popular choice in model-base behavioural attestation. It specifies the boundaries in which the attester is allowed to use some object and associates the attesting platform's trustworthiness to the object's policy, which is sent together with the object. Once the attester becomes untrusted, revocation of the access privilege to this object is relied on the *continuity of access decision*[1], a feature provided in UCON model.

Nauman et al. [61] also use UCON model, and their approach is more comprehensive. An additional component called *Behaviour Manager* is in charge of

[1]Control access to the object even if it is outside the direct control of the concerned stakeholder.

enforcing the UCON policy attached to the received objects. Meanwhile, it records attribute update logs and access logs of the object and extend the logs into PCRs. The verifier, after receiving the logs, can re-generate the attester's policy model and confront it against the UCON policy with an information flow check algorithm or directly with the UCON safety check algorithm.

Model-based behavioural attestation can provide very rich semantic information, which can help tracing the system behaviour modification, and verification of access control and usage control models. On the one hand, it helps identifying the "attack" point, hence to employ more explicit countermeasures. On the other, it could bring significant overhead of, not only evaluation process in the verifier, but also data collecting process in the attester, which makes it a less feasible choice in real-world cases.

As a conclusion, property-based attestation and model-based behavioural attestation are still in early stage. They are very difficult to be adopted directly in mainstream products. For instance, the behaviour specification and security property definition are not automatic, human intervention (e.g., an extensive analysis of the attester) is mandatory. Moreover, each system needs its own behaviour specification and configuration to fulfil certain security property, combined with the aforementioned issue, it will require a huge amount of efforts from system administrators.

Binary-based Attestation

Another approach to enhance the flexibility of configuration-based attestation without the complexity of property-based and model-based behavioural attestation is called *binary-based attestation*. It is a simplified variant of model-based behavioural attestation, with the main target to include runtime services at application layer when a computing platform is attested.

Binary-based attestation proposes to infer the behaviour of runtime service by identifying its executables and configurations through their digests. It uses the TPM as a trusted anchor, which implicitly proves the integrity of the measurement list which contains the executable and configuration digests. Thus, the main analysis target changes from the PCR values to the measurement list (Figure 3.2).

Compared to property-based and model-based behaviour attestation, binary-based attestation is easier to adopt because its scope is definite and narrower. The most influential work to measure static data (sometimes called structural data) in Linux kernel is called *Integrity Measurement Architecture* (IMA) [62]. Since 2009, this work has already been mainstreamed into Linux kernel. It has been used in several large scale trusted computing relevant projects, such as OpenTC [63] and SECURED [64].

In short, using *Linux Security Module* (LSM) [57] hooks (e.g., *file_mmap*), IMA measures static data when they are loaded into kernel memory and prior to be executed, and extends the measures into the TPM (Figure 3.3). Thus no file can hide the fact that it is loaded. With regards to binary-based attestation of software

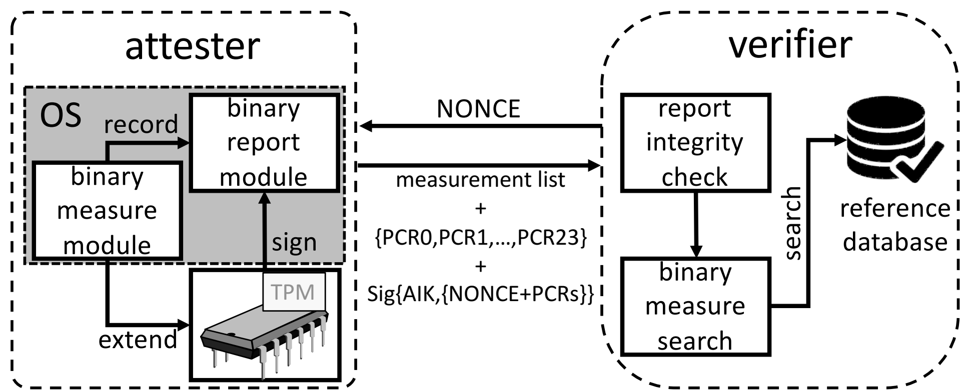

Figure 3.2. General architecture of binary-based attestation.

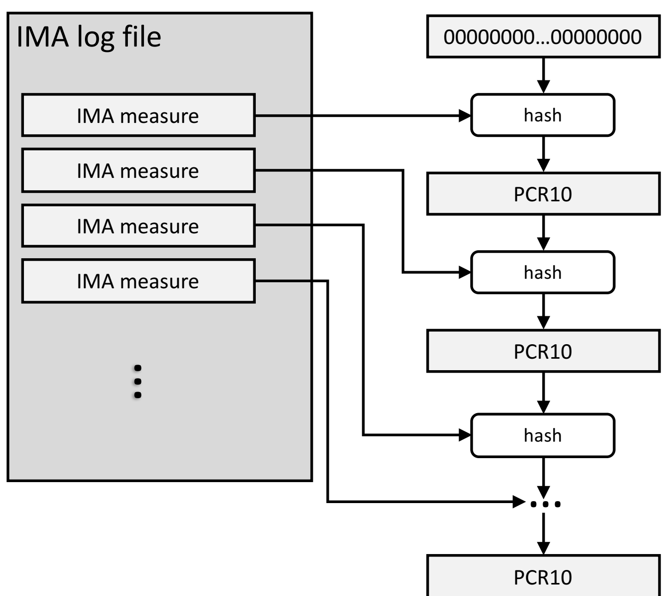

Figure 3.3. IMA measure extend operations.

services, IMA provides a viable approach for collecting binary digests of software services.

Based on its policy, IMA starts measuring static data upon various system events, e.g., binary execution, file access, kernel module/library loading. Thus, the LSM hooks may be frequently triggered when a great amount of files needs to be measured (e.g., a featured service such as Apache web service is started), hence degrades the performance. As a countermeasure, a cache is created in IMA to store the previous measurement operations, thus IMA only (re-)measure a file if the measurement operation of this file is not in the cache or an attribute of the file is changed, to reduce the occurrences of measurement.

When a file is measured and inserted into the measurement list, a PCR (default 10) accumulates the digest of the new measurement entry in the measurement list.

In this way, the measurement list can be implicitly authenticated by the TPM. This architecture is the first practical work to extend the TCG integrity measurement concept from the physical layer all the way up to the static data in the application layer in an operating system, which is a big step forward to reallocate TC technology from labs into real-world applications.

On the other hand, IMA can also appraise files based on their signatures stored in their extended attributes (xattrs). For instance, if the signature of an executable comes from an invalid key or the signature verification fails, e.g., the certificate of the key is not present in kernel's keyring, IMA prevents the executable to run, but the digest of the executable and its signature are still recorded in IMA measurement list. In [65], the authors combined the local appraisal feature of IMA with remote attestation, where the concatenation of file digest and its signature are extended into PCR10 in order to ensure their integrity. Further these data are sent to the verifier in order to show that only the executables with valid signature have been run in the platform.

However, IMA is not capable of dealing with unstructured/dynamic data, such as temporal program input/output or inter process communication. This drawback mainly brings three problems. First, the load-time measurement of executable code and static configuration files alone cannot accurately reflect runtime behaviour of the service. For example, a service can be compromised if a malicious input triggers a vulnerability, or a stateful program can be compromised if the dynamic state is not handled in a manner that preserves its high integrity (e.g., modified by other process). Second, all measured data (the digests of all acceptable scripts, executables and system configurations) need to be known by challenging party even if some measured components are not relevant to the service of interests. Third, the solution reveals sensitive information (e.g., what software is running in the attester), it may be misused for platform discrimination, that an attacker can use the data in remote attestation to launch remote attacks or limit user's choice of used software services.

Controlling unstructured data integrity has been studied for a long time but there is no optimal solution. The approaches to deal with this issue can be mainly categorised in two ways.

The solutions in the first category adopt a *Mandatory Access Control* (MAC) tool to ensure information flow integrity, e.g., *PRIMA* in [66] and *Dynamic Remote Attestation Framework and Tactics* (Dr@ft) in [67]. However, these solutions do not consider how an application internally handles its input, as a software may misuse assigned privileges if it operates on information at different security levels [68]. *Decentralised Information Flow Control* (DIFC) addresses the above problem by allowing developers to specify flexible policies, but it requires modifications to the applications or even the operating system [69]. In summary, when a platform is attested, not only the measures of data, but also the measure of the MAC policy need be provided to the challenging party in an authentic manner, in order to assess whether the policy enforcing the information flows is trusted. Thus, the solutions in this category are very difficult to be deployed and generalised, since every platform needs its own custom MAC policy.

The second category comprises solutions inferring runtime integrity of the targets by directly collecting measures of dynamic data. In particular, some solutions aim at evaluating the dynamic state of the Linux kernel, like LKIM [70] (further improved in [71] by employing a *Copy-on-Write* mechanism), and SBCFI [72]. Among the solutions at application level, we mention ReDAS [73], an architecture to attest two dynamic properties of applications (structural integrity and global data integrity) and *Dynamic Integrity Measurement Architecture* (DynIMA) [74], which detects return-oriented programming attacks with a new software module, called *Process Integrity Manager* (PIM). In order to take advantage of the easiness of deploying IMA, authors in [75] proposed another extended version of IMA called *Enhanced IMA*, which records all interactions occurring between different processes through regular files. Hence not only static data, but also a specific set of dynamic ones can be measured. However, it is inevitable that the solutions in this category introduce additional performance loss because of the newly added functionalities. Moreover, in some cases system kernel or the application of interests need to be modified to support the measurement of dynamic data feature, e.g., add hooks to function output and extend it into PCRs.

Implementation Perspective

From the practical perspective, remote attestation technique requires support from various components, from the hardware layer to the application layer. In each layer, different implementations are available.

In the hardware layer, besides hardware discrete TPM chip, software-based emulation of the hardware TPM is also popular, e.g., *swTPM2.0* [76], *TPM-emulator* [77, 78] and firmware TPM [79, 80]. They provide almost the same functions as the hardware TPM but have much worse security guarantee since they are still prone to software attacks.

Trusted boot can measure the integrity of the platform's boot phase, which forms a trusted computing base, where the operating system and the services are running in a platform with a known initial state. Moreover, this initial state can be used for sealing and full hard disk encryption systems, such as *Microsoft BitLocker Drive Encryption* [81]. Trusted boot is the most popular trusted computing feature nowadays with multiple implementations available, e.g., *GRUB-IMA* [82], *TBoot* [83], *TrustedGrub2* [84], Intel's Trusted Execution Technology [26] and so on. They provide almost the same functionalities, with the differences come only from the implementation perspective. These tools can prevent *Bootkit* attacks, such that the boot component code (e.g., Master Boot Record, Volume Boot Record or boot sector) is infected to compromise full disk encryption systems. An example is the *Evil Maid Attack*, in which an attacker installs a bootkit on an unattended computer, replacing the legitimate boot loader with one under their control [85]. However, trusted boot implementations can be directly used to prove the boot phase integrity of the platform to itself, but it cannot inform any remote party of its integrity state, thus a high level framework to convey the integrity evidence is mandatory.

From the framework point of view, *Open Platform Trust Service* (OpenPTS) is a proof-of-concept and experimental implementation of the trusted platform specification defined by TCG [86]. This framework is developed with the support of *TrouSerS* [54], which is intentionally written in C to support many types of target platform, including servers, PCs and embedded devices. Once the integrity evidence is prepared, it is sent back to the verifier through a *Secure Shell* (SSH) channel and processed there.

Along this vein, OpenAttestation SDK [87] and OpenCIT [88] SDK, both were initiated by Intel, partially adopted the TCG specifications in order to attest the boot phase integrity of the nodes running in a cloud. Thus an OpenStack [33] manager can be informed to deploy virtual instances on machines that has a known initial state, this is called *trusted compute pools*.

Trusted Network Connect (TNC) [32] is an open-source architecture proposed by TCG for network security, and it focuses on solving *Network Access Control* (NAC) problems, including network and endpoint visibility, network enforcement and device remediation. This standard provides a communication foundation for securing embedded systems such as network equipment, automotive, and IoT solutions. TNC can integrate with a TPM for secure authentication and attestation, addressing detection and mitigation of bootkits and other compromised software. To be more specific, during TNC handshake, the TPM of end point sends its PCR values to *Policy Decision Point* (PDP), subsequently the PDP compares them to a whitelist, and if not listed, the end point will be quarantined and remediated. The *Internet Engineering Task Force* (IETF) *Network Endpoint Assessment* (NEA) working group has published several *Request for Comments* (RFCs) based on TNC client-server protocols [89], e.g., RFC 5792 [90] and RFC 6876 [91]. Currently, this standard has been implemented in *strongSwan* as part of authentication criteria [92]. Recently, this set of specifications was extended to also cover the trust communication problem, and its name was changed to *Trusted Network Communication*.

The aforementioned tools can be used to attest the authenticity of a platform and whether a legitimate operating system is started in a trusted environment. They can be used to defend against software-based or remote attacks trying to steal sensitive data by corrupting the components loaded during the boot phase or modifying the platform's configurations. However, this protection is incomplete, because on the one hand, even the boot phase of a platform is trusted, the platform is still continuously exposed to remote attacks coming from the network. On the other, the aforementioned frameworks do not take into account the integrity of services running on these platforms, i.e. they do not integrate IMA or any other tools to measure the software services. The services are the front-end towards end users by providing answers to the user's inquiry, their integrity is of high relevance to the trustworthiness of the whole distributed system and the genuineness of the answers received.

3.1.1 Contribution

In this work, we propose a remote attestation framework, which provides a coverage of not only the integrity state of the boot phase of a physical platform but also the load time of software services running in it. Their integrity is continuously checked against a reference database containing all published legitimate packages and configuration whitelists. Thus if the platform is compromised by loading unknown boot components or the service is compromised with unknown configuration files or executables, the verifier in our framework can detect this fault in a timely fashion. Moreover, our framework provides simple property identification of whether the services running in attester have security and functional bugs.

3.2 Requirement Analysis

In this section, we follow the expected properties of remote attestation defined in [93], and analyse and define the basic requirements from security and functional perspectives.

3.2.1 Security Requirements

Platform boot phase integrity measurement: inherited from the transitive chain of trust model from TCG, our first step is to identify the root of trust for the chain and measure the initial state of the attester, which is the platform hosting the services. We follow the approach of trusted boot, that is a *Core Root of Trust for Measurement* (CRTM) is needed as the first component to be activated when an attesting platform is powered on. Subsequently, the other components loaded during the boot phase of the attester need to be measured and their measures need to be extended into designated PCRs by their previous active component.

Service load time integrity measurement: unique identification of loaded executables and configuration files of services is needed, and the measures of these structural data need to be inserted or appended in the integrity evidence signed by hardware-based identity key and evaluated by the verifier.

Evidence integrity and authenticity: the integrity evidence needs to be authenticated by a hardware-based identity, thus the evidence is bound to a physical platform and cannot be forged or modified. Additionally, the evidence should contain random temporal data to prove its freshness.

3.2.2 Functional Requirements

Fast deployment: the alteration to existing software and hardware environment should be minimal (if any) and additional concept introduced should make use of and adhere to existing specifications.

Minimal additional cost to management and original system: the introduced management overhead and performance loss to the original system should be minimal. Meanwhile, the constraints to the used software and operating systems should be minor.

Rich semantic information for integrity evidence: the remote attestation process should not be limited to a Boolean result (i.e. trusted and untrusted), but should provide a more informative result which allows decisions to be derived from several claims, e.g., property that the attester provides.

Backward compatibility: the remote attestation feature should generate no constraint for the original distributed system and its service. Moreover the original system should preserve its behaviour after the remote attestation feature is deactivated.

3.2.3 Possible Attacks

As no solution solves all problems, in this section we present the expected threats from attackers. The attacker may be a malicious third party, a system user or even the administration of the platform where the distributed system is running.

He may perform bootkit attack that replaces components loaded in the boot phase of a platform to corrupt ones, e.g., replace the authentic kernel to a cracked one. Or, after the system is running, the attacker may modify service's behaviour by loading customised service binaries (e.g., to store received data) or configuration file to his favour (e.g., lower the security of the distributed system by using obsolete cryptographic algorithms), or he can even perform attacks in the platform by launching scripts (e.g., to occupy the platform's resources by doing heavy computation or to inject malicious code to files).

Afterwards, the attacker may change the integrity evidence stored inside the platform operating system, e.g., removing or modifying the record belonging to the script he launched, but he cannot change the data already stored inside the hardware root of trust.

The adversary can also forward a remote attestation request to another genuine party to get a valid attestation result. Similarly, he can record the remote attestation response message transmitted between attester and verifier, and resend the attestation response after he compromises the attester.

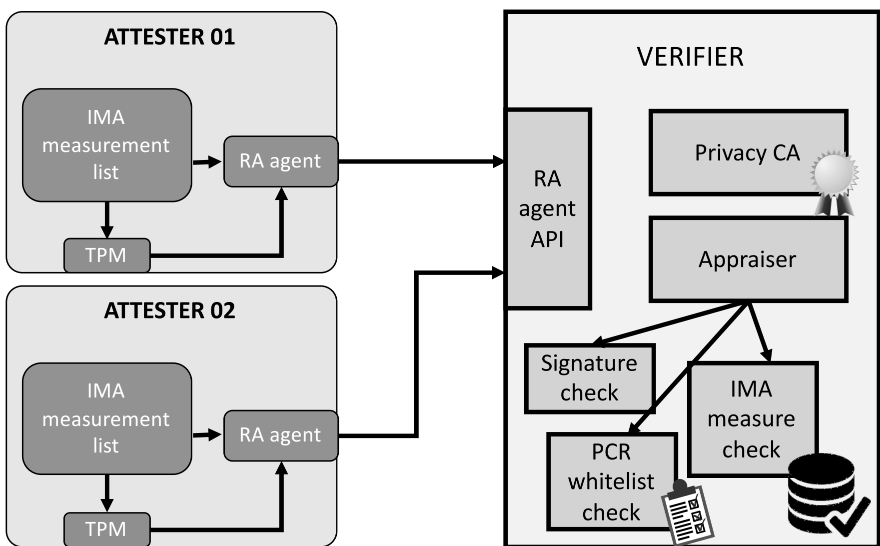

Figure 3.4. Overall remote attestation framework architecture.

For the sake of completeness, we need to mention that there is still one possible attack. Inherited from the attack model of TCG specifications, our design does not consider sophisticated invasive or non-invasive hardware attacks on any platform involved in distributed system.

3.3 General Architecture

Our solution aims to provide a complete solution of attesting the integrity state of boot phase of physical platforms and load time of software services running in distributed systems. In order to fulfil the requirements defined in Section 3.2, the verifier in our framework is placed in a node running as a third party.

This design is mainly for three reasons, (*i*), attestation of a platform with the services running in it is a computing intensive work, hence using a third party node can help offloading the workload from the attester or any other nodes in the system; (*ii*), a single point of management can significantly reduce the workload to monitor the whole system and the management point can also be in charge of certifying the AIK involved, which further simplifies the framework architecture; (*iii*), detaching remote attestation from the system enable backward compatibility, since the remote attestation feature will be used as an additional insurance for system integrity. Thus, the overall architecture of our framework is illustrated in Figure 3.4.

3.3.1 Attesting Platform

The attesting platform, or attester, in our design would host the services of the distributed system. They should be attested in order to check their integrity state

and that of the services running in their application layer.

Each attester needs a TPM working as a hardware root of trust. To measure structural data of the services when they are loaded, each attester needs to activate the IMA module in its operating system kernel. Further, IMA will insert the measures into its measurement list and extend the digest of measurement entry into a pre-defined PCR (PCR10 by default), in order to ensure that no field in the measurement entry can be changed without being detected.

A remote attestation agent is running in the attester, which is in charge of interacting with the verifier and preparing integrity reports sent to the verifier for evaluation. For simplicity, the remote attestation agent should be prepared by the verifier for all its attesters, thus it can be downloaded and installed at the registration phase of the attester.

3.3.2 Integrity Verifier

The integrity verifier in our framework is hosted in a third party node, it is in charge of checking the integrity state of all attesters registered with it. From the functional point of view, the verifier has two roles, and in the following section we will separate it as two components, i.e. *PrivacyCA* and *Appraiser*.

The PrivacyCA is the CA certifying the registered AIK of each attester, and its purpose is to allow the appraiser to identify the generator of the received integrity evidence. Although there are privacy preserving attestation algorithms available to hide the platform identity during the remote attestation process (e.g., Direct Anonymous Attestation [28] and its variant schemes [94, 95]), in our scenario, we do not consider the privacy issue of attesters. Because the attestation paradigm has been changed, the targets are servers instead of user terminals. As a matter of fact, we prefer the server's identity to be revealed to the end users, so they can have knowledge of the server he is interacting with using hardware-based identity. In this sense, the term "PrivacyCA" is a legacy one: it was used when attesting personal devices to protect the user privacy (as the user could register many different AIKs). In order to be consistent to the terminology of the TCG specifications and previous remote attestation work, we still use PrivacyCA in the rest of the thesis, although it acts as a normal CA which helps the appraiser to identify the generator of integrity evidence.

The appraiser is the core component in the verifier, and its job is to evaluate the integrity evidence provided by each attester in various ways. First of all, it needs to check the authenticity and integrity of the evidence, in order to check if the evidence has been compromised. Then it needs to check the boot phase of the attester in order to see if the attester has been booted in a trusted state. This step is mandatory because the chain of trust is needed from the hardware root of trust, and by evaluating the boot phase of an attester we can understand if the attester loaded all known components with known configurations, and a trusted computing base is created (i.e. all components loaded until this state are trusted).

Moreover, since the services running in the attesters are directly interacting with end users, their integrity is also critical. In our design, the appraiser also needs to provide evaluation of the integrity state of software service in the application layer. Software components are compiled into binaries and then the binaries and their associated configuration files are loaded into operating system kernel memory when the service is activated. Thus, the integrity of the loaded binaries and configuration files can partially guarantee the expected behaviour of the activated services. For this reason, the appraiser also needs to evaluate the binaries and the configuration files loaded into the kernel memory, i.e. the IMA measures of the attester.

Framework Components

To be more specific, the required components in our framework are the following:

- *TPM*, the core of our solution, which is the hardware root of trust;
- *RA agent*, which is in charge of generating integrity evidence and interacting with the verifier;
- *IMA*, which is mandatory to measure structural data (i.e. files) when they are loaded into memory;
- *(Privacy) CA* provisions and certifies attesting platform's AIK, which is required to identify the generator of the integrity evidence;
- *Appraiser* is the orchestrator of the overall attestation process and it is the one that actually evaluates the integrity evidence from each attester and gives result;
- *WhiteList table* is a collection of trusted PCR values, it can be derived from a clean setup state;
- *IMA measure reference database* is a collection of trusted IMA measures, it can be derived from a clean setup state or from packages released from their official repositories.

3.4 Remote Attestation Workflow

In this section, we present the overall workflow of our remote attestation framework. Mainly it is composed of two phases, *registration phase* and *remote attestation phase*.

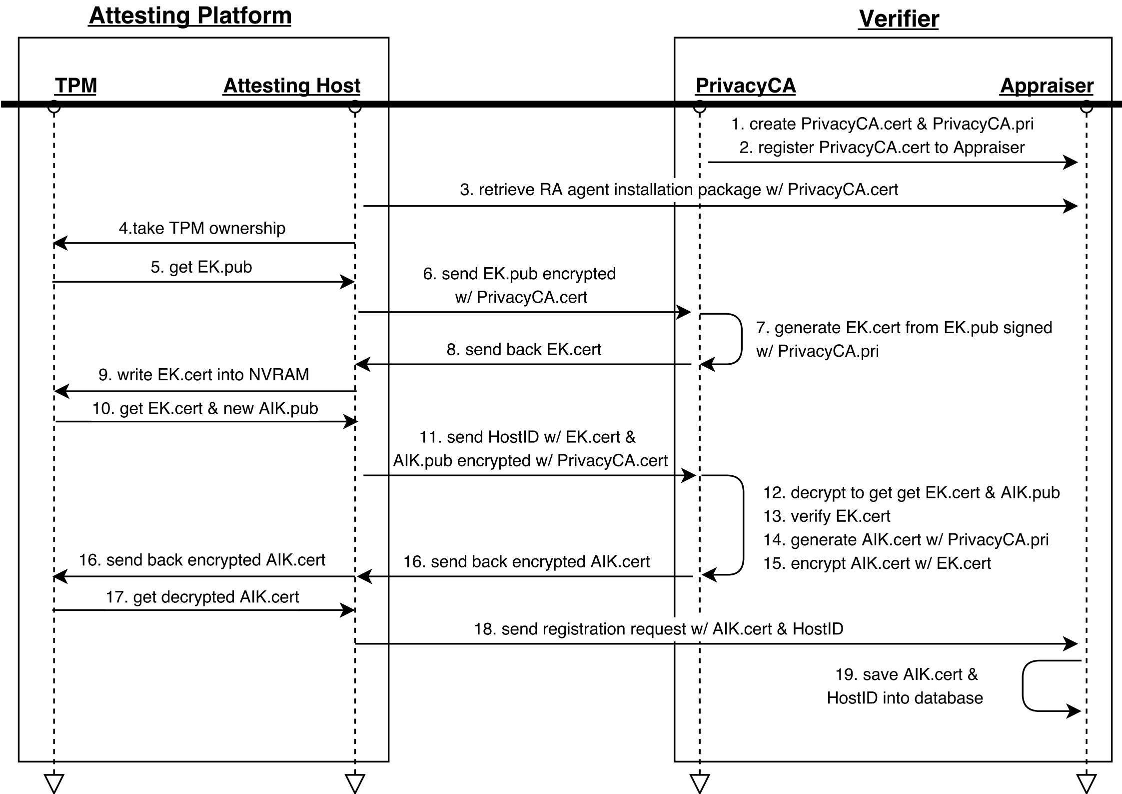

Figure 3.5. Registration phase for PrivacyCA and the first attesting platform.

3.4.1 Registration Phase

In registration phase, the involved components are *TPM* and *RA agent* running in *attesting host* installed in the attesting platform and *PrivacyCA* and *appraiser* installed in the verifier (Figure 3.5).

Each component has its own task during the registration process, and the purpose of this phase is for the appraiser to record the AIK certificate of a TPM, thus it is associated to a specific device to sign the integrity evidence, i.e. the PCR values.

This AIK certificate should be issued from a trusted PrivacyCA with the PrivacyCA's certificate known by the appraiser (i.e. steps 1-2). Since each TPM may have multiple AIKs but only one EK, thus the AIK must be endorsed by an EK and the AIK certificate can only be issued after the EK is proved to be genuine (i.e. steps 5-14). After the AIK certificate is created, it will be sent back to the TPM encrypted by the public part of its EK, ensuring that only the TPM with this EK can use this specific AIK and AIK certificate, and later the AIK certificate is registered to the appraiser (i.e. steps 15-19).

Step 1 and step 2 are only needed when the verifier is initiated, and the other steps are required for each attester in order to put its AIK certificate into the

appraiser's database.

Afterwards, the nodes should also register their trusted state (with trusted boot) in order for the appraiser to have golden PCR values stored in its whitelist database for further usage. Since the same software may have different binaries in different Linux distributions, e.g., the source code is compiled with different flags, the attester also needs to tell the appraiser which Linux distribution it is using. Later in the remote attestation phase, the appraiser can evaluate the trustworthiness of the attester using this parameter.

3.4.2 Remote Attestation Phase

The abstract level of the interactions in remote attestation is illustrated in Figure 3.6, while more detailed information is illustrated in Figure 3.7.

From Figure 3.6 we can see there are six steps to be completed in order to vet an attester. First, the verifier asks the remote attestation agent to send the integrity evidence of the attester (step 1). Once the request is received, the RA agent issues a *quote* operation to the TPM for getting the PCR values and a signature of them (step 2), and uses the result plus the recorded IMA measures (step 3) to create an integrity report.

When an integrity report is received by the verifier (step 4), it first checks the digital signature of the report with the public part of the registered AIK. Then it compares the received PCR values that contain the digests of the components loaded in the boot phase to a whitelist database in order to check that the boot phase of the attester is trusted (step 5). Afterwards, the verifier distils the IMA measures from the integrity report, and recomputes the final value following the extend operations illustrated in Figure 3.3. If the final value equals the PCR value in the received integrity report, this proves the IMA measurement list is intact. Finally, the verifier queries the IMA measures to a well-formed database containing the digests of the binaries found in the released packages from the official repositories with whitelisted custom configurations (step 6). In the case that the received PCR value does not match the whitelist or there is an unknown measure, the verifier can alert the system administrator that a certain node is not booted in a trusted state or certain service in the application layer is compromised.

Figure 3.7 is more informative, and it also shows how the freshness of the integrity report is assured with a nonce sent by the appraiser (step 1) and when it retrieves the AIK certificate registered by the attester based on its host ID and verify the quote signature (steps 5-6). Most importantly, it shows the step to check the IMA measurement list is intact (step 8).

3.5 Details of the Framework

In this section, we provide the details of our framework, showing the exact way how it works. In particular, we show how to enable attestation of the integrity of software

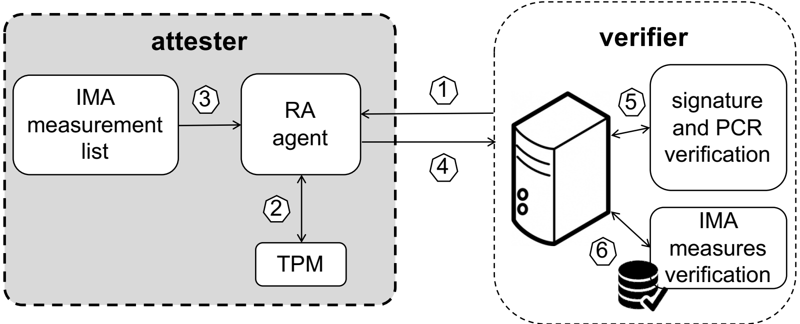

Figure 3.6. Remote attestation process.

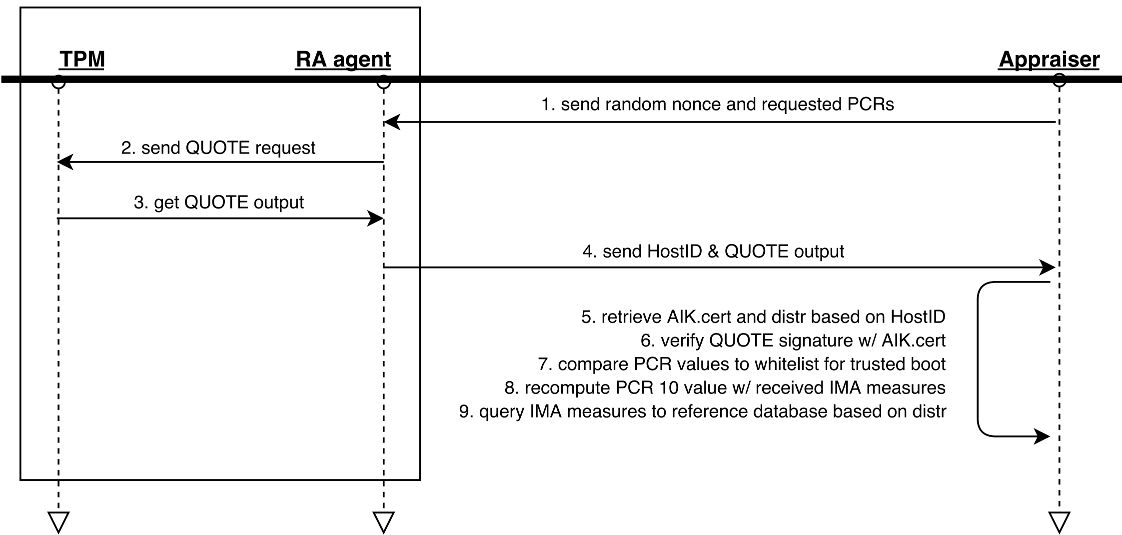

Figure 3.7. Diagram of remote attestation process.

services running in system application layer and how to create a reference database for multiple Linux distributions. In the end, since extended integrity reports will bring significant additional performance overhead, we also show the methods to minimise this overhead in both the attester and the verifier.

3.5.1 Trusted Boot

Trusted boot is used to ensure the attester is booted into a trusted state by loading known components (i.e. BIOS, boot loader, kernel and its modules) in the boot phase. This feature is meant to show a trusted computing base of the attester is available by storing digests of the components loaded in the boot phase into different PCRs. The following describes an example usage of PCRs during a trusted boot phase:

- PCRs 00-03: for use by the CRTM (e.g., initial EEPROM or PC BIOS);
- PCRs 04-07: for use by the boot loader stages;
- PCRs 08-15: for use by the booted base system (e.g., compartmentalisation system, hypervisor).

As shown in Chapter 2, the PCR values are fixed if and only if the same components are loaded in a fixed order. Thus, when the verifier receives a set of authentic PCR values, it can conclude that the attesting platform is booted with trusted components and its operating system is running in a trusted state, i.e., until this point, the attesting platform complies with its expected behaviour.

3.5.2 Service Load-time Integrity Measurement

TCG does not provide any implementation detail or constraint about how integrity measures are obtained. In our framework, the *Linux Integrity Measurement Architecture* (IMA) [62] is chosen for the purpose to measure the software service load time integrity because, on one hand, it does not require any modification to the Linux operating system (it is already integrated into the Linux kernel from version 2.6.30), on the other hand, it is one of the most accepted TCG-compliant solutions[2]. IMA is the state of the art static measurement mechanism. It is the first practical work to extend the TCG's trust measurement concepts to dynamic executable content from the BIOS all the way up to the application layer, which makes the remote attestation technique much more feasible in commodity platforms. It is an important step to introduce a TC-compliant system into a real-world scenario.

Once activated, IMA starts measuring the loaded files according to the criteria specified in its policy. System administrator can define the policy by writing all its statements into the special file `policy` in the `securityfs` file system (typically mounted at `/sys/kernel/security/` directory). Each static data is measured when it is loaded into the kernel memory, and immediately the digest of the measurement entry is extended into a PCR (default PCR10) to guarantee that the obtained measure cannot be tampered by any other component. Moreover, the list of measured files (with their digests) is visible any time through another special file `ascii_runtime_measurements` (Figure 3.8) from the same file system, encoded as ASCII text, or in binary form through the file `binary_runtime_measurements`.

The following information is provided in Figure 3.8: the template used is *ima*, which only has four attributes, i.e. PCR#, template-hash, filedata-hash and filename-hint. The PCR# indicates which slot is used for extend operation, i.e. PCR10. *Filename-hint* contains the measured file's name and its location. *Filedata-hash* is the SHA1 digest of the content of these files, and *template-hash* is the SHA1 digest

[2] 1137 citations in published papers by 1st Jan., 2017.

```
PCR# template-hash template filedata-hash   filename-hint
 10  4ad08...af759   ima     2517d...87ecc  /usr/bin/tail
 10  48327...9fed4   ima     9ee46...644ed  /usr/bin/ssh
```

Figure 3.8. Example of IMA measures in ASCII format log file.

of the other values concatenated (i.e. the digest of the measurement entry), in order to assure their integrity. Both filedata-hash and template-hash are encoded in hexdecimal.

In the case that the device is equipped with a TPM chip, an aggregation of the template-hash values are stored in one of the available registers, normally PCR10, instead of filedata-hash values, because the correctness of the template-hash value can imply the correctness of all other values in a measurement entry. The PCR value will be signed by the TPM with its AIK once a quote request is issued, and in this case the integrity of all the values in the measurement list can be cryptographically verified with the hardware-based AIK. A challenging party is able to simulate the extend operation, by hashing the concatenation of each template-hash value and the previous hashed value (Figure 3.3). If the result matches the authenticated value in PCR10, then the IMA measurement list has not been tampered.

3.5.3 Integrity Report

In order to attest the software integrity in the application layer, the IMA measures of loaded binaries and configuration files need to be put in integrity reports. In our framework, we follow the semantics proposed in the TCG Infrastructure Work Group Integrity Report Schema specification [31].

The integrity report is composed of two parts: (*i*), *QuoteData*, which contains the information required to ensure the integrity of the PCR values, such as the nonce, selected PCRs, and the signature of the PCR values generated with the AIK (Figure 3.9); (*ii*) *IMA measures*, which contains all the measures recorded by IMA after the platform is booted (Figure 3.10).

QuoteData follows an XML structure, which contains the following information:

- the PcrComposite is used to aggregate multiple PCR values in a single structure and represents the TPM TPM_PCR_COMPOSITE structure from a call to TPM quote;
  - PcrSelection - identifies which PCRs are quoted:
    - SizeOfSelect - the size in bytes of the PcrSelect structure;
    - PcrSelect - PcrSelect is a contiguous bit map that shows which PCRs are selected. Each byte represents 8 PCRs. Byte 0 indicates PCRs 0-7, byte 1 8-15 and byte 2 16-23. For each byte, the individual bits represent a corresponding PCR;

```
<QuoteData ID="_82897509-2D8A-4061-A2D9-DA2975998C70">
  <Quote>
    <PcrComposite>
      <PcrSelection SizeOfSelect="2" PcrSelect="AAQ="/>
      <ValueSize>15</ValueSize>
      <PcrValue PcrNumber="13">
        AqW6avizcmWIL0mTpWncin/NrPE=
      </PcrValue>
    </PcrComposite>
    <QuoteInfo VersionMinor="2" Fixed="QUOT"
      ExternalData="BqW335izcmWIL0m09Wncin/NrPE="
      DigestValue="AqW6avizcmWIL0mTpWncin/NrPE="
      VersionMajor="1" VersionRevMajor="1" VersionRevMinor="2"/>
  </Quote>
  <TpmSignature>
    <SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <SignatureValue>
      4Rxc/Nh/i6zYEumYbqhh8h+qTbGWowCKbEJgEH3rraxM1WMPYi3YdKR
      /D+2TNhBdPg3U7ydy6WwJ/c6uULq7ywUREG0zjxY4Vxe4wxv269VXtX
      QNXwzPCwfVEVhbc+wJw6HE4fhX6y4FCx2D6djD9r2geIBRil0IfrU=
    </SignatureValue>
  </TpmSignature>
</QuoteData>
```

Figure 3.9. QuoteData example.

  - ValueSize - the length in bytes of the array of PcrValue complex types;
  - PcrValue - the array of PcrValue structures. Each PcrValue contains a PCR number attribute to correspond to a PCR identified in PcrSelection;

- the QuoteInfo is the structure created inside the TPM and used in the calculation of the Quote signature;
  - ExternalData - the externally supplied nonce;
  - DigestValue - the composite hash value of the PCRs;
- TpmSignature - contains the quote signature value and signature method information.

The IMA measure is inserted into the XML attribute named *SimpleSnapshotObject* (Figure 3.10). The XML attribute is a fully TCG-compliant representation of IMA integrity measures and enables the analysis of service load-time integrity state in the attesting platform.

The integrity report includes the information from IMA measurement list. Each `<ns4:Objects>` corresponds to one IMA measure, with the *PCR number* stored in `<ns4:pcrindex>`, *template-hash* stored in `<ns4:Hash>`, *filedata-hash* stored in `<ns4:eventdigest>` and *filename-hint* stored in `<ns4:eventdata>` (also see example from Figure 3.8).

```
<ns3:SimpleSnapshotObject>
      <ns4:Objects>
            <ns4:Hash AlgRef="sha1" Id="PCR_10_LV1_0_0_EVENT">
                  4ad0868e88dc676f043d3367176b4af7788af759
            </ns4:Hash>
            <ns4:pcrindex>10</ns4:pcrindex>
            <ns4:eventtype>0</ns4:eventtype>
            <ns4:eventdata>/usr/bin/tail</ns4:eventdata>
            <ns4:eventdigest>
                  2517d0a40aaef7ef9092fc8c6086baa749087ecc
            </ns4:eventdigest>
      </ns4:Objects>
      <ns4:Objects>
            <ns4:Hash AlgRef="sha1" Id="PCR_10_LV1_0_0_EVENT">
                  4832768ae5310aa59663ca17eb280403ef69fed4
            </ns4:Hash>
            <ns4:pcrindex>10</ns4:pcrindex>
            <ns4:eventtype>0</ns4:eventtype>
            <ns4:eventdata>/usr/bin/ssh</ns4:eventdata>
            <ns4:eventdigest>
                  9ee465e3ed831f630228206866c0f010ce6644ed
            </ns4:eventdigest>
      </ns4:Objects>
</ns3:SimpleSnapshotObject>
```

Figure 3.10. Example of IMA measurements in an integrity report.

Partial Integrity Report

The remote attestation performance is limited due to the large size of the integrity reports: a new complete integrity report must be sent to the verifier and evaluated at each attestation request, otherwise the obtained attestation result may not reflect the current state of the attesting platform.

However, some properties of IMA and of the extend operation can be used to optimise the overall attestation process:

- once a measure has been extended to a PCR, no attacker can reverse the operation, because of the incremental nature of the *extend* operation;
- each step of the IMA measure validation process relies on the previous value stored in the used PCR, so this value can be used as a starting point for verifying subsequent measures (Figure 3.11);
- the production of IMA measures does not increase linearly, since a new digest is computed only if the extended component has never been measured or modified since last reboot. This implies a high rate of new measures at boot time, and a very low rate at run time.

The first two points permit the remote attestation agent at each attestation request not to send all IMA measures, but just those ones that have not already

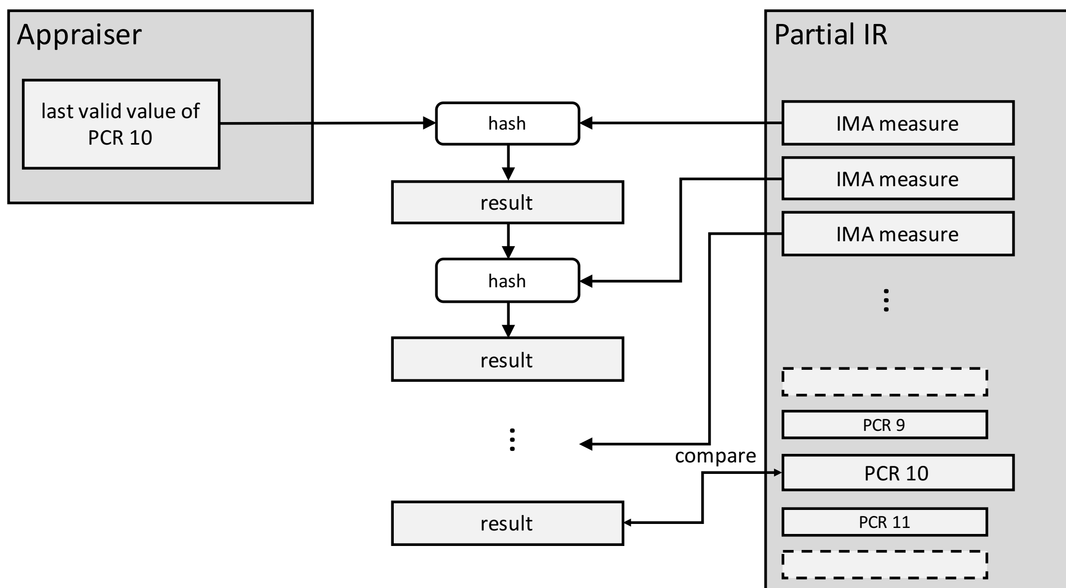

Figure 3.11. Partial integrity report verification.

been sent to the verifier. This greatly reduces the integrity report size.Also due to the third point above, the number of new measures will decrease significantly when the system is running in a stable phase. Indeed, the number of measures collected between two attestation requests may be very low when the system is in stable phase, thus making the integrity report size quite small.

The validation of a *partial integrity report* (i.e. an integrity report containing only a subset of IMA measures that have not been sent to the verifier) slightly differs from the verification of a complete integrity report. As shown in Figure 3.11, the main difference between the classic IMA validation process and the approach used with partial integrity report is the starting point for the hashing chain. Indeed, in the first case, it is a 20 B long sequence of zeros (Figure 3.3), while in the second case it is the last valid value of PCR10 which is obtained from the last integrity report received from the same attester.

In order to indicate the improvement of partial integrity reports, we give an example of a fresh minimal installation of CentOS7 attester, which has 290 IMA measures when the system is newly booted. With a full integrity report, the size is around 100 kB, while with a partial integrity report, the size can be reduced to around 4 kB when there is only one new IMA measure.

Periodic Attestation Requests

Even though with partial integrity reports the performance is improved, the verifier still needs a magnitude of seconds before providing an attestation result, which is infeasible in some use cases. For this reason, we introduced the concept of *periodic*

attestation, in order to provide users attestation results in a timely fashion, i.e. the framework user will immediately receive the latest attestation result not older than a threshold for certain target after he sends a request. The user has the possibility to submit a periodic remote attestation request, including two extra parameters:

- *timeThreshold* is the age limit of integrity report used to evaluate the trustworthiness of the target attester;
- *expirationTime* indicates the expiration time for the periodic attestation task.

Given these parameters, the verifier takes care of always having a valid attestation result that matches the user's requirements. It periodically requests a new integrity report from the attester and performs the requested analysis. Then, the latest result is available for asynchronous retrieval by calling a proper API.

3.5.4 Analysis Customisation

Once the integrity report is ready, the next step is to define how to analyse it. For this reason, in our remote attestation framework, we exposed the API for defining customised analysis tools to the received integrity reports. The administrator should specify the analysis type (i.e. its name) and the location of the analysis tool (either on-line or off-line). Thus, when the system administrator issues remote attestation requests, he needs only to define the analysis type in the request and the criteria that the integrity report should achieve in order to be trusted.

In the current version of our framework, we envision two analysis types that are mandatory. First, we need to compare the received PCR values to a whitelist in order to check the integrity state of the boot phase of the attester, which we call *VALIDATE_PCR*. Second, we need to compare the IMA measures to a reference database, which we call *IMA_VERIFY*. If the integrity report remains the same as the previous one (based on the PCR values), there is no need to analyse the integrity report again, but give the same result as previous one. Under this concern, we define another analysis type called *COMPARE_REPORT*, which compares the received report with the last one from the same attester.

However, no matter which analysis type is used, there are two steps that are mandatory. The first one is to check the authenticity and integrity of the quote output in the report with the attester's registered AIK certificate. The second one is to check the integrity of the IMA measurement list against the received PCR10 value.

Comparing the received PCR values in the integrity report to the whitelist is straightforward but it is a management nightmare, because the extend operation is order sensitive. For this reason we only use VALIDATE_PCR analysis type to verify the integrity of the boot phase of the attester where each component is loaded in a specific order and the parameters in each command are easily predefined and less dynamic, thus the final PCR values are predictable. However, not all PCRs are used

to store the digests of components loaded in boot phase, thus only a subset of PCR values are compared to the whitelist.

Verifying the integrity of load-time services is a much more complex task. In our framework, we adopt our previous solution [96], an IMA measurement analysis tool which is able to compare the IMA measures to a reference database, in order to check if the measures are known and the packages they belong are up-to-date. However, the previous solution has a problem since it lacks a standard method to transmit the IMA measures from an attester to the verifier and the quote output of the attester is also not available in the verifier, thus making this solution incomplete without a remote attestation framework. Hence, not only we integrate it into our framework to evaluate the IMA measures in the integrity report on the fly, but also with the remote attestation framework, the integrity of the IMA measurement list is verified against the received quote output of the attester.

In order to provide better flexibility, based on our previous solution, we extended the supported Linux distributions. Software packages are compiled from their source code, with slightly different configurations for different distributions. The compiled packages are published in various repositories that allow users to download and install directly, thus allows us to collect all binaries/executables in a convenient way. This approach makes TCG defined binary attestation possible, which identifies the running components (and their configuration) through their digests to infer the platform behaviour.

Measuring key structural data with IMA

IMA is able to measure structural data when it is loaded into kernel memory, however, not all structural data is relevant or meaningful. The first issue is verifying the custom configurations of running services, and the second one is that, the measured data may contain useless or temporary information, which is not able to be identified, e.g., random file opened for writing.

Regarding to the first issue, key custom configuration files, e.g., `sshd_config` needs to be taken into consideration. However, they are unpredictable for each platform (e.g., may have its own data like port number), so a static reference database is not able to identify them correctly. A computer system configuration management and change control system (like *CFEgine*[3]) is required to dynamically supplement the reference database for each node that is going to be attested.

In our case, we solve the second issues with an *Execution* policy, which sets IMA to measure the executables only, i.e. the main application's binary executed via *execve()* and its related shared libraries loaded through the *mmap()* system call by either the linker-loader (after finding the required dependencies in the ELF header), or by the programs themselves using the `glibc` function *dlopen()*. For instance, the execution policy statements in Figure 3.12 tell IMA to measure:

[3]`https://cfengine.com/`

```
# Measure all files mapped in the memory as executable
measure func=FILE_MMAP mask=MAY_EXEC
# Measure all files executed by the execve() syscall
measure func=BPRM_CHECK mask=MAY_EXEC
# Measure all files with object type equals CONFIG_t
measure obj_type=CONFIG_t
```

Figure 3.12. Execution policy of IMA.

- the files mapped in memory as executable;
- the files executed by the *execve()* system call in the kernel;
- the files loaded into kernel memory with obj_type equals to CONFIG_t (i.e. expected configuration files).

As an example, for a fresh *minimal* installation of CentOS 7 build 1503[4] distribution, the number of measures of loaded binaries is 290 with this execution policy (i.e. the first two entries). The third entry is used to identify and measure the expected configuration files of both the host operating system and the software services. In order to do so, there are three approaches. First, it is possible to use predefined SELinux[5] and AppArmor[6] labels. For instance, in Linux system, the configuration files are mostly labelled as `etc_t` type (i.e. environmental configurations). Then in order to measure these files, this label should be specified in the IMA policy. Second, it is possible to define a custom label, e.g., CONFIG_t. This approach is more suited in a service deployment scenario, where default configuration files labels are predefined and stored in packages. For instance, the IPsec service has its credential configuration files labelled as `ipsec_key_file_t` type. Third, if there is only one or a few configuration files to be measured and they are hard to be identified because their labels are generic, it is also possible to workaround and change the file owner to a fake user, and specifically define *measure fowner=UID_fake* in the IMA policy. With this trick, IMA will measure all the files belonging to this fake user when they are loaded into kernel memory.

Verifying IMA measures with a reference database

At this point, we can find the digests of all the loaded executables and configuration files of concern in the IMA measurement list. The next step is to define how they can be analysed.

First and uttermost, we need a reference database containing all digests which are considered as trusted. Moreover, in order to provide flexible attestation results,

[4] https://www.centos.org/

[5] http://selinuxproject.org/page/Main_Page

[6] http://wiki.apparmor.net/index.php/Main_Page

the stored trusted digests should be grouped based on which distribution they are located in, such solution is to tackle the issue of *code-diversity*, related to the problem that the same service may have different binaries in different distributions.

At the same time, the result for analysing the integrity report containing the IMA measures should provide as much useful information as possible, so the remote attestation result is not only Boolean, but can provide semantic information. For this reason, we define four trust/integrity levels based on the stored data in the reference database:

- **L1**: TPM and IMA functions are running correctly. The IMA measurement list is intact and it is implicitly authenticated with the signature of the TPM;
- **L2**: in addition to the **L1** achievements, the binaries executed and configurations applied are all known, i.e. found in the reference database of trusted components but they have at least one known security vulnerability;
- **L3**: in addition to the **L2** achievements, the binaries executed and configurations applied do not have any known security vulnerability, but have at least one known functional bug;
- **L4**: in addition to the **L3** achievements, the binaries executed and configurations applied do not have known security vulnerability or functional bug.

The aforementioned known vulnerability and known bug are included with the information provided by the reference database which contains the information of the updated packages, e.g., updated version, update type. For each binary executed, when its digest is found in the reference database, it will be mapped to the package version. If in the reference database there is an updated version of this package labelled as security fix, then the current version is considered as containing at least one security vulnerability. Similarly, if in the reference database there is an updated version labelled as bug fix, then the current version is considered as containing at least one functional bug. Another possible approach is to find the relevant information from a vulnerability database such as *National Vulnerability Database* (NVD)[7] and *Common Vulnerabilities and Exposures* (CVE)[8].

The distributed system administrator can choose their preferred trust level when sending the attestation requests. For example, he may use highest security requirement (i.e. L3 or L4) when attesting a server hosting security critical services (e.g., bank, stock trading) and use lower security requirement (i.e. L2) when the server is hosting security irrelevant services (e.g., gaming).

If some software fails to reach the trust level defined in the input, then a graph can be automatically generated to indicate which software is below the required

[7] https://nvd.nist.gov/

[8] https://cve.mitre.org/

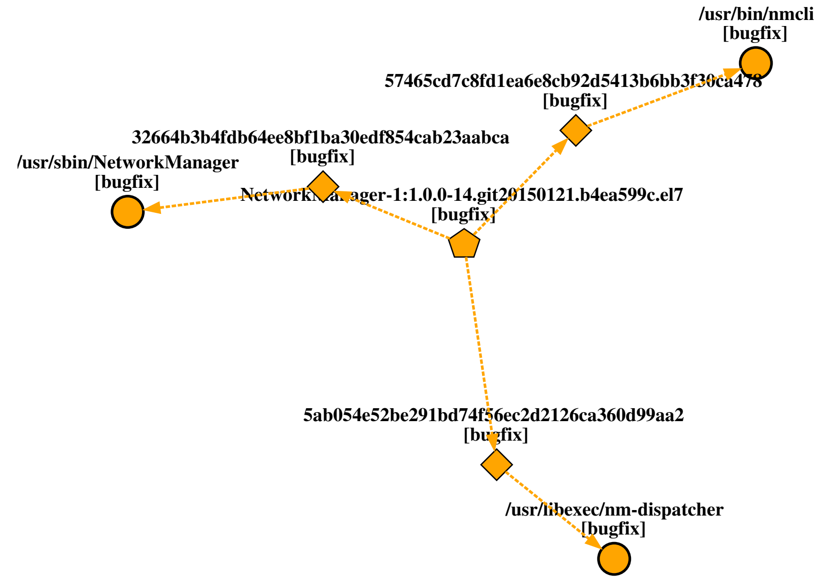

Figure 3.13. Example of L3 trust level with downgraded *NetworkManager*.

trust level, and what service is calling it as library, in order to present which service is compromised. Figure 3.13 shows an example, where the *NetworkManager-1.0.0-14.git20150121.b4ea599c.el7*, which is the one installed with CentOS7 build 1503 minimal installation by default, has a bug fixed in version *NetworkManager-1.0.0-16.git20150121.b4ea599c.el7_1*. Thus when the node is attested, the verifier can tell that this platform only achieves trust level L3 because the *NetworkManager* running in the platform has a functional bug. It first finds the IMA measures in the reference database, and map them to the packages they belong, then conclude whether the package has known vulnerability or functional bug by checking the latest package label. In this example, the squares represent the IMA measures found in the integrity report received from the attesting platform, which are also found in the reference database, their full path are represented as circles. All these three measures are mapped to the *NetworkManager* package, and there is an updated version of it which is labelled as *bugfix* in the reference database. Subsequently, the IMA verification tool labels the IMA measures as bugfix as well as their full paths, and concludes the attested platform only achieves L3.

However, it is important to say that this mechanism gives only an estimation of which executable may be affected. Since IMA does not report exactly which executable is affected by the loading of an unknown/untrusted library, we must conclude (the worst case) that all executables are affected. The IMA verification tool behaves this way and determines the integrity level of the system based on the worst update type assigned to measure list, i.e. unknown >security >bug.

In order to fully understand whether the executables and configuration files launched in the attesting platform are trusted or not, a reference database with the complete data of trusted measures is mandatory. We choose to use a highly-scalable

```
FilesToPackages = {
    file_hash: {
        distro_name-pkg_arch: {
            rpm_file_1: ``pkg_name_1'';
            rpm_file_2: ``pkg_name_2'';
            ...
            fullpath: ``path_name_full'';
        }
    }
}
```

Figure 3.14. FilesToPackages column family.

```
PackageHistory = {
    pkg_name-distro_name: {
        pkg_version-pkg_release: {
            name: ``pkg_name'';
            updatetype: ``newpackage''|``enhancement''|
                ``bugfix''|``security'';
        }
    }
}
```

Figure 3.15. PackageHistory column family.

NoSQL database, suited to manage huge amount of data with a *key-value* structure. In our database, we create two templates: `FilesToPackages` (Figure 3.14) and `PackagesHistory` (Figure 3.15).

`FilesToPackages` binds the digest of each file to its full path name and the packages in which it is contained. While `PackagesHistory` stores the update history of packages. Following the naming convention and release updates of software vendors, packages are keyed by the concatenation of their name and distribution. They contain information on the update type for each version and release number as delivered by repository maintainers.

The possible update types are: `newpackage`, which identifies new packages, `enhancement`, which means that the package contains new features, `bugfix`, which reports that non-security critical bugs have been corrected and, lastly, `security`, which indicates that security vulnerabilities found in an older version have been solved. In the last case, if some package are published but no information is available (in some rare cases), an `unknown` update type is defined for them. These update types are the key parameter to define the trust levels described above.

3.6 Application of Remote Attestation Framework in Network Policy Validation Scenario

In this section, we show one example application of the proposed remote attestation framework in a network policy validator. The remote attestation framework is used to ensure the integrity of the whole validator and the genuineness of the validator's results. The purpose of this section is to show the flexibility and extensibility of the remote attestation framework in various scenarios.

3.6.1 Motivations of A Trusted Network Policy Validator

In recent year, the adoption of server virtualisation, NFV and cloud computing techniques has brought several advantages such as service provisioning and deployment depending on user's requests. This approach enables elastic capacity to add or remove services reducing hardware cost. Although these techniques have several benefits, the management complexity of the entire system increases.

In order to tackle this issue, during the last decade, several approaches in the field of *Policy-Based Network Management* (PBNM) have been proposed to automatically configure services and applications. This typically provides automatic configuration of applications and services from scratch by defining high-level policies. Although this permits automatic provisioning of resources, by hiding refinement process details, these approaches typically do not support the management of enforced configurations, that is a crucial and complex task in production environments (e.g., data centres). It requires high accuracy (e.g., to identify and perform precise modification on configuration settings) and it must limit service downtime.

Although the automatic deployment of an application instance is a common feature of recent virtualisation platforms, on the other side, the monitoring and management of its configuration is currently not well addressed. A typical provisioning system does not periodically check the configuration settings of a instance. Often, this operation is performed manually, by an administrator, in case of failure or misbehaviour. For instance, think of updates to a firewall configuration.

This approach has at least two drawbacks. First, it is an error-prone and expensive task because it is performed by humans. For example, the administrator manually adds a new rule on a firewall that shadows another existing rule, modifying the resulting policy in an unexpected way. Second, it does not address misconfigurations that do not affect service operation. For example, an attacker could add a rule on a firewall to mirror traffic to another system. In this case, the service operates correctly but its configuration is altered and the attack succeeds. Therefore, to detect and avoid these situations, especially for complex scenarios, automatic monitoring and analysis of service configurations is mandatory.

Commonly resources are run on a specific platform with a dedicated node or as part of a cloud computing environment. However, these software are continuous exposed in the Internet. For example, an attacker could modify a software component

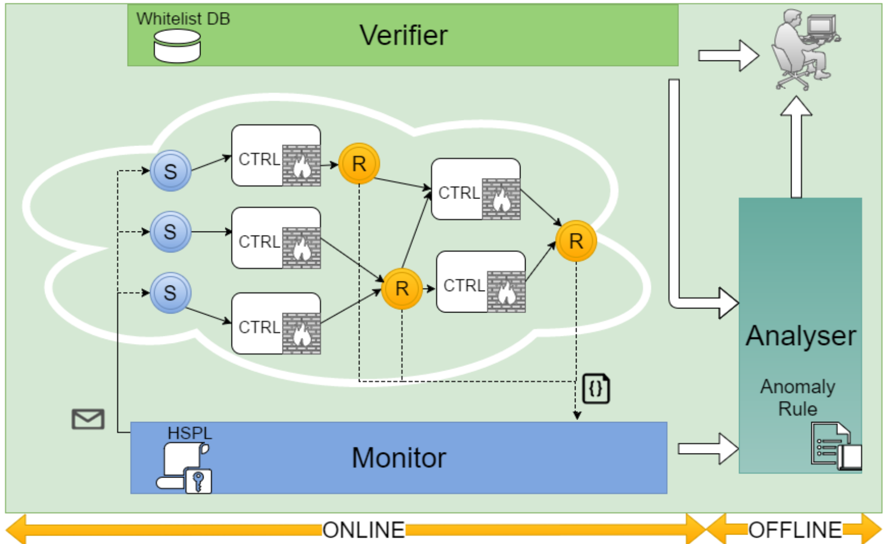

Figure 3.16. Policy validation workflow and the involved components.

to change its behaviour (e.g., to steal information from a virtual resource). Also in this case, these software changes may not affect the service operation, thus in order to evaluate the trustworthiness of these systems, the trusted computing techniques must be considered. In particular, the remote attestation technique makes it possible to verify the integrity of binaries and their configurations, although with some limitations.

3.6.2 Contribution

This work proposes a novel approach for validation and analysis of security policies, which uses trusted computing techniques to evaluate the trustworthiness of the security applications deployed into the network to enforce the policies, in order to guarantee the integrity state of the policy validator and the genuineness of the results.

3.6.3 Architecture

In this work, we present a unified approach to evaluate the enforcement of security requirements, expressed by a set of security policies. This evaluation enables the possibility to guarantee an adequate security level in the network, since the functional configuration of each deployed security control is checked to be consistent with the defined requirements. The presented approach consists of on-line policy enforcement validation by using monitoring and remote attestation. Remote attestation framework in this system is used to attest the nodes in the policy validation system, in order to assess the trustworthiness of the final result of the whole framework.

In our perception, the possible causes of failure in policy enforcement can be derived from: *(i)* errors in manual changes of the configuration rules for a security

function, that can alter the application behaviour or introduce policy anomalies; *(ii)* untrustworthiness of the security functions deployed into the network (e.g., modifying a software binary code to modify its behaviour).

In order to perform a comprehensive monitoring of the enforced security controls' [9] configurations, first of all, we need a *High Level Security Policy Language* (HSPL), useful to define security requirements by using an abstract and high-level approach. It can define the security requirements as a set of sentences close to natural language, e.g., "*do not access gambling sites*" or "*allow Internet traffic from 8:30 to 20:00 for employees*". In particular, the elements of a sentence (subject, object, etc.) can be selected by the administrator from a predefined set and implemented in an editor as different lists.

Starting from a set of HSPL statements, we designed a policy validation framework, composed of several modules, and its three-phases validation workflow. As depicted in Figure 3.16, two out of three phases are performed on-line (i.e. the involved processes work at run-time, when the network is enabled), while the third phase is performed off-line (i.e. it is triggered regardless of the network state). In particular, the framework includes:

- *Policy monitor* to periodically check the enforced HSPLs. In case of wrong enforcement, the process triggers another service to perform further analysis to detect the causes;
- *Conflict analyser*, triggered by the monitoring service to perform an exhaustive detection of anomalies among the configuration rules installed into the security functions;
- *Remote attestation verifier* to verify the trustworthiness of the network functions and the other components involved in the validation process.

The monitoring process is performed by the *monitor* module, which checks, by exploiting policy refinement techniques, if security functions are configured to enforce the HSPLs. This component relies on sender and receiver modules, deployed in the network. These nodes (depicted in Figure 3.16 respectively as "*S*" and "*R*") are transparent functions that generate, forward and monitor packets, without modifying them, and collect information for the monitor.

When the monitor detects a wrong enforced HSPL, it triggers the conflict analysis service to identify the causes of the misbehaviour. This service is implemented by another module, named *analyser*, which exploits formal techniques for detecting anomalies of configurations. By means of *First Order Logic* and *Boolean modelling*, the analyser is able to detect when an anomaly is triggered by one or more configuration rules. In case of anomaly, the administrator is notified by a report of the detected anomalies and related configuration rules.

[9] We use interchangeably the terms security controls (or controls) and security functions (or functions).

```
PCR# template-hash template filedata-hash    filename-hint
 10  fc465...848ee   ima    e4092...732e6c   /usr/sbin/iptables
 10  48327...9fed4   ima    f7655...43f45c   /etc/iptables-initial
 10  bd57b...e45b3   ima    810cf...f821d6   /usr/sbin/sshd
 10  94ea2...eff6b   ima    960f7...a9728a   /etc/ssh/sshd_config
```

Figure 3.17. IMA measures of *iptables* and *sshd* and their initial configuration files.

The workflow also includes a third module, the *verifier*, to check the integrity state of the network with remote attestation. The verifier works simultaneously with the monitor, as an on-line service. This component, thus, is in charge of verifying the trustworthiness of all the elements involved in the workflow, i.e. the network functions and the newly introduced components (and their sub-elements). If the remote attestation fails, the verifier immediately sends an alert to the administrator, specifying the causes.

In particular, in our validation workflow, the analyser is triggered only in case of wrong enforced HSPLs and positive attestation result. Therefore conflict analysis is not performed when there is at least one untrusted element. This choice has been proposed to avoid the overhead required by the analysis module and to increase the overall performance.

Remote Attestation Verifier

The remote attestation verifier used in this network policy validator is the remote attestation framework defined above. The primary goal of the verifier is to periodically attest the nodes hosting security applications in the framework and to cooperate with the conflict analyser, which analyses anomalies of the applied policies only when security controls are in trusted state. Furthermore, it can even attest every other components (e.g., senders, receivers) to ensure the whole framework is in trusted state, implying the genuineness of the test result.

The integrity state of a security control ensures that: *(i)* the system is booted with all known components in a predefined order, which implies the system is running in a trusted state without any bootkit attack; *(ii)* the services running in the application layer of each host are loaded with legitimate executables and known service configurations with reference to a well formed database, which implies the network functions are running in trusted state to deal with their inputs correctly. In case of compromised node, either because of remote attacks or wrong service configurations, the verifier alerts the administrator of this integrity state change.

For instance, in the policy validator, the security control functions (e.g., *iptables*) are essential tools to be used in the validator, thus their binary executables and initial configurations must be known by the remote attestation verifier. Moreover, *sshd*, the remote access tool and its configurations are also important, e.g., no password authentication is allowed. Thus their measures also must be present in the integrity

evidence provided by the attester and known by the remote attestation verifier (Figure 3.17).

3.7 Discussion

In this Chapter, we have shown our design of a remote attestation framework, which can attest not only the boot phase of a physical platform but also the services load-time integrity.

Regarding to the adversary model defined in Section 3.2, here we briefly discuss the countermeasures to these attacks in our framework and possible menaces.

The attacker can modify the behaviour of the distributed system by loading customised service binaries or configuration file to his favour, or he can even perform attacks in the platform by launching scripts. In our framework, we use IMA to measure and record the structural data when it is loaded into kernel memory. These measures are further extended into the PCR, which will be evaluated in the verifier.

Alternatively if the attacker can change the IMA measurement log in the platform, e.g., removing or modifying the measurement entries belonging to the scripts he launched. Because the digest of these measurement entries are extended into PCRs after being inserted into the measurement list, the integrity of these measures and the other values of the measurement list is protected. Any unauthorised modification to the measurement list will be detected by the verifier when it is reconstructing the final PCR value by simulating the extend operations of the digests of the measurement entry in the integrity report. Further, after the IMA measurement list integrity is checked, the verifier will query the measures of executables to the reference database which is generated with the help of the package update repositories for different Linux distribution, and the measures of configuration files to the whitelist, in case of unknown measure or the running software does not achieve the requirement (e.g., it is outdated), the verifier will detect the issue immediately.

The attacker can also perform a bootkit attack by replacing the authentic component with a corrupt one when the platform is booted, e.g., a cracked kernel in which IMA is disabled and no structural data is measured. However, this attack requires changing the content of the components loaded during the boot phase, and by using trusted boot and the PCR whitelist table, the modification is visible to the verifier during remote attestation process.

The attacker can also forward a remote attestation request to another genuine party to get a valid attestation result. To counter this attack, each platform in our framework needs to be pre-registered before they are deployed to host services, and their AIKs used for generating attestation signature will be certified by a PrivacyCA hosted in the verifier and each AIK certificate will stored in the verifier mapped to its host. Thus, each remote attestation result is associated to a specific node and forwarding a remote attestation request is not able to prove the "trust" integrity state of a corrupt device compromised by the attacker.

Similarly, the attacker can record the remote attestation response message transmitted between an attester and the verifier, and resend the attestation response after he compromises the attester. Since the verifier will send also a nonce to an attester along with the remote attestation request, the old response (even coming from the same attester) will be detected and vetted as an untrusted result.

For the sake of completeness, we need to mention that there are still two possible attacks. First, runtime memory corruption, the adversary may try to compromise a service by exploiting its vulnerabilities after it has been loaded into kernel memory. This issue exists because IMA can only measure structural data at their load time, but cannot give any information of the data already in kernel memory. However, memory corruption attacks can be prevented by adopting other operating system security mechanisms, like *Address Space Layout Randomisation* (ASLR) [97]. Second, since the attack model of TCG specifications does not take into account any physical attack to the attesting platform, we neither do nor consider sophisticated invasive or non-invasive hardware attacks on any platform involved in the distributed system. However, as described in Chapter 1, even when physical access is allowed, it is still very difficult to crack the TPM [13].

Chapter 4

Trusted Channel

Secure channels are widely adopted to provide secure communication between different nodes on the Internet. *Transport Layer Security* (TLS) [15] and *Internet Protocol Security* (IPsec) [16, 98] are the most used ones to provide secure communication between services and users. These security protocols protect data during its transmission and allow authentication of the channel endpoints, i.e. data confidentiality, integrity and authenticity. However, they do not provide any information of the integrity state of the platform and the software that generate or receive these data, e.g., maliciously modified software running may send fake or malicious data that infects another node. Also, attack and compromise secure channel endpoints with malware, e.g., trojan, would be easier to attack the secure channel directly. This leads to a severe problem of the current secure channel protocols, that a secure channel endpoint may be compromised and software maybe tweaked while the other peer have no information about it, thus opening a door for a wide range of attacks. The situation is depicted precisely by Prof. Gene Spafford [99] from Purdue University as:

> Using encryption on the Internet is the equivalent of arranging an armoured car to deliver credit card information from someone living in a cardboard box to someone living on a park bench.

In order to make the problem more clear, let's consider the following scenario: an employee wants to access a file stored in the company server from his home. Then he sets up a secure channel (either TLS or IPsec) to the internal network of his company and downloads this file. Everything seems fine, but the fact is that the server hosting this file has been attacked and compromised with a script injecting every file with a trojan, which will install itself automatically and silently when the file is opened. The employee, giving his full trust to the server, opens the downloaded file without being scanned by anti-virus software (if the trojan is a zero day malware, then the scanning is not helpful). As a result, his machine is infected by this trojan and the whole system is silently under the control of the attacker.

In the above example, the secure channel is deployed correctly and it works as expected, but the attack still succeeds because the integrity state information of the

server is missing. To avert such attack scenarios, integrity state of communication endpoints should be vetted before the channel is established, or at least the channel endpoints should have the chance to evaluate the "trustworthiness" of the other peer and make informed decisions.

In this Chapter we focus on the combination of TCG remote attestation technique and secure channel protocols to form *Trusted Channels*, in which the channel endpoints are attested and their identities are bound to their hardware platforms. Moreover, we present our implementation and performance evaluation in Chapter 6.

4.1 State of the Art and The Way Forward

The main feature provided by a trusted channel is the capability to provide integrity state information concerning the trustworthiness of a communication partner. With this capability, it is possible to enforce the security of data not only during transmission but also in the involved endpoints.

A naive and straightforward solution would be conveying the integrity evidence within the secure channel. Unfortunately, such simple solution is not practical and leaves space for relay attacks (where the integrity evidence is generated by a genuine platform but forwarded to other peers by a malicious platform, as an instance of *Man-in-the-Middle* attacks). Indeed, a strong linkage between the integrity evidence and the secure channel is mandatory.

The proposal of trusted channel has already been investigated in literatures by multiple work. It is often combined with SSL/TLS protocol (the most widely used secure communication channel for web servers) or IPsec protocol suite (the most widely used secure communication channels for Virtual Private Networks, VPNs). TCG also has a work group working on this issue, i.e. *Trusted Network Communication* (formerly known as *Trusted Network Connect*, TNC) [32].

One of the earliest and most influential work of linking platform integrity evidence to secure channels is [100]. This work proposed to include the SSL certificate used by the secure channel communication as part of the integrity measurements presented by the contacted endpoint during the remote attestation process. In this way, a malicious platform cannot relay the attestation request to another platform as its certificate will not be present in the measurement list of the genuine one (in the case that the genuine platform is not colluding with the malicious one). In addition, it also proposed to construct extended certificates, called *Platform Property Certificate*, which links the platform AIK to its SSL certificate, e.g., merge domain name and AIK public portion, thus simplifying the certifice creation, revocation and validation process.

Along the same vein, in [101] Gasmi et al. proposed a security architecture and mechanisms for establishing and maintaining a trusted channel. It allows the secure channel peers to cryptographically bind the measurements of their configurations, and an additional software module is introduced to force the changes in the configuration of one endpoint to be reported to the other peer when the channel is in place.

This solution relies on the binding feature of the TPM, and uses a certificate of the binding key (non-migrateable key) for authentication instead of a TLS certificate. Further this certificate is signed by the AIK of the TPM during remote attestation phase, which guarantees the binding key is indeed in this platform and no relay attack is possible. Further, the system is running in compartmental environment, where the session keys of the channel are store in the trusted computing base of the platform and will not be disclosed to other components.

However, this solution has some deficiencies. First, some features do not conform to the TLS specifications, e.g., sending attestation data within the key exchange message and including integrity data in session key computation, which may cause problem for backward compatibility and further requires re-specification. Second, the cost of certification is not considered. In [101], recertification is required every time the system is updated, which limits its feasibility in practice. Third, the trustworthiness of the platform is still evaluated based on golden binary values, thus the included components are limited, otherwise the golden values are hard to be achieved. Fourth, dynamic configuration changes are monitored by a software entity, and these changes do not reflect in the hardware-based evidence. Last but not least, each time an incoming connection is received by the server, it needs to prove its own trustworthiness on the fly, given the fact that TPM is a very slow device, the performance will degrade dramatically in a heavy loaded scenario.

To tackle the previous shortcomings, the authors in [102] proposed a design and a proof-of-concept implementation protocol, which adheres to the original TLS specifications and uses existing message extension formats to convey platform integrity information. It supports all relevant kinds of key exchange methods and provides forward secrecy of session keys. These keys are held protected by hardware, rendering their disclosure very difficult.

In [102], the trustworthiness of a platform is evaluated by comparing the platform component measurements to reference values provided by a trusted third party or certificates that can vouch for certain property of respective components. To guarantee strong isolation, the implementation is based on compartments, which consists of one or a group of software components that are logically isolated from other software components. Thanks to system compartmentation, a trusted computing base with minimal amount of software components is derived, where all security relevant operations of the common TLS protocol, like encryption, signing and handling credentials are moved to the trusted computing base. On the other hand, the protocol implementation remains in user space because there is no need to protect it. To be more specific, the attestation data is transmitted by introducing additional handshake message described in RFC-4680 [103] from the IETF networking group. This RFC defines the additional *SupplementalData* handshake message envisioned to carry additional generic data, whose format must be specified by the application that uses it, and whose delivery must be negotiated via Hello message extension. Following the same TLS extension, Yu et al. [104] proposed a trusted remote attestation model which combines the secure channel and the integrity measurement architecture.

In the case that anonymity is a requirement, in [105] the authors proposed to combine TLS with *Direct Anonymous Attestation* (DAA) to create anonymous authentication systems. If remote attestation is activated, then the system can turn into *anonymous trusted channels*, where the original attestation data transmitted in [102] changed to the values used in DAA.

Similar to [102], [106] tries to create session keys of trusted channel with the public part of the Diffie-Hellman (DH) key signed by the AIK of the TPM, ensuring the session key is bound to a specific platform. However, in [107], the authors pointed out the vulnerability of [102, 106] to a collusion attack that an untrusted server colludes with a genuine platform and proves itself trusted to others without the genuine party knowing the agreed session keys. To be more specific, the genuine platform uses its AIK to sign its PCRs and the public part of the key used by authenticating the attacker, and then the attacker sends this signature to others in order to prove its trustworthiness. In order to tackle this problem, the authors proposed *KEIA* protocol, which employs password-based authentication key exchange and TPM based attestation. Unlike previous solutions, in KEIA, the genuine platform can give a valid attestation result (trusted by others) if and only if it knows the seed to create the session key, thus the collusion attack can be foiled.

Similarly, in [108], the authors extended TLS with mutual attestation for platform integrity assurance. This solution proposed a unique identifier for each TLS session, and such identifier is included in the AIK certificate when the AIK is certified by the PrivacyCA. Thus in this solution, no host can relay other platform to generate integrity evidence for itself. However, for each TLS session, both entities need to generate a new AIK certificate and use the new AIK certificate for attestation, which makes this solution only feasible with long term TLS sessions.

Regarding to VPN, IPsec is a more popular option. In [109], the authors gave a comprehensive analysis of the problems of creating a trusted IPsec based VPN service, especially when security and complexity of the service are taken into concern. This solution uses a microkernel-based operating system and delegates all uncritical functionalities, such as network card driver and IP stack, to isolated software modules. Thus it allows to create IPsec gateways with a small trusted computing base.

Along this vein, the authors of [110] proposed an extension to the IPsec key exchange protocol, i.e. the Internet Key Exchange version 2 (IKEv2), to exchange attestation data before and after the IPsec channel is running. To be more specific, the first extension is to introduce an additional *Security Association* (SA) transformation type, called *remote attestation*, as an optional component of the IKE SA. This approach allows a peer to propose and select remote attestation as part of the negotiated set of algorithms. Then the second extension introduces a new IKE payload, called *attestation data*, it is the actual remote attestation data (i.e. TPM quote output) of the other peer in the channel. Since the derived session key from the negotiation messages is signed by the AIK of the channel endpoint, it guarantees the received integrity evidence belongs to the platform which is negotiating the SAs. However, in order to uniquely identify the channel endpoints, a public

key infrastructure is mandatory to certify the AIK used by the attester. Moreover, commercial products such as *TrustedVPN* [111] adopts similar solutions.

TCG also release several specifications on integration of remote attestation into existing secure channel protocols, which is named Trusted Network Communications (formerly known as Trusted Network Connect, TNC) [32]. The TNC architecture was originally introduced as a network access control standard with a goal of multi-vendor endpoint policy enforcement (i.e. focuses on connect). In 2009 TCG extended the specifications to systems outside of the enterprise network. And the focus of the working group changed from network access control to communication as the name suggests. In a nutshell, the main idea of TNC is as follows: there is an agent software installed in every terminal, and this software is used to collect integrity information of the terminal platform. The collected integrity information may be integrity status of all running processes and/or status of firewall or other components of concern. The TNC system will compare the information obtained from terminal with special policy made by network administrator when the terminal tries to connect the network. If the collected terminal integrity status matches with the given policy, the terminal is allowed to access network, otherwise it is denied.

In the extended specifications, part of the TNC architecture is defined as IF-T [112, 113], a standard for mapping the communications between TNC Clients and TNC Servers (TNCCS) onto existing protocols such as tunnelled Extensible Authentication Protocol (EAP) and TLS. The IF-TNCCS message, carrying the integrity measurement messages, are transported in the IF-T protocol, and its message format is defined in the IF-TNCCS specifications [114].

In summary, in order to create trusted channels, all the following major problems and requirements must be addressed: (*i*), the slowness of the TPM makes it impossible to send integrity evidence (i.e. quote data from the TPM) on the fly for multiple clients, because getting quote output from the TPM is time consuming. Hence the better option is to delegate the attestation work to a trusted party. (*ii*), the modification needs to adhere to the existing specifications, thus it can be backward compatible with tools without remote attestation. (*iii*), the evaluation of the trustworthiness should be widely spread to all connected clients. Meanwhile the attestation result should be semantically rich, instead of Boolean results. (*iv*), known attacks, e.g., relay attacks and collusion attacks, must be avoided.

4.1.1 Contribution

In this Chapter, we present our trusted channel design to combine TCG remote attestation technique and secure channel protocols. As one example, we describe how it works with IPsec protocol, but the same concept can be directly used with other secure channel protocols.

In trusted channel, the client can have the integrity state information of the remote IPsec server before connecting to it. Moreover, the credentials used by the IPsec server for authentication must be bound to a specific hardware platform, thus

an attacker cannot pretend to be a genuine IPsec server by forwarding the attestation result from another platform.

To further facilitate the management of the server component updates, we propose a single management entity (i.e. the remote attestation verifier defined in Chapter 3) to assess the integrity state of IPsec servers. Hence the information of the updates does not need to be spread in all connecting users, but only to the verifier. Furthermore, since the IPsec server is continuously exposed to the Internet even after the connection is established, in our trusted channel design, we envision that a client needs to query the verifier periodically about the latest integrity state information of the IPsec server, in order to tear down the channel immediately when the server is compromised.

4.2 Requirement Analysis

In the following section, we show the expected properties of trusted channel from both security and functional point of view.

4.2.1 Security Requirements

Secure channel properties: trusted channel should inherit the secure channel properties, such as integrity, confidentiality, authenticity and freshness of data during transmission.

Binding server identity to its hardware root of trust: the identity of trusted channel server should be bound to its integrity information and a specific hardware platform, during and after the channel establishment phase in order to prevent relay attacks and collusion attacks.

Privacy preservation: creation and maintenance of the trusted channels should adhere to the least information paradigm, i.e. each role only knows what is absolute necessary for proper integrity validation.

4.2.2 Functional Requirements

Fast deployment: the alteration to existing software and hardware environments should be minimal (if any) and additional concepts introduced should make use of and have to adhere to existing specifications.

Minimal overhead to channel establishment: the additional workload introduced to the original secure channel establishment should be negligible, i.e. the time needed to set up a trusted channel should be similar to set up a common secure channel.

Minimal overhead to channel performance: the performance of trusted channel should be similar to the original secure channel, i.e. remote attestation should not introduce any significant additional weight to both IPsec client and server.

Backward compatibility: the introduced remote attestation feature should incur no constraint with regards to the original secure channel establishment, and the secure channel establishment should preserve its original behaviour after remote attestation feature is disabled.

4.2.3 Possible Attacks

In trusted channel scenario, the attack model inherits the one from Section 3.2.3, hence we only emphasise the differences. The attacker may be a malicious third party, a trusted channel user or even the administrator of the IPsec server. He can eavesdrop and manipulate the communication between the IPsec client and server or control either one or both (e.g., creating a shadow server). Moreover, he can eavesdrop and manipulate the communications between the IPsec clients and the remote attestation verifier, but he cannot compromise the verifier, which in our design is always trusted and behaves correctly.

4.3 Trusted Channel Architecture

In this section we describe the general architecture of our trusted channel design, which follows a client-server model. Following this model, the involved roles are IPsec client(s), an IPsec server and a remote attestation verifier. The trusted channel is established between the clients and the server, and the verifier is running as a trusted third party that interacts with both entities (Figure 4.1).

Client

The IPsec client runs in a user terminal that directly interacts with an end user who intends to create an IPsec channel to a remote server whose integrity state is known.

In a heterogeneous scenario where every node has similar structure and computing power, both the IPsec client and server can be attested. However, in our scenario, we only attest the IPsec server but not the clients for the following two reasons: (*i*), the user may not wish to expose the configuration information of his device because of privacy issues. (*ii*), the user terminal may not have the ability (e.g., lack of the root of trust) to provide authentic integrity evidence about itself.

Since the IPsec client does not need to be attested, there is no specific requirement for the client machine, nor does it needs a hardware root of trust or a specific operating environment.

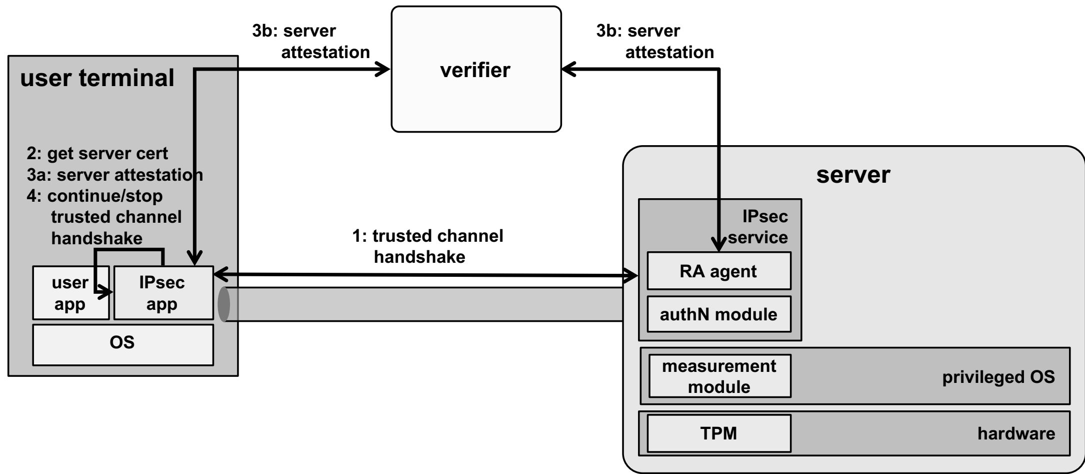

Figure 4.1. General architecture of trusted channel.

Server

The server is the other endpoint of trusted channel, and it hosts the IPsec service and allows end users to access certain services, thus in our design, this node is the one that needs to be attested. For instance it can be the front-end (i.e. gateway) of an enterprise's internal network, which authenticates the employees before allowing them to access the internal network or it may adopt security services like firewall or bandwidth control to the devices running in the internal network.

The requirement of the IPsec server is almost the same as the attester defined in Chapter 3, thus it needs a hardware root of trust (i.e. the TPM) and a measurement architecture (i.e. IMA) activated in the privileged operating system which runs on bare metal hardware, allowing the measures of executables and configuration files to be extended into the PCR in the TPM. Apart from the original requirements, in a trusted channel server, IPsec service and its authentication module are mandatory. These two parts should also be measured and assessed during the remote attestation process in the same way as other software running in the server.

Verifier

The verifier is the central point of our proposal, since it is in charge of periodically attesting the IPsec server, receiving remote attestation requests from clients and sending back the latest attestation result. It is the same verifier presented in Chapter 3, but it is extended in order to have the ability to bind the channel endpoint identity to its attestation result and a specific physical platform, thus an attacker cannot launch relay attacks or collusion attacks, that a compromised server is proven to be genuine by using the attestation result of another platform.

4.4 Creating Trusted Channel

The steps required for the establishment of a trusted channel are depicted in Figure 4.1. Initially the IPsec client begins the trusted channel handshake with its selected IPsec server (step 1) and gets the certificate of the latter (step 2); the received certificate will be used for verifying the binding between the attestation result and the contacted endpoint identity. Then, the IPsec client performs the server attestation itself (step 3a) or with the help of a trusted verifier (step 3b). If the result of the attestation is positive, the IPsec client continues the handshake and establishes the trusted channel. Otherwise, it closes the connection immediately (step 4). To be more specific, the steps to attest the IPsec server is illustrated in Figure 4.2 with detailed operations for each role.

Because of the variety of user terminals, we envision two possible approaches to attest the remote server: (i), the user terminal has enough computing power and the user has enough knowledge that the user terminal can attest the remote IPsec server by itself, thus the remote server can transfer its integrity report back to the client directly during channel authentication phase (similar to the case of TNC). (ii), the user terminal has less computing power required to attest the remote server by itself (e.g., a smart mobile device), thus it has to trust and offload the remote attestation work to a third party (i.e. verifier).

However, in trusted channel working scenario, the additional workload introduced by remote attestation in both the IPsec clients and the server should be minimised, otherwise it is less feasible from the practical point of view. For this reason, we chose the second approach that the clients offload the work of attesting the IPsec server to a trusted third party (e.g., a verifier set up by ISP or enterprise). But we still show the steps of the first approach in Figure 4.1 and Figure 4.2 with different suffix (a for the first approach and b for the second approach).

4.4.1 Extension to IPsec Authentication

IPsec is a protocol suite to secure *Internet Protocol* (IP) communications, and it works by authenticating and encrypting every IP packet of a communication session. Currently only IPsec protects all application traffic over an IP network, i.e. automatically secure applications at the IP layer. It supports network-level peer authentication, data-origin authentication, data integrity, data confidentiality (encryption), and replay protection [16, 98].

IPsec uses the concept of *Security Association* (SA) as the basis for building security functions into IP. An SA is simply a bundle of algorithms and parameters (e.g., keys) that is being used to encrypt and authenticate a particular flow in one direction. Therefore, in normal bi-directional traffic, the flows are secured by a pair of SAs. The framework for establishing SAs is provided by the *Internet Security Association and Key Management Protocol* (ISAKMP) [115], in order to negotiate, set-up, modify and delete a SA. Protocols such as *Internet Key Exchange* [116] and *Kerberized Internet Negotiation of Keys* [117] provide authenticated key exchanges.

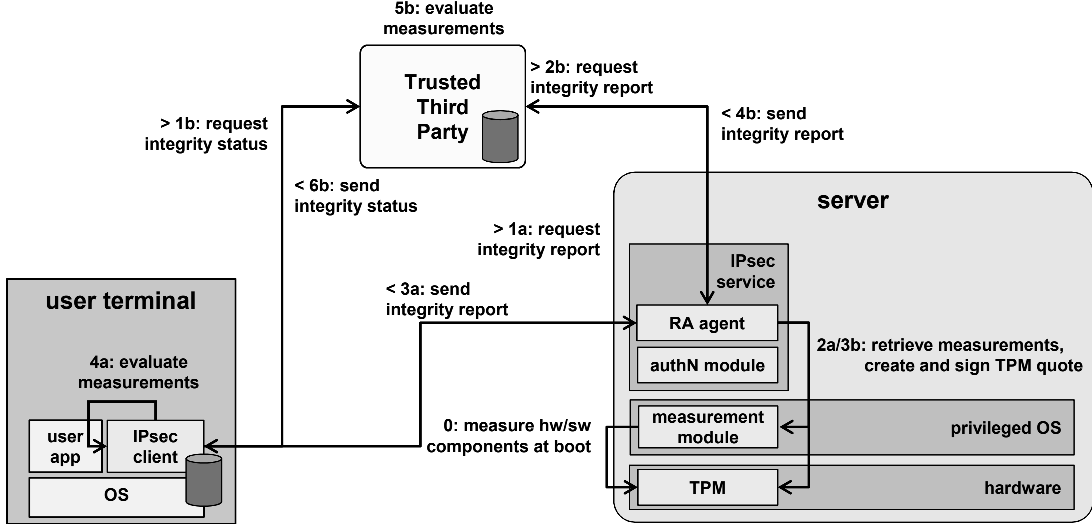

Figure 4.2. Trusted channel server attestation steps with and without a trusted third party.

In order to support additional remote attestation feature before the secure channel is created, the authentication phase needs to be extended. We chose to work with *Internet Key Exchange v2* (IKEv2) [118, 119], which creates a SA to protect ISAKMP exchanges. This SA is used to protect the negotiation of the SAs needed by IPsec traffic.

As shown in Figure 4.3, the step to assess the integrity state of the IPsec server is after `IKE_AUTH` response message and before `CREATE_CHILD_SA` message, i.e. before the client actually installs the IPsec SAs for communication.

Initial Phases in IKEv2 Exchange

IKEv2, compared to its predecessor IKEv1, has only two initial phases of negotiation: *IKE_SA_INIT* and *IKE_AUTH* exchanges.

IKE_SA_INIT is the initial exchange in which the peers establish a secure channel. This phase generates a shared secret key to encrypt further IKE communications. After the initial exchange is completed, all further exchanges are encrypted. As a matter of fact, the responder is computationally expensive (because of Diffie-Hellman key exchange algorithm) to process the IKE_SA_INIT packet and cannot leave to process the first packet, it leaves the protocol open to a *Denial of Service* (DoS) attack from spoofed addresses. In order to protect from this kind of attacks, IKEv2 has an optional exchange within IKE_SA_INIT to prevent against spoofing attacks. If a certain threshold of incomplete sessions is reached, the responder does not process the packet further, but instead sends a response to the initiator with a cookie. For the session to continue, the initiator must resend the IKE_SA_INIT packet and include the cookie it received.

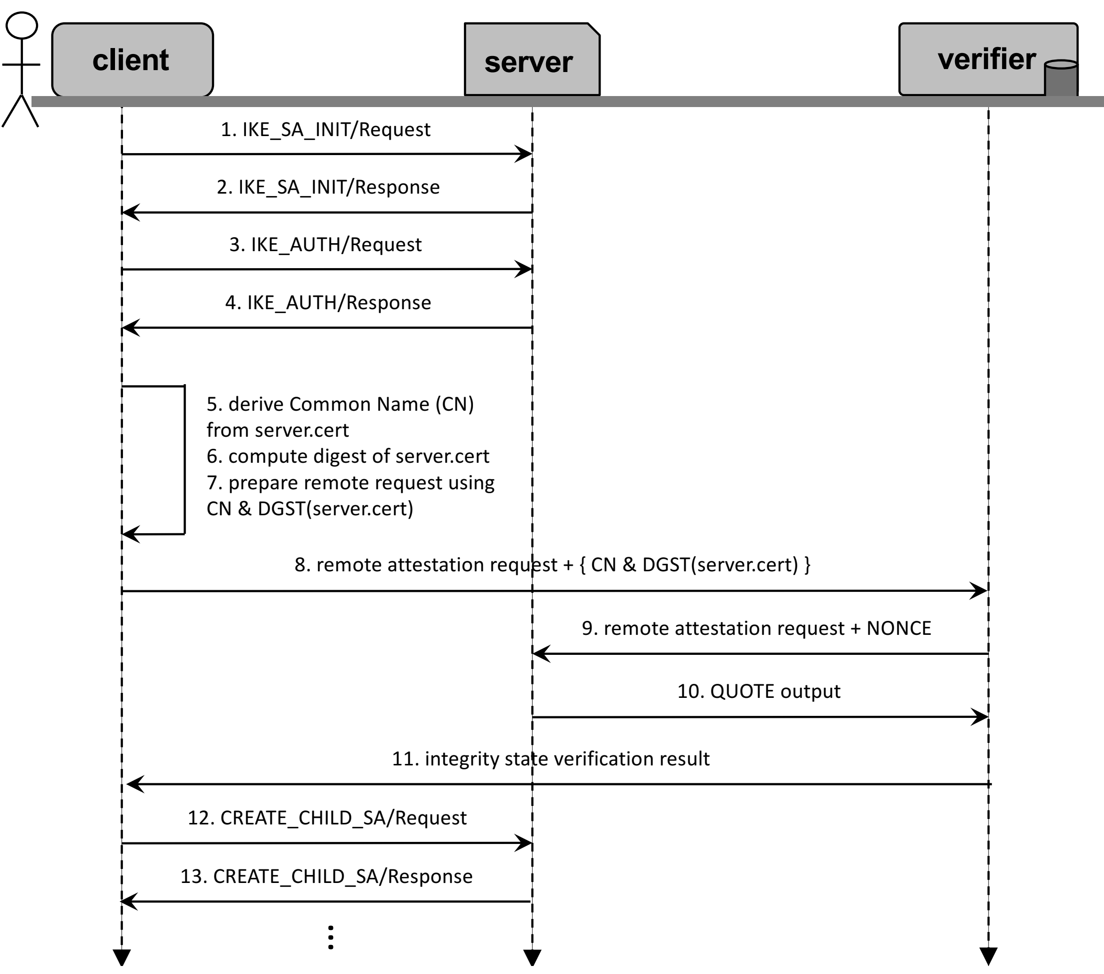

Figure 4.3. Extension to IKEv2 protocol with RSA-sig authentication.

After the IKE_SA_INIT exchange is complete, a pair of unidirectional ISAKMP SAs is generated; however, the remote peers have not been authenticated. The IKE_AUTH exchange is performed to authenticate the other peer. This exchange contains the ISAKMP ID along with an authentication payload. The contents of the authentication payload is dependent on the method of authentication, which can be Pre-Shared Key (PSK) [120], RSA certificates (RSA-SIG) [121], Elliptic Curve Digital Signature Algorithm certificates (ECDSA-SIG) [122], or Extensible Authentication Protocol (EAP) [123], etc. In addition to the authentication payloads, the exchange includes the SA and Traffic Selector payloads that describe the IPsec SA to be created (steps 1-4 in Figure 4.3).

In our design, we can use all supported authentication methods, and the only requirement is that there need to be structural data that can be measured by IMA in the IPsec server when the data is loaded into kernel memory. This is a strict requirement, otherwise the credentials of the channel endpoint cannot be bound to its hardware root of trust, which makes relay and collusion attacks possible. For simplicity, we use RSA signature authentication method in further description, which

uses a unique-identity digital certificate issued by a certificate authority. During authentication, each device digitally signs a set of fresh data and sends it to the other party along with its certificate. If the public key stored in the certificate matches the signature then the remote party is authenticated, otherwise the authentication fails. Specifically in IKE protocol, each initiator and responder of an IKE session using RSA signatures sends its own ID value, its identity digital certificate, and an RSA signature value consisting of a variety of IKE values, all encrypted by the negotiated IKE encryption method (e.g., DES or 3DES).

At this point, the IPsec client receives the certificate used by the server for authenticating itself. Afterwards, it derives the Common Name from the server certificate and computes the digest of the received certificate, in order to prepare a remote attestation request of the server. The request includes the server's identity (i.e. the common name in the certificate), the digest of the certificate, and required analysis types and their arguments. In the end, the IPsec client continues further IKEv2 exchanges if the received integrity evaluation result of the server is positive.

4.4.2 Extension to Remote Attestation Verifier

Thanks to the extended integrity reports (Section 3.5.3) and the analysis type customisation feature of the remote attestation verifier (Section 3.5.4), we introduce a new analysis type that allows the sender of remote attestation requests (i.e. the client in our schema) to define parameters in their requests which must be present in the attester's measurement list. In the trusted channel scenario, this new analysis type is called *check-cert*.

This feature is critical in the case that the remote attestation request sender knows that the attester must have applied some predefined custom configurations, thus the digests of these configuration files should be present in the integrity report generated by the attester (e.g., the public key certificate used to authenticate the IPsec server). If the digests of these custom configuration files are missing, then the attester should be evaluated as untrusted.

For instance, in the trusted channel scenario, if the public key certificate of the IPsec server is missing in the integrity report of the attester, then there may be an attack where the client is connecting to a compromised (shadow) server but the verifier is attesting the legitimate one. In this case, even though the client receives the positive attestation result from the verifier, his connection is still not trusted (Figure 4.4).

This check-cert feature of the remote attestation verifier to check if certain configuration of the attester is present mainly brings two benefits and one problem.

The first benefit is that, it binds custom configurations to a specific physical node. As an example, in trusted channel scenario, the certificate used by IPsec server to authenticate itself is unique and predefined, and the AIK used to sign the integrity report is also unique, thus when the verifier receives the digest of server certificate from the IPsec client and the integrity report from the server signed with

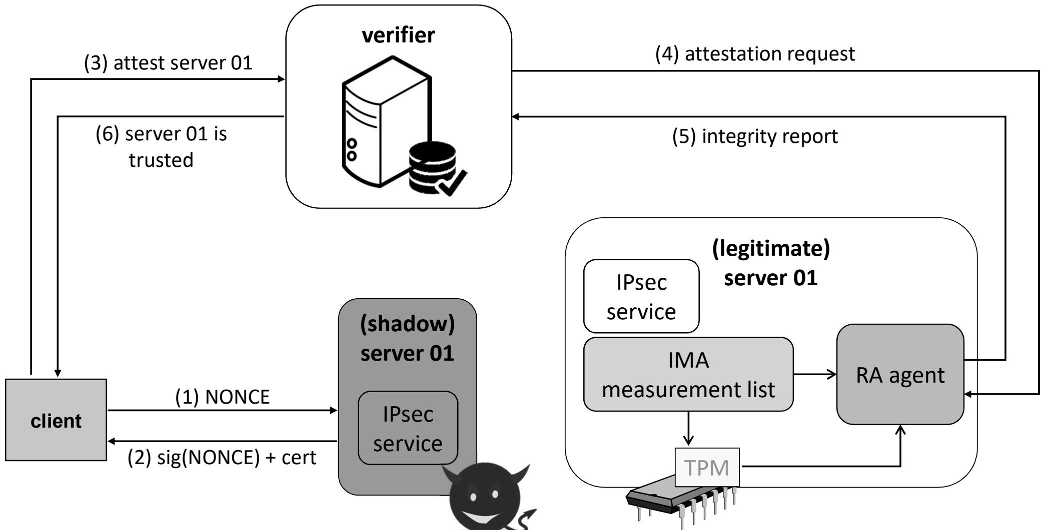

Figure 4.4. Shadow server attack without binding server certificate to its hardware root of trust.

the AIK containing the digest of its certificate, then the verifier knows that the client is trying to connect to this specific IPsec server (based on server certificate) which is running on this specific physical machine (based on AIK certificate). If an attacker successfully compromises the IPsec server, then its integrity state change will be detected with its integrity report. On the contrary if the attacker pretend to be a legitimate IPsec server and authenticates himself with another certificate (even if its common name is correct and issued by the trusted certificate authority), then the digest of the fake certificate will not be present in the integrity report signed with the AIK from the legitimate IPsec server. The scenario is similar in other distributed systems, such as 802.1X authentication system [124], where the authenticator asks the verifier to attest the authentication server to check whether (*i*) the authentication server (hosting RADIUS [125] or Diameter [126]) is compromised due to remote attacks and (*ii*) the authentication service is running with the correct configurations, before sending supplicant's credential to the authentication server for validation.

The second benefit is that, it prevents leaking any configuration information to random entities, because the entities querying about IPsec server's integrity state must have some specific information of remote servers (e.g., the digest of the IPsec server certificate), otherwise they will only receive "untrusted" as the integrity evaluation result.

On the other hand, the problem is caused because these custom configuration files (e.g., IPsec server certificate) are unknown to the verifier, thus the latter will evaluate the attester as untrusted without a remote attestation request containing the digests of these files. We envision there are three possible approaches to solve this issue.

In the first approach, the client sends his remote attestation requests along with

the digests of these configuration files (i.e. digest of the IPsec server certificate) to the verifier, then the latter uses the received digests to evaluate the attester's fresh integrity report and gives the attestation result on the fly. However, this approach is time consuming because the verifier needs to launch a fresh attestation process including all the needed steps (e.g., getting quote from TPM, generating integrity report), which needs at least several seconds before the result can be given back (more details in Chapter 6). Moreover, this approach has little scalability, for instance, when tens of clients connect to the same server at the same time would create a long queue before any one can successfully connect.

In the second approach, the system administrator decides which file(s) must be present in the attester, and registers their digests to the verifier as part of the attester's registration process. In this case, the verifier has the knowledge of the digests of these custom configuration files and can create the correct remote attestation requests by itself. Hence it can periodically attest the IPsec server and give the latest attestation result to the clients immediately when it receives a query from the client. However, in this approach the client cannot define his preferred analysis types in the requests, but can only accept the attestation results of the analysis type defined by the verifier. For instance, the client might use the analysis type COMPARE_REPORT, but the verifier only has the attestation result of VALIDATE_PCRs and load-time analysis types.

In the third approach, the verifier issues remote attestation requests to the attester without considering any custom configurations, and it stores the unknown digests in its file system. Later, when the verifier receives a remote attestation request containing the digests of custom configuration files from the client, it compares the digests in the request with the unknown digests it previously stored. If all the stored digests match the ones in the request, then the verifier gives "trusted" evaluation result to the client, otherwise if any of the digest mismatches, the result will be "untrusted". Although this approach requires less time than the first one, it is still slower than the second approach because the verifier needs to perform additional comparisons before giving the result, which means the scalability issue persists. Moreover, in this approach, the client neither can define his preferred analysis type freely in his requests.

The choice of these three approaches is relevant to the use scenario. For instance in the trusted channel scenario, the time needed to receive the attestation result is critical, asking a user to wait several seconds before he can establish the IPsec channel to the remote server is infeasible from the user experience point of view. Moreover, the only "must-have" file in an IPsec server should be its public key certificate that is used to authenticate itself. Any other custom configuration files, that may be also critical, but not strictly relevant in the trusted channel scenario. Under these concern, in our trusted channel design, we ask the IPsec server to register the digest of its public key certificate when it enrols to the verifier, then the verifier uses this digest to generate correct remote attestation requests and periodically attests the IPsec server. When it receives attestation request from a client with the correct digest value in the request, it immediately gives back the latest attestation result.

4.5 Discussion

In this section, we carry out a brief evaluation of our proposal based on the requirements defined in Section 4.2.

First of all, IPsec provides secure channel properties during data transmission, while the integrity state of the IPsec server (including both platform and service integrity) is offered by remote attestation process, which is provided by a trusted verifier. The positive attestation result implies the genuineness and security properties of the data within IPsec server. If the remote attestation framework is properly deployed in trusted channel, the IPsec server and its services should not be able to be tampered without being noticed by the verifier. And because of the periodic remote attestation requests issued by the client, the client can drop the connection immediately once the IPsec server is compromised. This solution is even able to provide protection against a malicious administrator, because the measures of the platform components, service executables and their configuration files cannot be faked without being detected by the remote attestation verifier.

Regarding to relay attacks, the attacker cannot forward the remote attestation request to other genuine platform because the credential used by secure channel endpoint to authenticate itself cannot be present in another platform, i.e. the authentication certificate of one platform cannot be present in another platform without being considered as unknown digests. Similarly, collusion attacks can also be avoided, because in the registration phase each platform gives its AIK certificate to the verifier, thus no platform can generate integrity report for another platform.

However, we envision a constraint for the attacker's behaviour, since he cannot steal the credentials of the IPsec server used for secure channel authentication. Otherwise he can pretend to be another IPsec server by using the stolen credentials even if the IPsec server certificate is present in the integrity report generated by the legitimate IPsec server.

On the other hand, protecting secure channel credentials is a challenging task from the practical point of view. Theoretically, it is possible to encrypt the credentials (but needs manual insertion of encryption password when the credentials are used), use ephemeral credential that is valid only for a short period of time (but needs to frequently regenerate credentials), or if possible, an external hardware, such as hardware security module or the TPM itself, to store the credentials (but the performance is slower).

So we assume the IPsec server can prevent its credentials to be stolen, and under this assumption, the public key certificate of the IPsec server can be indeed bound to the TPM attached to the physical platform. When a client sends a remote attestation request with the digest of the public key certificate it received in the authentication phase, the verifier knows the client is indeed contacting with a specific IPsec server. And the public key certificate is bound to a specific hardware device if the digest of the certificate is present in its integrity report.

With regards to the communication between the IPsec client and the remote

attestation verifier, a traditional SSL/TLS channel can be adopted to ensure the (*i*) the identity of the verifier and (*ii*) the attestation result released by the verifier is authentic.

To make a fast and widespread deployment of our approach, we extended the remote attestation framework presented in Chapter 3 to attest the IPsec server in our trusted channel. Even though the same approach can be used in all IPsec implementations, e.g., OpenVPN[1] and Openswan[2], in this thesis, we used the widely deployed IPsec implementation, *strongSwan* [127], and started our implementation based on it. Because the architecture of the strongSwan implementation is pluggable, the efforts that we had to be put into the implementation of our solution was moderate. Meanwhile, the development community of strongSwan is active, thus our questions can be resolved in a timely fashion. Additionally, we adhere to the mechanisms and concepts that have already been defined in existing specifications, in a way that original IPsec secure channel client can communicate with IPsec server without any modification if remote attestation feature is not required. The only difference is that the remote attestation plugin should be enabled when attestation of remote server is required. On the other hand, the IPsec service in the server side does not need any modification, and the platform where the service is deployed only needs a hardware root of trust, a measurement architecture and a remote attestation agent from our framework.

[1] https://github.com/OpenVPN/

[2] https://github.com/xelerance/openswan/

Chapter 5

Trusted Network

Software Defined Networking (SDN) [1, 2] and *Network Functions Virtualisation* (NFV) [3, 4] are modern techniques to implement networking infrastructures. Similar to cloud computing infrastructure, SDN and NFV exploit distributed system model and virtualisation technology to improve the feasibility, flexibility and elasticity of creating and deploying networks, and significantly reduce cost. Besides switching packets, they can also be used to provide other more advanced functionalities, e.g., security applications at the network edges.

The new paradigm of SDN/NFV brings the aforementioned benefits but also introduces a security gap in the network, because traditional hardware-based network functions are softwarised and SDN/NFV architectures heavily rely on specific software modules executed on highly distributed nodes, and those modules may act differently from their expected behaviour due to various errors and remote attacks. The use of remote attestation in network environments to ensure network node integrity is quite new and recent, currently it is raising interest not only in research community but also in the industry, as demonstrated by its consideration in the ETSI NFV standardisation efforts [128].

In this Chapter, we first discuss the techniques useful to evaluate the software integrity of a SDN/NFV node, especially the software running in virtual instances, and hence its trustworthiness to execute the desired applications. Further we present our solution to adopt hardware-based remote attestation of softwarised network nodes along with software modules running in virtual instances, making it feasible to create trusted softwarised networks. The actual implementation and performance evaluation of this solution are presented in Chapter 6.

5.1 Softwarised Network

Modern telecommunication networks contain an ever increasing variety of proprietary hardware. The equipment has evolved from a simple device that conveyed voice over modest distances (telephony) to complex media that transfer voice and

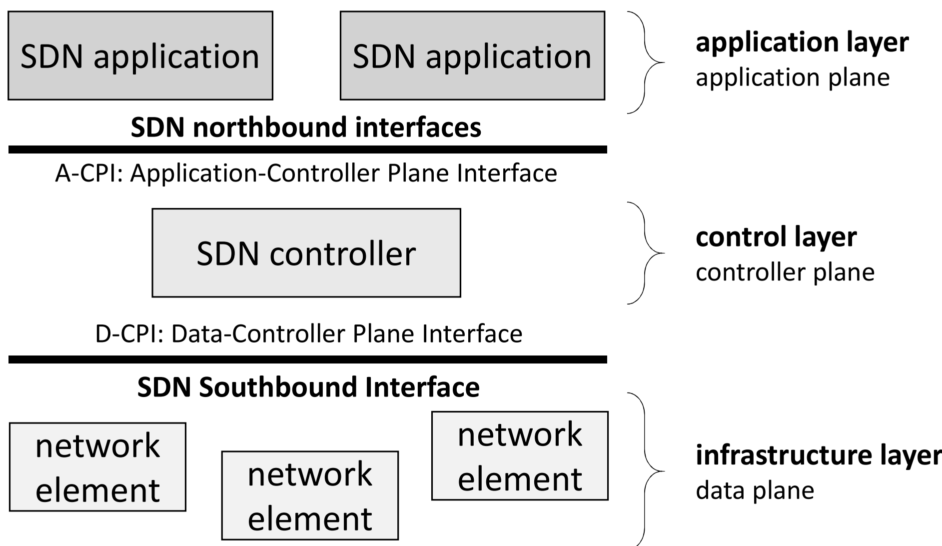

Figure 5.1. SDN architecture.

data worldwide. The launch of new services often demands network reconfiguration and on-site installation of new equipment that in turn requires additional space, power supply and trained maintenance staff. Hard-wired network with single functions devices are tedious to maintain, hard to evolve, and prevent service providers from offering dynamic services. However, the network innovation cycles currently accelerate and require greater flexibility and dynamism than hardware-based appliances allow.

In the same way as applications are supported by dynamically configurable and fully automated cloud environments, *Virtualised Network Functions* (VNF) allow networks to be agile and capable of responding automatically and quickly to the needs of the traffic and services running over it. The key enabling technologies for this vision include SDN and NFV. They are complementary but increasingly co-dependent for the benefits of softwarised networking to be fully achieved.

SDN is a particular approach to provide virtualised traffic routing and unified network flow management across hardware and software-based networking components. It is meant to address the fact that traditional network architecture is not suited any more in managing the current dynamic computing and scalable storage needs of enterprise data centres, campuses and carrier environments. The principal design of SDN is to decouple the existing control and data planes by moving the control part away from all network elements to a centralised node in the network, known as the SDN controller (Figure 5.1). The controller can then be programmable and manipulated by an upper layer called application plane, where the SDN applications give instructions to the SDN controller based on, e.g., policy defined by network administrator, in order to tell the latter how to distribute the flow rules. Thus, the underlying infrastructure can be abstracted from network applications and services.

On the other hand, NFV aims to transform the traditional networks by evolving

standard virtualisation technology to consolidate many network functions into virtual instances that can be run on industrial standard servers, switches and storages. These components could be located in data centres, network nodes and even end user premises, thus they can be moved to or instantiated in various locations in the network as required without the need for installing new equipment. NFV typically exploits SDN to create custom overlay networks connecting the various network functions and in turn SDN can use NFV to host its controllers and applications. An NFV service deployment requires the onboarding, activation and start-up of a set of virtualised elements that will be run on a uniform infrastructure supporting the virtualised execution environment, which is usually called *NFV Infrastructure* (NFVI).

In the softwarised networks, lightweight virtualisation technology is critical, especially in an NFV environment, as the network nodes usually have limited computational resources and VNF tends to have one to one relationship to the virtual instance. For this reason, recently a lot of interest has been raised by operating system level virtualisation techniques, as they incur much lower performance compared to full virtualisation techniques (i.e. hypervisor-based virtualisation) with smaller and more agile execution environments called *virtual containers*.

5.2 Security and Trust in Softwarised Networks

Without any doubt, the SDN/NFV infrastructure paradigm makes managing and programming the network much easier than hardware-based legacy networks. Especially, data handling rules in SDN are implemented as software modules instead of embedding them in the hardware, thus fine-grained network flows can be flexibly adapted within seconds and without involving any changes to the physical topology, which enables agile provisioning and removal of network services. However, SDN also brings its own set of problems, especially with regards to the security aspects, since the programmable nature introduces new risks that remote attacks could affect the behaviour of the network itself by modifying the loaded software module in network nodes (i.e. network integrity property).

Security of SDN is a hot topic nowadays. There are a lot of work surveying the security aspects of SDN in the last two years. Ahmad et al. [129] presented the evolution of SDN starting from its predecessor, i.e. active networking, then they discussed security challenges of different SDN planes (i.e. application, control and data planes), and categorised these threats and countermeasures based on their targets. For instance, the most prominent attack in the application plane would be fraudulent flow rules insertion, which is caused by malicious or compromised applications. While the most prominent attacks to the control plane is DoS, because of its visibility nature, centralised intelligence and limited resource. In data plane, fraudulent flow rules and flooding attacks are more relevant because the data plane is dumb and flow tables can only store a finite number of flow rules. While Alsmadi et al. [130] discussed the security threats to SDN according to their effects, and

listed possible security controls to counter, mitigate and recover from these threats. Moreover, they presented the current progress of different countermeasures.

Similarly, Shu et al. [131] discussed the security features of SDN as a whole and then analysed the security threats and countermeasures in detail from three aspects, based on which part of the SDN paradigm they target, i.e. the data forwarding layer, the control layer and the application layer. Li et al. [132] gave particular attention to OpenFlow-based SDN environment [133] and presented an up-to-date view to existing security challenges and countermeasures in the literature, i.e. the authors focused on the security issues of the lower two layers (i.e. control and data planes). Moreover, they mentioned the problem of software integrity, i.e. whether it will deal with the input correctly, regarding to network trust and software attestation. More recently, Dacier et al. [134] highlighted the pros and the cons of SDN regarding to the network security perspective. In particular they discussed whether SDN approach enriches the security of softwarised networks or sabotages it.

Apart from the generic security issues, the SDN controller has even more threats since it is centralised and has direct connections to each of the switches. The controller has an indirect view of the network topology based on the rules it has programmed to the switches. The latency incurred when updating the forwarding rules implies a synchronisation problem between the controller's view of the network and the actual network configuration. Existing work, such as [135] and [136] proposed incomplete solutions to this problem.

With regards to the software module integrity of SDN switches, Jacquin et al. [137] used trusted computing techniques to provide a strong hardware-based platform identity and dynamically monitor the low level configurations used to route virtual LANs. With the help of a newly introduced software component running in SDN switches, which collects the software measures and received flow rules and extend them into the hardware TPM, this architecture provides hardware-based evidence of the switches integrity state. This work is the first to use trusted computing in SDN switches, which provides a mechanism to check the network posture, bridging the gap between the remote attestation and virtual networking.

Similar to SDN, security of NFV is also a popular topic these days. In [138], Firoozjaei et al. analysed potential security threats in NFV environments and categorised them from the network and virtualisation points of view. To be more specific, the authors discussed infrastructure-target, VNF-target and user-target threats. Then they proposed reasonable countermeasures to cope with these dangers. Most importantly, this survey recognises the importance of software integrity in NFV environment, and promotes trusted computing (remote attestation) as the most promising solution to this issue. However, because the hardware TPM chip is improper to perform bulk data cryptographic operations due to its slow performance. The authors suggested to use firmware-based TPM [79, 80] platforms and TPM emulators [39, 78] because they may provide much better performance.

Ravidas et al. [139] provided an overview of the challenges to incorporate trust in NFV environment and proposed a basic and incomplete solution, that the image of the virtual instance hosting VNF is signed and the signature is checked before the

VNF is deployed. Further, they designed another approach to bind the VNF image to a NFVI based on the PCR values in the NFVI's TPM. However, no internal of the virtual instance is going to be probed and this solution does not check the integrity of the software VNFs.

The only issue is that the authors of [138, 139] have not discussed the problem of using remote attestation in virtualisation environment. The direct application of remote attestation to NFV environments is not possible without several enhancements. First, remote attestation is not virtualisation-friendly. As mentioned in Chapter 2, the major issue is that the hardware TPM chip cannot provide enough resources for multiple virtual instances running on a single platform, especially the PCRs. Second, NFV environments typically have a plethora of network nodes, management, scalability and performance must be carefully addressed .

These problems mostly concern traditional hypervisor-based virtualisation environment, even if the most widespread solution vTPM [39] is adopted. The vTPM solution tries to allow unmodified operating system and services running in virtual machines to use trusted computing techniques, not only remote attestation but also sealing, with the help of a virtual TPM. In general the threats to a virtual machine are essentially the same as for a physical node, plus those coming from its virtual machine manager (VMM) or hypervisor. A VM must completely trust its VMM, because the VM cannot defend itself in any way against VMM attacks, e.g., a VMM can temper its VM's memory without being detected by the latter. Thus the integrity of the VMM itself must be established in the first place, and this is part of the basic attestation which addresses those components directly executed on the hardware. Depending on the hypervisor type, it can be attested either as a component loaded at boot time (type I hypervisor) or as a service running in the host system (type II hypervisor). However this distinction is not influent on the result: the VMM's integrity is attested as part of the base operating environment of the node.

The actual implementation of vTPM has been mainstreamed in the Xen hypervisor, which is different from the proposal in [39]. For instance, the vTPM manager and multiple vTPM instances are running inside Xen's stub domains[1] instead of the domain 0 (dom0), which runs the privilege guest in the system to manage other guest domains (Figure 5.2).

This implementation architecture brings severe problems. First of all, the vTPM manager stub domain has its own TPM driver, which directly communicates with the physical TPM. This can cause a race condition if dom0 has its own TPM driver activated at the same time. Hence the privileged domain dom0 cannot use the TPM functionalities, neither using the physical TPM (which is not accessible due to conflict with the vTPM manager stub domain) nor via the vTPM (which is not yet available when dom0 is started). A more elegant and secure solution would be

[1]Stub domains are the same as other guest domains, but are dedicated for special purposes such as disaggregated device drivers

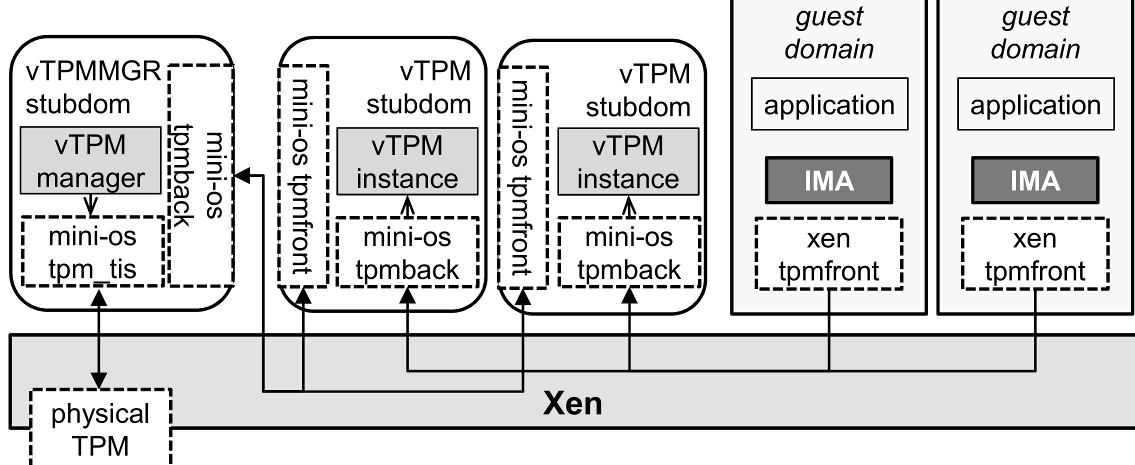

Figure 5.2. vTPM implementation architecture in the Xen hypervisor.

to set a different *locality*[2] of the physical TPM for dom0 and the vTPM manager stub domain. However, in this case, another practical problem appears: the vTPM manager fails to start if using a locality different from the default one [140]. This is a bug known to vTPM developers and, at the time of writing, no solution is available to resolve this problem. Therefore dom0 cannot be attested when using the vTPM feature, which is unacceptable since dom0 is at the heart of Xen.

The second problem is related to the vTPM client-side driver, which is compiled as a module in the guest VMs' kernel. As explained in [141], the problem is that IMA is loaded with the kernel and when it starts taking measurements, the client-side driver has not been loaded yet and the vTPM instance is not available, with the consequence that the IMA measures are not extended to any PCR. This problem also affects all the components loaded during the VM's boot process. For this reason, guest domains do not have access to a virtual root of trust with high integrity guarantee and this reduces the benefits of the remote attestation feature.

The third problem is that, without an entity guaranteeing the integrity state of each vTPM instance stub domain, it is possible that a stub domain is compromised (e.g., an attacker changes the vTPM instance behaviour to not record its malicious actions) and this fact goes undetected. This problem poses a serious question about the integrity guarantee provided by the vTPM solution.

Last but not least, the performance drop would increase with regard to the number of attested VMs, since a substantial portion of computing power would be used by each VM to prepare its own integrity report, especially if the remote attestation framework presented in Chapter 3 is used. In this case each VM needs

[2]Locality is an assertion to the TPM that a command is associated to a particular component. The purpose of setting different localities for dom0 and the vTPM manager is to permit them to use different PCRs and avoid conflicts.

to insert the IMA measurement list into its own integrity report even if there may be a lot of duplicates.

In conclusion, although vTPM is an attractive concept and it is currently promoted by the TCG as the solution to virtualisation environments [142], it has both performance and practical issues that severely limit its application. For this reason, in a VMM-based virtualised environment, normally integrity verification is available only for the host system and VMM, while the integrity guarantee for the VMs is limited to checking the image being loaded in a VM via a digital signature and/or download from a trusted repository. As a consequence, run-time integrity verification for the applications executed in the VMs is not provided. A detailed analysis and comparison of vTPM related work can be found in [143].

Along the same vein, another work [40] addresses the scalability issue by extending the original vTPM model. Since the conventional periodic polling model does not scale well, i.e. each VM adds effort to the attestation cost, Goldman et al. [40] proposed an event-based monitoring and pushing model. In this way, every time there is an extend operation to the vTPM instance, the monitor will notify the users that subscribed to the VM associated with this vTPM instance, thus achieving event-based attestation. The benefit of the pushing model is that it will eliminate the problem of *Time-of-Measure Time-of-Use* (ToMToU) attacks (a platform is genuine at the time of measuring, but it is compromised when it is actually used) and TPM reset attacks (fast rebooting the system after malicious script execution, to reset TPM PCR values).

In general event-based monitoring is more convenient and the feedback time can be much faster than periodic attestation model, since the hypervisor itself has full control of its guests, and can be extended to support event-based monitoring. Following this idea, the authors of [42] directly offloaded the remote attestation work to the hypervisor, which is called *Integrity Verification Proxy* (IVP). The VMs running on top of the hypervisor are in *Debug Mode*, so a debugging tool (e.g., gdb) can be used to set *watch points* (e.g., locations in memory) triggered by integrity relevant operations such as IMA measurement list insertion. When a watch point is triggered, the hypervisor can pause the VM and calls an additional module to check its integrity. If the integrity test result is positive, then the VM is resumed for its execution, otherwise it is stopped. However, this solution suffers a severe performance penalty due to executing the VMs in debug mode and the remote attestation feature, and the integrity evidence of the services running in VMs are not present in the hardware TPM chip, which lower the security guarantees.

Lauer et al. [144] combined vTPM and IVP solutions. Their approach uses the vTPM to record integrity evidence for each VM and an additional module embedded within the hypervisor to generate the integrity reports for itself and all its guests. As recognised by [142], the integrity evidence of VMs needs to be bound to the integrity evidence of its hypervisor, which is called *deep attestation*. Thus the authors of [144] proposed to concatenate the nonce received from the challenger, the PCR values from all vTPM instances and the measurement list of all VMs, and then the concatenation result will be signed by the physical TPM as the external data of

the quote operation. In this way, the challenger can check the integrity of all VM's measurement lists against the PCR values of vTPM instances based on signature from the physical TPM. Meanwhile the integrity state of the hypervisor is also available in this integrity report as presented in the hardware PCR values.

However, this solution has two problems. First it is unfriendly to the privacy of the VM, because each challenger can have the integrity information of all VMs running in this platform, i.e. what software is running and what version. Second, this solution is computing power demanding to both the challenger and the attester, because the challenger needs to distil useful information from a massive amount of data and the attester needs to generate a very large integrity report, which makes the ToMToU problem more severe.

In [145], the authors presented a new design of a TPM that supports hardware-based virtualisation techniques by using functionalities of the Intel VT-X/I architecture [146]. This architecture augments the x86-processor architecture with two new forms of CPU mode, VMX root in which the VMM runs and VMX non-root in which the guest VMs run. Because of this new mode, VMM can use a second privilege level to issue scheduling and management commands. In this proposal, the information about the state of a physical TPM is stored in *TPM Control Structure* (TPMCS), it is loaded into the corresponding TPM by the VMM each time a particular VM operates on the TPM, this ensures that the state of on VM's TPM does not corrupt the state of another VM's TPM, although the TPM is the same one. However, the TPM specification and its implementation need to be extended in order to support the second privilege level and the TPMCS, such that the TPM context of each VM can be saved and loaded. Additionally, it can only work with virtualisation capable hardware TPM, introducing another limitation to its usage.

Similarly, taking advantage of the Intel VT technology, Intel proposed its own container solution called *Intel Clear Containers* [147]. It provides an alternative approach which delivers the benefits of both by combining the hardware-assisted isolation of hypervisor-based VMs with the high performance of Linux containers. In short, it is a container for a workload wrapped inside its own VM based on a minimal QEMU hypervisor [148]. The QEMU hypervisor retains the ability to enforce workload isolation of containers in hardware using Intel VT technology. All container workloads are spawned by unmodified Docker [36] or Rocket [149] software stacks and run unmodified within their own VM. Further the TPM helps applications or VMs to validate the hardware they run on. If the BIOS, physical or virtual machine configuration doesn't meet a baseline, the operating system simply doesn't boot.

Taking advantage of the para-virtualisation technique, where software running in guest VM needs to make specially hypercalls, the authors of [150] tried to para-virtualise the TPM in order to share the TPM amongst several operating systems. This solution requires the software components running in guest VM to be ported in order to use the para-virtualised TPM interface instead of the original one. Then the hypervisor multiplexes the TPM commands, schedule the access to the physical TPM and prevent unauthorised access to the TPM functionalities.

However, several important resources still need to be virtualised, including PCRs and EK. Because of this, this solution is not suited for remote attestation, especially to attest the integrity state of software running in guests, because the vPCRs and vEK do not provide enough authenticity and integrity guarantees. Apart from this issue, the only implementation related details disclosed by the authors are the number of lines of code that was required to achieve TPM para-virtualisation. No details regarding the platform used for implementation are available in the paper.

With the massive and very successful adoption of Docker [36], operating system level virtualisation (i.e. virtual container) comes back to public view. Various work try to extend the chain of trust into the internal of virtual containers. In [151], the authors proposed the idea to integrate vTPM with virtual containers with two different models, i.e. either put the vTPM instances in the host operating system kernel but beneath the container manager or in one of the privileged container. However, the authors only presented their proposals with no actual implementation.

The authors of the original vTPM are also implementing the vTPM support for Linux containers [152]. In short, they implemented a *vtpm_proxy* driver in the Linux kernel that enables to spawn a TPM device with the front-end being the `/dev/tpmX` (where X=0,1,2...) on the host and the back-end being a file descriptor returned from an *ioctl* on `/dev/vtpmx`. The former is moved into the container by creating a character device with the appropriate major and minor numbers while the latter is passed to the TPM emulator. This allows programs to interact with a TPM in a container using the character device and the emulator will receive the commands via the file descriptor and use it for sending back responses.

However, this solution does not link the internal of virtual containers to the physical TPM, thus the integrity guarantee provided is lower. Moreover, because the integrity state of the underlying host system is not included in a single remote attestation request to a virtual container, deep attestation is mandatory. This means in order to attest a virtual container, two remote attestation requests are needed.

CoreOS team[3] developed another container engine which is more security focused. It supports measuring container state and configuration into the TPM event log, and the measured data include the container root filesystem, the contents of the container manifest and the arguments passed to container engine [153]. This provides a cryptographically verifiable audit log of the containers executed on a physical node, including the configuration of each container. However, the solution is still incomplete because not all software modules running containers are measured and extended into the PCR.

An alternative of the TPM as the root of trust is the Intel *Software Guard eXtensions* (SGX) [154, 155], which is available in the 6th generation Intel CPUs. It provides similar functionalities as the TPM, but software isolation and remote attestation are provided using CPU instructions, i.e. the root of trust is built into the CPU. The software isolation capabilities allow for the software to have private

[3] `https://coreos.com/`

data in memory that cannot be accessed by another process, even in the face of a root-level exploit, as the access control is enforced by the processor. The remote attestation feature allows a challenger to confirm that certain software is executing in one of these protected memory regions (called *enclaves*). When combined together, a challenger can verify that certain software is running in its enclave before transmitting private data to it, and can then be assured that the data cannot be accessed by the rest of remote system, even the operating system.

Nowadays, this topic is very popular in NFV environments, especially because the SGX enclaves are suited to be trusted execution environment for operations like perform packet processing. With the help of Intel SGX, the authors in [156] proposed a secure framework for NFV applications. The framework provides an interface to move the VNF state and state processing code inside the enclave. Since VNF functionality is tightly coupled with their state, what needs to be done by the framework is to provide a model to only allow relevant state, which needs protection, to be moved into enclaves. In [157], the authors proposed a packet processing software module running in SGX enclaves in order to improve the privacy of NFV applications. Also they experimented the performance overhead introduced by SGX, and the result shows the computational overhead of using SGX is negligible in a realistic development scenario. As long as the data is stored inside of an SGX enclave and makes use of remote attestation to establish a secure channel between the enclave and the rest of the network, any private data that the VNF needs to access will be protected.

In [158], the author proposed *SCONE*, a secure container mechanism for Docker that uses the SGX trusted execution support to protect container processes from outside attacks. It offers a standard library interface that transparently encrypts/decrypts I/O data, in order to reduce the performance impact of thread synchronisation and system calls inside SGX enclaves. However, the performance overhead is more than expected, being around 40% with regards to the service running in native container environment.

The current limitations of adopting remote attestation in NFV environments can be summarised as following: (*i*) the internal of VMs or virtual containers is not present in the hardware TPM, it is only available in vTPM (e.g., vPCR and vAIK). (*ii*) the performance impact of hypervisor-based solutions (e.g., IVP) is huge, making them less feasible in real-world application. (*iii*) Intel SGX based solution, while promising, is only supported by a very limited number of devices. Moreover, SGX is known to be vulnerable to side-channel attacks [159, 160], such as *Conceal Cache Attacks*.

5.2.1 Contribution

Softwarised networks, that heavily rely on software modules running in virtual instances on distributed nodes and virtualisation technology, eagerly need reliable means to report the software integrity running in virtualised instances. In this Chapter, we present our design to attest the integrity state of the software service

running not only on the physical platform hosting virtual containers but also inside these containers (e.g., VNFs). This solution can boost the trustworthiness of network nodes to execute the desired network functions, which is an important step to create trusted networks.

Our solution has four advantages compared to previous work; (*i*) virtual container instead of virtual machine is chosen as the virtualisation technique because it is more lightweight and agile, thus more feasible in SDN/NFV environments; (*ii*) the integrity evidence of the software services running in virtual container is based on a hardware-based entity, thus the integrity guarantee is as strong as if they are running directly on bare metal physical hardware; (*iii*) for as many virtual containers can be launched on a single platform, only one integrity report is generated, which significantly reduces the performance loss compared to other solutions. (*iv*) the virtual containers are differentiated from each other, thus the managing entity can restore trust integrity state of the whole platform by replacing the compromised virtual container instead of resetting the whole platform.

As a matter of fact, our solution was presented and well appreciated in the ETSI NFV security working group in the the ETSI NFV meeting hosted in Bilbao on February 21-24, 2017.

5.3 Requirement Analysis

In this Section, we present the expected properties of trusted softwarised networks and the requirements to be achieved.

5.3.1 Security Requirements

Hardware-based root of trust: network nodes need to be attested based on hardware-based evidence. This is a strict requirement to provide the evidence integrity and authenticity, that elevates the integrity guarantees of the attestation results.

Platform boot integrity measurement: the boot phase of a network node need to follow the trusted boot approach that a CRTM is activated as the first component when the platform is booted. Subsequently, other components loaded during platform boot phase needs to be measured and the measures need to be extended into corresponding PCRs by its previous loaded component.

Service load-time integrity measurement: unique identification of loaded executables and configuration files of services (e.g., VNFs) is needed, and the measures of these structural data need to be inserted or appended in the integrity evidence signed by hardware-based identity key and evaluated by the verifier.

Attestation of virtual containers: not only the services running in the physical platform but also the VNFs running inside virtual containers need to be attested with hardware root of trust. Because in trusted softwarised networks, both the integrity of the host platforms and the network functions are crucial, hence the internal of virtual containers must be probed.

5.3.2 Functional Requirements

Fast deployment: the alteration to existing software and hardware environments should be minimal (if any) and additional concepts introduced should make use of and have to adhere to existing specifications. Meanwhile, the introduced remote attestation feature should be transparent to the network function developers, thus with no additional requirement from their side, as they should use the virtual containers as the unmodified ones.

Minimal performance loss: new features usually bring performance overhead, because they need to occupy CPU cycles. In the case of trusted softwarised networks, the network nodes are typically short of computing power, thus the performance overhead in the network elements' side should be negligible.

Differentiate attestation of virtual container: running in commodity devices, not all virtual containers are running network functions, since some may be irrelevant to security sensitive tasks. For this reason, virtual containers need to be identified, thus it is possible to differentiate whether the compromised container is relevant or not.

Simple roll back strategy: after a virtual container is compromised, the roll back strategy to restore the trusted state should be simple, instead of rebooting the whole platform as it is used to be.

Backward compatibility: the introduced remote attestation feature should incur no additional constraint. Meanwhile the original network should preserve its behaviour after the remote attestation feature is deactivated.

5.4 Remote Attestation in Lightweight Virtualisation Environments

Considering the requirements defined above, we made the following choices. First of all, we chose the TPM as the hardware root of trust, because it is widely deployed in modern commodity devices and its usage is free of charge. Second, we chose operating system level virtualisation technique (i.e. virtual containers) instead of

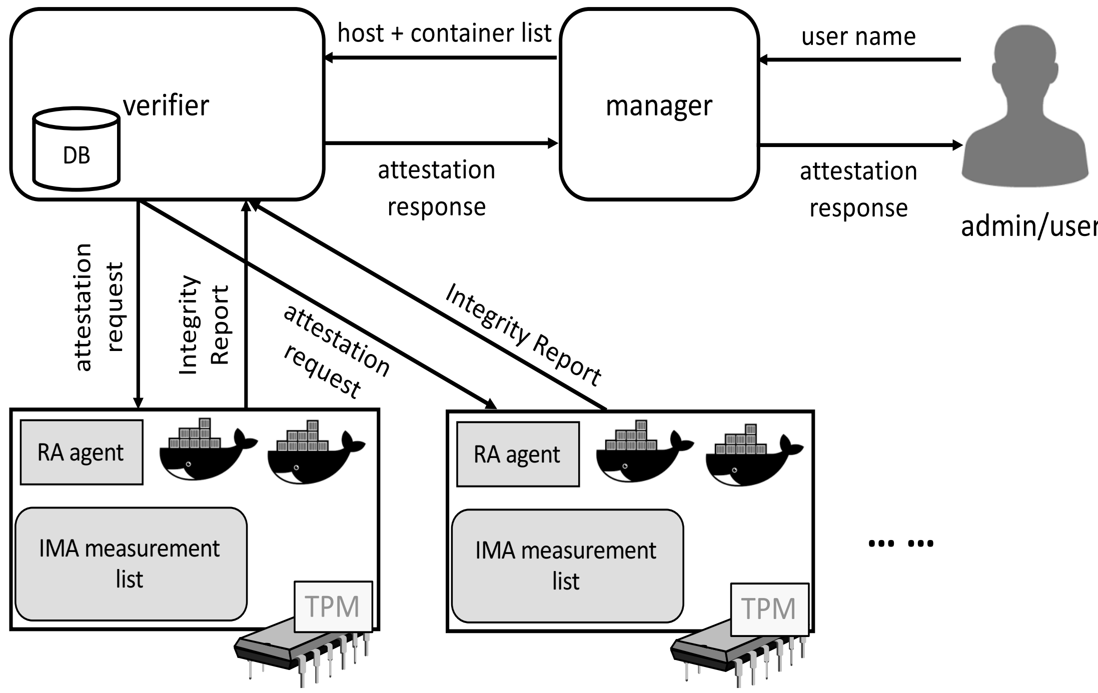

Figure 5.3. Overall architecture of Docker attestation system.

hypervisor-based one (i.e. virtual machines) because the latter is usually much more heavier than the former [44]. To be more specific, we worked with Docker virtual container implementation due to its open source nature and extreme active community. Third, we chose the remote attestation framework described in Chapter 3 as the basis, because it already supports attestation of trusted boot and services running on the physical platform, thus the expected extension work is minimised.

5.4.1 General Architecture

The general architecture of our Docker attestation system is illustrated in Figure 5.3. In our schema there are three actors: the *verifier*, the *attester* (i.e. virtual container host) and *manager*. These three components, as their names suggest, play different roles in our system.

The manager creates the required virtual containers for deploying network functions and keeps track of both the *universally unique identifier* (UUID) of the virtual container (hereafter called virtual container ID or container ID for simplicity) and the network node ID which hosts the virtual containers.

When the network administrator asks for the integrity of a network function deployed in some virtual containers running in the network, the manager initiates a remote attestation process. It sends to the verifier the list of virtual containers to be checked, along with the list of the network nodes hosting these containers, and asks for their integrity state.

The verifier is the one defined in Chapter 3, but it needs extension to fulfil the requirements defined above. It is the core of the architecture, since it is in charge of evaluating the integrity state of the network nodes and the virtual containers running in them. After it receives the remote attestation request from the manager, the verifier first contacts the network nodes in the host list, asking them to send back their integrity reports, which include the integrity evidence of both the host system and all virtual containers running in them. With the knowledge of the container list, the verifier then starts to check the integrity state evidence belonging to the virtual containers of interest.

Since the manager is in charge of managing all the virtual containers hosting network functions, it can start a rollback strategy if it notices a compromission of any virtual container or host. For example, the manager can pause the compromised container and start a new one to take over its job without rebooting the whole platform. However, if the host operating system (i.e. the one hosting the virtual containers) is compromised, then the whole platform must be rebooted because all containers could potentially have been compromised in subtle and unpredictable ways (e.g., in their application data).

In order to offer the aforementioned features, several existing components have to be modified and extended.

5.4.2 Extension of Linux IMA

The first important modification we made is related to the Linux IMA module. We extended the original module with a new template, with the capability of putting an identifier of a virtual container in the measurement list. The extension permits to discern between the IMA measures produced by the host system and those produced inside a certain virtual container, i.e. to differentiate where the executables and configuration files are loaded.

Given the internal working of virtual containers, the containers invoke directly the functions in the underlying kernel of the host system. Thus, when executables and configuration files in the containers are loaded into kernel memory, these operations will be automatically captured by the IMA module running in the host system, so that the corresponding digests are added to the IMA measurement list and the digest of measurement entry is extended into the PCR10 to ensure the integrity of the IMA measurement list. This standard feature is the basis of our solution and, in order to adapt its usage for a container-enabled platform, we created the new IMA template.

Our template adds a new attribute, called *dev-id*, which records in which virtual device the structural data is loaded, thus allowing to identify the corresponding container. For example, Figure 5.4 displays five measures recorded with our new template: two of them belong to the host system, which has the dev-id 8:19, while the other three belong to two containers, specifically those with dev-id 253:1 and

```
PCR# template-hash  template     dev-id   filedata-hash        filename-hint
10   ccd75...21c04  ima-cont-id    8:19   sha1:1bc28...aab2c   /usr/bin/ping
10   742f2...1f79c  ima-cont-id    8:19   sha1:b53bc...c0740   /usr/sbin/sshd
10   318cc...4fdcb  ima-cont-id  253:1    sha1:1bc28...aab2c   /usr/bin/ping
10   66789...916a4  ima-cont-id  253:1    sha1:9fb4b...a02f3   /badScript.sh
10   3fcee...28826  ima-cont-id  253:2    sha1:1bc28...aab2c   /usr/bin/ping
```

Figure 5.4. Example of the extended measures (from the IMA ASCII log file).

253:2 (fourth column)[4]. More in detail, after connecting to the host with SSH, we launched the `ping` command (i.e. load `ping` executable into kernel memory) three times, one each in the host and in two different containers. Since the `ping` executable is the one provided by the host system and shared by all the containers, the digest recorded in the *filedata-hash* column is the same for all the records.

The *template-hash* column is the digest of all the other values in the row concatenated and is extended into PCR10 to ensure the integrity of those values. As a consequence, since the dev-id for the third and the fifth records are different, hence they have different template-hash values even if the filedata-hash is the same.

Note that IMA records information not only for standard binaries but also for custom applications, such as the script in the fourth record. As an example, we assume that this script has been injected into the container exploiting a bug in a service. When the whole measurement list is sent to the verifier, it will decide if the container with dev-id 253:1 needs to be checked. If this is the case, the verifier will announce that this container is untrusted, because the digest of `badScript.sh` is unknown in the reference database. In this way, not only the verifier knows what binaries have been executed, but it can also identify the container or the host system where they are executed.

This feature brings a key advantage: if a container loses at any point its trusted state, then the manager does not need to reset the whole attester platform to restore trust state. On the contrary, it just needs to pause the untrusted container and start a new one. This feature makes the remote attestation technique much more appealing from a performance point of view, hence applicable in real cases.

5.4.3 Extension of Remote Attestation Framework

The second component to be extended is the remote attestation agent of our framework. The extension is to support the mapping of the container ID to its device number (i.e. dev-id) assigned by the host system.

Our remote attestation framework is the core tool for providing the remote attestation function. We extended the integrity report template of the original framework

[4]The *dev-id* is defined by the file system driver in the host kernel.

```
<Container Id="8948d6f37d41">
    <DevId>253:1</DevId>
</Container>
<Container Id="1beb7b9c05s6">
    <DevId>253:2</DevId>
</Container>
<Container Id="2f9695f9db36">
    <DevId>253:3</DevId>
</Container>
<Host>
    <DevId>8:0</DevId>
    <DevId>8:1</DevId>
</Host>
```

Figure 5.5. The extended part of an integrity report.

defined in Chapter 3 by adding new attributes to map the container ID with the corresponding device number.

This extension is mandatory because the verifier only knows the list of the container ID to be checked, which is sent from the manager. But in the integrity report received from the network node, there is no direct reference to the container ID, since the integrity report provides only the dev-id assigned by the host system.

Luckily, this modification is simple, since our remote attestation framework provides support for the transmission of the IMA measurement list to the verifier. What remains to be done is to add new elements and types in the XML schema used by the integrity reports. Additionally, the remote attestation agent is also modified so that it inserts these new elements in the reports.

As shown in Figure 5.5, the new XML elements added are:

- a *Container* element providing the mapping between a container's ID and the associated virtual device number created in the host system;
- a *Host* element containing the list of all physical device numbers associated to the host system.

5.4.4 Extension of IMA Verification Procedure

Finally, the last component to be modified is the remote attestation verifier, by extending its original behaviour to distil the containers to be attested out of the full integrity report when it receives a remote attestation request from the manager.

As presented in Chapter 3, the verifier has the ability to attest the services running on the physical platform by using a reference database storing the digests of

all known "good" executables and configuration files for a certain purpose. For instance, it is possible to initially populate the database with all the elements in the packages available in the official repository for multiple Linux distributions. However, this approach has a drawback that it only considers the platform as a whole. If the integrity report contains just one unknown digest for a loaded binary or applied configuration file, the verifier will evaluate the whole platform as untrusted and a trusted state can be restored only by resetting the whole platform. Unfortunately, resetting a physical platform hosting tens or hundreds of VMs or virtual containers is not acceptable in a real-world scenario when the problem is affecting just one virtual instance.

For this reason, we need to extend the original implementation. We introduce a new analysis type in the verifier, named *cont-check*, which requires as parameter a list of the containers to be checked. We call this list *cont-list* and it is created by the manager based on the virtual container to be verified for a specific service or user. The new cont-check analysis type works cooperatively with *load-time* analysis type, it reduces the full IMA measurement lists received from the network nodes by keeping for each node only the measures related to the containers of interest plus the measures generated by the host system (since it is the base for running the virtual containers). Afterwards, the verifier compares this reduced list of measures with the known good values in the reference database in order to decide if there is unknown measure(s) or if all active software services achieve the required trust level based on their IMA measures.

As previously explained, each measure in the extended IMA template is linked to its virtual device ID, so the verifier knows if a measure is related to the host system or to a container (and in this case, which one). In this way, the result of the verification is related only to specific containers. For example, if one container is compromised and it is present in the cont-list, then the attestation result will be *untrusted*. Or if the software service running in a container of interest does not achieve the required trust level. On the contrary, if the compromised container is not in the cont-list (i.e. the manager does not care about its state, for example because it is not involved in any sensible operation), the overall system status will remain *trusted*.

In the case of untrusted result, the verifier and the manager know which measures are unknown or bring security and functional bugs from the reference database and which container they belong to, so the manager can proceed to replace only the corrupted container in the attested host, with no need of rebooting the whole network node.

5.5 Discussions

In this Chapter, we presented our solution, an architecture to support integrity verification of operating system level virtualisation instances, i.e. virtual containers, using trusted computing techniques.

With this solution, the service running in the virtual containers can be practically attested as if they would be running on a physical platform, and the integrity state of the virtual containers can be well understood by a third party in a reliable manner. The capability to directly interact with the hardware root of trust makes the integrity reports from the virtual container host non-forgeable, which has strong resistance against remote attacks.

Another important feature of our solution is that, the verifier is able to know exactly which element (container or host system) is compromised. Thus the network manager can take informed decision about the rollback strategy, e.g., pause just the compromised virtual container or stop all containers and restart the whole platform.

Moreover, our solution is transparent to the services running in virtual containers, that they don't need to be modified in any way as if they are interacting with a normal operating system level virtualisation environment. All modifications to enable remote attestation are small and performed directly in the host system. This makes our solution very easy to be adopted and without any impact to the hosted services. As a matter of fact, our solution can be directly applied in cloud computing environment to monitor more general service running in virtual containers.

Last but not least, the images in current official Docker repository contains various known vulnerabilities because of out-of-date software, according to [161], 24% of latest Docker images have significant vulnerabilities. This situation justifies the needs of the integrity level in our remote attestation framework, which is able to identify which software and container has security vulnerability or functional bugs.

Chapter 6

Implementation Details and Performance

Besides theoretical proposals, we also provide details about the implementation. Our implementation is based on existing and widely adopted open source software, mostly released under open source license in the *SECURED* [64] European project and available to be deployed directly[1].

Moreover, we also provide extensive performance details of our systems, including the performance overhead in the attester with the additional remote attestation feature, the overall time needed of a remote attestation operation in our framework, the time needed to establish a trusted channel and the performance impact in virtual container environment.

6.1 Remote Attestation Framework

Our remote attestation framework is developed based on Intel's OpenAttestation (OAT) SDK v1.6[2], which we largely extended for our purpose. The original OAT framework aims to deal with the integrity state of the boot phase of physical platforms equipped with a TPM, thus a machine is considered trusted to deploy virtual machines if the machine is booted with all known components in the right order. However, it does not take into account the integrity of the platform after they are running, which leaves a gap for a wide range of remote attacks.

In the following paragraphs, we first briefly describe the original OAT framework, pointing out its drawbacks and advantages in order to justify our choice of using it as the basis of our development. Then we present our enhanced version of OAT (hereafter referred as OAT v1.7), detailing our extensions and performance efficiency of the new version.

---

[1] https://github.com/SECURED-FP7/

[2] https://github.com/OpenAttestation/OpenAttestation/

6.1.1 OpenAttestation SDK

First of all, the original OAT framework is well designed and provides all components necessary to adopt TCG-defined remote attestation feature, of which we took advantage in order to avoid reinventing the wheel. It provides three important features that reduce the amount of work from our extension, which are *PrivacyCA*, *WhiteList tables* and *integrity report portal*.

As described in Chapter 3, PrivacyCA is required to certify a TPM generated AIK associated with the its EK when the platform is registered to the verifier. We chose PrivacyCA approach instead of Direct Anonymous Attestation (DAA) [28, 95] because in our working scenario, the attestation targets are servers instead of user terminals, thus we do not consider the privacy issue. As a matter of fact, in our scenario we prefer the server identity to be revealed to the end users, so they can have a hardware-based identity proof of the server, hence removing the chance of a shadow server attack. Also, DAA is a very complex protocol which requires multiple interactions with the TPM in order to generate a valid quote output that introduces unnecessary performance overhead in our solution.

The native support of the Whitelist table is another advantage, since this feature can be directly used to evaluate the boot integrity of attesters. As described in Section 3.5.1, the final PCR values are fixed and predictable if the components loaded in the boot phase are intact.

Integrity report portal summarises all previous reports may not be a mandatory feature for a remote attestation framework, but it is a very useful feature because in our case the integrity report is largely extended and a portal can help to identify the compromising point of the attester.

All the data of registered AIK certificate, whitelist PCR values and the received integrity reports are stored inside a well structured *MySQL* database[3], which makes efforts of the management and our extension moderate.

After presenting the advantages of OAT v1.6, we then have a look at the way OAT v1.6 works, especially the remote attestation process and why it needs to be extended.

The attestation process proposed by OAT v1.6 relies on the appraiser's orchestration role and the polling feature implemented on the remote attestation agent. The latter periodically polls the remote attestation API for new actions to be performed. If an attestation request is submitted, the appraiser defines a new `send_report` action for all involved agents and the corresponding agents start their work. For instance, when the agent receives a `send_report` command, it generates an integrity report which contains just the output of a Quote operation from the TPM, obtained through the mediation of the TrouSerS software stack [54]. In particular, the PCR values are stored into an element named *QuoteData* in the integrity report, which contains the necessary information to prove the integrity and freshness of the PCR

[3] https://www.mysql.com/

values, such as the nonce, the selected PCRs, and the AIK signature of the PCR values.

All integrity reports from remote attestation agents are collected by the appraiser, which executes a three-step validation process as a whole. In the first step, the appraiser validates the authenticity of the integrity report by verifying the AIK's signature with its registered public key. Then it compares the received integrity report to the last one from the same agent, and if the PCR values in both reports are the same it means the integrity state of the platform has not changed, so the appraiser just gives back the same result as the last one. Otherwise, the appraiser compares the received PCR values to the ones stored in the WhiteList table, in order to conclude the integrity state of the platform.

However, this validation approach has two drawbacks. First, it has no flexibility, since the appraiser can give only Boolean decisions, either *trusted* or *untrusted*. There is no intermediate level and the result contains no useful information (e.g., why the platform is untrusted). Second, the whitelist table used by the appraiser is a set of binary cumulative measures, and by nature, the extend operation is order sensitive, i.e. extend(A,B) $\neq$ extend(B,A). Thus in order to get a positive result, not only "good" components are loaded, but also they need to be loaded in a specific order.

Both problems limit the feasibility of remote attestation in real-world applications. For example, the whitelist approach is infeasible to be used to prove the integrity state of services running in application layer, since it is practically impossible to set the executables and configuration files to be loaded in a predefined order and no modification is allowed afterwards (e.g., no change of their configurations).

6.1.2 Enhancements to OpenAttestation

Extensions to the original OAT framework have been designed following the ideas presented in Chapter 3. Because the original OAT framework does not allow pluggable components, thus our extension has to dig into its internal source code.

First of all, we extended the appraiser web service in order to expose the received integrity reports to external tools and storage with a RESTful API through a function called *fetchReport*. The API receives the integrity report ID and gives back the raw integrity report in XML format from the MySQL database, which can be invoked by external analysis tools that have been registered to the appraiser.

Then we modified the remote attestation agent that appends the IMA measures in the integrity reports. Especially we added a new *SnapshotType* and its sub-attributes in the integrity report, which contains the IMA measures. However, in order to send arbitrary length bytes without getting corrupted by, e.g., a couple of special symbols, in which case may break the transmission, all information retrieved from the operating system (e.g., executable names) are encoded in Base64 format, thus the raw integrity report is less human readable (Figure 6.1).

```
<Values>
    <ns4:SimpleObject>
        <ns4:Objects Image="BQAAA...RvcgA=" Type="ima">
            <ns4:Hash AlgRef="sha1" Id="PCR_10_LV1_0_1_EVENT">KYh...N8Y=</ns4:Hash>
        </ns4:Objects>
    </ns4:SimpleObject>
</Values>
```

Figure 6.1. Example of an IMA measure in an OAT integrity report.

```
bash oat_pollhosts -h verifier '{"hosts":["attester"],
    "analysisType":"VALIDATE_PCR;load-time,l_req=l4_ima_all_ok|=="}'
```

Figure 6.2. A remote attestation request calling two analysis types.

Afterwards, the *hash* value (i.e. the digest of the measurement entry) in the report will be extended one after another and the final value will be compared to the one stored in the quote output, in order to prove the IMA measures are intact.

After the integrity report is extended to include IMA measures and exposed through *fetchReport* function, we modified the internal source code of OAT appraiser in order to change the hardcoded analysis types (i.e. compare reports and validate PCR) into adjustable parameters and exposed another RESTful API (called *analysisTypes*) for the registration of external integrity report analysis tools (Figure 6.3).

From now on, attestation request is adjustable. For instance, in Figure 6.2, the remote attestation request is calling two analysis types, i.e. VALIDATE_PCR and load-time. Thus, it will compare the received PCR values to the whitelist table and call the external tool which is registered with load-time analysis type (i.e. in our case, the IMA measure verification script registered as shown in Figure 6.3). Meanwhile the load-time analysis type has its own input parameters, i.e. *l_req=l4_ima_all_ok*|==, it means that the required integrity level is 4 with all IMA measures must be known and the software services must be up-to-date regarding to the reference database.

In order to minimise the performance impact introduced by the largely extended integrity reports, we introduced a new feature called *partial integrity reports*, which only contains data have not been sent to the appraiser. As mentioned in Chapter 3, we can achieve this feature thanks to the nature of the extend operation. In order to do so, we added a new field to the attestation log, called *firstReport*, which stores the ID of the report containing the first part of the list of measures. With this additional information, the appraiser is able to build a report with the complete set of measures from partial reports. On the remote attestation agent side, it is extended to store the number of bytes read from the measurement list file and the last measure read, thus it can ignore data which have been sent previously when the appraiser requests a report of type ‘continue’ (i.e. an identification of partial integrity report).

In Figure 6.4, there is only one file newly measured by IMA, and we simulate

```
addAnalysisType() {
  curl -H "Content-Type: application/json" -X POST \
  -d '{"name":"load-time","module":"IMAVerify","url":"'$RAPATH' -H '$DBIP'"}' \
  "https://verifier:8443/WLMService/resources/analysisTypes"
}
```

Figure 6.3. Registering load-time as a new analysis type.

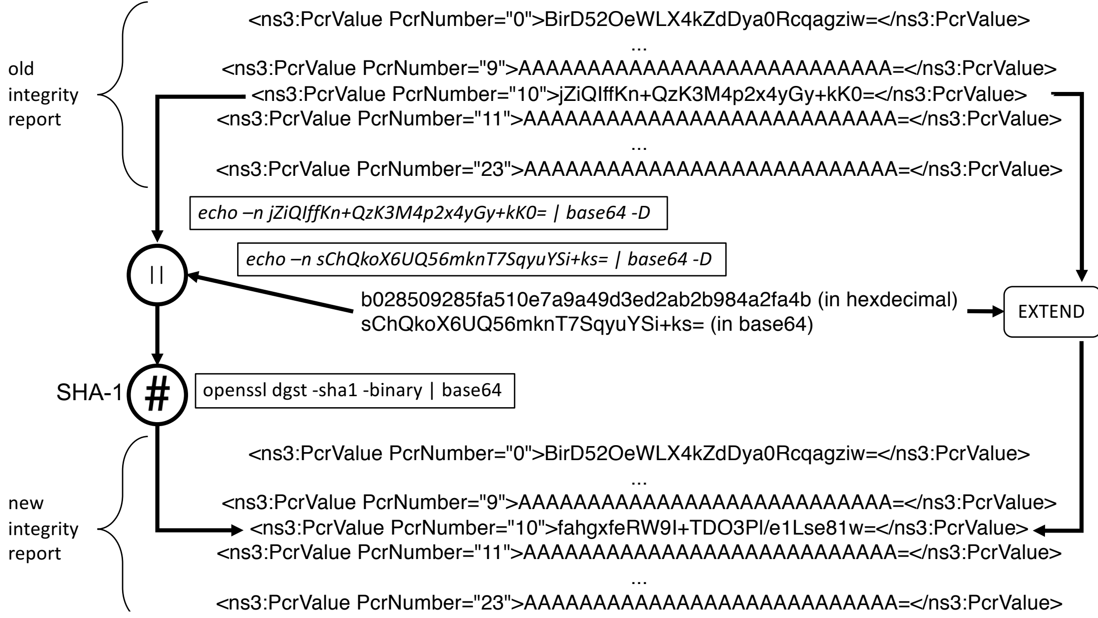

Figure 6.4. Extend operation with partial integrity report.

the extend operation with software tools (i.e. *openssl* and *echo*). The data are operated in bit level, thus the concatenation operation needs to first decode the base64 data, then the concatenation result are inputted to openssl with the binary option and later the binary output are encoded to base64. The PCR10 value in the new integrity report shows that the final result of our simulated operations is the same one as the TPM extend operation.

Either with full or partial integrity reports, every time the remote attestation agent receives a `send_report` request, it generates a fresh quote output and sends it back to the appraiser. This operation is to prove the remote attestation agent is not lying about if there are new executables loaded into kernel memory, i.e. measured by IMA. The difference between full and partial integrity reporting is only relevant to the IMA measures to be inserted into the integrity report.

Moreover, if the PCR values in the integrity report is the same as the previous report, it means the integrity state of the attester has not changed, thus the new integrity report will not be stored in the attestation log in order to save the disk space of the verifier.

Periodic attestation request has also been implemented because getting a fresh remote attestation result on-the-fly is time consuming, since it needs several seconds. Hence we introduce periodic attestation request to always have a valid attestation result and give back the result immediately. A new function named *getPendingRequests* was introduced to obtain a list of attestation requests to be served from the OAT MySQL database for either periodic or non-periodic requests. This function will be invoked by the attestation API to know if there is any attestation to be performed for a certain attestation agent.

In the case there is a periodic attestation request, the appraiser first checks if the request has expired by comparing the associated expiration time to the age of the request. Afterwards, it checks if the time threshold is less than the difference between the current time to the last request time, if so, the appraiser sets the flag asking the remote attestation agent to send its integrity report, otherwise the period threshold is not reached and the agent needs to do nothing. In the end, the appraiser updates the request time associated with the request in the OAT database in order to tell if the attestation threshold is reached in the next time. While in the case there is a non-periodic attestation request, the appraiser directly sets a flag asking the remote attestation agent to send its latest integrity report.

IMA Measure Verification

After the description of our OAT framework extension, in the rest of this section we will have a deep look at the IMA measure verification implementation (i.e. load-time analysis type) and its reference database creation.

The IMA measure verification tool is developed using python. After it is registered in the new OAT framework (Figure 6.3), it can be invoked through the remote attestation API by calling load-time analysis type. Then the script accesses the integrity report received from attester and evaluates the IMA measures in the report based on the defined arguments. Since the script is invoked through the remote attestation API, its evaluation result influences the global attestation result of the attester.

In the first step, the IMA measure verification tool gets the received integrity report by calling the *fetchReport* function with the latest report ID of the attester. Then it distils the IMA measures from the integrity report and parses them from Base64 to ASCII, and creates a dictionary containing all IMA measures. Since OAT has already checked the integrity state of the IMA measures in the integrity report, then the verification tool can rest assure and query these IMA measures in the dictionary directly to a reference database and give assessment result.

Development of the IMA measure verification tool is straightforward and the difficult part is creating the reference database. We chose to use Apache Cassandra[4],

[4] http://cassandra.apache.org/

a highly-scalable NoSQL database, suited to manage huge amount of data with a *key-value* structure (i.e. the digest values).

The Cassandra data model is based on the concept of *Column*, an elementary data structure with a name (or key), a value and a timestamp. It also supports a more complex data structure, called *SuperColumn*, whose value consists in a map of *Columns* instead of plain data. Moreover, since the structure is dynamic, the *row* (i.e. the key), its *Column* or *SuperColumn* can be inserted together.

Following the definition in Chapter 3, our database is organised around two main *Column Families*: `FilesToPackages` and `PackagesHistory`. In `FilesToPackages`, each digest is organised as a *row* in the database and its full path name and the packages are organised as *Columns*. The package name is further grouped by the distribution name (e.g., CentOS) and the processor architecture (e.g., x86) as a *SuperColumn*. Meanwhile in `PackagesHistory`, the *row* is the concatenation of the package name and its distribution. Further the package's version and its release number is organised as another *SuperColumn* in the same way as released by repository maintainers. In the end, the package full name and its update type are organised as *Column*. In this approach, the packages with same name and distribution can have a sorted order based on the package version. Thus when the reference database is queried, it can give result immediately without sorting the package version on the fly.

After the reference database is structured, the next step is to find the correct data and fill the database. The task is more complicated, especially to find the update type of each package, which will be used to conclude the integrity level of the attested platform. For different distributions, we have to use different methods and tools to get this information.

Software packages of *Ubuntu*[5] are published and stored according to their types, which can be:

- `trusty`: normal release;
- `trusty-backports`: unsupported updates, it offers a way to selectively provide newer versions of software for older Ubuntu releases;
- `trusty-proposed`: pre-released updates, i.e. the testing area for updates;
- `trusty-security`: important security updates; patches for security vulnerabilities in Ubuntu packages;
- `trusty-updates`: recommended updates; updates for serious bugs in Ubuntu packaging that do not affect the security of the system.

[5] https://www.ubuntu.com/

The packages are downloaded using *debmirror*[6] and stored in the local disk before further processing. Their update types are derived from the repository name[7]. Then a python script named *client_insert_pkg_hash.py* is called to dissect the compiled packages, computing the SHA1 digests of the binaries inside, and insert the digests, package names and versions, update types into the reference database.

The software packages of other three distributions are processed with a similar approach. *Fedora*[8] and *CentOS*[9] are downstream distributions of *Red Hat*[10]. They share some similarities inherited from Red Hat, e.g., they both use *RPM Package Manager* and install `.rpm` suffix packages. Sometimes, the update packages are compiled from the same source file used for patching the upstream Red Hat installation. *Extra Packages for Enterprise Linux* (EPEL)[11] is a Fedora Special Interest Group that creates, maintains, and manages a high quality set of additional packages for Enterprise Linux, including, but not limited to, Red Hat Enterprise Linux (RHEL), CentOS and Fedora. It is usually a supplementary repository for CentOS and Fedora, thus we also create a reference database for it.

These three distributions do not have a unified tool to download the compiled packages from their repositories, but luckily most repositories can work with the *rsync*[12] protocol.

Fedora community (including EPEL) has a very nice website[13] for interacting between developers and users. In this website, each package publishes its *submitter*, *release*, *status*, and most importantly, the *update type* (Figure 6.5). Thus, we developed a new python script to parse the web pages in order to find the update package name and retrieve its update type directly from the web site. When Red Hat downstream distributions are chosen to host the verifier, then we can directly use *bodhi-cli*[14] tool to display the update type for each package, but this tool is not available for Debian-based distributions.

Unfortunately, CentOS does not have such website, hence a different approach to retrieve the update type for each package is needed. There are two possibilities: searching for the information from the upstream Red Hat update website[15] or the

[6] https://linux.die.net/man/1/debmirror/

[7] Lately Ubuntu repository changes the way to publish update packages, the actual packages are released in a single directory but the information of these packages is still stored in separated directories as it used to be, so the approach is the same, but needs some modification to the implementation.

[8] https://getfedora.org/

[9] https://www.centos.org/

[10] https://www.redhat.com/

[11] https://fedoraproject.org/wiki/EPEL/

[12] https://linux.die.net/man/1/rsync/

[13] https://bodhi.fedoraproject.org/updates/

[14] https://fedoraproject.org/wiki/Bodhi/

[15] https://rhn.redhat.com/errata/rhel-server-7-errata.html

FEDORA-2015-15798

Submitter	lvrabec
Builds	selinux-policy-3.13.1-128.13.fc22
Update ID	FEDORA-2015-15798
Release	F22
Karma	+2 stable threshold: 3 unstable threshold: -2
Type	bugfix
Severity	
Status	stable
Suggestion	
Submitted	8 days ago, 2015-09-14 09:59:10.428460

Figure 6.5. Fedora update system.

CentOS package announce mailing list[16].

Although most of CentOS update packages are compiled from the source package of Red Hat, still quite a lot of CentOS packages are missing in the Red Hat website. On the contrary, the mailing list is a better option. In the mailing list, we can have the update package name, the update version and the update type (Figure 6.6). The format of the mail is consistent, which saves our effort to adapt the script to find the update type information.

Similar to Ubuntu, all downloaded packages from the repositories of these three distributions are dissected, the SHA1 digests of all binaries will be computed and inserted into the reference database, along with the package name, package version and package update type.

Collecting and filling all the information into the reference database is a time consuming process, for example, *Ubuntu trusty* (14.04 LTS) alone has more than 10,000 packages in 60 GB that need to be downloaded and processed. However, only the first time to create the reference database is time consuming, afterwards, only the newly uploaded packages will to be appended into the database, which requires very little time.

Creating the reference database is more difficult than developing the standalone IMA measurement verification tool. It heavily relies on the support from the distribution repository maintainers. Finding the relevant information from Fedora and EPEL website is more straightforward, but the format and the URL address change from time to time, and a lot of efforts are required to adapt to these changes. On

[16] https://lists.centos.org/pipermail/centos-announce/

```
CentOS Errata and Bugfix Advisory 2015:1542

Upstream details at : https://rhn.redhat.com/errata/RHBA-2015-1542.html

The following updated files have been uploaded and are currently
syncing to the mirrors: ( sha256sum Filename )

x86_64:
5c1f6a42a94fb5b12dcde973990023487aac346d837414dd330153bc24bee85a
unixODBC-2.3.1-11.el7.i686.rpm
932850e7a115ce7f35e47ea1bf29da70c28a36bf7ba2f8650554c8bf1fb3985f
unixODBC-2.3.1-11.el7.x86_64.rpm
6a03a1740c5b5e6d6a9facddca6a5e86c87e204f6878470df9c0601d1c1cae10
unixODBC-devel-2.3.1-11.el7.i686.rpm
48d78784659d95b2800ffaebd7f43677e5646eb3be16646cc2cf2c1ba9d642d0
unixODBC-devel-2.3.1-11.el7.x86_64.rpm

Source:
70b40153fb4659ec6177b65d5f948d16e0e8342656d18c4c294547eb3acff1d3
unixODBC-2.3.1-11.el7.src.rpm



--
Johnny Hughes
CentOS Project { http://www.centos.org/ }
irc: hughesjr, #centos at irc.freenode.net
```

Figure 6.6. Example mail from CentOS-announce mailing list.

another hand, for CentOS update packages, it may happen that some packages are published, but they cannot be found in the announce mailing list, so we have to find the update type of these packages manually from the upstream website. If a package has no update type information, an *unknown* update type (treated the same as *newpackage*) is defined for them, indicating that this package is released in official repository, but there is no update type information available.

6.1.3 Performance Evaluation

In order to assess the performance of our implementation, we present the data collected in our testbed.

The setup

We use two different machines in our setup, acting as the *attester* and the *verifier*. In order to simplify the scenario, the reference database is running inside the same machine hosting the OAT appraiser (i.e. verifier), but it can be deployed in another machine.

One of the advantages provided by our approach is that the attestation burden lies in the machine hosting the verifier, while approximately no performance impact exists on the attester side. For this reason, the devices deployed are as following: one very old (i.e. powerless) device is used as the attester to exacerbate the performance

impact, while one powerful virtual machine is used as the verifier and hosts the reference database. The individual specifications are those:

- *attester*: Intel Core 2 6400 CPU @ 2.4 GHz + 2 GB RAM + Infineon TPM v1.2 + 160 GB hard disk
- *verifier*: a KVM virtual machine assigned with 2 Intel Xeon CPU cores @ 2.4 GHz + 4 GB RAM and 160 GB hard disk

To reduce the network influence in the test, both machines are connected to the same switch. Although multiple Linux distributions are supported by in our approach, in our test, we use CentOS 7 with distribution kernel version 3.10.0-229.el7.x86_64 as the operating system in the attester. Because CentOS is the first Linux distribution compiled with IMA enabled by default and it is one of the most widely adopted operating system in servers. Similarly, the verifier is also running CentOS 7, but the distribution kernel version is 3.10.0-123.20.1.el7.x86_64.

Test cases

In our work, the performance of concern is the attestation cost, mainly the performance penalty introduced to the attester with IMA and RA and the time needed for the verifier to finish its analysis.

In order to accurately test the performance penalty in practice, we simulated a real-world use case, with the attester set to perform operations similar to those of a HTTP server reacting to incoming requests. To discard the influence of the network communication, which may be unstable because of various reasons, we deliberately decided to perform only local I/O operations with a local trigger to start all the operations. In this test, the attester creates a single 1 MB file, then starts an infinite loop to repeatedly compute the SHA512 digest of this file. At every computation a counter is updated in a file on disk. This test mimics the case that a server reads a request, performs a computing operation and writes log data on disk.

We repeated the test ten times for each of the three different settings: basic configuration (i.e. no IMA and no RA), execution IMA policy configuration (i.e. IMA activated with execution policy described in Section 3.5.4 but no RA) and finally IMA remote attestation configuration (i.e. IMA activated with execution policy and RA requested every 10 s). Each run of the test lasts 300 s and the recorded results are presented in Table 6.1.

As it is shown that the differences among these three configurations are minor, the overall performance is only slightly affected by the introduction of IMA and RA. Considering as a reference the average value of performed operations without neither IMA nor RA, the performance only drops by 0.66% in average with IMA and RA enabled and in worst case the performance drops about 2.8%. So we conclude that the performance impact of our proposed integrity verification feature is nearly optimal and suitable for application in real-world cases, with respect to the overhead on the attesting platform.

	no IMA, no RA	IMA, no RA	IMA, RA
MIN	49,705	49,604	48,702
AVG	50,109	49,894	49,778
MAX	50,584	50,600	50,366
index	100%	99.57%	99.34%

Table 6.1. Number of operations and performance index in three configurations.

The verifier is hosted on a (trusted) third party, which also hosts the reference database and needs to query the latter with the IMA measurement list received from the attester. Although its performance has no direct influence on the attester, but its behaviour affects the overall performance of remote attestation process.

As described above, our updated remote attestation framework supports multiple analysis types of integrity reports, the user can define his preferred analysis type in the attestation request, individually (only one analysis type) or cumulatively (overall analysis with all supported analysis types).

The most important analysis type is load-time, i.e. the IMA verification script is invoked to analyse the IMA measurement list to check the if the binaries executed and the configurations applied are known in the reference database, meanwhile it concludes the integrity level of the attesting platform. This analysis is more time consuming, while the other two analysis types, *VALIDATE_PCR* and *COMPARE_REPORT* take less time (the functions are also much simpler). Thus we launched ten remote attestation requests with each analysis type with a fresh CentOS 7 minimal installation without any software running on it, in this case the number of IMA measures in the attester is 290.

Table 6.2 presents our test results of elapsed time (the period between user submits a remote attestation request until he receives the result) and processing time (the period between the OAT verifier receives the integrity report until it produces the result) for each attestation analysis type.

ANALYSIS TYPE	ELAPSED TIME (s)	PROCESSING TIME (s)
COMPARE_REPORT	5.672	4.606
VALIDATE_PCR	5.842	4.758
load-time	6.757	5.738

Table 6.2. Average elapsed and processing time for each analysis type.

As presented, the time required by load-time analysis type is about 1 s longer than the other two analysis types. But be reminded that in all three cases, the remote attestation agent needs to insert all 290 IMA measures into the integrity report and the difference comes from the verifier side, i.e. if the verifier needs to call the IMA verification tool or not. The time needed in all three analysis types will increase with the number of IMA measures increases, and this behaviour is more clear in load-time analysis type because every new IMA measure introduces

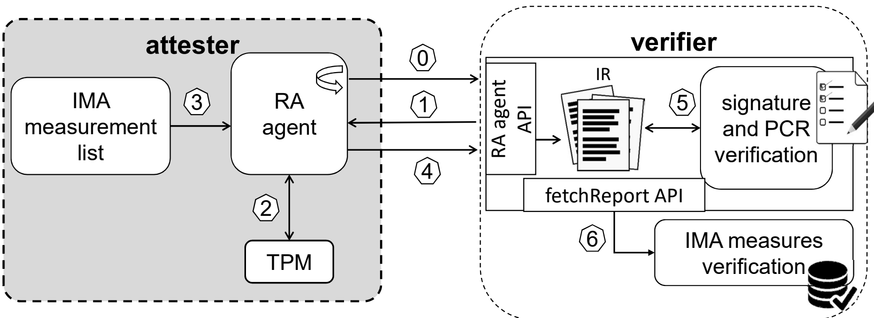

Figure 6.7. Actual remote attestation process in the developed framework.

additional workload to be queried to the reference database and evaluated with the integrity level.

Now we are going to have a deep look at the load-time test result presented in Table 6.2. When the verifier receives a remote attestation request from a user, the following seven steps must be performed before the verdict can be issued (Figure 6.7). The details of time used in each step in our framework are the following: in order to minimise performance loss and attack surface of the attesting platform, the remote attestation agent does not accept incoming attestation requests from other parties, rather it periodically polls the verifier (it registered to) at a predefined interval (step 0). Once a remote attestation request is present as a flag (step 1), the agent issues a *quote* request to the TPM (step 2) for getting the signed PCR values and uses the result plus the recorded IMA measures to create the integrity report (step 3). The time to perform a quote operation is fixed while the time to prepare the IR is proportional to the number of IMA measures because each measure along with its name must be encoded in Base64 and then appended to the XML template defined in the TCG specifications. Thus the size of the integrity report is also proportional to the number of IMA measures and so is the time needed to transmit the report from the attester to the verifier (step 4). After the report is received, the verifier checks the digital signature of the quote output against the public key certificate of the attester, stored when the host machine was registered. Next the consistency of the IMA measurement list is checked against the PCR values (i.e. PCR10), as previously shown in Figure 3.3 (step 5). When these steps are completed, the measurement list is passed to the IMA verification tool to be compared to the reference database, in order to evaluate the integrity level of attester (step 6). At this point, the verifier is able to return the integrity verification result with load-time analysis type.

In our setup, each remote attestation agent running in attester polls to the verifier every 2 s (an adjustable parameter), so the time cost for step 0 and step 1 is non-deterministic, and it is relevant to the time instant of sending the remote attestation request from the verifier. This is the reason why in Table 6.2 the average elapse time is about 1 s more than the process time in all three analysis types.

In step 2, the remote attestation agent needs to get the quote result from the TPM. This operation is performed through the *TrouSerS* TCG software stack [54]

and it takes around 2 s for the remote attestation agent to get a quote result from the TPM in CentOS 7, which is consistent with the results in [162].

The time to create a TCG compliant integrity report (step 3) in the attester is proportional to the number of IMA measures in the list. It takes about 2.4 s in case of 290 IMA measures. But be reminded that the attester is running on an old machine, and with a more powerful device the time could be reduced. When the integrity report is ready with these 290 IMA measures, its size is about 96.5 kB.

The network quality (step 4) impacts the time that the attester sends its integrity report to the verifier. In order to evaluate the network impact to each attestation operation, we used *iperf3*, a tool for active measurements of the maximum achievable bandwidth on IP networks[17] to evaluate the network connection speed. The average bandwidth from the attester to the verifier ranges from 11.0 MB/s to 11.8 MB/s, and the average is 11.3 MB/s. Considering the integrity reports, whose maximum size is about 100 kB, the time for transferring the reports is negligible (around 0.01 s).

In step 5, the verifier needs to perform two operations. First it checks the AIK signature in the integrity report, then it extends the measures in order to see if the IMA measures have been modified. This step is rather fast, as it takes around 0.4 s.

Finally, in step 6, the IMA verification tool is invoked to query the IMA measures with the reference database. With 290 IMA measures, the time needed to assess the measures and conclude the final result takes around 0.95 s.

In order to present the performance improvement of partial IR (mainly in step 3, since extend operation in step 5 is very fast which makes the improvement negligible), we launch RA requests in both normal mode and in partial IR mode, and the results are presented in Table 6.3.

OPERATION	normal mode	partial IR mode	improvement
create IR time	2.385 s	1.738 s	0.647 s (27.1%)
IR size	96.5 kB	3.7 kB	92.8 kB (96.2%)

Table 6.3. Performance improvements with partial integrity reports.

In normal mode, the integrity report sent from the attesting platform comprises 290 IMA measures, and its size is about 96.5 kB. On the contrary, a partial integrity report is sent when a new measure is detected, then it only contains one IMA measure and its size is about 3.7 kB. The time to generate a full integrity report is about 2.38 s and it is reduced to 1.74 s with a partial report (27.1%).

The partial IR feature can significantly reduce the time cost in step 3 by discarding the duplicated IMA measures to be processed, and slightly improve the performance of step 5 by discarding duplicated extend operations. Potentially step 6 (query IMA measures to the reference database) can also benefit from partial integrity report, because only a subset of IMA measures (i.e. IMA measures in the

[17] http://software.es.net/iperf/

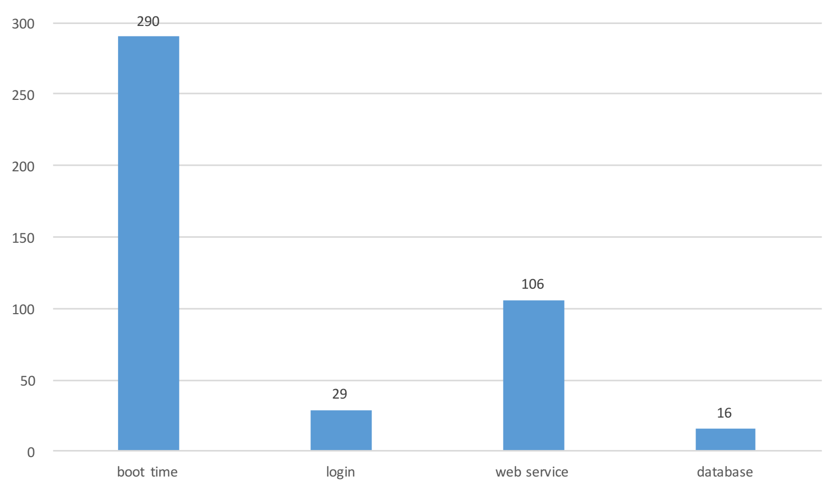

Figure 6.8. The number of the IMA measures in a typical web server.

latest partial integrity report) needs to be queried to the reference database. However, this benefit brings an additional problem with regard to unknown digests in the previous integrity reports. If the usage scenario is simple that all IMA measures are equivalent, then this problem can be solved with an additional flag showing if there is unknown digest in each report. However, this is not always the case, as we will show in the trusted channel and the virtual container attestation scenarios. Hence step 6 is not taking advantage of the partial integrity report feature, each time the IMA verification tool is called, all IMA measures collected starting from the platform boot are queried to the reference database.

In a typical situation, when the attester is booted, the OAT appraiser attests it in order to ensure the integrity of booting process (measured by TrustedGRUB2[18] and validated with *VALIDATE_PCR* analysis type[19]), at that time, the number of the IMA measures is 290. Afterwards, IMA measures another 29 executables with a local login operation, 106 executables with the activation of the Apache web service and 16 executables with the activation of an SQL database (Figure 6.8). In total, the full IMA measurement list contains at least 441 measures when the web server system is up and running in a stable phase without considering configuration files.

In order to show the benefit of the periodic attestation feature of our remote attestation framework, we set the verifier to attest the attester every 10 s for one day starting from 15:00 and at that time there were 441 measures. The increment of the IMA measure number is shown in Figure 6.9. At 00:02 on the second day, there were two new measures of *anacron*, which performs periodic commands scheduling. Then at 03:21, 12 new measures showed up, including *nice* to set the priority of the

[18] https://github.com/Rohde-Schwarz-Cybersecurity/TrustedGRUB2/

[19] We have no modification or extension of TrustedGRUB2, so we omit its description in this thesis. The introduction can be found in Chapter 3.

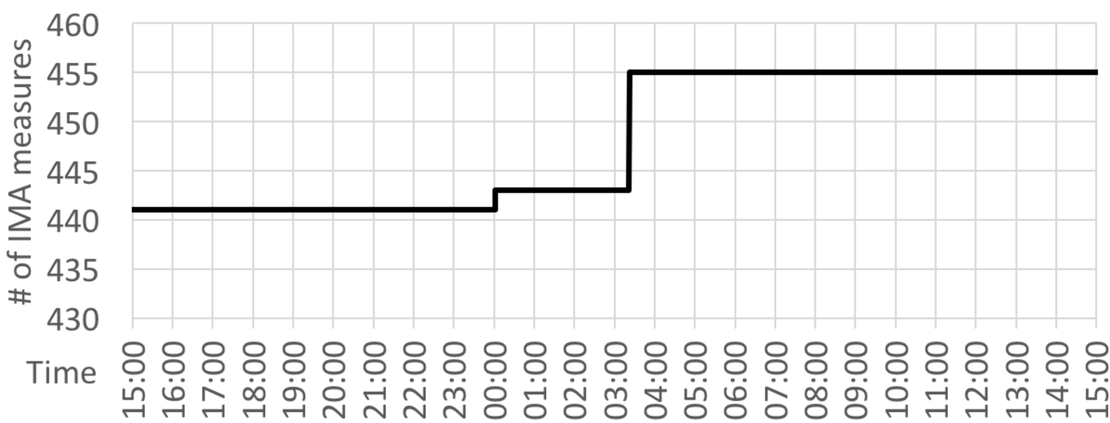

Figure 6.9. Number of IMA measures in a day.

process, *0yum-daily.cron* to check if *yum* needs to automatically update the system, *logrotate* to compress the generated system logs, *renice* and *ionice* to set the priority of processes, *mandb* to update the manual page caches and some dependencies.

These 14 new measures were all called by the automatic services enabled by the operating system, they can be disabled by carefully configuring the attester. However, we note that, even after these 14 unexpected measures were recorded by IMA, the attester was still in *trusted* state, because all these measures were all in the reference database when the corresponding packages were published.

6.2 Trusted Channel

Trusted channel is secure channel with its endpoints attested by each other or by a trusted third party. It can bind the identity of secure channel endpoint to its integrity state and to a specific physical node. In our work, we chose *strongSwan* IPsec implementation because it is open source and supports various operating systems, e.g., Linux 2.6 and later, Android, Mac OS X and Windows.

6.2.1 strongSwan

strongSwan [127] was launched in 2005, and it was developed based on the discontinued FreeS/WAN project[20]. Since the first target platform was Linux, the source code of strongSwan are written in C. Currently its latest version is 5.5.1, which has been ported in other popular operating systems.

strongSwan implements both IKEv1 and IKEv2 key exchange protocols. Starting from version 5.x, it uses a single monolithic IKEv1 and IKEv2 daemon, which is

[20] http://www.freeswan.org/

```
oat-attest {
    load = yes
    oat_server = verifier
    ima_level = 14
    attest_interval = 15
}
```

Figure 6.10. The configuration file of oat_attest plugin in strongSwan.

```
bash oat_pollhosts -h verifier '{"hosts":["ned"],
        "analysisType":"load-time+check-cert,l_req=l4|==,
        cert_digest=cb0c80994ddced19f401debda6e216dd9a6f8c90"}'
```

Figure 6.11. An remote attestation request example to attest IPsec server.

named *charon*. Our trusted channel design requires IKE protocol to authenticate each other and to transfer the public key certificate of the IPsec server to the client, thus our extension starts from the charon daemon.

6.2.2 Extension of strongSwan

Taking advantage of the modular plugin-based architecture of strongSwan, our extension requires no change of the internal code of the charon daemon, we only need to implement a plugin attached to this daemon. Once the charon daemon is activated, it will invoke all plugins set to enabled in order to check different criteria to decide whether key exchange should continue or not.

Following this idea, we developed a plugin called *oat_attest*. It consists of two parts, `oat_attest_plugin` (towards the charon daemon) and `oat_attest_listener` (towards the OAT verifier). `oat_attest_plugin` first creates a plugin interface to the charon daemon, then it generates a listener instance and registers this listener. Afterwards, it destroys the listener to free the memory space when the latter finishes its job. `oat_attest_listener` is invoked by the oat_attest_plugin, and this listener is in charge of generating remote attestation requests to the verifier and authorising further operations only when the received attestation result is positive, otherwise the SA will not be installed or the plugin instructs the daemon to uninstall the current SA.

The configuration file of the oat_attest plugin is shown in Figure 6.10. It tells the charon daemon whether the plugin should be loaded (i.e. load option), which verifier the plugin should contact (i.e. oat_server), which integrity level should be expected (i.e. ima_level) and how often the plugin should contact the verifier asking for the latest attestation result of the IPsec server (i.e. attest_interval in second).

When the oat_attest plugin is invoked, it asks the certificate used by the IPsec server through the charon daemon, afterwards it extracts the common name and computes the SHA1 digest of the certificate. Both values are used in generating the remote attestation request in order to tell the verifier which IPsec server it is

```
Extracted Common Name from peer certificate: ned
SHA1 of the peer certificate: cb0c80994ddced19f401debda6e216dd9a6f8c90

---------- Remote Attestation begins ----------

User terminal -> verifier:
request: { "hosts": [ "ned" ], "analysisType": "load-time+check-cert,l_req=l4|>=, \
    cert_digest=cb0c80994ddced19f401debda6e216dd9a6f8c90" }

Verifier -> user terminal:
response: {"status":"success","code":200,"results":[{"host_name":"ned", \
    "trust_lvl":"trusted"}]}

Host:ned, status:  trusted

---------- Remote Attestation ends ----------

IKE_SA test-ned[15] established between 10.0.2.15[tao]...130.192.1.76 \
    [C=CH, O=strongSwan, CN=ned]
scheduling reauthentication in 9785s
maximum IKE_SA lifetime 10325s
installing DNS server 8.8.8.8 via resolvconf
installing new virtual IP 10.2.1.1
CHILD_SA test-ned{15} established with SPIs c010bd49_i cac30078 and \
    TS 10.2.1.1/32 === 0.0.0.0/0
connection 'test-ned' established successfully
```

Figure 6.12. Remote attestation result from strongSwan log.

connecting to and what certificate the server uses for authentication (Fig 6.11). The request will be filled into a JSON object and periodically sent to a REST API hosted in the verifier via POST requests (Figure 6.12).

6.2.3 Extension of verifier

The extension of the verifier consists of two parts. The first part is to extend the IMA verification script to assess the input parameter as a *must-have* measure of the attester. This extension is straightforward, the verifier only needs to check if the must-have parameter is in the measurement list before giving the final result. The second part is to bind the identity of the IPsec server to the physical node without degrading user experience of the client.

As discussed in Section 4.4.2, there are three possible approaches for the verifier to attest the IPsec server and send back the attestation result to the client: (*i*) the verifier attests the server on-the-fly after receiving request from the client, (*ii*) the verifier periodically attests the server with pre-registered parameters and gives immediate result, and (*iii*) the verifier periodically attests the server and compares the parameters in received attestation request to the stored unknown digests, then gives back the result.

In the trusted channel scenario, the time needed to establish the channel is crucial, i.e. asking a user to wait several seconds before he can connect to the remote server is unacceptable from the user experience point of view. Meanwhile, in the trusted channel scenario, we envision that the only must-have measure of the

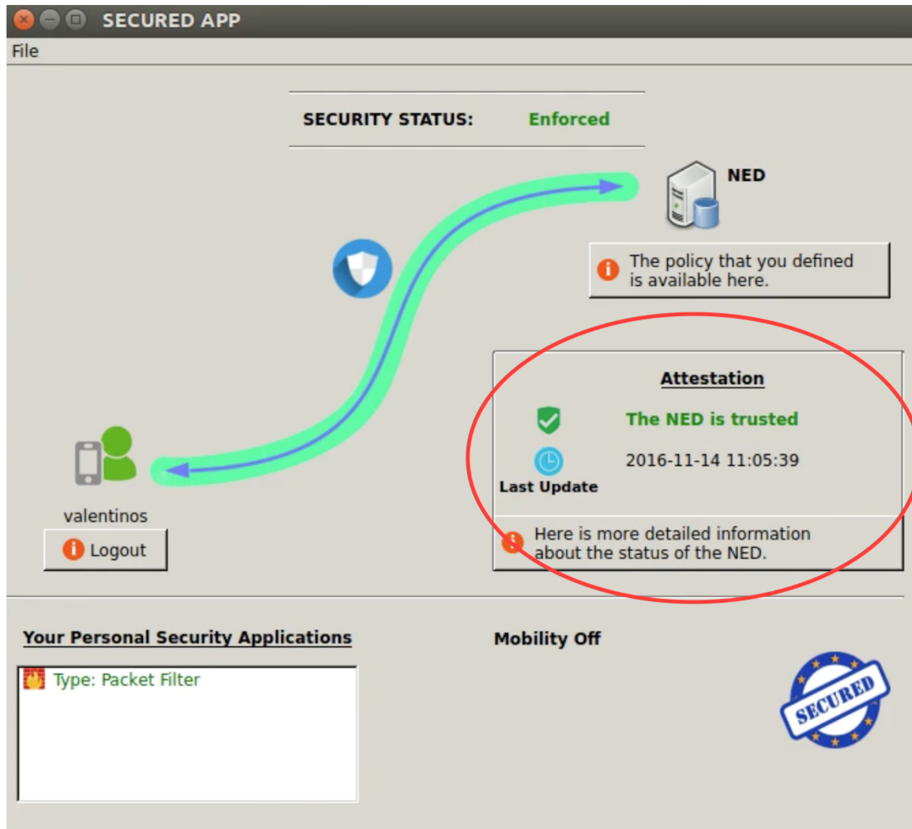

Figure 6.13. The SECURED application with remote attestation enabled.

IPsec server is its public key certificate, which is used to bind the server's identity to its TPM. Thus we chose the second approach and set the IPsec server to register the digest of its certificate when it is enrolled to the OAT verifier, then the verifier uses this digest to generate the remote attestation requests and periodically attest the IPsec server (with the periodic attestation feature introduced in our new OAT framework), and finally, it gives the latest attestation result immediately when it receives a request from clients. For this reason, a new REST API is developed for receiving incoming remote attestation requests and giving the latest attestation result. In order to be consistent with the development language, the new API was developed using *PHP*. Its task is to process the received remote attestation request, distil the IPsec server name (i.e. attester's name) and the digest of its certificate, and finally find the latest attestation result of this server from the OAT verifier's attestation log.

This aforementioned trusted channel implementation is adopted in SECURED project, where the IPsec client now supports Linux (Figure 6.13).

6.2.4 Performance Evaluation

In order to evaluate the performance of our solution, we set up the following testbed:

- an IPsec client running in a virtual machine on top of VirtualBox hypervisor, the virtual machine is assigned with one Intel Core i5-5287U CPU @ 2.9 GHz, 4 GB DDR3 RAM and 16 GB fixed size virtual hard disk;
- an IPsec server (i.e. attester) running on a physical node with Intel Core i7-4600U CPU @ 2.1 GHz, 16 GB DDR3 RAM and 120 GB SSD hard disk;
- an OAT verifier running in a virtual machine on top of KVM hypervisor, the verifier is assigned with 2 Intel Xeon CPU @ 2.4 GHz, 4 GB RAM and 160 GB hard disk.

The IPsec server is running in CentOS 7 operating system using kernel version 3.10.0-327.el7.x86_64, the verifier is running in CentOS 7 using kernel version 3.10.0-123.20.1.el7.x86_64 and the IPsec client is running in Ubuntu 14.04 LTS using kernel version 3.19.0-25-generic. Meanwhile, in order to reduce the network influence in our test, all machines are connected to the same switch.

Test case

The major performance impact of concern in trusted channel scenario is the additional time introduced in the channel setup phase. Thus we provide the time detected in establishing a trusted channel with and without the oat_attest plugin enabled in the IPsec client side.

In strongSwan, the establishment of IPsec channel can be separated into two steps. The first step is to launch an executable called *starter*, and it is used to start, stop, and configure the IKE daemon using its configuration file, i.e. loads the IPsec connection configuration from the `ipsec.conf` file and passes the configuration to the stroke plugin in the keying daemon. The second step is to launch the *stroke* executable, which is invoked to actually start the key exchange process for a specific connection defined in `ipsec.conf`.

These two steps design facilitates our test. Since our primary goal is to test the time difference in the key exchange phase between the IPsec client and the server, we only need to measure the time needed to finish the key exchange phase (i.e. the time needed by stroke) with and without the oat_attest plugin.

We launched the *starter* executable and then use *stroke* to set up a test connection and tear down it after 10 s. We repeated the same operation 10 times with the oat_attest plugin activated and 10 times without the plugin. The minimum, average and maximum time detected in each configuration are presented in Table 6.4.

Using the average time to finish the key exchange without the attestation plugin as the base, the time needed with the plugin increases from 0.357 s to 0.455 s in average, which causes 0.098 s (27.5%) raise. And in the worst case, the time increases 0.151 s (42.3%).

Even if the numbers seem to be non-negligible, we remind that this additional time is only introduced when an IKE authentication is performed, afterwards, the

	w/o oat_attest (s)	w/ oat_attest (s)
MIN	0.290	0.398
AVG	0.357	0.455
MAX	0.424	0.508
index	100%	127%

Table 6.4. Performance difference between attestation plugin disabled and enabled.

attestation requests are sent in background without incurring any other degradation of the user experience. Moreover, since our design and implementation do not require any additional cryptographic operation, there is no performance loss compared to the original secure channel implementation. While from the security point of view, the integrity guarantee of the connected IPsec server (including the platform and the services) and the binding between the server's identity to a specific hardware node are deemed to be crucial and strong, which can protect the clients from various remote attacks.

In the IPsec server side, the newly launched strongSwan service introduces 60 IMA measures which will be inserted in the integrity report and sent to the verifier for evaluation. Especially the public key certificate (i.e. peerCert.der) used by the server to authenticate itself is in the integrity report to bind the identity of the server to the AIK generated by its TPM. Since the remote attestation requests from the verifier are periodic, the overall remote attestation time has no impact to the trusted channel setup. Meanwhile, the performance overhead introduced by IMA and RA is the same as the results shown in Section 6.1.3. Hence in order to omit the duplicates, we do not present the test results for these two aspects in trusted channel scenario.

6.3 Trusted Networks

As described in Chapter 5, in our design of trusted network, we chose the operating system level virtualisation technique to host the network functions. Because the additional hypervisor layer and operating system kernel of each virtual instance are removed, virtual containers are much more lightweight and agile than the conventional hypervisor-based virtual machines.

In our development, we chose *Docker* as the virtual container implementation because it had a boosting increase in the market share during the last years. According to a survey conducted by Datadog, 10.7% of its customers are now using Docker containers in production by June 2016, which is a 30% year-over-year growth in the number [163]. Currently, a lot of Internet service provider companies are researching about SDN/NFV technology based on virtual containers. As a matter of fact, Deutsche Telekom has started experimenting to deploy NFV networks with the help of Docker containers [164].

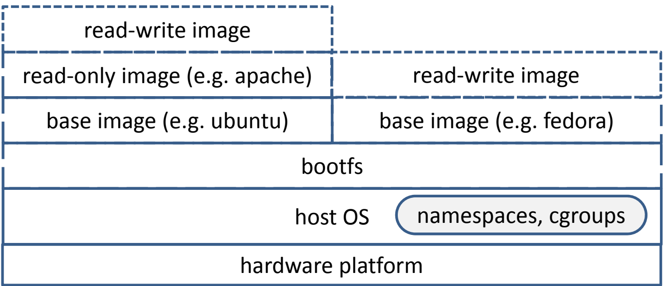

Figure 6.14. The image hierarchy in Docker.

6.3.1 Docker

Docker is an open-source virtual container implementation, and it aims to simplify the deployment of services running inside virtual containers with the help of an additional layer of abstraction called the *Docker Engine*. It uses the resource isolation features of Linux kernel (such as *cgroups* and kernel *namespaces*) and a union-capable filesystem (such as *Device Mapper*) to allow independent containers to run within a single Linux instance.

A fundamental building block in Docker is the concept of *image*: containers themselves are launched from images. Images can be considered as the "source code" for containers, i.e. pre-arranged blocks for providing special functions. Images are built in a layered way, using a *union filesystem* that implements a union mount for different file systems (Figure 6.14).

At the base layer there is a boot filesystem image, called *bootfs*, that works like a typical Linux boot filesystem. In general, bootfs mainly contains the *bootloader* and the *kernel* of a container. When a container has booted and its kernel has been loaded into the memory, this layer is unmounted to free space in memory.

The second layer is composed by one or more root filesystems images, called *rootfs* or *base image*. Each base image hosts one kernel of the specific container's operating system. These layers are read-only, and on top of them other additional read-only layers are hosted. Then, every time a new container is launched, the Docker daemon or Docker engine constructs a read-write layer on the top of the image hierarchy, in which all processes belonging to this container are executed.

Docker works in a "copy-on-write" pattern: all lower layers are set with read-only access, thus when a file is going to be modified, it is beforehand copied into the topmost read-write layer and then modified there. The original version of the modified file still exists after the change, but hidden behind the copy. In this way, each new container only has a read-write layer at top of the image hierarchy, and more containers can share the same image layers underneath. This permits Docker to launch containers in a timely fashion and saves a lot of resources, e.g., disk and memory.

Each image is managed by a storage driver which is responsible to manage the image layers and perform the copy-on-write operations. Storage drivers used by

Docker in general are *Aufs*[21] (the first supported by Docker), *Device Mapper*[22] and *Btrfs*[23], where the former two are the most popular ones. Depending on the Linux distribution, Docker uses different storage drivers as its default option (e.g., Aufs on Ubuntu, Device Mapper on CentOS), but this is not a constraint as it is a configurable parameter during installation. Comparing Aufs and Device Mapper, they work at different abstraction level. Aufs operates at file level: if a file is going to be modified, then the file is entirely copied inside the read-write layer and modified there. Device Mapper operates at block level: when a file is going to be modified, only the blocks of interests are moved to the read-write layer. Thus, the latter one is a better choice for performance, because the latency for the copy-on-write operations is reduced. For this reason, in the rest part of this section, we refer Device Mapper as the storage driver of Docker.

At this point, when a virtual container is activated by the Docker daemon (with `run` command), Device Mapper creates a new virtual device for the container, giving it a device number (hereafter also referred as dev-id) that is composed of a major number and a minor number. This behaviour is inherited from Linux kernel, where a device is identified by a major number that typically identifies the driver used for managing the device, and a minor number that exactly identifies the device referred. The identifier is made up of 32 bits, where the most significant 12 bits represent the *major number* and the least significant 20 bits identify the *minor number*.

In particular, Device Mapper uses either 252 or 253 as the major number, depending on the package compiled for each Linux distribution, while as the minor number it assigns a growing positive number for each new virtual device. For example, considering 253 as the major number (as it happens in CentOS), the starting pool will have a device number like 253:0, the first container created will be identified as 253:1, the second as 253:2 and so on. The device number is an important parameter in our solution as it is used to identity each virtual container in the host operating system.

6.3.2 Enabling Remote Attestation in Docker containers

As described in Chapter 5, we have to make modification in three different components in our previous remote attestation framework, the IMA module, the OAT agent, and the IMA verification script.

The modification to IMA

Regarding to the IMA module, we introduced a new "field" to represent the device number associated with the measured file with the help of a new IMA template

[21] http://aufs.sourceforge.net/

[22] ftp://sources.redhat.com/pub/dm/

[23] https://btrfs.wiki.kernel.org/index.php/Main_Page/

called *ima-cont-id*. Since our goal is to introduce a new IMA template with the new field, we started our modification with the file `ima_template.c`, in which there are two arrays we need to modify to add our new template name and its fields:

- `ima_template_desc`: which defines a template and inside it stores the name, the format string and an array of type "ima_template_field structures" (one for each field that makes up the template);
- `ima_template_field`: which defines a specific field and inside it stores the name and the pointer to two functions, one to retrieve information to be assigned to the field, and one to print the output.

Afterwards, we extended the `ima_template_lib.c` file by introducing a new function called *ima_eventdev_id_init*, which is the initial function to include the file device number as part of the template data. Once the new template is activated, this function receives two parameters, which are automatically passed by the IMA module at the time of a new measure needs to be inserted into the list. They are *event_data* (contains information about the file being measured) and *field_data* (contains information to be filled in the measurement list). Following the newly defined IMA template, this function extends the *field_data* by adding a new attribute containing the dev-id associated with the file via its *inode* retrieved from the *event_data*.

After the implementation is finished, we only need to recompile the kernel and activate the new template by adding *ima_template=ima-cont-id* in the attester's boot configuration.

The modification to the OAT framework

The modification to the OAT framework is straightforward, since only the remote attestation agent needs to be modified to add the new attributes in the integrity reports.

`Core_Integrity_Manifest_v1_0_1.xsd` is the XML template used to create integrity reports, and the new elements need to be introduced into the template are *Container* and *Host*. The Container element has an additional attribute called *Id*, which is to store the container's UUID, that recorded by its manager. Further, the element has a child tag called *DevId*, which is the dev-id associated to this container assigned by Device Mapper in the container host. In the same way, Host element also has a child tag called *DevId*, which is the dev-id associated to the devices belonging to the host operating system.

Moreover, in order to be backwards compatible, we added a new property called *AddContainerAnalysisSupport* in the configuration file of the remote attestation agent. Once the property is set to true, *retrieveHostDevices* (to obtain a list of all the dev-id associated to host devices) and *retrieveMapDmContainers* (to obtain the mapping between container id and its associated dev-id) are invoked to fill the new elements in integrity reports.

```
bash oat_pollhosts -h verifier '{"hosts":["ned"],
        "analysisType":"VALIDATE_PCR;load-time+cont-check,l_req=l2_ima_all_ok|==,
        cont-list=7b7e912abda3+3a5f603b86c4"}'
```

Figure 6.15. A remote attestation request example to attest Docker container host.

The modification to IMA verification script

As presented in Chapter 3, the OAT verifier in our new framework is capable of defining custom analysis type. Thus we introduced a new analysis type called *cont-check*, which is specifically called to analyse the IMA measures belonging to a set of virtual containers. In order to call this analysis type, an additional parameter is required in the remote attestation request, which is called *cont-list*. This new parameter lists the container IDs which need to be attested (e.g., Figure 6.15).

After the verifier receives the integrity report from an attester, it calls a new function called *ContainerCheckAnalysis*, which corresponds to the cont-check analysis type. This function is developed based on the original IMA measure verification tool, and at first it looks for the newly introduced elements in the reports and creates a mapping between container ID and its associated device ID. Then when the IMA measurement entries stored in integrity report are decoded from Base64 to ASCII, the parser calls the generated mapping to distil the IMA measures belonging to the containers in the cont-list and the ones belonging to the host operating system with the help of the newly introduced elements in integrity reports. Further, these IMA measures will be queried to the executable reference database and configuration whitelist. If any of these measures is unknown, the verifier will tell on which device the unknown measure is generated, i.e. either on the host or on a device belonging to a specific container and the container's ID.

6.3.3 Performance Evaluation

In order to evaluate the performance of our solution, we set up the following testbed:

- a Docker host running on a node with Intel Core i7-4600U CPU (2 core, 4 threads), @ 2.1 GHz, 16 GB DDR3 RAM and 120 GB SSD hard disk;
- an OAT verifier running in a virtual machine on top of a KVM hypervisor. The verifier is assigned with 2 Intel Xeon CPU @ 2.4 GHz, 4 GB RAM and 160 GB hard disk.

Both machines are running the CentOS 7 operating system, with the Docker host using a custom compiled kernel based on version 4.4, Docker version 1.9.1 with Device Mapper version 1.02.107, and the verifier using original kernel version 3.10.0-123.20.1.el7.x86_64.

```
docker run -d centos /bin/sh -c "while true; do echo hello world; sleep 1; done"
```

Figure 6.16. Starting a Docker container with simple task.

```
# sudo docker run -d centos service
33c01aa75a3402eb012b7aaba4601ecdf781fc21682e6130877b1e0122360227
Error response from daemon: Cannot start container
33c01aa75a3402eb012b7aaba4601ecdf781fc21682e6130877b1e0122360227:
adding interface veth767e1e6 to bridge docker0 failed:  exchange full
```

Figure 6.17. The error given when the maximum number of active Docker containers is reached.

Performance limitation of Docker

In order to evaluate the maximum number of containers that Docker can run in a single host, we created a simple script to launch as many containers as possible, each of them is executing just a basic test function, writing to standard output the "hello world" string every second (Figure 6.16).

At first, we assumed the maximum number of launched containers on a single platform is limited by the size of the minor number in the dev-id assigned by the Device Mapper, i.e. 20 bits, or the resources can be offered by this platform. On the contrary, we found out that the maximum number of active containers in a single platform is limited to 1023, even if the computing resources of the platform are not saturated. And when the maximum number of container is reached, an error is shown (Figure 6.17).

We discovered that this limitation lies on the number of network interfaces the Linux kernel can assign to the virtual instances, either virtual containers or virtual machines. Since the size of the network bridge identifier BR_PORT_BITS in the Linux kernel is 10 bits, the maximum number of containers or any sort of virtual instances that can be launched in a single default platform is 1023 [165].

However, according to the discussion in [166], the maximum number of Docker instances can be increased through two practical approaches:

- use Open vSwitch[24] to assign the network resources, which allows for activating 2^{16} ports;
- disable the Spanning Tree Protocol and tweak the BR_PORT_BITS size, and theoretically it also allows for activating 2^{16} ports.

[24] http://openvswitch.org/

In any case, the maximum number of supported container instances activated on a single platform is less than the minor number of dev-id supported by the Device Mapper.

Performance impact of IMA and RA

The general performance impact of the attester with IMA and RA has been presented in Section 6.1.3, here we show our test results of specific Docker environment. Especially we show the time differences in sending `run`, `stop` and `remove` commands with and without the two additional features.

Similar to previous test cases, to provide adequate coverage and avoid measuring unnecessary files, we used the execution policy presented in Section 3.5.4. In our test, with the execution policy, each container adds seven new IMA measures in the integrity report, because the function running inside the containers is very simple (Figure 6.16). More specifically, the following measures are added by each container: four for various *glibc* standard and shared libraries (used by the Linux kernel), one for the *ncurses* library (providing the text-based user interface), one for the *shell* itself and a last one for the *sleep* binary (as it is not a built-in command in the shell). Thus, the number of IMA measures grows linearly with the number of containers launched in the platform.

If more complex services are deployed inside the container, the number of measured binaries would increase even by the orders of magnitude. For example, with a newly built Apache server image (without any additional component, like a SQL database) the number of detected IMA measures is 123.

To test the performance penalty to the basic Docker container operations introduced by IMA and RA in the attesting platform, we launched 512 containers sequentially and then we stopped them one by one. We repeated this test ten times, to get statistically meaningful results, and repeated the test once with both IMA and RA deactivated, once with IMA activated and RA deactivated and once with both IMA and RA activated.

Three essential operations are involved in the test: *run* (to start a container), *stop* (to send `SIGTERM` to the process running inside a container) and *remove* (to remove a container along with its assigned resources, e.g., disk storage and network bridge). The minimum, average and maximum time to finish these three operations are given respectively in Table 6.5, 6.6, and 6.7. The average time for these operations is presented graphically in Figure 6.18, 6.19, and 6.20, where the X-axis shows the number of active containers in the attesting platform and the Y-axis is the time to finish the operation. These tables and graphs are to be interpreted as time needed to perform an operation given a certain number of containers already active. For example, values for 256 containers represent respectively the time needed to create a new container (hence to have 257 active containers), to stop a container (hence to have 256 containers with only 255 active), and to remove a container (hence to have 255 active containers).

start # containers		*1*	*2*	*4*	*8*	*16*	*32*	*64*	*128*	*256*	*512*
w/o IMA w/o RA	min	1.19	1.13	1.14	1.19	1.17	1.27	1.23	1.42	1.62	2.14
	avg	1.28	1.22	1.17	1.24	1.26	1.37	1.34	1.56	1.75	2.39
	max	1.37	1.37	1.24	1.29	1.44	1.45	1.41	1.77	1.86	2.73
w/ IMA w/o RA	min	1.29	1.21	1.24	1.27	1.25	1.32	1.42	1.52	1.66	2.37
	avg	1.39	1.31	1.38	1.32	1.34	1.40	1.50	1.55	1.75	2.45
	max	1.53	1.55	1.49	1.39	1.40	1.51	1.69	1.57	1.90	2.59
w/ IMA w/ RA	min	1.24	1.23	1.22	1.28	1.25	1.37	1.39	1.43	1.65	2.16
	avg	1.37	1.24	1.32	1.36	1.44	1.44	1.46	1.51	1.77	2.38
	max	1.81	1.25	1.40	1.55	1.72	1.51	1.67	1.70	2.03	2.58
difference w/ IMA		+0.11	+0.09	+0.19	+0.08	+0.08	+0.03	+0.16	-0.01	+0.00	+0.06
difference w/ IMA+RA		+0.09	+0.02	+0.15	+0.12	+0.18	+0.07	+0.12	-0.05	+0.02	-0.01

Table 6.5. Time (in seconds) to start a container with and without IMA and RA, and the difference computed between the average values.

stop # containers		*1*	*2*	*4*	*8*	*16*	*32*	*64*	*128*	*256*	*512*
w/o IMA w/o RA	min	10.55	10.55	10.55	10.46	10.48	10.46	10.45	10.50	10.50	10.57
	avg	10.59	10.57	10.58	10.55	10.54	10.56	10.52	10.57	10.57	10.68
	max	10.69	10.59	10.59	10.59	10.56	10.63	10.57	10.63	10.66	10.84
w/ IMA w/o RA	min	10.54	10.49	10.48	10.51	10.49	10.53	10.46	10.46	10.50	10.60
	avg	10.57	10.55	10.56	10.56	10.55	10.58	10.56	10.59	10.58	10.69
	max	10.60	10.59	10.62	10.60	10.58	10.64	10.66	10.66	10.67	10.83
w/ IMA w/ RA	min	10.56	10.54	10.55	10.56	10.55	10.57	10.57	10.61	10.63	10.69
	avg	10.58	10.56	10.57	10.58	10.58	10.58	10.58	10.61	10.66	10.77
	max	10.65	10.59	10.59	10.62	10.62	10.60	10.60	10.63	10.70	10.85
difference w/ IMA		-0.02	-0.02	-0.02	+0.01	+0.01	+0.02	+0.04	+0.02	+0.01	+0.01
difference w/ IMA+RA		-0.01	-0.01	-0.01	+0.03	+0.04	+0.02	+0.06	+0.04	+0.09	+0.09

Table 6.6. Time (in seconds) to stop a container with and without IMA and RA, and the difference computed between the average values.

The main difference in performance with and without IMA is related to the start of a new container (Table 6.5 and Figure 6.18) which requires about 79 ms more with IMA and 71 ms in the case both IMA and RA are activated. Apart for statistical fluctuations, it is independent of the number of active containers. This is expected because the start operation directly adds workload to the attesting platform and each start operation needs to measure and extend seven new IMA measures into the TPM, so we can estimate that each new measure requires about 11 ms.

An interesting behaviour is that the start time of the first four containers without IMA is decreasing (Figure 6.18) because the CPU of the attesting machine has two cores and supports four threads, hence the first four containers have the privilege to share the resources equally without competing for CPU cycles. But when IMA comes into play the behaviour changes and is more unpredictable as another important process (IMA itself) comes into play.

On another hand, the differences for the stop and remove operations with and without IMA and RA are minor (if any), since these two operations do not involve any IMA nor RA functionality.

The time to stop a container is always around 10 s (Table 6.6 and Figure 6.19) because the Docker stop operation sends `SIGTERM` to the main process of the container. However, in our test, this signal cannot stop the process executed in the container, so it needs to wait the predefined threshold (with default value 10 s) to send `SIGKILL` and kill the whole container process.

remove # containers		*1*	*2*	*4*	*8*	*16*	*32*	*64*	*128*	*256*	*512*
w/o IMA w/o RA	min	2.66	2.61	2.64	2.74	2.88	3.11	3.66	5.35	7.39	11.73
	avg	2.72	2.71	2.75	2.82	2.96	3.26	3.72	5.45	7.69	12.03
	max	2.83	2.84	2.81	2.90	3.09	3.54	3.84	5.55	8.14	12.17
w/ IMA w/o RA	min	2.56	2.62	2.63	2.75	2.89	3.10	3.62	5.32	7.40	11.87
	avg	2.64	2.67	2.69	2.84	2.95	3.15	3.90	5.42	7.68	12.06
	max	2.72	2.70	2.77	2.97	3.09	3.29	4.59	5.49	8.11	12.29
w/ IMA w/ RA	min	2.63	2.62	2.60	2.74	3.00	3.14	3.63	5.34	7.46	11.66
	avg	2.77	2.73	2.77	2.85	3.07	3.24	3.81	5.55	7.59	11.81
	max	2.95	2.87	3.01	2.93	3.15	3.41	4.08	5.79	7.77	11.97
difference w/ IMA		-0.08	-0.04	-0.06	+0.02	-0.01	-0.11	+0.18	-0.03	-0.01	+0.03
difference w/ IMA+RA		+0.05	+0.02	+0.02	+0.03	+0.11	-0.02	+0.09	+0.10	-0.10	-0.22

Table 6.7. Time (in seconds) to remove a container with and without IMA and RA, and the difference computed between the average values.

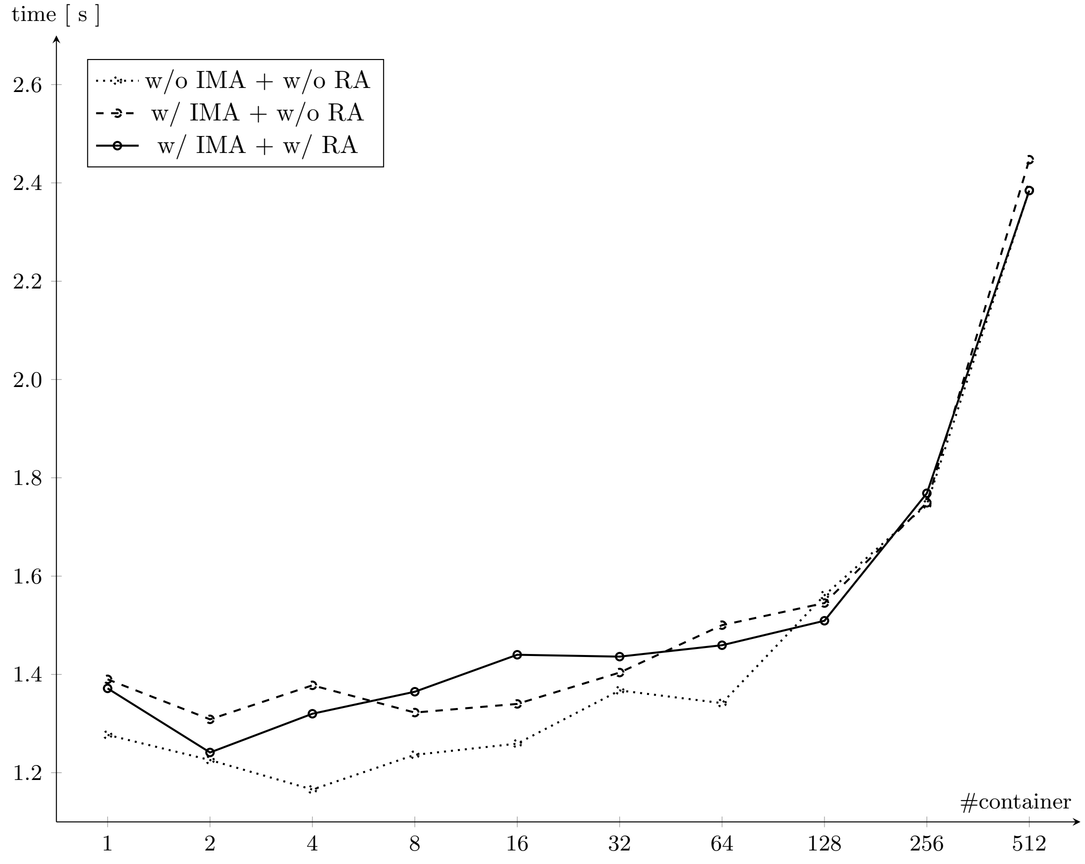

Figure 6.18. Time to start a container with and without IMA and RA.

The remove operation removes the resources assigned to a container and the time needed grows linearly with the number of active containers (Table 6.7 and Figure 6.20). As described in Section 6.3.1, Device Mapper is a block-level copy-on-write system, which creates a "pool" of space with the help of two sparse files, *data* and *metadata*. When a container is removed, Device Mapper needs to find the location of the blocks belonging to this container from the metadata file and then remove these blocks in the data file. So, the performance of sparse files limits the

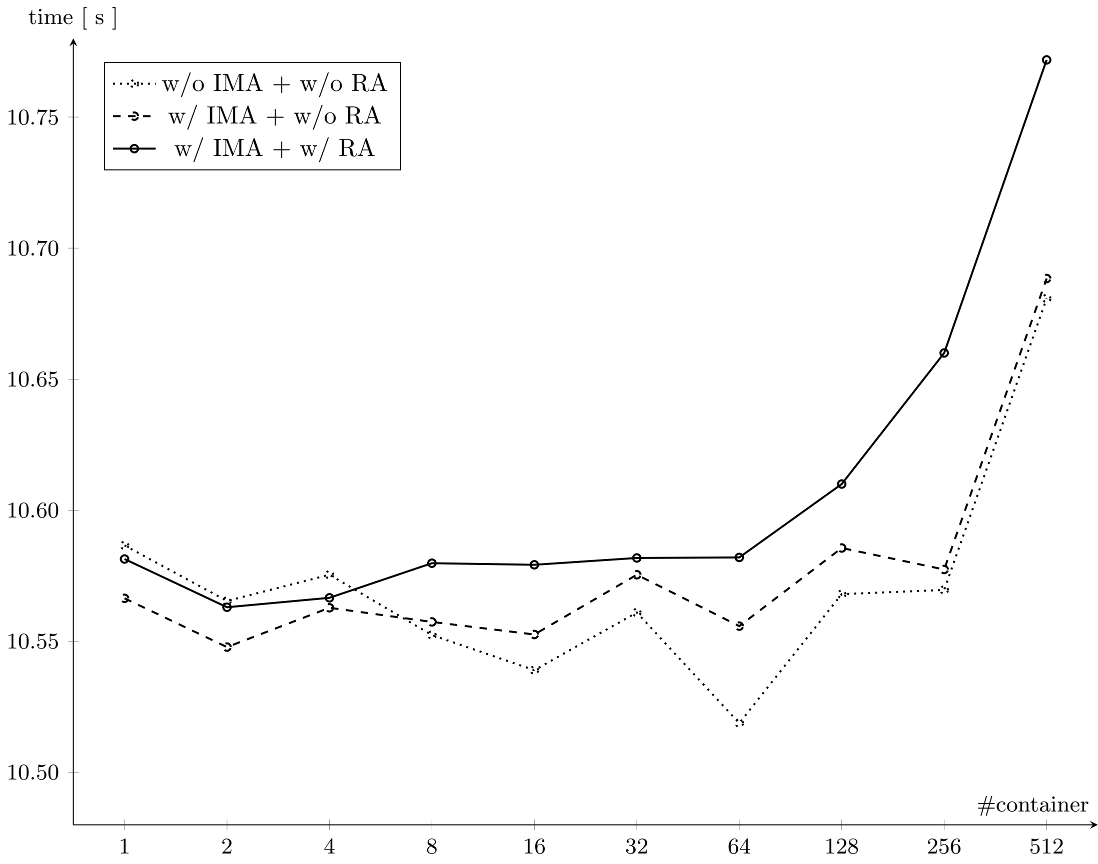

Figure 6.19. Time to stop a container with and without IMA and RA.

overall performance of the remove operation.

Performance in a real-world test case

Similar to the previous test case (Section 6.1.3), we also simulated a real-world application, with the containers set to perform operations similar to those of a HTTP server reacting to incoming requests. In this test, each container creates a single 1 MB file and then starts an infinite loop to repeatedly compute the SHA512 digest of this file. At every computation a counter is updated on disk. This approach mimics the case of a server reading a request, performing a computation and writing log data on disk. To discard the influence of the network interface (which could be a bottleneck with many containers) we deliberately decided to perform only local I/O operations.

To start and stop all operations simultaneously (as we have seen that containers must be started sequentially and this takes a lot of time), all the containers are mapped to a volume linked to the same folder, waiting for a trigger file to be present. Once the trigger is created, all the containers start their computation simultaneously and at every step test if the trigger file still exists. If not, the container enters in

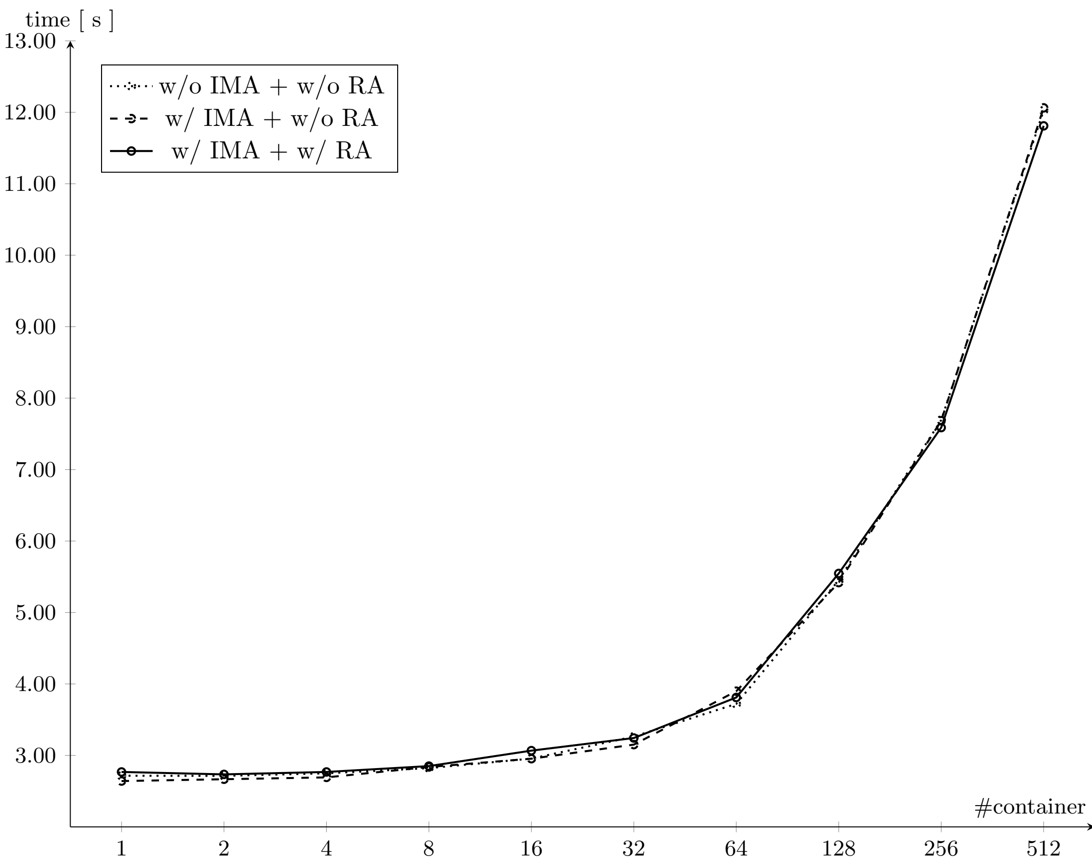

Figure 6.20. Time to remove a container with and without IMA and RA.

	no IMA, no RA	IMA, no RA	IMA, RA
min	134,855	133,849	133,799
avg	135,284	134,623	134,440
max	135,602	135,626	134,769
index	100%	99.51%	99.37%

Table 6.8. Number of operations and performance index under three settings in Docker environment.

a sleep mode, stopping the computation and checking periodically if the trigger is re-created to repeat the process again.

In this test we started 256 containers to execute the aforementioned task. Same as the previous test, we repeated the test ten times for each of three different settings: plain Docker environment, Docker with IMA, and finally Docker with IMA and RA agent. Each run of the test lasts 300 s, and the recorded results are presented in Figure 6.8.

It is clear that the differences among these three settings are minor: the overall performance is only slightly affected by the introduction of the remote attestation. Considering as a reference the average value of performed operations without IMA

# containers		*0*	*32*	*128*	*256*	*512*
time	attester	3.71	3.80	4.05	4.99	6.56
	verifier	1.56	1.63	1.77	1.95	2.35
	total	5.27	5.43	5.82	6.94	8.91

Table 6.9. Average time (in seconds) required to complete a remote attestation request with different number of active containers.

and RA, the performance only drops by 0.63% in the full case (i.e. with IMA and RA enabled). So we conclude that the performance impact of our proposed integrity verification feature is nearly optimal and suitable for application in real-world cases, with respect to the overhead on the attested platform.

Overall remote attestation performance evaluation

Let's now look at the other main component of our architecture: the verifier, which hosts the reference database and needs to query it with the IMA measures received from the attesters.

Table 6.9 presents the results of our test about the global performance of the remote attestation process with different number of active containers (all active containers and the host system are attested in this case). Be reminded that each container introduces seven new IMA measures, and in the initial state the Docker host has around 900 IMA measures (i.e. produced by the host system).

The processing time is given as a total and also split into two components related to the attester and the verifier, since the time for transmitting the integrity report is very small and hence negligible. With 512 containers the number of IMA measures in our test is around 4500 and the size of the report is about 1.38 MB, which only needs around 0.1 s to be transmitted.

The total time (and also the individual components) grow roughly linearly with the number of active container (given that in this case each container produces the same number of IMA measures). The time taken by the attester is related to the quote operation and the generation of the integrity report. In this test scenario, the attester is much more powerful than the attester in the Section 6.1.3, thus the time to generate an integrity report is much shorter. While the time of the verifier includes the efforts to verify the signature of the quote output, to validate the integrity of all IMA measures in the report and to compare the IMA measures with the reference database. Even with 512 active containers, the time needed to attest the internal of all of them is less than 10 s that we deem a good result weighting the integrity guarantee provided by our solution.

In case this performance is not considered adequate, there are margins for improvement. On the attester side, the hard limit is the time taken to perform the digital signature over the integrity report by the TPM, which in our machine takes about 2 s. The rest of the time is spent in creating the integrity report and this

time can be reduced by creating partial integrity reports (i.e. containing only those measures added since the last integrity report). In case a high-frequency verification is desirable, then dedicating a whole core to the Docker host system would be an option, so that the remote attestation agent would not have to compete with the containers for the CPU. Additionally re-implementing the remote attestation agent as native code would improve performance as current implementation is in Java, which is notoriously slower than C programs.

Chapter 7

Conclusion

In this work, we have proposed an innovative remote attestation framework, which is able to attest the integrity state not only of the boot phase of a physical platform but also of the services running in its application layer, with hardware-based methods. The framework allows user of the services or administrator of the distributed system to know the integrity state of the nodes. Thus they can make an informed decision whether to trust the system with their private data or the responses of the system based on its integrity state. Moreover, in our framework, the remote attestation burden is offloaded to a trusted third party, i.e. the verifier, leading to a minimum amount of performance loss in the attesting platform side, where the actual services are running.

Based on our remote attestation framework, we presented our solution to create trusted channels, which are secure channels with the endpoint integrity attested and bound to a specific hardware platform. This solution allows secure channel users to know the integrity state of the server they are going to connect before the connection is actually established, i.e. before the potential damage is done.

We also proposed a solution to attest the services running in virtual containers with hardware-based evidence. This feature is especially important in SDN/NFV environments, since the software modules are in general running in virtual instances instead of directly on the hardware platform. Moreover, in this solution virtual containers can be differentiated from each other. Thus in the case that one container is compromised, only this container is required to be paused to restore the trust state without the need to reset the whole physical platform. Implicitly, with the hardware-based integrity evidence to ensure the correct behaviour of the network functions, it enables the possibility of creating trusted softwarised networks.

We not only developed the theoretical models, but also provided their actual implementations. The source code of the prototypes are released under open source licenses of the SECURED project. The prototypes have been presented in two ETSI NFV security group meetings and they are going to be deployed in the new

SHIELD project[1]. The performance evaluation of all three prototypes indicates their feasibilities to be applied in real-world scenarios.

Regarding to future work, although the integrity of the software module and the distributed system is able to provide certain level of trust, trust is much more difficult to achieve. The correct execution of software modules does not make them invulnerable to system design errors or implementation problems. Hence, formal verification [167] and continuous integration [168] techniques must also be considered in real system design and implementation. On the other hand, with regards to the current limitation of measuring dynamic or unstructural data, which is not able to be measured with original IMA, we are investigating the *enhanced Berkeley Packet Filtering* (eBPF) tracing tool [169], which is mainstreamed to the Linux 4.x series kernels. We are expecting to complement eBPF with IMA in order to provide a complete coverage of both structural and unstructural data which affect the behaviour of the software services.

[1] https://www.shield-h2020.eu/

Acronyms

AIK	Attestation Identity Key
BIOS	Basic Input/Output System
CA	Certificate Authority
CPU	Central Processing Unit
CRTM	Core Root of Trust for Measurement
CSA	Cloud Security Alliance
DoS	Denial of Service
EK	Endorsement Key
ETSI	European Telecommunications Standards Institute
HSPL	High Security Policy Language
ICT	Information and Communications Technology
ID	IDentity
IKE	Internet Key Exchange
IMA	Integrity Measurement Architecture
IoT	Internet of Things
IP	Internet Protocol
IPsec	Internet Protocol Security
IR	Integrity Report
MITM	Man In The Middle
NAPT	Network Address and Port Translation
NAT	Network Address Translation

NFV	Network Functions Virtualization
NFVI	Network Functions Virtualization Infrastructure
NIST	National Institute of Standards and Technology
PBNM	Policy-Based Network Management
PCR	Platform Configuration Register
RA	Remote Attestation
RFC	Request for Comments
RoT	Root of Trust
RTM	Root of Trust for Measurement
RTR	Root of Trust for Reporting
RTS	Root of Trust for Storage
SA	Security Association
SDN	Software-Defined Network
SGX	Software Guard eXtension
SRK	Storage Root Key
SSH	Secure Shell
SSL	Secure Sockets Layer
TC	Trusted Computing
TCG	Trusted Computing Group
TLS	Transport Layer Security
TP	Trusted Platform
TPM	Trusted Platform Module
TTP	Trusted Third Party
TXT	Trusted eXecution Technology
VM	Virtual Machine
VPN	Virtual Private Network
vTPM	virtual Trusted Platform Module
XML	Extensible Markup Language

Bibliography

[1] B.McCouch, "SDN, Network Virtualization, And NFV In A Nutshell" 2014, http://www.networkcomputing.com/networking/sdn-network-virtualization-and-nfv-nutshell/1655674152.

[2] K.Kirkpatrick, "Software-defined Networking", Communications of the ACM, Vol. 56, No. 9, September 2013, pp. 16–19, doi:10.1145/2500468.2500473.

[3] C.Donley, J.Berg, M.Kloberdans, "Network function virtualization (nfv)" 2016, https://www.google.com/patents/US20160006696.

[4] European Telecommunications Standards Institute, "Network Functions Virtualisation" https://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[5] M. K.Srinivasan, K.Sarukesi, P.Rodrigues, M. S.Manoj, P.Revathy, "State-of-the-art Cloud Computing Security Taxonomies: A Classification of Security Challenges in the Present Cloud Computing Environment", ICACCI'12: International Conference on Advances in Computing, Communications and Informatics, Chennai, India, August 3-5 2012, pp. 470–476, doi:10.1145/2345396.2345474.

[6] K.Hashizume, D. G.Rosado, E.Fernández-Medina, E. B.Fernandez, "An analysis of security issues for cloud computing", Journal of Internet Services and Applications, Vol. 4, No. 1, February 2013, pp. 1–13, doi:10.1186/1869-0238-4-5.

[7] S.Singh, Y.-S.Jeong, J. H.Park, "A survey on cloud computing security: Issues, threats, and solutions", Journal of Network and Computer Applications, Vol. 75, September 2016, pp. 200–222, doi:10.1016/j.jnca.2016.09.002.

[8] P. B.Kurtz, "Cybersecurity: Change or Die" 2016, https://blog.cloudsecurityalliance.org/2016/09/09/cybersecurity-change-die/.

[9] Akamai, "Akamai's Internet security executive review" 2016, https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/akamai-q2-2016-internet-security-executive-review.pdf.

[10] A.Robertson, "Google Docs users hit with sophisticated phishing attack" 2017, https://www.theverge.com/2017/5/3/15534768/google-docs-phishing-attack-share-this-document-with-you-spam.

[11] Trusted Computing Group https://www.trustedcomputinggroup.org/.

[12] Trusted Computing Group, "Trusted Platform Module (TPM) Summary" http://www.trustedcomputinggroup.org/trusted-platform-module-tpm-summary/.

[13] W.Jackson, "Engineer shows how to crack a 'secure' TPM chip" 2010, https:

//gcn.com/articles/2010/02/02/black-hat-chip-crack-020210.aspx?m=1.

[14] S.Thielman, "Yahoo hack: 1bn accounts compromised by biggest data breach in history" 2016, https://www.theguardian.com/technology/2016/dec/14/yahoo-hack-security-of-one-billion-accounts-breached.

[15] T.Dierks, E.Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC-5246, August 2008.

[16] S.Bellovin, "Guidelines for Specifying the Use of IPsec Version 2", RFC-5406, February 2009.

[17] T.Su, A.Lioy, N.Barresi, "Trusted Computing Technology and Proposals for Resolving Cloud Computing Security Problems", Cloud Computing Security: Foundations and Challenges (J. R.Vacca), Taylor & Francis Group, CRC Press 2016, pp. 345–358, doi:10.1201/9781315372112-27.

[18] A.Filograna, P.Smiraglia, C.Gilsanz, S.Krco, A.Medela, T.Su, "Cloudification of Public Services in Smart Cities the CLIPS project", ISCC'16: IEEE Symposium on Computers and Communication, Messina, Italy, June 27-30 2016, pp. 153–158, doi:10.1109/ISCC.2016.7543731.

[19] R.Bonafiglia, F.Ciaccia, A.Lioy, M.Nemirovsky, F.Risso, T.Su, "Offloading personal security applications to a secure and trusted network node", NetSoft'15: 1st IEEE Conference on Network Softwarization, London, UK, April 13-17 2015, pp. 1–2, doi:10.1109/NETSOFT.2015.7116171.

[20] L.Jacquin, A.Lioy, D. R.Lopez, A. L.Shaw, T.Su, "The Trust Problem in Modern Network Infrastructures", Cyber Security and Privacy: 4th Cyber Security and Privacy Innovation Forum, CSP Innovation Forum 2015, Brussels, Belgium April 28-29, 2015, Revised Selected Papers (F.Cleary, M.Felici), Springer 2015, pp. 116–127, doi:10.1007/978-3-319-25360-2_10.

[21] F.Valenza, T.Su, S.Spinoso, A.Lioy, R.Sisto, M.Vallini, "A formal approach for network security policy validation", Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), Vol. 8, No. 1, March 2017, pp. 79–100.

[22] P.England, B.Lampson, J.Manferdelli, M.Peinado, B.Willman, "A Trusted Open Platform", Computer, Vol. 36, No. 7, July 2003, pp. 55–62, doi:10.1109/MC.2003.1212691.

[23] Trusted Computing Group, "TCG Specification Architecture Overview" http://www.trustedcomputinggroup.org/wp-content/uploads/TCG_1_4_Architecture_Overview.pdf.

[24] Trusted Computing Group, "TPM Main Design Principles - revison 116" 2011, https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-1-Design-Principles_v1.2_rev116_01032011.pdf.

[25] B.Schneier, "SHA-1 Collision Found" 2017, https://www.schneier.com/blog/archives/2017/02/sha-1_collision.html.

[26] Intel Corporation, "Solutions and Products with Intel Trusted Execution Technology (Intel TXT)" http://www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/where-to-buy-isv-txt.html.

[27] Fedora, "What is UEFI Secure Boot?" https://docs.fedoraproject.org/en-US/Fedora/18/html/UEFI_Secure_Boot_Guide/chap-UEFI_Secure_Boot_Guide-What_is_Secure_Boot.html.

[28] E.Brickell, J.Camenisch, L.Chen, "Direct Anonymous Attestation", CCS'04: 11th ACM conference on Computer and Communications Security, Washington DC, USA, October 25-29 2004, pp. 132–145, doi:10.1145/1030083.1030103.

[29] J.Camenisch, A.Lysyanskaya, "Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials", CRYPTO'02: 22th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22 2002, pp. 61–76, doi:10.1007/3-540-45708-9_5.

[30] J.Camenisch, A.Lysyanskaya, "A Signature Scheme with Efficient Protocols", SCN'03: 3rd International Conference of Security in Communication Networks, Amalfi, Italy, September 11-13 2003, pp. 268–289, doi:10.1007/3-540-36413-7_20.

[31] Trusted Computing Group, "TCG Infrastructure Working Group Integrity Report Schema, Specification Version 2.0" 2011, http://www.trustedcomputinggroup.org/wp-content/uploads/IWG_Integrity_Report_Schema_v2.0.r5.pdf.

[32] Trusted Computing Group, "Trusted Network Communication" http://www.trustedcomputinggroup.org/work-groups/trusted-network-communications/.

[33] OpenStack, "Trusted Compute Pools" http://docs.openstack.org/admin-guide/compute-security.html.

[34] "Kernel-based Virtual Machine" http://www.linux-kvm.org/page/Main_Page.

[35] "Xen project" https://www.xenproject.org/.

[36] "Docker - home page" https://www.docker.com/.

[37] J.Geffner, "Virtualized Environment Neglected Operations Manipulation" http://venom.crowdstrike.com/.

[38] S.Berger, "Trusted Platform Module Support Phase I" http://wiki.qemu.org/Features/TPM.

[39] S.Berger, R.Cáceres, K. A.Goldman, R.Perez, R.Sailer, L.van Doorn, "vTPM: Virtualizing the Trusted Platform Module", USENIX'06: 15th Conference on USENIX Security Symposium, Vancouver, B.C., Canada, July 31-August 04 2006, pp. 305–320.

[40] K.Goldman, R.Sailer, D.Pendarakis, D.Srinivasan, "Scalable integrity monitoring in virtualized environments", STC'10: 5th ACM workshop on Scalable Trusted Computing, Chicago, Il, USA, October 4-8 2010, pp. 73–78, doi:10.1145/1867635.1867647.

[41] T.Garfinkel, B.Pfaff, J.Chow, M.Rosenblum, D.Boneh, "Terra: a virtual machine-based platform for trusted computing", SOSP'03: 19th ACM symposium on Operating Systems Principles, Bolton Landing, NY, USA, October 19-22 2003, pp. 193–206, doi:10.1145/1165389.945464.

[42] J.Schiffman, H.Vijayakumar, T.Jaeger, "Verifying System Integrity by Proxy",

TRUST'12: 5th International Conference on Trust and Trustworthy Computing, Vienna, Austria, June 13-15 2012, pp. 179–200, doi:10.1007/978-3-642-30921-2_11.

[43] S.Soltesz, M. E.Fiuczynski, L.Peterson, M.Mccabe, J.Matthews, "Virtual Doppelgänger: On the Performance, Isolation, and Scalability of Para- and Paene- Virtualized Systems" 2006, http://people.clarkson.edu/~jnm/publications/paenevirtualization.pdf.

[44] H.Ali, "Performance of Docker vs VMs" 2014, http://www.slideshare.net/Flux7Labs/performance-of-docker-vs-vms.

[45] E.Messmer, "Details emerging on Hannaford data breach - Malware loaded onto Hannaford servers let attackers intercept credit card data" 2008, http://www.networkworld.com/article/2284998/lan-wan/details-emerging-on-hannaford-data-breach.html.

[46] A. R.Sadeghi, C.Stüble, "Property-based attestation for computing platforms: caring about properties, not mechanisms", NSPW'04: Workshop on New Security Paradigms, Nova Scotia, Canada, September 20-23 2004, pp. 67–77, doi:10.1145/1065907.1066038.

[47] J. A.Poritz, "Trust[ed|in] Computing, Signed Code and the Heat Death of the Internet", SAC'06: ACM Symposium on Applied Computing, Dijon, France, April 23-27 2006, pp. 1855–1859, doi:10.1145/1141277.1141716.

[48] A. W.Appel, E. W.Felten, "Models for Security Policies in Proof-Carrying Code", Princeton University 2001, ftp://ftp.cs.princeton.edu/reports/2001/636.pdf.

[49] G. C.Necula, "Proof-carrying Code", POPL'97: 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Paris, France, January 15-17 1997, pp. 106–119, doi:10.1145/263699.263712.

[50] A.Bernard, P.Lee, "Enforcing Formal Security Properties", Carnegie Mellon School of Computer Science 2001, April, http://reports-archive.adm.cs.cmu.edu/anon/usr0/anon/usr0/ftp/2001/CMU-CS-01-121.pdf.

[51] J.Poritz, M.Schunter, E.Herreweghen, M.Waidner, "Property Attestation–Scalable and Privacy-friendly Security Assessment of Peer Computers", IBM Zurich Research Laboratory 2004, October, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.128.5802&rep=rep1&type=pdf.

[52] U.Kühn, M.Selhorst, C.Stüble, "Realizing property-based attestation and sealing with commonly available hard- and software", STC'07: 2nd ACM workshop on Scalable Trusted Computing, Alexandria, Virginia, USA, October 29-November 02 2007, pp. 50–57, doi:10.1145/1314354.1314368.

[53] R.Korthaus, A. R.Sadeghi, C.Stüble, J.Zhan, "A practical property-based bootstrap architecture", STC'09: 4th ACM workshop on Scalable Trusted Computing, Chicago, IL, USA, November 9-13 2009, pp. 29–38, doi:10.1145/1655108.1655114.

[54] "TrouSerS, The open-source TCG Software Stack" http://trousers.sourceforge.net/.

[55] X.Li, C.Shen, X.Zuo, “An Efficient Attestation for Trustworthiness of Computing Platform”, IIH-MSP’06: International Conference on Intelligent Information Hiding and Multimedia, Pasadena, CA, USA, December 18-20 2006, pp. 625–630, doi:10.1109/IIH-MSP.2006.265080.

[56] M.Alam, X.Zhang, M.Nauman, T.Ali, J. P.Seifert, “Model-based behavioral attestation”, SACMAT’08: 13th ACM Symposium on Access Control Models And Technologies, Estes Park, CO, USA, June 11-13 2008, pp. 175–184, doi:10.1145/1377836.1377864.

[57] W.Chris, C.Crispin, S.Stephen, M.James, K.-H.Greg, “Linux Security Modules: General Security Support for the Linux Kernel”, USENIX’02: 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9 2002, pp. 213–226, doi:10.1109/FITS.2003.1264934.

[58] L.Gu, X.Ding, R. H.Deng, B.Xie, H.Mei, “Remote attestation on program execution”, STC’08: 3rd ACM workshop on Scalable Trusted Computing, Alexandria, Virginia, USA, October 27-31 2008, pp. 11–20, doi:10.1145/1456455.1456458.

[59] J.Park, R.Sandhu, “Towards Usage Control Models: Beyond Traditional Access Control”, SACMAT’02: 7th ACM Symposium on Access Control Models and Technologies, Monterey, California, USA, June 3-4 2002, pp. 57–64, doi:10.1145/507711.507722.

[60] X.Zhang, F.Parisi-Presicce, R.Sandhu, J.Park, “Formal Model and Policy Specification of Usage Control”, ACM Transaction on Information and System Security, Vol. 8, No. 4, November 2005, pp. 351–387, doi:10.1145/1108906.1108908.

[61] M.Nauman, M.Alam, X.Zhang, T.Ali, “Remote Attestation of Attribute Updates and Information Flows in a UCON System”, TRUST’09: 2nd International Conference on Trust and Trustworthy Computing, Oxford, UK, April 6-8 2009, pp. 63–80, doi:10.1007/978-3-642-00587-9_5.

[62] R.Sailer, X.Zhang, T.Jaeger, L.van Doorn, “Design and Implementation of a TCG-based Integrity Measurement Architecture”, USENIX’04: 13th USENIX Security Symposium, San Diego, CA, USA, June 27-July 02 2004, pp. 223–238.

[63] “Open Trusted Computing (OpenTC)” `www.opentc.net/`.

[64] “The SECURED project (SECURity at the network EDge)” `http://www.secured-fp7.eu/`.

[65] S.Berger, K.Goldman, D.Pendarakis, D.Safford, E.Valdez, M.Zohar, “Scalable Attestation: A Step toward Secure and Trusted Clouds”, IEEE Cloud Computing, Vol. 2, No. 5, September 2015, pp. 10–18, doi:10.1109/MCC.2015.97.

[66] T.Jaeger, R.Sailer, U.Shankar, “PRIMA: policy-reduced integrity measurement architecture”, SACMAT’06: 11th ACM Symposium on Access Control Models And Technologies, Lake Tahoe, CA, USA, June 7-9 2006, pp. 19–28, doi:10.1145/1133058.1133063.

[67] W.Xu, G.-J.Ahn, H.Hu, X.Zhang, J.-P.Seifert, “DR@FT: Efficient Remote Attestation Framework for Dynamic Systems”, ESORICS’10: 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22 2010, pp. 182–198, doi:10.1007/978-3-642-15497-3_12.

[68] B.Hicks, S.Rueda, T.Jaeger, P.McDaniel, "From trusted to secure: building and executing applications that enforce system security", ATC'07: USENIX Annual Technical Conference, Santa Clara, CA, USA, June 17-22 2007, pp. 1–14.

[69] W.Sze, R.Sekar, "A Portable User-level Approach for System-wide Integrity Protection", ACSAC'13: 29th Annual Computer Security Applications Conference, New Orleans, Louisiana, USA, December 9-13 2013, pp. 219–228, doi:10.1145/2523649.2523655.

[70] P. A.Loscocco, P. W.Wilson, J. A.Pendergrass, C. D.McDonell, "Linux kernel integrity measurement using contextual inspection", STC'07: 2nd ACM workshop on Scalable Trusted Computing, Alexandria, Virginia, USA, October 29-November 02 2007, pp. 21–29, doi:10.1145/1314354.1314362.

[71] M.Thober, J. A.Pendergrass, C. D.McDonell, "Improving coherency of runtime integrity measurement", STC'08: 3rd ACM workshop on Scalable Trusted Computing, Alexandria, Virginia, USA, October 27-31 2008, pp. 51–60, doi:10.1145/1456455.1456464.

[72] N. L.Petroni, Jr., M.Hicks, "Automated detection of persistent kernel control-flow attacks", CCS'07: 14th ACM conference on Computer and Communications Security, Alexandria, Virginia, USA, October 29-November 02 2007, pp. 103–115, doi:10.1145/1315245.1315260.

[73] C.Kil, E.Sezer, A.Azab, P.Ning, X.Zhang, "Remote attestation to dynamic system properties: Towards providing complete system integrity evidence", DSN'09: IEEE/IFIP International Conference on Dependable Systems Networks, Lisbon, Portugal, June 29-July 02 2009, pp. 115–124, doi:10.1109/DSN.2009.5270348.

[74] L.Davi, A. R.Sadeghi, M.Winandy, "Dynamic integrity measurement and attestation: towards defense against return-oriented programming attacks", STC'09: 4th ACM workshop on Scalable Trusted Computing, Chicago, IL, USA, November 9-13 2009, pp. 49–54, doi:10.1145/1655108.1655117.

[75] R.Sassu, G.Ramunno, A.Lioy, "Practical Assessment of Biba Integrity for TCG-Enabled Platforms", TrustCom'14: 13th International Conference on Trust, Security and Privacy in Computing and Communications, Beijing, China, September 24-26 2014, pp. 495–504, doi:10.1109/TrustCom.2014.63.

[76] IBM, "Software TPM 2.0" https://sourceforge.net/projects/ibmswtpm2/.

[77] "Software-based TPM Emulator" https://sourceforge.net/projects/tpm-emulator.berlios/.

[78] M.Strasser, H.Stamer, "A Software-Based Trusted Platform Module Emulator", Trust'08: 1st International Conference on Trusted Computing and Trust in Information Technologies, Villach, Austria, March 11-12 2008, pp. 33–47, doi:10.1007/978-3-540-68979-9_3.

[79] H.Raj, S.Saroiu, A.Wolman, R.Aigner, J.Cox, P.England, C.Fenner, K.Kinshumann, J.Loeser, D.Mattoon, M.Nystrom,

D.Robinson, R.Spiger, S.Thom, D.Wooten, "fTPM: A Firmware-based TPM 2.0 Implementation", Microsoft Research 2015, November, https://www.microsoft.com/en-us/research/publication/ftpm-a-firmware-based-tpm-2-0-implementation/.

[80] H.Raj, S.Saroiu, A.Wolman, R.Aigner, J.Cox, P.England, C.Fenner, K.Kinshumann, J.Loeser, D.Mattoon, M.Nystrom, D.Robinson, R.Spiger, S.Thom, D.Wooten, "fTPM: A Software-Only Implementation of a TPM Chip", USENIX'16: 25th USENIX Security Symposium, Austin, Texas, USA, August 10-12 2016, pp. 841–856.

[81] "BitLocker Drive Encryption Overview" https://technet.microsoft.com/en-us/library/cc732774(v=ws.11).aspx.

[82] "Open Platform Trust Services, GRUB-IMA" https://osdn.net/projects/openpts/wiki/GRUB-IMA.

[83] "Trusted Boot" https://sourceforge.net/projects/tboot/.

[84] "TPM enabled GRUB2 Bootloader, TrustedGRUB2" https://github.com/Rohde-Schwarz-Cybersecurity/TrustedGRUB2.

[85] B.Schneier, "Evil Maid Attacks on Encrypted Hard Drives" https://www.schneier.com/blog/archives/2009/10/evil_maid_attac.html.

[86] "Open Platform Trust Services" https://osdn.net/projects/openpts/wiki/FrontPage.

[87] "OpenAttestation SDK" https://github.com/OpenAttestation/OpenAttestation.

[88] "Open Cloud Integrity Technology" https://github.com/opencit/opencit.

[89] "Trusted Network Communication and Internet Engineering Task Force" http://trustedcomputinggroup.org/work-groups/trusted-network-communications/tnc-and-ietf/.

[90] P.Sangster, K.Narayan, "PA-TNC: A Posture Attribute (PA) Protocol Compatible with Trusted Network Connect (TNC)", RFC-5792, March 2010.

[91] P.Sangster, N.Cam-Winget, J.Salowey, "A Posture Transport Protocol over TLS (PT-TLS)", RFC-6876, February 2013.

[92] "Trusted Network Connect (TNC) HOWTO" https://wiki.strongswan.org/projects/1/wiki/trustednetworkconnect.

[93] G.Coker, J.Guttman, P.Loscocco, A.Herzog, J.Millen, B.O'Hanlon, J.Ramsdell, A.Segall, J.Sheehy, B.Sniffen, "Principles of remote attestation", International Journal of Information Security, Vol. 10, No. 2, April 2011, pp. 63–81, doi:10.1007/s10207-011-0124-7.

[94] L.Chen, P.Morrissey, N. P.Smart, "Pairings in Trusted Computing", Paring'08: 2nd International Conference of Pairing-Based Cryptography, Egham, UK, September 1-3 2008, pp. 1–17, doi:10.1007/978-3-540-85538-5_1.

[95] E.Brickell, L.Chen, J.Li, "Simplified Security Notions of Direct Anonymous Attestation and a Concrete Scheme from Pairings", International Journal of Information Security, Vol. 8, No. 5, September 2009, pp. 315–330, doi:10.1007/s10207-009-0076-3.

[96] E.Cesena, G.Ramunno, R.Sassu, D.Vernizzi, A.Lioy, "On Scalability of Remote Attestation", STC'11: 6th ACM Workshop on Scalable

Trusted Computing, Chicago, IL, USA, October 17-21 2011, pp. 25–30, doi:10.1145/2046582.2046588.

[97] "Address Space Layout Randomization" https://pax.grsecurity.net/docs/aslr.txt.

[98] S.Frankel, S.Krishnan, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap", RFC-6071, February 2011.

[99] E. H.Spafford, "Spaf's Home Page" http://spaf.cerias.purdue.edu/index.html.

[100] K.Goldman, R.Perez, R.Sailer, "Linking Remote Attestation to Secure Tunnel Endpoints", STC'06: 1st ACM Workshop on Scalable Trusted Computing, Alexandria, Virginia, USA, November 03 2006, pp. 21–24, doi:10.1145/1179474.1179481.

[101] Y.Gasmi, A. R.Sadeghi, P.Stewin, M.Unger, N.Asokan, "Beyond Secure Channels", STC'07: 2nd ACM Workshop on Scalable Trusted Computing, Alexandria, Virginia, USA, November 02 2007, pp. 30–40, doi:10.1145/1314354.1314363.

[102] F.Armknecht, Y.Gasmi, A. R.Sadeghi, P.Stewin, M.Unger, G.Ramunno, D.Vernizzi, "An Efficient Implementation of Trusted Channels Based on Openssl", STC'08: 3rd ACM Workshop on Scalable Trusted Computing, Alexandria, Virginia, USA, October 31 2008, pp. 41–50, doi:10.1145/1456455.1456462.

[103] S.Santesson, "TLS Handshake Message for Supplemental Data", RFC-4680, September 2006.

[104] Y.Yu, H.Wang, B.Liu, G.Yin, "A Trusted Remote Attestation Model Based on Trusted Computing", TrustCom'13: 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Melbourne, Australia, July 16-18 2013, pp. 1504–1509, doi:10.1109/TrustCom.2013.183.

[105] E.Cesena, H.Löhr, G.Ramunno, A. R.Sadeghi, D.Vernizzi, "Anonymous Authentication with TLS and DAA", Trust'10: 3rd International Conference on Trust & Trustworthy Computing, Berlin, Germany, June 21-23 2010, pp. 47–62, doi:10.1007/978-3-642-13869-0_4.

[106] F.Stumpf, O.Tafreschi, P.Röder, C.Eckert, "A robust integrity reporting protocol for remote attestation", WATC'06: 2nd Workshop on Advances in Trusted Computing, Tokyo, Japan, November 31-December 1 2006, pp. 308–317.

[107] L.Zhou, Z.Zhang, "Trusted Channels with Password-Based Authentication and TPM-Based Attestation", CMC'10: International Conference on Communications and Mobile Computing, Shenzhen, China, April 12-14 2010, pp. 223–227, doi:10.1109/CMC.2010.232.

[108] N.Aziz, N.Udzir, R.Mahmod, "Extending TLS with Mutual Attestation for Platform Integrity Assurance", Journal of Communications, Vol. 9, No. 1, January 2014, pp. 63–72, doi:10.12720/jcm.9.1.63-72.

[109] S.Schulz, A. R.Sadeghi, "Secure VPNs for Trusted Computing Environments", Trust'09: 2nd International Conference on Trusted Computing, Oxford, UK, April 6-8 2009, pp. 197–216, doi:10.1007/978-3-642-00587-9_13.

[110] A. R.Sadeghi, S.Schulz, "Extending IPsec for Efficient Remote Attestation", FC'14: 14th International Conference on Financial Cryptography and Data Security, Tenerife, Canary Islands, Spain, January 25-28 2010, pp. 150–165, doi:10.1007/978-3-642-14992-4_14.

[111] "TrustedVPN" https://cybersecurity.rohde-schwarz.com/en/products/secure-networks/trustedvpn.

[112] Trusted Computing Group, "TNC IF-T: Protocol Bindings for Tunneled EAP Methods Specification" 2014, https://trustedcomputinggroup.org/tnc-if-t-protocol-bindings-tunneled-eap-methods-specification.

[113] Trusted Computing Group, "TNC IF-T: Binding to TLS" 2013, https://trustedcomputinggroup.org/tnc-if-t-binding-tls/.

[114] Trusted Computing Group, "Trusted Network Connect Client-Server" 2014, https://trustedcomputinggroup.org/tnc-if-tnccs-specification/.

[115] D.Maughan, M.Schertler, M.Schneider, J.Turner, "Internet Security Association and Key Management Protocol (ISAKMP)", RFC-2408, November 1998.

[116] D.Harkins, D.Carrel, "The Internet Key Exchange (IKE)", RFC-2409, November 1998.

[117] S.Sakane, K.Kamada, M.Thomas, J.Vilhuber, "Kerberized Internet Negotiation of Keys (KINK)", RFC-4430, March 2006.

[118] C.Kaufman, "Internet Key Exchange (IKEv2) Protocol", RFC-4306, December 2005.

[119] D.Black, D.McGrew, "Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol", RFC-5282, August 2008.

[120] D.Harkins, "Secure Pre-Shared Key (PSK) Authentication for the Internet Key Exchange Protocol (IKE)", RFC-6617, June 2012.

[121] T.Kivinen, J.Snyder, "Signature Authentication in the Internet Key Exchange Version 2 (IKEv2)", RFC-7427, January 2015.

[122] D.Fu, "IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC-4754, January 2007.

[123] H.Tschofenig, D.Kroeselberg, A.Pashalidis, Y.Ohba, F.Bersani, "The Extensible Authentication Protocol-Internet Key Exchange Protocol version 2 (EAP-IKEv2) Method", RFC-5106, February 2008.

[124] IEEE Standard Association, "802.1X - IEEE Standard for Local and metropolitan area networks-Port-Based Network Access Control" 2010.

[125] A.DeKok, A.Lior, "Remote Authentication Dial-In User Service (RADIUS) Protocol Extensions", RFC-6929, April 2013.

[126] V.Fajardo, J.Arkko, J.Loughney, G.Zorn, "Diameter Base Protocol", RFC-6733, October 2012.

[127] "strongSwan" https://www.strongswan.org/about.html.

[128] ETSI NFV Security ISG, "Security and Trust Guidance" http://docbox.etsi.org/ISG/NFV/Open/Drafts/SEC18/NFV-SEC003ed112v002.doc.

[129] I.Ahmad, S.Namal, M.Ylianttila, A.Gurtov, "Security in Software Defined Networks: A Survey", IEEE Communications Surveys Tutorials, Vol. 17, No. 4, August 2015, pp. 2317–2346, doi:10.1109/COMST.2015.2474118.

[130] I.Alsmadi, D.Xu, "Security of Software Defined Networks: A survey", Computers & Security, Vol. 53, September 2015, pp. 79–108, doi:10.1016/j.cose.2015.05.006.

[131] Z.Shu, J.Wan, D.Li, J.Lin, A. V.Vasilakos, M.Imran, "Security in Software-Defined Networking: Threats and Countermeasures", Mobile Networks and Applications, Vol. 21, No. 5, January 2016, pp. 764–776, doi:10.1007/s11036-016-0676-x.

[132] W.Li, W.Meng, L. F.Kwok, "A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures", Journal of Network and Computer Applications, Vol. 68, April 2016, pp. 126–139, doi:10.1016/j.jnca.2016.04.011.

[133] N.McKeown, T.Anderson, H.Balakrishnan, G.Parulkar, L.Peterson, J.Rexford, S.Shenker, J.Turner, "OpenFlow: Enabling Innovation in Campus Networks", SIGCOMM Computer Communication Review, Vol. 38, No. 2, April 2008, pp. 69–74, doi:10.1145/1355734.1355746.

[134] M. C.Dacier, H.Konig, R.Cwalinski, F.Kargl, S.Dietrich, "Security Challenges and Opportunities of Software-Defined Networking", IEEE Security & Privacy, Vol. 15, No. 2, March/April 2017, pp. 96–100, doi:10.1109/MSP.2017.46.

[135] W.Liu, R. B.Bobba, S.Mohan, R. H.Campbell, "Inter-flow consistency: A novel SDN update abstraction for supporting inter-flow constraints", CNS'15: IEEE Conference on Communications and Network Security, Florence, Italy, September 28-30 2015, pp. 469–478, doi:10.1109/CNS.2015.7346859.

[136] L.Schiff, S.Schmid, P.Kuznetsov, "In-Band Synchronization for Distributed SDN Control Planes", Journal of SIGCOMM Computer Communication Review, Vol. 46, No. 1, January 2016, pp. 37–43, doi:10.1145/2875951.2875957.

[137] L.Jacquin, A. L.Shaw, C.Dalton, "Towards trusted software-defined networks using a hardware-based Integrity Measurement Architecture", NetSoft'15: 1st IEEE Conference on Network Softwarization, London, UK, April 13-17 2015, pp. 1–6, doi:10.1109/NETSOFT.2015.7116186.

[138] M. D.Firoozjaei, J. P.Jeong, H.Ko, H.Kim, "Security challenges with network functions virtualization", Future Generation Computer Systems, Vol. 67, February 2017, pp. 315–324, doi:10.1016/j.future.2016.07.002.

[139] S.Ravidas, S.Lal, I.Oliver, L.Hippelainen, "Incorporating trust in NFV: Addressing the challenges", ICIN'17: 20th Conference on Innovations in Clouds, Internet and Networks, Paris, France, March 7-9 2017, pp. 87–91, doi:10.1109/ICIN.2017.7899394.

[140] Xen Project mailing list, "vtpmmgr bug: fails to start if locality not 0" 2014, https://lists.xen.org/archives/html/xen-devel/2014-11/msg00606.html.

[141] M.Fioravante, D. D.Graaf, "Virtual TPM interface for Xen" https://www.kernel.org/doc/Documentation/security/tpm/xen-tpmfront.txt.

[142] Trusted Computing Group, "Virtualized Trusted Platform Architecture Specification, Version 1.0, Revision 26" 2011, https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_VPWG_Architecture_V1-0_R0-26_FINAL.pdf.

[143] X.Wan, Z.Xiao, Y.Ren, "Building Trust into Cloud Computing Using Virtualization of TPM", MINES'12: 4th International Conference on Multimedia Information Networking and Security, Nanjing, China, November 2-4 2012, pp. 59–63, doi:10.1109/MINES.2012.82.

[144] H.Lauer, N.Kuntze, "Hypervisor-Based Attestation of Virtual Environments", UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld'16: International IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress, Toulouse, France, July 18-21 2016, pp. 333–340, doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0067.

[145] F.Stumpf, C.Eckert, "Enhancing Trusted Platform Modules with Hardware-Based Virtualization Techniques", SECURWARE'08: 2nd International Conference on Emerging Security Information, Systems and Technologies, Cap Esterel, France, August 25-31 2008, pp. 1–9, doi:10.1109/SECURWARE.2008.23.

[146] R.Uhlig, G.Neiger, D.Rodgers, A. L.Santoni, F. C. M.Martins, A. V.Anderson, S. M.Bennett, A.Kagi, F. H.Leung, L.Smith, "Intel virtualization technology", Computer, Vol. 38, No. 5, May 2005, pp. 48–56, doi:10.1109/MC.2005.163.

[147] Intel Corporation, "Intel Clear Containers" 2017, https://clearlinux.org/features/intel-clear-containers/.

[148] "QEMU is a generic and open source machine emulator and virtualizer" 2017, www.qemu.org/.

[149] "rkt - the pod-native container engine" 2017, https://github.com/rkt/rkt.

[150] P.England, J.Loeser, "Para-Virtualized TPM Sharing", TRUST'08: 1st International Conference on Trusted Computing and Trust in Information Technologies, Villach, Austria, March 11-12 2008, pp. 119–132, doi:10.1007/978-3-540-68979-9_9.

[151] S.Hosseinzadeh, S.Laurén, V.Leppänen, "Security in Container-based Virtualization Through vTPM", UCC'16: 9th International Conference on Utility and Cloud Computing, Shanghai, China, December 6-9 2016, pp. 214–219, doi:10.1145/2996890.3009903.

[152] S.Berger, "Virtual TPM Proxy Driver for Linux Containers" https://www.kernel.org/doc/html/v4.10/security/tpm/tpm_vtpm_proxy.html.

[153] "rkt and the Trusted Platform Module" 2017, https://github.com/rkt/rkt/blob/master/Documentation/devel/tpm.md.

[154] Intel Corporation, "Intel Software Guard Extensions Programming Reference" 2015, https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

[155] V.Costan, S.Devadas, "Intel SGX explained" https://pdfs.semanticscholar.org/2d7f/3f4ca3fbb15ae04533456e5031e0d0dc845a.pdf.

[156] M.Shih, M.Kumar, T.Kim, A.Gavrilovska, "S-NFV: Securing NFV States by Using SGX", SDN-NFV Security'16: ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, New Orleans, Louisiana, USA, March 11 2016, pp. 45–48,

doi:10.1145/2876019.2876032.

[157] M.Coughlin, E.Keller, E.Wustrow, "Trusted Click: Overcoming Security Issues of NFV in the Cloud", SDN-NFVSec'17: ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, Scottsdale, Arizona, USA, March 24 2017, pp. 31–36, doi:10.1145/3040992.3040994.

[158] S.Arnautov, B.Trach, F.Gregor, T.Knauth, A.Martin, C.Priebe, J.Lind, D.Muthukumaran, D.O'Keeffe, M. L.Stillwell, D.Goltzsche, D.Eyers, R.Kapitza, P.Pietzuch, C.Fetzer, "SCONE: Secure Linux Containers with Intel SGX", OSDI'16: 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, GA, USA, November 2-4 2016, pp. 689–703.

[159] F.Brasser, U.Müller, A.Dmitrienko, K.Kostiainen, S.Capkun, A. R.Sadeghi, "Software Grand Exposure: SGX Cache Attacks Are Practical" 2017.

[160] M.Schwarz, S.Weiser, D.Gruss, C.Maurice, S.Mangard, "Malware Guard Extension: Using SGX to Conceal Cache Attacks" 2017.

[161] J.Sulinski, "Docker Image Vulnerability Research" 2017, https://www.federacy.com/docker_image_vulnerabilities.

[162] N.Kuntze, C.Rudolph, J.Paatero, "Establishing Trust between Nodes in Mobile Ad-Hoc Networks", INTRUST'12: 4th International Conference on Trusted Systems, London, UK, December 17-18 2012, pp. 48–62, doi:10.1007/978-3-642-35371-0_4.

[163] Datadog, "8 surprising facts about real Docker container adoption" 2016, https://www.datadoghq.com/docker-adoption/.

[164] Business Cloud News, "Deutsche Telekom experimenting with NFV in Docker" 2016, http://www.businesscloudnews.com/2015/02/09/deutsche-telekom-experimenting-with-nfv-in-docker/.

[165] "Docker User in Google group" https://groups.google.com/forum/#!msg/docker-user/k5hqpNg8gwQ/00mvrB2nIkJ.

[166] "Lingering processes (and containers) when writing to stdout" https://github.com/docker/docker/issues/1320.

[167] A.Atzeni, T.Su, T.Montanaro, "Lightweight Formal Verification in Real World, A Case Study", WISSE'14: 4th International Workshop on Information Systems Security Engineering, Thessaloniki, Greece, June 17 2014, pp. 335–342, doi:10.1007/978-3-319-07869-4_31.

[168] T.Su, J.Lyle, A.Atzeni, S.Faily, H.Virji, C.Ntanos, C.Botsikas, "Continuous Integration for Web-Based Software Infrastructures: Lessons Learned on the webinos Project", HVC'13: 9th International Haifa Verification Conference: Verification and Testing, Haifa, Israel, November 5-7 2013, pp. 145–150, doi:10.1007/978-3-319-03077-7_10.

[169] B.Gregg, "Linux Enhanced BPF (eBPF) Tracing Tools" 2017, http://www.brendangregg.com/ebpf.html.