SECURELY LAUNCHING VIRTUAL MACHINES ON TRUSTWORTHY PLATFORMS IN A PUBLIC CLOUD

An Enterprise's Perspective

Mudassar Aslam[1], Christian Gehrmann[1], Lars Rasmusson[1], Mats Björkman[2]
[1]*Swedish Institute of Computer Science, Isafjordsgatan 22, Box 1263, SE-164 29 Kista,Sweden*
[2]*Mälardalens Högskola, Box 883, SE-721 23 Västerås, Sweden*
{*mudassar, chrisg, lra*}*@sics.se, mats.bjorkman@mdh.se*

Keywords: Security: Trusted Computing: Virtualization: Cloud Computing: IaaS.

Abstract: In this paper we consider the Infrastructure-as-a-Service (IaaS) cloud model which allows cloud users to run their own virtual machines (VMs) on available cloud computing resources. IaaS gives enterprises the possibility to outsource their process workloads with minimal effort and expense. However, one major problem with existing approaches of cloud leasing, is that the users can only get contractual guarantees regarding the integrity of the offered platforms. The fact that the IaaS user himself or herself cannot verify the provider-promised cloud platform integrity, is a security risk which threatens to prevent the IaaS business in general. In this paper we address this issue and propose a novel secure VM launch protocol using Trusted Computing techniques. This protocol allows the cloud IaaS users to securely bind the VM to a trusted computer configuration such that the clear text VM only will run on a platform that has been booted into a trustworthy state. This capability builds user confidence and can serve as an important enabler for creating trust in public clouds. We evaluate the feasibility of our proposed protocol via a full scale system implementation and perform a system security analysis.

1 INTRODUCTION

In recent years, there has been a tendency to migrate IT services like email, storage, and other applications into the clouds due to cost and maintenance benefits. Several different cloud sourcing models exist. In this paper we focus on the cloud model where the cloud consumer is able to deploy full software systems (including the operating system) on a shared common infrastructure, or what is often referred to as the Infrastructure-as-a-Service (IaaS) cloud model (Krutz and Vines, 2010). This paper presents a novel technique for the secure launch of virtual machines in a public cloud.

A small company without security skills or an ordinary IT service consumer might trust a public cloud service provider and in some cases prefer cloud services over self-hosted services with a belief that his or her cloud provider can offer better security by recruiting specialized staff and equipment. In contrast, most large or medium size enterprises have higher security requirements for their own or their business users sensitive data; and if their data is compromised due to a security breach in the cloud provider network, it will results in serious legal and business setbacks. Therefore, these enterprises are reluctant to host their services in a public cloud unless they get trusted ways to validate the contractual security guarantees provided by the cloud provider.

The focus of our work is to introduce technical ways to verify the security guarantees provided by the cloud service provider. We achieve this by allowing the cloud user to cryptographically bind the user virtual machine (VM) to a trustworthy state of the provisioned cloud platform. Furthermore, we ensure that the whole launch process meets all expected major security requirements of a high quality public service with respect to authentication and secure transfer. According to our suggested VM launch protocol, a particular VM is not even sent to the provider network if no platform with the expected security guarantees can be offered by the IaaS cloud. Our proposed protocol uses a unique combination of state-of-the art techniques for authentication, and remote attestation together with TPM's[1] sealing and a compact single transfer protected VM launch command. The main

[1]Trusted Platform Module

contributions of this paper are the following:

- We design and propose a secure VM launch protocol for the clouds which enables verifiable trust between the cloud user and provider.
- We implement the proposed protocol to evaluate the feasibility of using TCG[2] mechanisms in terms of implementation effort and complexity.
- We perform a security analysis of the proposed protocol and show how expected security requirements are fulfilled.

The paper is organized as follows. In Section 2 we present the scenario we are addressing and relevant security threats. In Section 3 we cover the existing body of literature in the area. In Section 4, we describe the underlying architecture of the resource platform which provides necessary foundation to design a trusted platform. We present out launch protocol in Section 5 followed by its implementation in Section 6. Finally, we present a security analysis of the proposed launch protocol in Section 7 and conclude in Section 8.

2 SCENARIO AND THREATS

2.1 Scenario

We are considering the IaaS cloud model from an enterprise's perspective which is depicted in Figure 1. There are two main stakeholders - the cloud *Provider* and the cloud *User*. The provider can be the resource/platform manufacturer or a third party cloud provider who provisions bare virtual machines to the users through a *Procurement Server*. The users can install any software stack, including OS on the procured VM. We consider an enterprise cloud user that wants to outsource its business processes to the cloud *without* compromising the security of its data and applications. In this paper, we consider secure VM management for this scenario and in particular the initial process when the user wants to launch a VM on the offered provider infrastructure. We analyze this VM launch phase from a security perspective and suggest protocols and principles for protecting it. This covers *one* phase of the VM life-cycle. Other important phases are *VM management* once the VM has been launched on the target platform and *VM migration* from one cloud platform to another in order to allow efficient resource management. These later phases are

[2]Trusted Computing Group, http://www.trustedcomputinggroup.org/

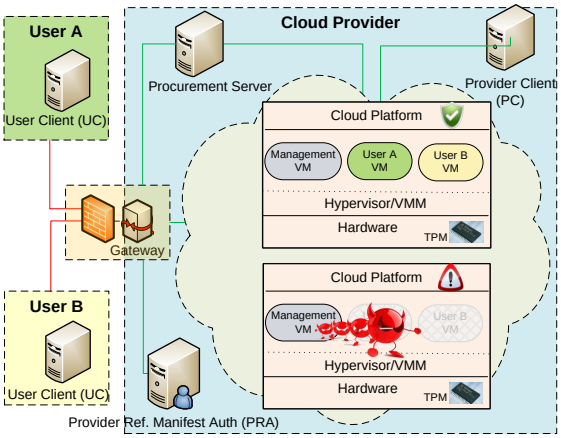

Figure 1: System Overview.

out of scope of the current paper and subject to future work.

2.2 Threats

As we have previously discussed, migration from traditional business models to a cloud-based model has business benefits such as cost savings etc. However, this comes together with new security threats that prevent capitalizing the cloud advantages for many enterprises (CircleID, 2009). These threats have been identified by the research community (Aslam and Gehrmann, 2011), (Vaquero et al., 2011) and the alliance for cloud security (CSA Threats, 2010). Below, we shortlist the major threats relevant to virtual machine launch in a public cloud.

1. Insecure Interfaces: Insecure interfaces and APIs to cloud services could result in various security issues related to confidentiality, integrity, availability and accountability. This means that the risk of security breach exists right from the beginning, that is, when the cloud user launches a VM. This risk remains even after the launch due to the insecure management interfaces (CSA Threats, 2010).

2. User-client Authenticity: The attacker tries to impersonate a legitimate user-client.

3. Provider Network Attacks: Internal (inside the provider network) and external network attacks in the cloud, e.g. DoS attack.

4. Disowning Malicious VM: The cloud user repudiates the launch of a VM instance, for example to deny the ownership of a misbehaving or malicious VM.

5. Cloud Platform Authenticity: The attacker tries to impersonate a cloud platform.

6. Insider Attacks: A rogue employee with administrative access within a cloud provider network can install malicious code/software on the cloud platform that can then easily leak user's (i.e. enterprise) confidential data.

7. Mis-configured Cloud Platform: There is a risk that the provider unintentionally installs a hostile software package on the cloud platform or the cloud node is not compliant to the promised security profile, that is, the cloud platform is not updated with the latest security patches. These risks expose the platform and hence the cloud users to known vulnerabilities which could expose the platform for possible attacks.

8. User-data Leak: The cloud provider tries to run a user VM instance using received VM image on an in-secure platform. The user starts using such a potentially in-secure VM and risks of losing confidential data.

The first four threats mentioned above are general and well known for any distributed system while the last four are more cloud-specific for which we propose mechanisms and solutions. The goal with the overall security architecture (Section 4) and our secure launch protocol (Section 5) is to address all these listed threats. We revisit the threats list in the security analysis in Section 7 and discuss if/how the threats have been mitigated in our design.

3 RELATED WORK

The importance of security for clouds has been identified very early by the concerned stakeholders, and the research community has responded by identifying security threats and proposing many different kinds of solutions. Consortia like Cloud Security Alliance[3], Trusted Computing Group[4] and other industrial-academic research alliances such as TClouds[5] and EuroCloud[6] have already taken initiatives to address and solve security issues in the clouds. As we have discussed in Section 2.2, several reports and papers have been published that analyze cloud security threats (CSA Guide, 2009), (CSA Threats, 2010), (Aslam and Gehrmann, 2011) and (Vaquero et al., 2011).

The TCG initiative for practical deployment of trusted clouds (TMI, 2010) aims at proposing specifications and reference models. The TMI working group, at its infancy, has only released a short white paper so far (Cloud Computing and Security, 2010) which identifies the areas where TCG mechanisms can provide security and trust in the clouds. While details about *how* are still unavailable publically, we have used the basic TCG mechanisms of secure boot, remote attestation and sealing capabilities to propose a mechanism that enables security and trust in the clouds. There also exist some industrial proprietary solutions like the Intel® TXT (Intel Trusted Execution Technology) which uses TCG and proprietary hardware extensions to set up trustworthy cloud platforms. The acTvSM Platform (acTvSM, 2010) is one such prototype implementation. Our solution can be built upon any such platform and additionally gives the ability to the cloud user to verify the trustworthiness of the platform before using the cloud.

There are other proposed solutions to implement remote attestation as defined by the TCG. (Huang and Peng, 2009) and (Sailer et al., 2004) use TPM based cryptographic attestations by sending Integrity Measurement Log (IML) in combination with TPM_Quote which is securely computed by the Trusted Platform Module (TPM Commands, 2007). The verifier compares the quoted response with a self-computed hash value from the IML to check and decide about the integrity of the target platform. (Landfermann et al., 2006) propose an alternate scheme for remote attestation by introducing property-based attestations instead of cryptographic attestations to make it a scalable solution. (Haldar et al., 2004) also propose a similar remote attestation technique to fulfill platform integrity requirements. All these papers mainly discuss remote attestation techniques which could be used in performing the integrity verification step of our proposed protocol.

The architecture introduced by (Jansen et al., 2006) presents ways to protect the confidentiality of the user VM by leveraging on the sealing mechanism supported by TPM. We enforce sealing of the user VM to a platform state in a slightly different way. Our approach allows a *remote* user to seal the user VM to the target platform by encrypting his/her VM using a special purpose TPM-based bind key, which can only be used for decryption if the target platform is in a known good state. The user can also validate the required sealing properties (i.e. attested platform state, non-migratable TPM key) through our proposed approach of certifying the bind key. Finally, (Gasmi et al., 2007) propose a trusted channel which focuses on the integrity of the target platform in establishing a secure session. The trusted channels also features trusted computing mechanisms for establishing trust between end entities. Different from such an *on-*

[3]http://www.cloudsecurityalliance.org
[4]http://www.trustedcomputinggroup.org/
[5]http://www.tclouds-project.eu/
[6]http://www.eurocloud.org/

line solution, we propose a VM launch procedure that does not require that the VM is scheduled when actually transferred to the IaaS provider from the user, but it can be stored protected until it is actually scheduled on the target platform.

4 CLOUD PLATFORM SECURITY ARCHITECTURE

Naturally, the most important component in creating trust in an IaaS model, is the design of the shared resources, i.e. the cloud platform. The cloud platform must be both flexible enough to allow multiple users to run their services, and at the same time it should be easy to manage and have the needed security components to prove to the user that it is trustworthy and can protect the user data in a good way. Below, we discuss the platform security architecture we assume and that we partly[7] have designed. Key parts of this architecture have been verified in an implementation which is discussed in Section 6.

4.1 Cloud Platform Architecture

In order to allow multiple users to run their services on a shared cloud platform, we consider the XEN hypervisor (Chisnall, 2007) for platform virtualization. However, the platform architecture is flexible to use other hypervisors like KVM[8] or VMware[9] with some adaptations. The platform runs a *Management Virtual Machine* in the privileged domain i.e. dom0 along with other less privileged user domains (domU) running *User Virtual Machines* belonging to different cloud users (see Figure 2). The architecture divides the platform into four layers of abstraction where the *Services Layer* presents our implemented modules (see Section 6) which are fundamental in the trusted VM launch.

The management VM runs our implemented *Management Agent* at *Application Layer* which interacts with other entities of the network (internal and external) and performs services for them. The *Services Layer* has four major components which interact with each other and perform all security and trust management operations. The *Compartment Manager* (CM) performs the leading role by using services of other components. Other than the management of overall VM life-cycle, the main tasks performed by the CM

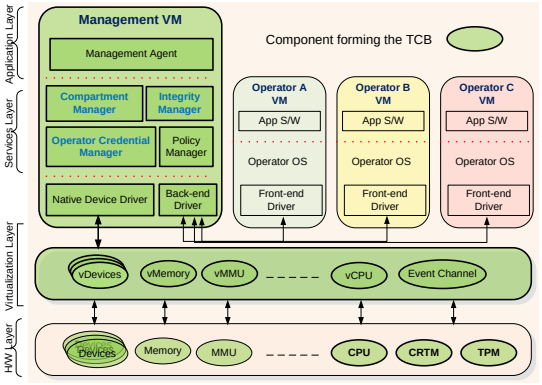

Figure 2: Cloud Platform Architecture.

are unsealing and running/launching user VMs. The CM uses the services of the *Integrity Manager* (IM) which performs various tasks pertaining to platform trust establishment. It uses TPM features and performs tasks like making runtime integrity measurements and reporting platform state for remote attestation. In principle, the CM calls IM functions for validating trust before launching a user VM. The *Credential Manager* (CredM) keeps the credentials of the licensed users to make access checks of VMs prior to launch, logs launch events and transfers licensing information to the licensing engine. The *Policy Manager* (PM) handles platform security policies for the cloud platforms, for example, a platform access policy which allows/disallows particular operations on the platform according to the client (user or provider) access rights. Furthermore, PM can be used to enforce licensing policies for different softwares and/or resource usage.

The underlying *Virtualization Layer* makes virtual instances of hardware resources so that every user VM can have its own virtual hardware. However, the architecture design choice for the hardware TPM to virtualize it or not depends upon its intended use. Virtualizing TPM instances as proposed by (Berger et al., 2006) allows every user VM to use the features offered by TPM (e.g. platform attestation, integrity reporting etc.). Though virtual TPMs allow the cloud users to benefit from the security features offered by TPM, it results in increased size of *Trusted Computing Base* (TCB) which can expose security vulnerabilities due to *hierarchical trust dependencies* (van Doorn, 2007). Therefore, we consider not to virtualize the TPM and limit its use to the hypervisor and Management VM which suffice the user requirement of validating target cloud platform *prior* to his/her VM launch. Similar models are proposed by (Haldar et al., 2004) and (Jansen et al., 2006) where the TCB is only up until the level of the hypervisor.

[7] we use a combination of state-of-the-art technologies and new designs

[8] http://www.linux-kvm.org/

[9] http://www.vmware.com/virtualization/

4.2 Platform Credentials

In the process of platform integrity verification (see Section 5.2), the cloud platform is required to report *runtime measurements* to the verifying user-client. The reported values can only be trusted if they are signed by the TPM. The cloud platform should therefore have an asymmetric key pair for attestation, a so called Attestation Identity key pair, K_{AIK}. The secret part of this key (SK_{AIK}) can ONLY be used inside the TPM chip and never leaves the TPM unencrypted. The corresponding public part, PK_{AIK} is registered with a trusted third party usually called *Privacy CA* who issues an identity certificate for the platform, a so called *AIK-Certificate* according to the TCG attestation key and certificate principles (TCG Architecture Overview, 2007).

Our proposed VM launch protocol (see section 5) also supports the *confidentiality* of user data and programs/applications - a security requirement identified in (Aslam and Gehrmann, 2011). This can be achieved by cryptographically binding the user VM to a cloud platform which is booted into a trustworthy state. We create a non-migratable (TPM 1.1) or certified-migratable (TPM 1.2) asymmetric key pair in the TPM which is used for such cryptographic binding. We refer to this key as *bind key* and denote it by K_{bind}. The public part of this key is denoted by PK_{bind}, and the corresponding private part by SK_{bind}. We create this key pair using the `TPM_CreateKey()` command (TPM Commands, 2007), and also state the required value of Platform Configuration Registers (PCRs) to use this key. A TPM key created in this way can only be used for decryption if the cloud platform is booted into a trustworthy state. In order to prove to a remote user that K_{bind} is a TPM key and is bounded to a trustworthy platform state, it is signed together with the defined PCRs using SK_{AIK} though the `TPM_CertifyKey()` command (TPM Commands, 2007) (see Section 6).

4.3 Platform Secure Boot

We assume *trusted boot* (IMM, 2006), (Sailer et al., 2004) for our cloud platform bootstrapping in which the hash of every loaded code component is recorded in the TPM PCRs by the TPM *extend* operation. In order to make the hash values in the PCRs meaningful, a corresponding *Event Structure* which contains the component's meta-data as well, is also recorded in a so called Integrity Measurement Log (IML) file. The values in the TPM PCRs along with IML are securely reported to the user as *runtime measurements* in the platform integrity verification (see Section 5.2).

5 SECURE VM LAUNCH PROTOCOL

The cloud platform security architecture presented in the previous sections and the performed pre requisites provide a basis for implementing the TCG mechanisms like remote attestation, sealing and key binding. We use these features of the TCG to design and implement a secure VM launch protocol. The VM launch process is divided into three main and distinct phases which are shown in Fig. 3 and described in the following sections.

5.1 Connect and Discovery

In order to protect the internal virtual resources, the provider needs to authenticate all resource requests towards the network through the provider gateway, which authenticates the user-client and establishes a secure session with the user-client. Similarly, the user-client must make sure that it connects to a trusted cloud provider network. We use IPsec (Frankel and Krishnan, 2011) to establish a standard VPN session between the user-client and the provider network. We use certificate-based key exchange for the session establishment for which both entities are assumed to have their digital certificates issued by a mutually trusted CA.

After the establishment of the IPsec connection, the user-client sends a resource discovery request to the *Procurement Server* which pools a list of available resources and presents them to the user-client. The user-client selects one of the presented resources which matches his criteria (feature-set, security, policy, resource location etc.). The user-client then goes through the billing and payment cycle which is not part of our protocol and is not covered here. Once the payment is done, the user-client gets a resource definition file which is a provider-signed XML file containing all important information about the procured resource. This includes 1) the URI of the resource which is used to connect to the resource, 2) a link to access *reference measurements* of the procured platform which are used in the platform integrity verification to be performed in the next step, 3) links to platform usage policies which are enforced on the platform, e.g. licensing, 4) a list of clients (user or provider) who can access this platform along with their access rights, and 5) links to any other Service Level Agreements (SLAs). The externally linked files are integrity protected by including their checksums in the provider signed resource.xml file. This resource definition file is important in the sense that it presents the provider statement about the integrity state of the

procured platform. The novelty of our solution lies in that the user can validate the proclaimed integrity of the provisioned resource. This validation is performed in the next step of the VM launch protocol.

5.2 Platform Integrity-Verification

Before launching the VM to the procured platform, the user-client needs to validate the integrity of the target platform as proclaimed by the provider. This is done according to the TCG attestation procedure (TCG Architecture Overview, 2007). We use an attestation protocol similar to the one described in (Huang and Peng, 2009) and (Sailer et al., 2004). The attestation starts with the user-client sending a nonce (N) to the cloud platform with a request to report its integrity state. The Management Agent requests the Integrity Manager to generate an Attestation Response. The attestation response contains the current state of PCRs and the *Integrity Report*. The IM requests the TPM to get the current state of PCRs signed with SK_{AIK} (see Section 4.2). This is done using the `TPM_Quote` (TPM Commands, 2007) command. The Integrity Report, current PCR state and the AIK-Certificate are sent back to the user-client. The user-client validates the AIK-Certificate, nonce and the signatures and compares the received Integrity Report with the Reference Manifests (RM) collected from Provider Reference Manifest Authority (PRA) using the link provided in resource definition XML file in the previous step (see Figure 1). The role of PRA is to create/aggregate, sign and publish reference metrics for each cloud platform (IMM, 2006). Once the validations are successful, the user-client then computes the hash (h1) of the received *Attestation Response* which is used to bind the next VM launch step to the current integrity verification step of the launch protocol.

5.3 VM Launch

The final stage of the proposed protocol is the actual launch of the user VM. We protect user data and applications (user programs) by cryptographically binding them to the cloud platform using the TPM. Such binding mechanism fulfills an important security requirements, that is, the user data is available only if the cloud platform is running trusted software components thus protecting data confidentiality against malware. The platform-sealed launch introduced here is done by using a specially created non-migratable or certified-migratable, asymmetric key (K_{bind}) which is created in the pre-launch phase (see Section 4.2). The user-client requests for the PK_{bind} along with assurances that the K_{bind} is a non-migratable or certified-migratable TPM key, and it can only be released/used if the cloud platform is in the trusted state as attested in the previous step. The cloud platform loads K_{bind}

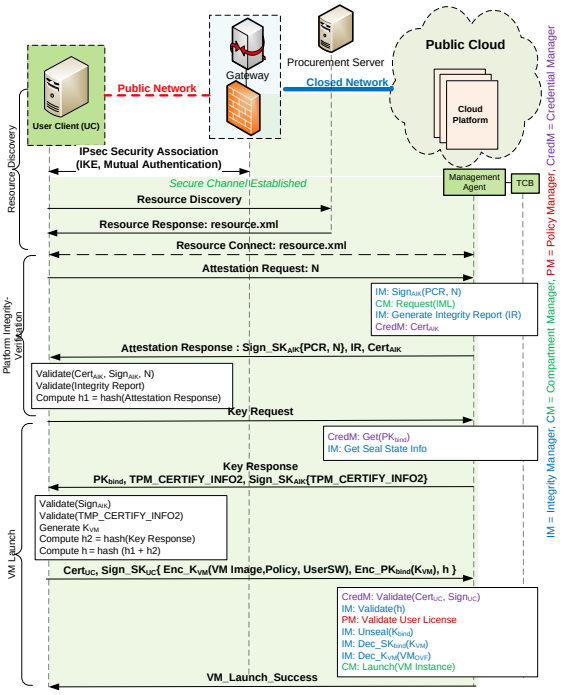

Figure 3: VM Launch Protocol.

and calls `TPM_CertifyKey()` command (TPM Commands, 2007). The validation result of this command includes a `TPM_CERTIFY_INFO` structure which is sent to the user-client along with K_{bind}. The `TPM_CERTIFY_INFO` structure is K_{AIK} signed which certifies that K_{bind} is a non-migratable TPM key. The `TPM_CERTIFY_INFO` structure includes the digest for K_{bind} which is compared against the computed digest of the received K_{bind} to authenticate the received K_{bind}. The `TPM_CERTIFY_INFO` structure also includes the required `PCR_INFO` digest which is compared against the expected/trusted PCR hash value to ensure that K_{bind} can only be used when the platform is in a trusted state.

After all validations, the user-client generates a symmetric key K_{VM} which is used to encrypt the VM package[10] which contains the VM image, any user credentials and user applications which will run in the user VM instance. The K_{VM} is encrypted with PK_{bind} to ensure the required platform-binding properties stated earlier. In order to cryptographically bind the different protocol steps - attestation, bind key exchange and launch - the user-client computes the hash of *Key Response* message(h2), appends it to h1(computed in previous stage), and computes h

[10]The VM is packaged using Open Virtualization Format (OVF)

which is also sent to the cloud platform. Finally, the user-client signs the VM Launch command (shown below) so that the user cannot repudiate the VM launch later on.

$$Cert_{uc}, Sign\_SK_{uc}[K_{VM}\{VM_{OVF}\}, PK_{bind}\{K_{VM}\}, h]$$

The cloud platform performs the necessary validations and checks the user license for VM launch rights, and (optionally) also updates the corresponding charging records. It then loads the bind key, decrypts the K_{VM}, decrypts the VM package and asks the Compartment Manager to run the user VM instance. Finally, a SUCCESS or FAIL launch message is returned to the user-client. After successful VM launch the user-client can then connect to the running VM instance for any management operation.

6 IMPLEMENTATION

The Secure VM Launch Protocol, as presented in the previous section, requires multiple entities/platforms communicating with each other in a complete IaaS provisioning system (see Figure. 1). In order to setup a complete framework and test a running prototype of the launch protocol, we implemented the required functionalities for the following entities:

- User-Provider Client
- Procurement Server
- Cloud Platform: Management Agent

In the following sections, we discuss the implementation of these different entities in more details.

6.1 User-Provider Client

The cloud user needs a client for resource procurement, VM launch and its later management[11]. Similarly, the provider also needs a client for the management and administrative tasks on the provisioned cloud platform. In our prototype, we have implemented a single client that can run in two operational modes, i.e. `launch` and `manage`, and thus can be used by the user or provider. In this paper, we only focus on the cloud user using the client program in the `launch` mode. This is done by specifying the 'launch' argument along with a 'user certificate' which authenticates the cloud user to the *Procurement Server* and *Cloud Platform*. The client program can be executed as follows:

*# **runClient** launch user.cer*

[11] VM management is not covered in this paper

The client program running in `launch` mode by the cloud user connects to the *Procurement Server*(see Section 6.2) and receives a `resource.xml` file (see Section 5.1) if a resource is procured successfully. The `resource.xml` file is then parsed to locate the URI of the procured cloud platform and connects to it for *Integrity Verification* before finally sending the *VM Launch* request.

6.2 Procurement Server

Conceptually, the *Procurement Server* can be a combination of multiple entities like cloud controller which pools and allocates cloud resources, Payment and Licensing engines to sell available cloud resources. However, our implementation of the *Procurement Server* is limited to listening to the user-client requests for available resources and generating the signed `resource.xml` file to be sent back to the user-client who can then use this file to connect to the procured resource.

6.3 Cloud Platform: Management Agent

The cloud platform which runs the VM instance for the user, performs the integrity verification and VM launch functionalities for the user. It runs a *Management Agent* in the privileged domain which can access TPM services. The Management Agent and its modules can use TPM services only through the TCG Software Stack (TSS, 2007) for which we use a Java-based implementation of the TSS called jTSS, developed at the Institute for Applied Information Processing and Communication (IAIK), Graz University of Technology.

The *Management Agent* listens for the client requests to *launch* his/her VM and runs the VM launch protocol. The sequence of messages between the user-client and the Management Agent perform *Platform Integrity Verification* (see Section 5.2) and *VM Launch* (see Section 5.3). For this purpose, three important *response messages* are generated by the Management Agent:

- Attestation Response
- Bind-key Response
- Launch Response

These response messages are generated using the modules in the *Services Layer*, most notably the *Integrity Manager* and the *Compartment Manager*, and are discussed in the following sections.

6.3.1 Attestation Response

The *Attestation Response* contains the 1) PCR_Quote, 2) Integrity Report and 3) the AIK Certificate. The PCR_Quote is generated by the hardware TPM using SK_{AIK} which is accessed by calling the `getQuote(expectedPcrComposite, N)` method of the *Integrity Manager*. The *Compartment Manager* uses the `requestIML()` method to load the Integrity Report and the AIK Certificate is loaded using the `getAikCertificate()` method of the *Credential Manager*. Finally, the generated *Attestation Response* is sent to the user-client for integrity verification.

6.3.2 Bind-Key Response

The purpose of sending the *Bind-Key Response* to the user-client is to allow the user to validate that the K_{bind} is a TPM-based key and it can only be used if the platform is in the state verified by the user-client in the previous step of integrity verification. This is achieved by calling the `certifyBindKey()` method of the *Integrity Manager* which loads K_{bind} and K_{AIK} to generate a K_{AIK} signed `TPM_CERTIFY_INFO` structure. The bind-key response message is sent to the user-client which includes PK_{bind} along with a K_{AIK} signed `TPM_CERTIFY_INFO` structure which can only be generated by the TPM thereby certifying K_{bind}.

6.3.3 Launch Response

The *Launch Response* is a simple *SUCESS* message for the user-client or a detailed *ERROR* message in case of a failed launch. The *Launch Response* message is generated by the `getLaunchResponse()` method of the Management Agent which performs validations for user certificate, signature and the received hash value (h). Furthermore, it calls the `decryptKvm()` method of the Integrity Manager to get the symmetric key K_{VM}, which is required to decrypt the received VM package. Since K_{VM} is encrypted with PK_{bind} (a TPM-based key), the Integrity Manager requests the TPM to use the corresponding SK_{bind} to decrypt K_{VM}. The bind key (K_{bind}) is created (see Section 4.2) with the property that it can be used for decryption *only* if the TPM PCRs have the same value as stated at the time of key creation[12]. This implies that only a platform in a good known state (i.e. verified integrity state) can get decrypted K_{VM} and hence can decrypt the VM package. Once the VM package is successfully decrypted, the *Compartment Manager* uses the `unpackage()` and `launchVM()` methods to launch the user VM.

[12]The platform is not required to be in the expected state at the time of key creation

7 SECURITY ANALYSIS

We analyze the security of our launch protocol in the following sections along with reflections on the way threats identified in section 2.2 are addressed. The summary of the security analysis is presented in the Table 1.

7.1 Authentication

Due to the well known vulnerabilities, (Somorovsky et al., 2011), we decided to not fully rely on and use the existing cloud interfaces and APIs for VM launch. Instead, we used existing well established protocols such as IKE (with certificate-based key exchange) and IPSec for the establishment of a virtual private network (VPN) between the user-client and the provider network. The mutual authentication of both entities protects against any impersonation attempts by the attacker (Threats 2 and 3). The user-client also checks the authenticity of the cloud platform during platform integrity verification (Section 5.2). We also ensure that only the authenticated cloud platform receives and runs the user VM by sealing the VM image to the target cloud platform (Threat 5)

7.2 Confidentiality

The confidentiality of the user applications and data is ensured by binding the VM image to the trusted cloud platform (Threat 8). This is done by using the bind key which is cryptographically bound to the provider-stated and user-validated state of the platform. This means that the user data and applications cannot be accessed even if the VM image is used to create a VM instance on another compromised platform because it cannot be decrypted on any platform other than the trusted one. However, a provider with administrative access to the cloud platform can get a memory dump of the running user VM to extract user credentials (Rocha and Correia, 2011). Similarly, an insider can migrate the running VM to a compromised platform to generate more possibilities of breach of confidentiality. Our proposed approach to protect against such insider attacks is to restrict the access of such sensitive administrative tasks (e.g taking snapshot of the running VM, VM migration, etc.) by enforcing access rights through the *Policy Manager* module. The `resource.xml` file, which represents the procured resource, contains a list of clients (provider and user) with their access permissions which could be reviewed by the user before launching his VM. The details of a trusted *Policy Manager* with *Access Control Module*, which could allow sensitive administra-

Table 1: Security Analysis of our Proposed Mechanisms to Mitigate Threats.

Threat	Security Service	Applied Mechanism
1. Insecure Interfaces	Authentication	IPSec, IKE association
2. User Client Authenticity	Authentication	IPSec, IKE association
3. Provider Network Attacks	Authentication	Gateway, IPSec
4. Disowning Malicious VM	Non-repudiation	User-signed VM Launch
5. Cloud Platform Authenticity[1]	Authentication	Remote Attestation, VM Sealing
6. Insider Attacks[1]	Platform Integrity	Remote Attestation
7. Mis-configured Cloud Platform[1]	Platform Integrity	Remote Attestation
8. User-data Leak[1]	Confidentiality	VM Sealing

[1] solved by our proposed techniques

tive tasks like VM migration, is a subject of future work.

Another aspect of data and applications confidentiality is the protection against attacks from a malicious VM running in parallel on the same physical platform. It is very hard to give any firm guarantees with respect to the secure isolation provided by any virtualization layer (Ormandy, 2007). We use XEN which is a widely used open virtualization layer that has been and is subject to continuous in depth security analysis.

7.3 Integrity

The cloud platform integrity is an important security requirement for a cloud user which is achieved by using the TCG mechanism of *remote attestation*, thereby addressing threat 6 and 7. We presented the complete architecture (see Section 4) required for the platform integrity verification performed in the launch protocol which allows the cloud user to verify the compliance of the cloud platform to the promised security profile (Threat 8). However, the *Boot Security* and provisioning of the *Reference Measurements* are critical in defining the overall integrity of the platform. For example, if the published reference metrics include bad software or softwares with vulnerabilities, the security of the platform becomes questionable. Hence, provisioning of high quality reference metrics assumes *good* softwares as well, which is a challenge in the proposed architecture.

7.4 Non-Repudiation

Both parties should have mechanisms to prove the transaction/action in case of repudiation by the other party (Threat 4). Two such critical actions and corresponding security requirements are 1) a proof that the cloud user has requested and paid for the resource to launch his VM, and 2) a proof that the cloud user has actually sent a request to run his VM instance (so that he cannot disown it later). In our protocol, non-repudiation is achieved by sending a provider signed `resource.xml` for the first action and the signature of the cloud user on the `VM_Launch` request for the second action.

7.5 Replay Protection

The cloud user sends a nonce (N) in the attestation request which is returned in the attestation response (signed `TPM_Quote`). The *Attestation Response* and the *Bind-key Response* are hashed to keep session binding and protect against replay attacks. The randomness in the hash values (h,h1 and h2) comes from the original nonce (N) sent in the attestation request.

8 CONCLUSION

In this paper we have addressed one of the major hurdles for enterprises to shift their business processes to the clouds, i.e. lack of security guarantees. We identified major security threats with respect to virtual machine provision in IaaS. Many of the identified threats are general and well known for any distributed system for which standard security mechanisms already exist. We proposed a novel *launch protocol* which uses standard security mechanisms for the known threats and combined those with trusted computing techniques to fill the trust deficit between cloud stakeholders. In addition, we have designed a complete software architecture needed to run the protocol and protect the VM data. Our protocol enables *technical trust* which allows cloud users to validate the security guarantees given by the cloud provider *before* using the provisioned service. We also implemented the designed protocol to evaluate its feasibility in terms of implementation effort and complexity. Finally, we performed security analysis to validate that all security requirements are fulfilled and all identified threats have been addressed comprehensively. In our analysis, we showed how a user VM can strongly be bounded to a particular platform to fulfill users' security requirements. However, this strong

binding also limits the flexibility with respect to load management, platform upgrade, etc. These limitations can be reduced by adding trusted VM migration mechanism to the solution. Our suggested architecture allows opportunities for such extensions but the details are left for future research.

REFERENCES

acTvSM (2010). Advanced Cryptographic Trusted Virtual Security Module. http://www.iaik.tugraz.at/content/research/trusted_computing/actvsm/.

Aslam, M. and Gehrmann, C. (2011). Security Considerations for Virtual Platform Provisioning. In *ECIW '11: Proceedings of the 10th European Conference on Information Warfare and Security*, pages 283–290, UK. The Institute of Cybernetics at the Tallinn University of Technology, Academic Publishing Limited.

Berger, S., Cáceres, R., Goldman, K. A., Perez, R., Sailer, R., and van Doorn, L. (2006). vTPM: Virtualizing the Trusted Platform Module. In *USENIX-SS'06: Proceedings of the 15th Conference on USENIX Security Symposium*, Berkeley, CA, USA. USENIX Association.

Chisnall, D. (2007). *The Definitive Guide to the Xen Hypervisor (Prentice Hall Open Source Software Development Series)*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

CircleID (2009). Survey: Cloud Computing 'No Hype', But Fear of Security and Control Slowing Adoption.

Cloud Computing and Security (2010). Cloud Computing and Security - A Natural Match. http://www.trustedcomputinggroup.org/resources/cloud_computing_and_security_a_natural_match.

CSA Guide (2009). Security guidance for critical areas of focus in cloud computing. http://www.cloudsecurityalliance.org/guidance/csaguide.v2.1.pdf.

CSA Threats (2010). Top Threats to Cloud Computing. Technical Report Version 1.0, Cloud Security Alliance.

Frankel, S. and Krishnan, S. (2011). IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071 (Informational).

Gasmi, Y., Sadeghi, A.-R., Stewin, P., Unger, M., and Asokan, N. (2007). Beyond Secure Channels. In *STC '07: Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing*, pages 30–40, New York, NY, USA. ACM.

Haldar, V., Chandra, D., and Franz, M. (2004). Semantic Remote Attestation - A Virtual Machine directed approach to Trusted Computing. In *USENIX Virtual Machine Research and Technology Symposium*, pages 29–41.

Huang, X. and Peng, Y. (2009). An Effective Approach for Remote Attestation in Trusted Computing. In *WISA 2009 : Proceedings of the 2nd International Symposium on Web Information Systems and Applications*, pages 80–83, FIN-90571, OULU, FINLAND. Academy Publisher.

IMM (2006). TCG Infrastructure Architecture Part-II - Integrity Management. http://www.trustedcomputinggroup.org/resources.

Jansen, B., Ramasamy, H. V., and Schunter, M. (2006). Flexible Integrity Protection and Verification Architecture for Virtual Machine Monitors. In *The Second Workshop on Advances in Trusted Computing (WATC 06 Fall*.

Krutz, R. L. and Vines, R. D. (2010). *Cloud Security: A Comprehensive Guide to Secure Cloud Computing*. Wiley Publishing.

Landfermann, R., Kuhlmann, D., Kuhlmann, D., L, R., Ramasamy, H. V., Ramasamy, H. V., Schunter, M., Schunter, M., Ramunno, G., Ramunno, G., Vernizzi, D., and Vernizzi, D. (2006). D.: An Open Trusted Computing Architecture – Secure Virtual Machines Enabling User-defined Policy Enforcement. www.opentc.net.

Ormandy, T. (2007). An empirical Study into the Security Exposure to Hosts of Hostile Virtualized Environments. In *CanSecWest*.

Rocha, F. and Correia, M. (2011). Lucy in the sky without diamonds: Stealing confidential data in the cloud. *Dependable Systems and Networks Workshops*, 0:129–134.

Sailer, R., Zhang, X., Jaeger, T., and van Doorn, L. (2004). Design and implementation of a tcg-based integrity measurement architecture. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 16–16, Berkeley, CA, USA. USENIX Association.

Somorovsky, J., Heiderich, M., and Jensen, M. (2011). All your clouds are belong to us: security analysis of cloud management interfaces. *computing security*.

TCG Architecture Overview (2007). TCG Specification Architecture Overview. http://www.trustedcomputinggroup.org/resources.

TMI (2010). TCG Trusted Multi-Tenant Infrastructure. http://www.trustedcomputinggroup.org/developers/trusted_multitenant_infrastructure.

TPM Commands (2007). TPM Specification, TPM Main Part-III Design Principles. http://www.trustedcomputinggroup.org/resources.

TSS (2007). TCG Software Stack (TSS) Specification. http://www.trustedcomputinggroup.org/resources.

van Doorn, L. (2007). Trusted Computing Challenges. In *STC '07: Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing*, pages 1–1, New York, NY, USA. ACM.

Vaquero, L. M., Rodero-Merino, L., and Morán, D. (2011). Locking the sky: a survey on iaas cloud security. *Computing*, 91:93–118.