Towards Trusted Cloud Computing

Nuno Santos Krishna P. Gummadi Rodrigo Rodrigues

MPI-SWS

Abstract

Cloud computing infrastructures enable companies to cut costs by outsourcing computations on-demand. However, clients of cloud computing services currently have no means of verifying the confidentiality and integrity of their data and computation.

To address this problem we propose the design of a *trusted cloud computing platform* (TCCP). TCCP enables Infrastructure as a Service (IaaS) providers such as Amazon EC2 to provide a closed box execution environment that guarantees confidential execution of guest virtual machines. Moreover, it allows users to attest to the IaaS provider and determine whether or not the service is secure before they launch their virtual machines.

1 Introduction

Companies can greatly reduce IT costs by offloading data and computation to cloud computing services. Still, many companies are reluctant to do so, mostly due to outstanding security concerns. A recent study [2] surveyed more than 500 chief executives and IT managers in 17 countries, and found that despite the potential benefits, executives "trust existing internal systems over cloud-based systems due to fear about security threats and loss of control of data and systems". One of the most serious concerns is the possibility of confidentiality violations. Either maliciously or accidentally, cloud provider's employees can tamper with or leak a company's data. Such actions can severely damage the reputation or finances of a company.

In order to prevent confidentiality violations, cloud services' customers might resort to encryption. While encryption is effective in securing data before it is stored at the provider, it cannot be applied in services where data is to be computed, since the unencrypted data must reside in the memory of the host running the computation. In Infrastructure as a Service (IaaS) cloud services such as Amazon's EC2, the provider hosts virtual machines (VMs) on behalf of its customers, who can do arbitrary computations. In these systems, anyone with privileged access to the host can read or manipulate a customer's data. Consequently, customers cannot protect their VMs on their own.

Cloud service providers are making a substantial effort to secure their systems, in order to minimize the threat of insider attacks, and reinforce the confidence of customers. For example, they protect and restrict access to the hardware facilities, adopt stringent accountability and auditing procedures, and minimize the number of staff who have access to critical components of the infrastructure [8]. Nevertheless, insiders that administer the software systems at the provider backend ultimately still possess the technical means to access customers' VMs. Thus, there is a clear need for a technical solution that guarantees the confidentiality and integrity of computation, in a way that is verifiable by the customers of the service.

Traditional trusted computing platforms like Terra [4] take a compelling approach to this problem. For example, Terra is able to prevent the owner of a physical host from inspecting and interfering with a computation. Terra also provides a remote attestation capability that enables a remote party to determine upfront whether the host can securely run the computation. This mechanism reliably detects whether or not the host is running a platform implementation that the remote party trusts. These platforms can effectively secure a VM running in a single host. However, many providers run data centers comprising several hundreds of machines, and a customer's VM can be dynamically scheduled to run on any one of them. This complexity and the opaqueness of the provider backend creates vulnerabilities that traditional trusted platforms cannot address.

This paper proposes a *trusted cloud computing platform* (TCCP) for ensuring the confidentiality and integrity of computations that are outsourced to IaaS ser-

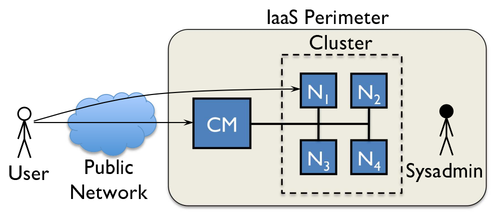


Figure 1: Simplified architecture of Eucalyptus.

vices. The TCCP provides the abstraction of a closed box execution environment for a customer's VM, guaranteeing that no cloud provider's privileged administrator can inspect or tamper with its content. Moreover, before requesting the service to launch a VM, the TCCP allows a customer to reliably and remotely determine whether the service backend is running a trusted TCCP implementation. This capability extends the notion of attestation to the entire service, and thus allows a customer to verify if its computation will run securely.

In this paper we show how to leverage the advances of trusted computing technologies to design the TCCP. Section 2 introduces these technologies and describes the architecture of an IaaS service. Section 3 presents our design of TCCP. Although we do not yet have a working prototype of TCCP, the design is sufficiently detailed that we are confident that a solution to the problem under discussion is possible.

2 Background

2.1 Infrastructure as a Service

Today, myriads of cloud providers offer services at various layers of the software stack. At lower layers, Infrastructure as a Service (IaaS) providers such as Amazon, Flexiscale, and GoGrid allow their customers to have access to entire virtual machines (VMs) hosted by the provider. A customer, and user of the system, is responsible for providing the entire software stack running inside a VM. At higher layers, Software as a Service (SaaS) systems such as Google Apps offer complete online applications than can be directly executed by their users.

The difficulty in guaranteeing the confidentiality of computations increases for services sitting on higher layers of the software stack, because services themselves provide and run the software that directly manipulates customer's data (e.g., Google Docs). In this paper we focus on the lower layer IaaS cloud providers where securing a customer's VM is more manageable.

While very little detail is known about the internal organization of commercial IaaS services, we describe (and base our proposal on) Eucalyptus [6], an open source IaaS platform that offers an interface similar to EC2. Figure 1 presents a very simplified architecture of Eucalyptus. This system manages one or more clusters whose nodes run a virtual machine monitor (typically Xen) to host customers' VMs. Eucalyptus comprehends a set of components to manage the clusters. For simplicity, our description aggregates all these components in a single *cloud manager* (CM) that handles a single cluster; we refer the reader to [6] for more details.

From the perspective of users, Eucalyptus provides a web service interface to launch, manage, and terminate VMs. A VM is launched from a virtual machine image (VMI) loaded from the CM. Once a VM is launched, users can log in to it using normal tools such as ssh. Aside from the interface to every user, the CM exports services that can be used to perform administrative tasks such as adding and removing VMIs or users. Xen supports live migration, allowing a VM to shift its physical host while still running, in a way that is transparent to the user. Migration can be useful for resource consolidation or load balancing within the cluster.

2.2 Attack model

A sysadmin of the cloud provider that has privileged control over the backend can perpetrate many attacks in order to access the memory of a customer's VM. With root privileges at each machine, the sysadmin can install or execute all sorts of software to perform an attack. For example, if Xen is used at the backend, Xenaccess [7] allows a sysadmin to run a user level process in Dom0 that directly accesses the content of a VM's memory at run time. Furthermore, with physical access to the machine, a sysadmin can perform more sophisticated attacks like cold boot attacks and even tamper with the hardware.

In current IaaS providers, we can reasonably consider that no single person accumulates all these privileges. Moreover, providers already deploy stringent security devices, restricted access control policies, and surveillance mechanisms to protect the physical integrity of the hardware. Thus, we assume that, by enforcing a security perimeter, the provider itself can prevent attacks that require physical access to the machines.

Nevertheless, sysadmins need privileged permissions at the cluster's machines in order to manage the software they run. Since we do not precisely know the praxis of current IaaS providers, we assume in our attack model that sysadmins can login remotely to any machine with root privileges, at any point in time. The only way a sysadmin would be able to gain physical access to a node running a costumer's VM is by diverting this VM to a machine under her control, located outside the IaaS's security perimeter. Therefore, the TCCP must be able to 1) confine the VM execution inside the perimeter, and 2) guarantee that at any point a sysadmin with root privileges remotely logged to a machine hosting a VM cannot access its memory.

2.3 Trusted Computing

The Trusted Computing Group (TCG) [10] proposed a set of hardware and software technologies to enable the construction of trusted platforms. In particular, the TCG proposed a standard for the design of the *trusted platform module* (TPM) chip that is now bundled with commodity hardware. The TPM contains an endorsement private key (EK) that uniquely identifies the TPM (thus, the physical host), and some cryptographic functions that cannot be modified. The respective manufacturers sign the corresponding public key to guarantee the correctness of the chip and validity of the key.

Trusted platforms [1, 4, 5, 9] leverage the features of TPM chips to enable *remote attestation*. This mechanism works as follows. At boot time, the host computes a measurement list ML consisting of a sequence of hashes of the software involved in the boot sequence, namely the BIOS, the bootloader, and the software implementing the platform. The ML is securely stored inside the host's TPM. To attest to the platform, a remote party challenges the platform running at the host with a nonce n_U. The platform asks the local TPM to create a message containing both the ML and the n_U, encrypted with the TPM's private EK. The host sends the message back to the remote party who can decrypt it using the EK's corresponding public key, thereby authenticating the host. By checking that the nonces match and the ML corresponds to a configuration it deems trusted, a remote party can reliably identify the platform on an untrusted host.

A trusted platform like Terra [4] implements a thin VMM that enforces a *closed box* execution environment, meaning that a guest VM running on top cannot be inspected or modified by a user with full privileges over the host. The VMM guarantees its own integrity until the machine reboots. Thus, a remote party can attest to the platform running at the host to verify that a trusted VMM implementation is running, and thus make sure that her computation running in a guest VM is secure.

Given that a traditional trusted platform can secure the computation on a single host, a natural approach to secure an IaaS service would be to deploy the platform at each node of the service's backend (see Figure 1). However, this approach is insufficient: a sysadmin can divert a customer's VM to a node not running the platform, either when the VM is launched (by manipulating the CM), or during the VM execution (using migration). Consequently, the attestation mechanism of the platform does not guarantee that the measurement list obtained by the remote party corresponds to the actual configuration of the host where the VM has been running (or will be running in the future). Therefore, the TCCP needs to provide a remote attestation that guarantees the immutability of the platform's security properties in the backend.

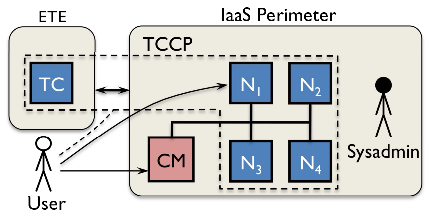


Figure 2: The components of the trusted cloud computing platform include a set of *trusted nodes* (N) and the *trusted coordinator* (TC). The untrusted *cloud manager* (CM) makes a set of services available to users. The TC is maintained by an *external trusted entity* (ETE).

3 Trusted Cloud Computing Platform

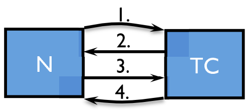
We present the *trusted cloud computing platform* (TCCP) that provides a closed box execution environment by extending the concept of trusted platform to an entire IaaS backend. The TCCP guarantees the confidentiality and the integrity of a user's VM, and allows a user to determine up front whether or not the IaaS enforces these properties. Next section gives an overview of TCCP, and Section 3.2 presents a detailed design.

3.1 Overview

TCCP enhances today's IaaS backends to enable closed box semantics without substantially changing the architecture (Figure 2). The trusted computing base of the TCCP includes two components: a *trusted virtual machine monitor* (TVMM), and a *trusted coordinator* (TC).

Each node of the backend runs a TVMM that hosts customers' VMs, and prevents privileged users from inspecting or modifying them. The TVMM protects its own integrity over time, and complies with the TCCP protocols. Nodes embed a certified TPM chip and must go through a secure boot process to install the TVMM. Due to space limitations we will not go into detail about the design of the TVMM, and we refer the reader to [5] for an architecture that can be leveraged to build a TVMM that enforces local closed box protection against a malicious sysadmin.

The TC manages the set of nodes that can run a customer's VM securely. We call these nodes *trusted nodes*. To be trusted, a node must be located within the security perimeter, and run the TVMM. To meet these conditions, the TC maintains a record of the nodes located in the security perimeter, and attests to the node's platform to verify that the node is running a trusted TVMM



1. n_N
2. $\{ML_{TC}, n_N\}_{EK^p_{TC}}, n_{TC}$
3. $\{\{ML_N, n_{TC}\}_{EK^p_N}, TK^P_N\}_{TK^P_{TC}}$
4. $\{accepted\}_{TK^P_N}$

Figure 3: Message exchange during node registration.

implementation. The TC can cope with the occurrence of events such as adding or removing nodes from a cluster, or shutting down nodes temporarily for maintenance or upgrades. A user can verify whether the IaaS service secures its computation by attesting to the TC.

To secure the VMs, each TVMM running at each node cooperates with the TC in order to 1) confine the execution of a VM to a trusted node, and to 2) protect the VM state against inspection or modification when it is in transit on the network. The critical moments that require such protections are the operations to *launch*, and *migrate* VMs. In order to secure these operations, the TCCP specifies several protocols (see Section 3.2). Due to space constraints, we do not address other critical operations such as *suspend/resume* allowed by Xen.

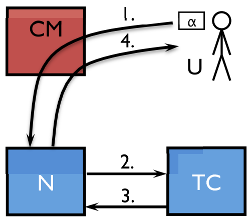
We assume an *external trusted entity* (ETE) that hosts the TC, and securely updates the information provided to the TC about the set of nodes deployed within the IaaS perimeter, and the set of trusted configurations. Most importantly, sysadmins that manage the IaaS have no privileges inside the ETE, and therefore cannot tamper with the TC. We envision that the ETE should be maintained by a third party with little or no incentive to collude with the IaaS provider e.g., by independent companies analogous to today's certificate authorities like VeriSign.

3.2 Detailed Design

In this section we detail the most relevant TCCP mechanisms. We describe the protocols that manage the set of nodes of the platform that are trusted (Section 3.2.1), and the protocols that secure the operations involving VM management, namely launching and migrating VMs (Section 3.2.2). In these protocols, we use the following notation for cryptographic operations. The pair $\langle K^p, K^P \rangle$ represents the private-public keys of an asymmetric cryptography keypair. Notation $\{y\}_{K^x}$ indicates that data y is encrypted with key K^x. We use a specific notation for the following keys: EK_x denote endorsement keys, TK_x indicate trusted keys, and K_x denote session keys. Nonces n_x, unique numbers generated by x, help detect message replays.

3.2.1 Node management

The TC dynamically manages the set of trusted nodes that can host a VM by maintaining a directory containing, for each node within the security perimeter, the public endorsement key EK^P_N identifying the node's TPM, and the expected measurement list ML_N. The ETE makes some properties of the TC securely available to the public, namely the EK^P_{TC}, the ML_{TC}, and the TK^P_{TC} (identifying the TC). Both the ML_N and the ML_{TC} express the canonical configurations that a remote party is expected to observe when attesting to the platform running on a node N or on the TC, respectively.



1. $\{\alpha, \#\alpha\}_{K_{VM}}\{n_U, K_{VM}\}_{TK^P_{TC}}$
2. $\{\{\{n_U, K_{VM}\}_{TK^P_{TC}}, n_N\}_{TK^p_N}, N\}_{TK^P_{TC}}$
3. $\{\{n_N, n_U, K_{VM}\}_{TK^P_N}\}_{TK^p_{TC}}$
4. $\{n_U, N\}_{K_{VM}}$

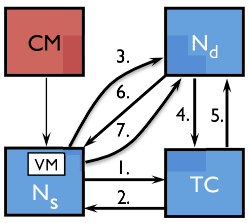
Figure 4: Message exchange during VM launch.

In order to be trusted, a node must register with the TC by complying to the protocol depicted on Figure 3. In steps 1 and 2, N attests to the TC to avoid an impersonation of the TC by an attacker: N sends a challenge n_N to the TC, and the TC replies with its bootstrap measurements ML_{TC} encrypted with EK^p_{TC} to guarantee the authenticity of the TC. If the M_{TC} matches the expected configuration, it means the TC is trusted. Reversely, the TC also attests to N by piggybacking a challenge n_{TC} in message 2, and checking whether the node is authentic, and is running the expected configuration (step 3). The node generates a keypair $\langle TK^p_N, TK^P_N \rangle$, and sends its public key to the TC. If both peers mutually attest successfully, the TC adds TK^P_N to its node database, and sends message 4 to confirm that the node is trusted. Key TK_N certifies that node N is trusted.

In the case that a trusted node reboots, the TCCP must guarantee that the node's configuration remains trusted, otherwise the node could compromise the security of the TCCP. To ensure this, the node only keeps TK^p_N in memory causing the key to be lost once the machine reboots. The node is thus banned from the TCCP, since it will not be able to decrypt messages encrypted with the previous key, and must repeat the registration protocol.

3.2.2 Virtual machine management

We present the TCCP protocols to secure the VM launch and migration operations. When launching a VM, the TCCP needs to guarantee that 1) the VM is launched on a trusted node, and 2) the sysadmin is unable to inspect or tamper with the initial VM state as it traverses the path between the user and the node hosting the VM. The initial VM state α contains the VM image (VMI) (that can be personalized and contain secret data) and the user's



1. $\{\{N_d, n_{s1}\}_{TK_N^p}, N_s\}_{TK_{TC}^P}$
2. $\{\{n_{s1}, TK_{N_d}^P\}_{TK_{N_s}^P}\}_{TK_{TC}^p}$
3. $\{\{K_S, n_{s2}\}_{TK_{N_s}^p}, N_s\}_{TK_{N_d}^P}$
4. $\{\{N_s, n_d\}_{TK_{N_d}^p}, N_d\}_{TK_{TC}^P}$
5. $\{\{n_d, TK_{N_s}^P\}_{TK_{N_d}^P}\}_{TK_{TC}^p}$
6. $\{n_d\}_{K_S}$
7. $\{VM_{id}, \#VM_{id}\}_{K_S}$

Figure 5: Message exchange during VM migrate.

public key (used for ssh login)[1]. In practice, the user can decide to use a VMI provided by the IaaS.

To enforce these requirements, the parties involved in launching a VM follow the protocol depicted in Figure 4. The protocol is designed on the fact that, before launching the VM, a user does not know which physical node the VM will be assigned, and, among the components of the service, only trusts the TC. First, the user generates a session key K_{VM}, and sends message 1 to the CM containing: α and α's hash encrypted with the session key (to protect the confidentiality and integrity of the initial state), and K_{VM} encrypted with TK_{TC}^P. Encrypting the session key with the TC's public key ensures that only the TC can authorize someone to access α. The TC only authorizes trusted nodes.

Upon receiving the request to launch a VM, the CM designates a node N from the cluster to host the VM, and forwards the request to N. Since the node needs to access α in order to boot the VM, it sends message 2 to TC which decrypts K_{VM} on N's behalf. This message is encrypted with TK_N^p so that the TC can verify whether N is trusted. If the corresponding public key is not found in the TC's trusted node database, the request is denied. This would have been the case had the CM diverted the request to a node controlled by a malicious sysadmin. Otherwise, the node is reckoned to be trusted; the TC decrypts the session key, and sends it to the node in message 3, such that only N can read the key. N is now able to decrypt α, and boot the VM. Finally, the node sends message 4 to the user containing the identity of the node running the VM.

In live migration [3], the state of an executing VM is transfered between two nodes: a source N_s and a destination N_d. To secure this operation, both nodes must be trusted, and the VM state must remain confidential and unmodified while it is in transit over the network. Figure 4 shows the sequence of messages involved in securing the migration of a VM. In steps 1 and 2, N_s asks TC to check whether N_d is trusted. In message 3, N_s negotiates a session key K_S with N_d that will be used to secure the transfer of the VM state. Before accepting the key, N_d first verifies that N_s is trusted (steps 4 and 5). If both nodes mutually authenticate successfully, N_d acknowledges the acceptance of the session key to the K_S (step 6), and, in message 7, N_s finally transfers the encrypted and hashed VM state to the N_d, guaranteeing the confidentiality and integrity of the VM.

[1] In current IaaS services, the user public key is injected in the VM at launch time. A possible attack could be to inject more keys or other malicious software.

4 Conclusions and Future Work

In this paper, we argue that concerns about the confidentiality and integrity of their data and computation are a major deterrent for enterprises looking to embrace cloud computing. We present the design of a *trusted cloud computing platform* (TCCP) that enables IaaS services such as Amazon EC2 to provide a closed box execution environment. TCCP guarantees confidential execution of guest VMs, and allows users to attest to the IaaS provider and determine if the service is secure before they launch their VMs. We plan to implement a fully functional prototype based on our design and evaluate its performance in the near future.

References

[1] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn. vTPM: virtualizing the trusted platform module. In *Proc. of USENIX-SS'06*, Berkeley, CA, USA, 2006.

[2] Survey: Cloud Computing 'No Hype', But Fear of Security and Control Slowing Adoption. `http://www.circleid.com/posts/20090226_cloud_computing_hype_security/`.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. of NSDI'05*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[4] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A Virtual Machine-Based Platform for Trusted Computing. In *Proc. of SOSP'03*, 2003.

[5] D. G. Murray, G. Milos, and S. Hand. Improving Xen security through disaggregation. In *Proc. of VEE'08*, pages 151–160, New York, NY, USA, 2008.

[6] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus: A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems. Technical Report 2008-10, UCSB Computer Science, 2008.

[7] B. D. Payne, M. Carbone, and W. Lee. Secure and Flexible Monitoring of Virtual Machines. In *Proc. of ACSAC'07*, 2007.

[8] T. R. Peltier, J. Peltier, and J. Blackley. *Information Security Fundamentals*. Auerbach Publications, Boston, MA, USA, 2003.

[9] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. L. Griffin, and L. v. Doorn. Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor. In *Proc. of ACSAC '05*, Washington, DC, USA, 2005.

[10] TCG. `https://www.trustedcomputinggroup.org`.