Chapter 21
Software Architectures for Self-Protection in IaaS Clouds

K. R. Jayaram, Aleksandar Milenkoski and Samuel Kounev

Abstract In this chapter, we focus on software architectures for self protection in IaaS clouds. IaaS clouds, especially hybrid clouds are becoming increasingly popular because of the need for developers and enterprises to dynamically increase/decrease their use of computing resources to adapt quickly to market forces and customer demands, reduce costs and increase fault tolerance. However, the adoption of public IaaS and hybrid clouds by enterprises is slower than expected because current hybrid cloud infrastructures do not provide scalable and efficient mechanisms to prevent software tampering, configuration errors and ensure the trustworthiness and integrity of the software stack executing a hybrid application workload; or to enforce governmental privacy and audit regulations by ensuring that remote data and computation do not cross specified geographic boundaries. We discuss recent research on integrating intrusion detection systems in IaaS infrastructures, as well as hardware-rooted integrity verification and geographic fencing to address the concerns outlined above.

21.1 Introduction

Enterprises have been adopting cloud computing in a variety of different abstraction levels, namely Software-, Platform- and Infrastructure-as-a-Service systems and in different deployment models (Private, Public, Hybrid, and Community). Security/protection issues/concerns associated with cloud computing differ according the the abstraction-level and deployment model but fall into two broad categories:

K. R. Jayaram
IBM T. J. Watson Research Center, e-mail: `jayaramkr@us.ibm.com`

Aleksandar Milenkoski
University of Wurzburg e-mail: `aleksandar.milenkoski@uni-wuerzburg.de`

Samuel Kounev
University of Wuerzburg e-mail: `samuel.kounev@uni-wuerzburg.de`

- security issues faced by cloud providers (organizations providing software-, platform-, or infrastructure-as-a-service via the cloud) and
- security issues faced by their customers (companies or organizations who host applications or store data on the cloud).

The responsibility for security and protection, thus goes both ways, however it is often the responsibility of the provider to:

- (as much as possible) ensure that their infrastructure is secure and that client data and applications are protected
- make it easy for the customer to take measures to fortify their application; and encourage the use of strong passwords and authentication methods as general policy in their cloud

Before discussing system and software architectures for self-protection in cloud computing, let us address the main security threats faced by enterprise applications deployed in the cloud. We focus on threats that arise *due to the cloud*, i.e., they add to threats already faced by the application when it is deployed in a private datacenter in a customer-facing scenario. It is generally recommended that information security controls be selected and implemented according and in proportion to the risks, typically by assessing the threats, vulnerabilities and impacts. While cloud security concerns can be grouped into any number of dimensions, some are outlined below:

- **Configuration and patching errors**. The extensive use of virtualization in cloud infrastructures brings unique security concerns for tenants of a public, private and hybrid cloud services. Virtualization alters the relationships between applications and the underlying hardware; especially given the fact that even networking functions are virtualized in modern datacenter. The virtualization layers themselves have to be properly configured, managed and secured. Specific concerns include compromises to the virtualization layer, especially the hypervisor. For example, a breach in the administrator workstation with the management software of the virtualization software can cause the whole datacenter to go down or be reconfigured to an attacker's liking. Typically the cloud provider asks the customer to "trust" that the provider has performed all the necessary configurations, patching, etc. accurately. However, instead of blindly "trusting" the cloud provider, a better approach is to follow the classic "trust, but verify" paradigm, where the cloud provider offers some visibility into the internal configuration and attests to its correctness and security compliance.
- **Software tampering**. When an organization elects to store data or host applications on the public cloud, it loses its ability to have physical access to the servers hosting its information. As a result, potentially business sensitive and confidential data is at risk from insider attacks. According to a recent Cloud Security Alliance report, insider attacks are the third biggest threat in cloud computing. Therefore, cloud service providers must ensure that thorough background checks are conducted for employees who have physical access to the servers in the data center. Additionally, data centers must be frequently monitored for suspicious activity. But, such manual methods may not always be ef-

fective. We need automated integrity verification of the software stack executing applications in the cloud. A key hurdle to increased adoption of cloud computing infrastructures is concern about the security of the software stack executing the workload. In a survey of large enterprises by Forrester, 46% of respondents indicated that "Ensuring that the security policies on applications, operating systems and workloads are the same in the public cloud as our data center" as a key hurdle for increased adoption of cloud computing infrastructures. There is broad agreement that verbal or written assurances of cloud providers in advertisements and service-level agreements regarding security and compliance are not sufficient. Encryption of all data and homomorphic computations can assure data confidentiality only if the code performing encryptions does not maliciously do something else with the data. Certificates of computation based on probabilistically checkable proofs (PCPs) and trusted computing platforms based on said proofs are still not practical, requiring customization for each application and incurring high overheads.

- **Isolation and Security Management**. In order to conserve resources, cut costs, and maintain efficiency, cloud service providers often store more than one customer's data on the same server. As a result, there is a chance that one user's private data can be viewed by other users (possibly even competitors). To handle such sensitive situations, cloud service providers should ensure proper data isolation and logical storage segregation. A cloud self-protection architecture is effective only if the correct defensive implementations are in place. An efficient self-protection architecture should recognize the issues that will arise with security management. The security management addresses these issues with security controls. These controls are put in place to safeguard any weaknesses in the system and reduce the effect of an attack.
- **Identity management** : Every enterprise will have its own identity management system to control access to information and computing resources. Cloud providers either integrate the customer's identity management system (e.g., Enterprise Kerberos) into their own infrastructure, using federation or SSO technology, or provide an identity management solution of their own (e.g., OpenStack KeyStone).
- **Physical security** : Cloud service providers physically secure the IT hardware (servers, routers, cables etc.) against unauthorized access, interference, theft, fires, floods etc. and ensure that essential supplies (such as electricity) are sufficiently robust to minimize the possibility of disruption. This is normally achieved by serving cloud applications from 'world-class' (i.e. professionally specified, designed, constructed, managed, monitored and maintained) data centers.
- **Application security** : Application security (short: AppSec) encompasses measures taken throughout the code's life-cycle to prevent gaps in the security policy of an application or the underlying system (vulnerabilities) through flaws in the design, development, deployment, upgrade, or maintenance of the application. This includes measures against software tampering, where an attacker modifies an existing application's runtime behavior to perform unauthorized ac-

tions; exploited via binary patching, code substitution, or code extension. This also includes preventing unauthorized access to administration interfaces; unauthorized access to configuration stores; retrieval of clear text configuration data; lack of individual accountability; over-privileged process and service accounts

While there are many types of defense mechanisms behind a cloud self-protection architecture, they are broadly classified into:

- *Deterrent mechanisms* : These techniques are intended to reduce attacks on a cloud system. Much like a warning sign on a fence or a property, deterrent controls typically reduce the threat level by informing potential attackers that there will be adverse consequences for them if they proceed. (Some consider them a subset of preventive controls.)
- *Prevention mechanisms* : Preventive controls strengthen the system against incidents, generally by reducing if not actually eliminating vulnerabilities. Strong authentication of cloud users, for instance, makes it less likely that unauthorized users can access cloud systems, and more likely that cloud users are positively identified.
- *Detection mechanisms* : Detection mechanisms are intended to detect and react appropriately to any incidents that occur. In the event of an attack, a detective control will signal the preventative or corrective controls to address the issue. System and network security monitoring, including intrusion detection and prevention arrangements, are typically employed to detect attacks on cloud systems and the supporting communications infrastructure.
- *Corrective mechanisms* : Corrective controls reduce the consequences of an incident, normally by limiting the damage. They come into effect during or after an incident. Restoring system backups in order to rebuild a compromised system is an example of a corrective control.

In this chapter, we focus on software architectures for self protection in IaaS clouds. IaaS clouds, especially hybrid clouds are becoming increasingly popular because of the need for developers and enterprises to dynamically increase/decrease their use of computing resources to adapt quickly to market forces and customer demands, reduce costs and increase fault tolerance. However, the adoption of hybrid clouds by enterprises is slower than expected because current hybrid cloud infrastructures do not provide scalable and efficient mechanisms to prevent software tampering, configuration errors and ensure the trustworthiness and integrity of the software stack executing a hybrid application workload; or to enforce governmental privacy and audit regulations by ensuring that remote data and computation do not cross specified geographic boundaries.

In traditional data centers, workloads and data were often static and had a hard binding to the physical systems on which they resided and executed. However, virtualization has made the migration of data and computing resources across physical hosts easy. Virtual machine (VM) migration is performed by datacenter and cloud management systems for a host of reasons including load balancing, consolidation and during maintenance. A private datacenter can employ sophisticated cluster management middleware and algorithms to reliably identify the location of physical servers on which the data and workloads reside during initial placement and migration, and can put security policies in place to ensure that the management middleware is not compromised. But, this is infeasible in current IaaS cloud implementations, because it would require the cloud provider to expose its systems to a customer/developer who may be malicious.

Organizations considering hybrid clouds however need to produce audit trails of data and application movement, as well as carry out effective forensics when the occasion demands it. In particular, the workload location identification and attestation capability needs to be verifiable and auditable by the customer and/or his designated third party; and preferably anchored in hardware. These capabilities enable workload and data boundary control in hybrid clouds, effectively conferring users control over where workloads and data are created, where they are run, and where they migrate to for performance, optimization, reliability, and high- availability purposes.

Given the enormous overheads involved in proof carrying code and in PCP-based verifiable computation, in real-world deployments, ensuring trust in the public IaaS cloud's software stack (hypervisor, VM, and libraries) often involves *software integrity verification* – checking whether specific "known bug-free" or formally verified/model-checked versions of hypervisors, OS and libraries are used by the IaaS provider, preventing injection of malicious code into the software stack and preventing low-level attacks on the hypervisor like those involving root kits. Hence a self-protection architecture should provide:

- Ability for developers to specify, at a high level, the integrity requirements of the software stack executing his applications/workloads
- Ensure compliance at runtime by certifying the integrity of the software stack through certificates verifiable by the developer or a third party trusted by him.
- Allow the specification of geographic locations and boundaries pertaining to computations in both IaaS and PaaS applications, and ensure their compliance during placement of virtual machines (at the start of execution) and during VM lifecycle events like re-sizing, migration and fault-recovery.
- Perform all the available tasks at the scale of massive data centers and IaaS clouds, which contain hundreds of thousands of virtual machine instances.

21.2 Intrusion Detection Systems

An intrusion detection system (IDS) is a device or software application that monitors network or system activities for malicious activities or policy violations and produces reports to a management station. IDS come in a variety of flavors and approach the goal of detecting suspicious traffic in different ways. There are network based (NIDS) and host based (HIDS) intrusion detection systems. NIDS is a network security system focusing on the attacks that come from the inside of the network (authorized users). When we classify the design of the NIDS according to the system interactivity property, there are two types: on-line and off-line NIDS.

On-line NIDS deals with the network in real time; it analyses TCP/IP and UDP packets and applies some rules to decide if there exists an attack or not. Off-line NIDS deals with stored data and runs some analytics on the data to decide if it an attack or not. Some systems may attempt to stop an intrusion attempt but this is neither required nor expected of a monitoring system. Intrusion detection and prevention systems (IDPS) are primarily focused on identifying possible incidents, logging information about them, and reporting attempts. In addition, organizations use IDPSes for other purposes, such as identifying problems with security policies, documenting existing threats and deterring individuals from violating security policies. IDPSes have become a necessary addition to the security infrastructure of nearly every organization using cloud computing systems.

IDPSes typically record information related to observed events, notify security administrators of important observed events and produce reports. Many IDPSes can also respond to a detected threat by attempting to prevent it from succeeding. They use several response techniques, which involve the IDPS stopping the attack itself, changing the security environment (e.g. reconfiguring a firewall) or changing the attack's content

21.2.1 NIDS

Network Intrusion Detection Systems (NIDS) are placed at a strategic point or points within the network to monitor traffic to and from all devices on the network. An NIDS performs an analysis of passing traffic on the entire subnet, and matches the traffic that is passed on the subnets to the library of known attacks. Once an attack is identified, or abnormal behavior is sensed, the alert can be sent to the administrator. An example of an NIDS would be installing it on the subnet where firewalls are located in order to see if someone is trying to break into the firewall. Ideally one would scan all inbound and outbound traffic; however doing so might create a bottleneck that would impair the overall speed of the network. OPNET and NetSim are commonly used tools for simulation network intrusion detection systems. NID Systems are also capable of comparing signatures for similar packets to link and drop harmful detected packets which have a signature matching the records in the NIDS. Providing NIDS systems is typically the responsibility of the IaaS cloud provider.

It is important to emphasize that the wide adoption of virtualization, a key enabling technology of cloud computing, has lead to the emergence of the practice of deploying conventional NIDS as virtualized network functions (VNFs); that is, a network-based IDS may be deployed in a designated VM and configured to tap into the physical network interface card used by all VMs (see Chapter 22). Thus, the IDS can monitor the network activities of all VMs at the same time while being isolated from, and transparent to, their users.

21.2.2 HIDS

Host Intrusion Detection Systems (HIDS) run on individual hosts or devices on the network. A HIDS monitors the inbound and outbound packets from the device only and will alert the user or administrator if suspicious activity is detected. It takes a snapshot of existing system files and matches it to the previous snapshot. If the critical system files were modified or deleted, an alert is sent to the administrator to investigate.

In the case of IaaS clouds, HIDS should be deployed at both the hypervisor-level and virtual-machine level. In contrast to HIDS deployed at virtual-machine level, providing HIDS at hypervisor-level is a responsibility of the IaaS cloud provider.

Typical IaaS cloud IDS architectures have to provide good visibility into the state of the monitored host, while still providing strong isolation for the IDS, thus lending significant resistance to both evasion and attack. Typical hypervisor-based approaches leverage virtual machine monitor (VMM) technology. This mechanism enables the IDS to be placed external to the host it is monitoring, into a completely different hardware protection domain, providing a high-confidence barrier between the IDS and an attacker's malicious code. The VMM also provides the ability to directly inspect the hardware state of the virtual machine that a monitored host is running on. The VMM provides the ability to interpose at the architecture interface of the monitored host, yielding even better visibility than normal OS-level mechanisms by enabling monitoring of both hardware and software level events. This ability to interpose at the hardware interface also allows the IDS to mediate interactions between the hardware and the host software, allowing it to perform both intrusion detection and hardware access control.

An IDS running outside of a virtual machine only has access to hardware-level state (e.g. physical memory pages and registers) and events (e.g. interrupts and memory accesses), generally not the level of abstraction where we want to reason about IDS policies. Typical hypervisor based IDSes leverage three properties of VMMs:

- Isolation : Software running in a virtual machine cannot access or modify the software running in the VMM or in a separate VM. Isolation ensures that even if an intruder has completely subverted the monitored host, he still cannot tamper with the IDS.
- Inspection : The VMM has access to all the state of a virtual machine: CPU state (e.g. registers), all memory, and all I/O device state such as the contents of storage devices and register state of I/O controllers. Being able to directly inspect the virtual machine makes it particularly difficult to evade a VMM IDS since there is no state in the monitored system that the IDS cannot see.
- Interposition : Fundamentally, VMMs need to interpose on certain virtual machine operations (e.g. executing privileged instructions). A VMM IDS can leverage this functionality for its own purposes. For exam- ple, with only minimal modification to the VMM, a VMM IDS can be notified if the code running in the VM attempts to modify a given register. VMMs offer other properties that

are quite useful in a VMM IDS. For example, VMMs completely encapsulate the state of a virtual machine in software. This allows it to easily take a checkpoint of the virtual machine. Using this capability we can compare the state of a VM under observation to a suspended VM in a known good state, easily perform analysis off-line, or capture the entire state of a compromised machine for forensic purposes.

21.3 Trustworthy Geo-Fenced IaaS and Hybrid Clouds

This section described recent research on trustworthy geographically fenced hybrid clouds (TGHC), a generic, scalable and extensible middleware system to automatically bridge the gap between applications with their integrity and geo-fencing policies, and raw hardware infrastructure. It describes TGHCs modularly, by (a) outlining the challenges in certifying the trustworthiness of cloud computing infrastructures and in geo-fencing computation, including scalability limitations of existing solutions, (b) presenting scalable mechanisms to transform bare metal servers into trusted IaaS computing pools through integrity measurement, management and monitoring that leverage open, off-the-shelf hardware technologies like Intel TPM, (c) introducing workload specification languages to specify integrity and geo-fencing policies on hybrid workloads, and (d) extending IaaS systems to ensure that workload bursting from private data centers to public clouds uses trusted computing pools and respects geographic boundaries during initial placement of virtual machines (VMs) and further migration. TGHCs are expected to be:

- *Generic*, targeted at a variety of distributed applications programmed in different languages and employing different software architectures e.g., event-based and loosely coupled, service oriented and tightly coupled, legacy software, etc.
- *Fine-grained and Flexible,* offering developers the flexibility to chose between two granularities in their specification of integrity and "trustworthiness" policies, i.e., either using (1) application level abstractions like classes, components, executables or (2) using IaaS abstractions like VMs, VM pools/patterns.
- *Scalable,* enabling it to be applied to modern distributed applications and deployed to real world clouds.
- *Easy to use,* requiring minimal effort from the developer, and requiring no more effort than the specification of integrity and geo-location policies in a high-level policy language/template and no more runtime configuration than state of the art orchestration systems like Puppet, Chef, etc.

21.3.1 A Modular Hybrid IaaS Cloud

Figure 21.1 illustrates the architecture of TGHCs:

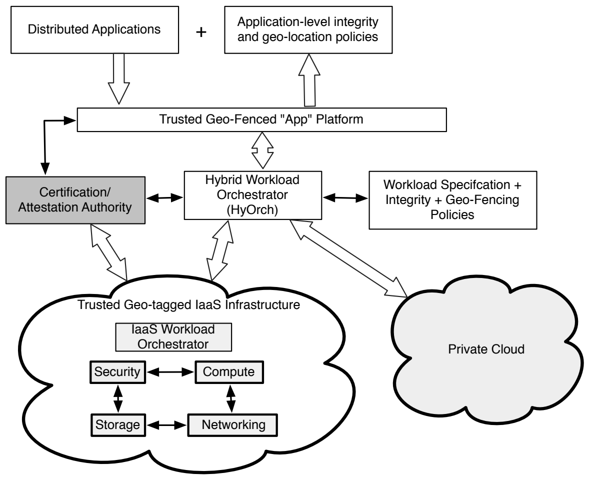
Distributed Applications
+
Application-level integrity and geo-location policies
Trusted Geo-Fenced "App" Platform
Certification/ Attestation Authority
Hybrid Workload Orchestrator (HyOrch)
Workload Specifcation + Integrity + Geo-Fencing Policies
Trusted Geo-tagged IaaS Infrastructure
IaaS Workload Orchestrator
Security
Compute
Storage
Networking
Private Cloud

Fig. 21.1: A high level overview of our Trusted Geo-Fenced Hybrid Clouds Vision

1. An IaaS infrastructure based on *open* standards, where the integrity and geographic location of each component – VM provisioning, migration, network provisioning, block and object storage, etc. – is attested by certification authorities and can be trusted by developers and enterprises (modulo their trust in certification authorities).
2. Hybrid IaaS orchestration middleware, which leverages the infrastructure above to (a) enable the specification of integrity and geo-location policies at the level of IaaS cloud abstractions, i.e., using VM instances, virtual LANs, monitoring components and patterns, and (b) ensure compliance with said policies at all stages during the lifetime of the pattern by certifying bare metal compute infrastructure and the entire software stack executing on them – hypervisors, libraries, guest OS in VM instances, etc., along with the application binaries.
3. Ensure that the design and implementation of the components above is extensible and modular, enabling future design of trustworthy hybrid application platforms, that use the trusted IaaS cloud. This paper, however, focusses on trusted hybrid IaaS, and describes the research challenges in engineering hybrid geo-fenced application platforms.

21.3.2 Trusted Geo-Tagged IaaS Infrastructure

As illustrated by Figure 21.1, a trusted geographically tagged IaaS infrastructure forms the basis of TGHCs. It consists of a trusted pool of servers running a trusted software stack, on which both customer workloads and management systems (i.e. those responsible for workload placement, migration, network provisioning, security and storage) are executed. Each server in this trusted computing pool (TCP) runs a trusted software stack, i.e., the infrastructure ensures that neither the hypervisor, operating system or any of the packages on it have been maliciously modified and are “known-good” versions, as specified by the developer. Furthermore, the infrastructure measures the integrity of all additional patches, application code and application middleware (e.g., web servers, database servers) that are loaded at run time and attests their integrity to an attestation service run by a trusted third party.

The key challenge is to perform all these functions automatically. We propose using hardware-based security techniques to ensure the integrity of the software stack on the trusted IaaS. Specifically, Section 21.3.4 describes how we leverage off-the-shelf hardware chips (Trusted Platform Modules or TPMs) to measure the integrity of hypervisors, operating systems and software packages and interact with the attestation service to get these measurements checked against those supplied by the manufacturer of the operating system or hypervisor. Section 21.3.7 describes how TPMs can be further leveraged to securely store and transmit geographic location tags to the attestation service. Scalability of both measurement and attestation is key, because a trusted computing pool can potentially contain thousands of servers. Hardware-rooted attestation helps reduce trust placed in cloud providers, because the customer can use the attestation service to independently verify the integrity and geo-location of the servers executing his workloads.

21.3.3 IaaS Orchestration Middleware

The next component, as illustrated by Figure 21.1 is the IaaS workload orchestrator, which interacts with the trusted geo-tagged compute pool, the third party attestation service and optionally, a workload specification from the developer. The IaaS cloud provider deploys the workload orchestrator on a trusted server, i.e., the cloud provider first uses the techniques mentioned above and detailed in Section 21.3.4 to boot a trusted operating system on a server and certify it to the attestation service. Then, it employs the techniques described in Section 21.3.4 to verify the integrity of the workload binares and deploys the workload on the trusted server connecting it to the attestation service. If the developer prefers to specify the public cloud portion of his hybrid cloud application as a pattern of virtual machines, e.g., using template languages like OpenStack HOT, the IaaS workload orchestrator parses this specification and interacts with the attestation service to determine the subset of nodes in the IaaS provider’s datacenter that are trusted in the geographic location specified by the developer. Then orchestrator then deploys the VMs on trusted nodes while respecting geographic boundaries, and instantiates the public cloud component of the application. Furthermore, all subsequent elastic scaling and VM migration only occurs on trusted servers and respects geographic boundaries (geo-fencing).

21.3.4 Converting Raw Servers to Trusted Servers

The first step in engineering TGHCs is to create a pool of trusted servers that can be managed by the IaaS management software, especially the IaaS workload orchestrator. Converting raw servers to trusted servers involves leveraging and extend the best ideas from all existing system integrity verification techniques. A modular approach can be employed to create a trusted server, by starting from a small piece of trusted code stored securely in a crypto processor to boot a trusted operating system and hypervisor; and then modularly use the trusted OS to validate all software packages on the server.

21.3.4.1 A Hardware Root of Trust

In current cloud computing systems, security (including malware prevention, detection, and remediation) is handled primarily by software – either at the operating system level or by the cloud management, authentication and authorization systems. This requires customers to place higher levels of trust in cloud providers. A better approach is the creation of a root of trust at foundational layer of the system – hardware. Like other proposals [13], we use the Trusted Platform Module (TPM) chip [11], a standard crypto-processor established by the Trusted Computing Group (TCG) [10] to instantiate a root of trust at every server. Then, that root of trust grows upward, into and through the operating system, applications, and services layers. TPM provides many security functions including special registers, called Platform Configuration Registers (PCRs) [11] which hold various measurements in a shielded location in a manner that prevents spoofing. Most laptops and desktops have been sold with a TPM module since 2006, and server motherboards are increasingly incorporating TPM chips.

21.3.4.2 OS + Hypervisor Boot Integrity

Most existing solutions for trustworthy clouds, including the EU TClouds [2] project the proposal by Intel [13] use trusted boot [9] based on trusted BIOS code – Intel Trusted Execution Technology (TXT) code [3, 13]. The TPM chip containing the TXT code is manually installed into servers in a data center. When OS booting starts, TXT measures (hashes) the initial boot code, and commits the measurement to the TPM chip, which stores the measurement in PCRs. TXT measures the next piece of code to be executed and extends a platform configuration register (PCR) in the TPM based on the measurement in the PCR before the control is transferred to the next program loaded during the boot process. If each new code module loaded during the booting process in turn measures the next one before transferring control, there is a chain of trust established. If this measurement chain continues through the entire boot sequence, the resultant PCR values will reflect the measurement of all files used. TXT and TPM can attest (prove) this measurement to a third party by signing the measurement with a private key known only inside the TPM. The corresponding public key can be used by the third party (attestation server) to verify the signature, and thus the measurement of the boot code. To protect against replay of measurements, the attestation protocol uses cryptographic nonces. The TPM is designed so that once a measurement is added to a PCR register, no operating system or user-level software can reset or remove the value; only a hardware reboot resets the chip, and thus restarts the booting process, re-instantiating the root of trust. This trusted This *measurement before execution* model therefore leads to a chain of trust that is observable by the attestation service.

In trusted boot, measurements can be of code, data structures, configuration, information, or anything that can be loaded into system memory. To further protect the integrity of the measurements, hash measurements are not written to PCRs, but rather a PCR is "extended" with a measurement. This means that the TPM takes the current value of the PCR and the measurement to be extended, hashes them together, and replaces the content of the PCR with that hash result. The effect is that the only way to arrive at a particular measurement in a PCR is to extend exactly the same measurements in exactly the same order. Therefore, if any module being measured has been modified, the resulting PCR measurement will be different and thus it is easy to detect if any code, configuration, data, etc. that has been measured had been altered or corrupted. However, this mechanism has scalability issues when applied to a large ensemble of machines in a data center, which we address in the following sections. We explain two alternatives, and then propose a mechanism for scalable verifiability.

21.3.4.3 Verifiability, but no Scalability with Native TXT

Trusted boot using Intel TXT and TPM, in their existing forms, can be used to prove the integrity of the operating system and hypervisor and all subsequent software packages that are installed on them to perform various functions (e.g, profiling, ssh, web server, etc.), as proposed by, e.g., [2] and [13]. If the chain of measurements during the boot process is kept by the attestation service in a software maintained list, it can be attested and verified, by the attestation service against the signed PCR value. If at any point, a piece of code is malicious, the model guarantees that the measurement of the malicious code is included in the list. Any attempt by malicious code to modify the measurement list would cause the validation to fail.

However, *this approach used by [13] and [2] simply does not scale*. The attestation service, which has to check the measurements should have a list of all possible "known-good" measurements for all versions of all operating systems, and all versions of each trusted software package, but also all combinations of operating systems and subsets of software packages. This is because of the way trusted boot extends integrity measurements in PCRs – assuming that there are K operating systems and each operating system has at most P compatible packages, each server may chose to install an operating system and any subset of the P packages. So, the attestation server has to store $O(2^P)$ hash measurements for each OS, and the check the measurement reported by the TPM chip against these. The size of the *whitelist* is consequently $O(K \times 2^P)$. The attestation server has to perform this checking for thousands of machines in a data center, and should accommodate multiple versions of packages to be of any real use.

To alleviate this, there have been proposals to modify TXT to record the sequence in which each code module is hashed and send the sequence to the attestation server. Then the attestation server can compute expected integrity measurement by hashing the $p < P$ stored hash measurements corresponding to the p packages actually installed by the server containing TPM and TXT. Although, this only requires the storage of $O(K \times P)$ hash measurements, it still has scalability limitations – we created a whitelist database for recent Fedora and RHEL distributions (Fedora 15 - 19, and RHEL 6). One such typical release has over 5000 available packages, with around 400,000 files. In addition, these packages are frequently updated within a release. The kernel packages are particularly large, each containing some 3,000 kernel modules, and each was updated on average 10 - 20 times. After a few months, the database grew to over 30GB, growing progressively slower, and exceeding the partition allocated for it.

21.3.4.4 Scalability, but no Verifiability with Linux IMA

A popular integrity model is secure boot [12], which stores one or more public keys in the TPM chip. The keys can only be updated with physical presence. Secure boot assumes that all code is signed (which is reasonable for most modern software systems), and the public keys stored in the TPM are used to appraise (verify) signatures on the boot code, and if the signatures are not correct, the boot is terminated. Secure boot can also root a chain of trust, in which each stage verifies the signature on the next stage. The Linux integrity subsystem also supports digital signature appraisal with the IMA-appraisal module [7], thus extending the secure boot chain of trust up into the Linux file systems. Digital signatures provide both integrity and authenticity, and they do so in a much more scalable way than simple measurements. However, this creates the problem of managing keys in the TPM module. Due to the fact that they can be updated only with physical presence, it is difficult to add or remove software vendors to a cloud application. One way to get around this is to sign all files with just the IaaS cloud provider's private key, but that defeats the whole purpose of reducing trust in the cloud provider. Also, secure boot model does not provide a hardware rooted attestation to a third party, so a centralized management system cannot tell if a system has been compromised.

21.3.4.5 Scalable Verifiability

A better approach is to use Intel's TXT technology to measure and attest the boot loader, kernel and hypervisor code, along with the public keys of all trusted software and data vendors. If all public keys are stored in one file, the attestation server should only store $O(K)$ measurements, assuming that there are K (bootloader+kernel+hypervisor) combinations. Further, integrity verification of the software packages can be accomplished by checking their digital signatures with the public keys of trusted software vendors through Linux's IMA appraisal module. This does not require developers to exclusively use software signed by the IaaS provider and also scales to a large number of software packages and versions by avoiding the storage of 2^P measurements. However, the TPM module is required in the physical compute host to certify the host's integrity to the attestation service, and verifying digital signatures through IMA does not accomplish this. Consequently, the IaaS provider and user should only trust hypervisors and operating systems with IMA code that verifies the digital signature of each executable before loading it during the boot process or before executing it in response to user needs. In other words, the way to ensure that the operating system/hypervisor on a server does not execute untrusted code is to (1) choose operating systems with kernels that check the signatures of all executables and do not have mechanisms to circumvent the digital signature verification process, and (2) verify that the server has booted one of the *said kernels* at runtime by measuring its integrity through TXT and attesting to the same through a trusted third party attestation service. Once this is done, the booted kernel ensures that all subsequent software is digitally signed obviating the need to individually attest the hash of each package. Hardware-rooted attestation helps reduce trust placed in cloud providers, because the customer can use the attestation service to independently verify the integrity and geo-location of the servers executing his workloads. Furthermore, such techniques protect against and detect any attack on persistent data (files), regardless of whether it comes through the main processor's network interface or the management modules interface. Even a remote root is unable to forge a valid signature. This is because of the trusted BIOS code (TXT) which hashes each component during the boot process to boot a secure kernel, which subsequently doesn't allow any unsigned program to execute. Hence, attacks are impossible unless the attacker has the ability to break strong cryptographic primitives. ‘

21.3.5 vTPMs and VM Launch Integrity

Through trusted hypervisor boot, we can ensure the integrity of pre-launch and launch components on a platform, from the BIOS to the operating system and hypervisor. However, no specific claims can be made about the virtual machines being launched, other than indicating that they are being launched on a measured and attested hypervisor platform. Although virtual machine monitors (VMM) or hypervisors are naturally good at isolating workloads from each other because they mediate all access to physical resources by virtual machines, they cannot by themselves attest and assert the state of the virtual machine that is launched. Each virtual machine launched on a virtual machine manager and hypervisor platform can benefit from a hardware root of trust by storing its launch measurements in the TPM. However, this requires virtualizing the TPM, with a virtual TPM (vTPM) for each of the virtual machines. Each of these virtual TPM (vTPM) instances then emulates the functions of a hardware TPM.

One approach is to use the open-source IBM vTPM [1] system for TGHCs. It's is to provide a TPM functionality to a virtual guest operating system. This allows programs to interact with a TPM in a virtual system the same way they interact with a TPM on the physical system. Each guest gets its own unique, emulated, software TPM. However, each of the vTPM's secrets (Keys, NVRAM, etc) are managed by a vTPM Manager domain, which seals the secrets to the Physical TPM. If the process of creating each of these domains (manager, vTPM, and guest) is trusted, the vTPM subsystem extends the chain of trust rooted in the hardware TPM to virtual machines in the hypervisor. Each major component of vTPM is implemented as a separate domain, providing secure separation guaranteed by the hypervisor.

Once vTPMs are instantiated, the launch of guest operating systems can be trusted and so can the packages installed on the guest operating systems. This is done using the scalable verifiability technique presented in Section 21.3.4.5.

21.3.6 Scalable Third Party Attestation

The attestation server is typically managed by a trusted third party (TTP). Each server in a trusted infrastructure pool connects to the attestation server to provide TXT integrity measurement hash values, which are then checked by the attestation server against its database of integrity measurements. The most common attestation server is Intel's OpenAttestation server (OAT) which uses a `mysql` database to store hash values. The attestation server can be run either by a TTP or by the developer himself, if the IaaS cloud provider is willing to trust the developer with cryptographically secure hashes of its operating systems, software and hypervisors. This is not uncommon, especially in aerospace, defense and other sensitive workloads from governmental agencies. The main drawback of the existing OAT server and its implementation is that it still requires storage of hashes of all combinations of operating systems and software packages (2^P) as discussed above, and that it is a standalone server which is neither highly available nor elastic. To scale to trusted IaaS pools, these mechanisms for scalable verifiability have been incorporated into Intel's OAT implementation, while additionally replacing the storage layer with a distributed three-way replicated key-value store (HyperDex), running on servers that have been manually verified to be trustworthy.

21.3.7 Trusted Geo-Location

The next step in engineering TGHCs is to identify the geographic location of each trusted server and to be able to certify its location to a trusted third party that also attests to the integrity of the software stack. In turn, cloud management software uses the trusted server location information to enforce constraints of sensitive applications with geo-fencing requirements. Geo-fencing ensures that the requirements on geographic boundaries are not violated and, furthermore, if violations are detected, that appropriate remedial actions are taken. In this section, we first outline an existing manual location provisioning technique, and propose an RFID-based geo-location technique to reduce trust placed in a cloud provider. Manual location provisioning and the use of RFID tags described below are the only two solutions currently feasible using *off-the-shelf* hardware.

21.3.7.1 Manual Location Provisioning using the Trusted Platform Module (TPM)

The IaaS cloud provider administrator or a trusted third party, can securely provision the location (geo-tag) of the server motherboard (postal code, GPS coordinates, city/state/country) to the TPM chip on the motherboard when the server is installed in the cloud data center. There are no hardware changes required in this approach; the existing TPM takes care of secure storage for the geo-tags. [13] provides an overview of this proposal – by reusing the PCR22 of the TPM. PCRs are *non-volatile* RAM, and consequently the location stored survives re-boots unless PCR22 is manually erased and the TPM re-provisioned. The TCG specifications [9] allocate PCR22 in the TPM for OS/VMM use, and popular hypervisors like VMWare ESX, Xen and KVM do not use it for other purposes. The only drawback with manual provisioning is that it requires the developer to place some trust in the IaaS provider and its employees – to not physically remove the motherboard and ship it to another data center.

21.3.7.2 RFID Tags

A solution to slightly reduce the trust placed in an IaaS provider is to use Radio-frequency identification (RFID) asset tags in servers in combination with a a specialized hardware tamperproof asset tracker. The asset tracker is installed by a trusted third party (TTP) in a datacenter and connects to the TTP's attestation service through a network connection separate from the IaaS provider's network, e.g., using a different wired ISP other than the IaaS provider or through high speed wireless network (4G LTE) SIM cards or through satellite internet. The tamperproof asset tracker consists of an RFID tracker, a GPS locator chip and a trusted server that can read the GPS chip and RFID tracker to communicate with the TTP's attestation service. Each server in the data center is provisioned with an active secure RFID asset tag (costs about $30 per tag) which periodically transmits a tupple, consisting of the RFID tag's identifier and the server motherboard's universally unique identifier (UUID), encrypted by a shared key (shared between the active RFID tag and the tamperproof asset tracker) or through the asset tracker's public key. The asset tracker, in turn hashes each of the elements of this tuple and sends them to the attestation service. TXT is also extended to hash the UUID of the motherboard which is securely stored in the TPM module and send this to the attestation service, which can then attest to the location of the server motherboard because it knows the location of the asset tracker.

21.3.8 Trusted Hybrid IaaS

The next step in engineering TGHCs is to transform a group of trusted servers to a trusted IaaS compute pool, which can be used by a hybrid cloud application. Engineering a trusted IaaS is challenging because of the complexity of the components of a modern IaaS cloud manager (e.g., OpenStack). We rely on our modular design approach to realize trustworthy hybrid IaaS clouds as shown in figure 21.2.

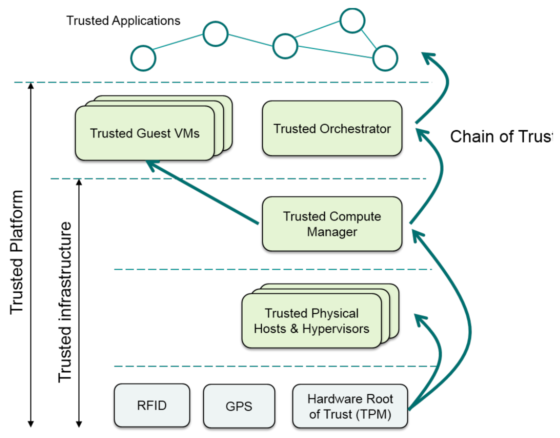

Fig. 21.2: A modular approach to create a trusted cloud using other trusted components

IaaS clouds consist of VMs, VM instances and the IaaS management system. The main function of an IaaS cloud is to instantiate a virtual machine (VM) image or a pattern of VM images in response to authenticated user requests, and network the instantiated VMs according to the user's specifications. Hence, the main components of an *IaaS management system*, with examples from the open-source IaaS platform OpenStack [6] are (1) authentication and identity management (Keystone) (2) compute provisioning, migration and resource management (Nova) (3) network provisioning (Neutron), (4) VM image storage and management (Glance), (5) block and object storage and (6) cloud orchestrator (Heat) and dashboard (Horizon). To realize a trusted IaaS cloud, each of these components must in turn be made trustworthy. In this section, we describe how to create a trusted IaaS using OpenStack as an example, and then detail how a trusted IaaS can be used by a hybrid cloud application.

21.3.8.1 IaaS Management System

For an IaaS cloud to be trusted, its management system should also be trusted. When the IaaS cloud is set up, trusted servers are set up using the techniques in Section 21.3.4. Then, various IaaS management packages are installed on some of these trusted servers and validated using the scalable attestation techniques described in Section 21.3.4. The customer, in turn, can verify from the attestation service that the management components of the IaaS cloud are running on trusted servers. In the case of OpenStack, Nova, Glance, Swift, Neutron, KeyStone and Horizon are installed on trusted servers. The open-source nature of OpenStack, whose architecture and implementation have been vetted by an active community of developers, along with attestation of its installation through the third party attestation service ensures that the management layer performs as expected. Thus, the root of trust now extends to the IaaS management system, thus ensuring the integrity of the IaaS cloud software stack. A trusted IaaS cloud may contain both trusted and untrusted servers – the only requirements are that it contain a non-zero, non-trivial number of trusted servers and that it execute all its management functions on trusted servers. For this, the basic capability needed is resource management.

21.3.8.2 Resource Management

A trusted compute pool in an IaaS cloud is simply a collection of trusted servers obtained from the process outlined in Section 21.3.4. An IaaS cloud typically has a resource management system, e.g., Nova in the case of OpenStack [5]. These resource managers have different architectures (centralized vs decentralized) and matching algorithms to match available resources to requests and reserve resources for user requests. In this paper, we describe how to implement a trusted computing pool in a centralized/replicated resource manager like Nova. Nova is centralized because it uses a single database (mySQL) to store information about all servers and other components in a data center. When a request (r) arrives at Nova for servers to schedule VM instances, Nova queries the database for available server capacity, retrieves matching server resources to allocate to the VM (e.g., 2 cores, 4GB RAM on server s), reserves these resources for request r, and instantiates the requested VM on s. Nova can be replicated, i.e., using replicated mySQL, both for fault tolerance/high availability as well as to scale to a large number of servers and VM instantiation/migration requests. Resource allocation then becomes a distributed transaction handled by the replicated mySQL implementation.

To implement a trusted computing pool, the key idea is to connect the resource manager to the attestation service to check the veracity of a server's claims, and then use trustworthiness and geographic location as additional filters during placement and migration. We have extended Nova to communicate with the attestation service to verify whether a server that claims to be running a trusted hypervisor and our scalable attestation software (described in Section 21.3.4) is actually doing so. Our Nova implementation then stores information in its database about trusted servers in the computing pool that it manages. Additionally, our Nova also obtains the geographic location of the server and verifies it with the location reported by the attestation service. When a VM instantiation request requires a trusted server in a specific location, our extended Nova modifies its query accordingly to chose a server that has been attested and filters out untrusted servers. Similarly, during the migration of a VM, the new physical server should be attested as a trusted server, before Nova can move a VM to it. Similar techniques can also be used in other cloud resource managers, like Omega [8], Mesos [4], etc., by connecting the resource manager to the attestation service, modifying its matching algorithm to distinguish between trusted and untrusted servers and adding geo-location as a data attribute to each trusted server.

21.3.8.3 Authentication and Storage Management

The user authentication and identity management components do not need any changes to work with trusted hybrid IaaS, i.e, KeyStone in the case of OpenStack does not have to be changed. The image management system (Glance) requires minimal changes to ensure that images are stored on trusted geo-located servers, i.e, its API and functionality are extended to specify the location where an image has to be stored. The block store and object store should be extended to ensure that (1) storage requests from a trusted VM instance are handled by and the data stored on a trusted server, (2) the geo-fencing constraints on storage requests from a trusted VM remain the same as the constraints on the VM instance, i.e., if a VM is restricted to execute in the USA, then all its object and block storage resides in the USA.

21.3.8.4 A Trusted IaaS

Though resource management and server management are key components of every IaaS cloud, modern IaaS implementations provide several other functionalities, including but not limited to automatic elastic scaling at the level of virtual machine instances, floating IPs, security groups, virtual LANs, volume storage, object storage, etc. All such components of the IaaS implementation should be instantiated on trusted servers. Otherwise, malicious code in the other components of the IaaS system can circumvent the protections offered by the trusted compute pool.

In the case of the OpenStack IaaS platform, this includes all its management components, including the authentication and security subsystem (KeyStone), image management system (Glance), network management (Neutron), etc. In summary, we propose to modularly use TPM and our scalable attestation to convert raw servers into trusted servers attested by a trusted third party. Then, we instantiate OpenStack Nova on one or more trusted servers for resource management, which communicates with each of the servers and with the attestation service to determine whether they are a trusted software stack and their geo-location. We then use trusted Nova to instantiate the other components on trusted servers or trusted Linux VMs.

21.4 Conclusions and Future Research Directions

In this chapter, we have presented two key technologies for self-protection in IaaS clouds, and describe their software architecture. This includes Intrusion Detection Systems and hardware-rooted integrity verification of software stack executing applications in the cloud. We have illustrated how these technologies serve as preventive, detective and deterrent controls for the various threats faced by applications executing in the cloud as outlined in Section 1.

References

1. Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vtpm: Virtualizing the trusted platform module. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.
2. EU Framework 7 – TClouds Project. *Trustworthy Clouds Privacy and Resilience for Internet-scale Critical Infrastructure*, 2013. `http://www.tclouds-project.eu/index.php/published-results/public-deliverables`.
3. W. Futral and J. Greene. *Intel Trusted Execution Technology for Server Platforms*, 2014. `http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/trusted-execution-technology-security-paper.pdf`.
4. Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308, Berkeley, CA, USA, 2011. USENIX Association.
5. OpenStack. *Nova Developer Documentation*, 2014. `http://docs.openstack.org/developer/nova/`.
6. OpenStack. *OpenStack Architecture*, 2014. `http://docs.openstack.org/training-guides/content/module001-ch004-openstack-architecture.html`.
7. Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. 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, 2004. USENIX Association.
8. Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 351–364, New York, NY, USA, 2013. ACM.
9. Trusted Computing Group. *Trusted Boot*, 2014. `http://www.trustedcomputinggroup.org/resources/trusted_boot`.

10. Trusted Computing Group. *Trusted Computing Group Web Portal*, 2014. `http://www.trustedcomputinggroup.org`.
11. Trusted Computing Group". *Trusted Platform Module Specification*, 2014. `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`.
12. R. Wilkins and B. Richardson. *UEFI Secure Boot in Modern Computer Security Solutions*, 2013. `http://www.uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf`.
13. R. Yeluri and E. Castro-Leon. *Building the Infrastructure for Cloud Security A Solutions View*. Apress Inc., 2014.