Enabling Cloud Customers to Trust the Cloud

Ashwin Shashidharan and Jitesh Shah
{ashashi3, jhshah}@ncsu.edu

December 2, 2011

Abstract

In recent times, cloud computing has emerged into a very popular [1] means of scaling up quickly without the associated infrastructure costs. The promise is complete freedom from maintaining own server infrastructures and moving to third-party infrastructure providers. Despite it's increasing popularity, a significant chunk of applications haven't yet found a place in the cloud. The reason is the lack of guarantees from the cloud provider, about the security of customer data stored on the cloud. Even in the presence of guarantees, the cloud customer would want to ensure that a very small chunk of *auditable* infrastructure on the cloud side is trustworthy. Using that as a Trusted Computing Base (TCB), the customer would build his own security to protect his data, thus, having the guarantees. Note that the threat to the customer data, in our model, is assumed *not* to be from the cloud provider, but, attacks on the cloud provider by competitors using the same cloud service or other external attackers.

This paper proposes a way to enable the customer to protect his data in the face of a cloud compromise. Two key technologies are used to build the security architecture: SELinux (Security Enhanced Linux) [21] and TPM (Trusted Platform Modules) [6]. Both of these technologies allow confining the root user. We also describe a range of feasible attacks and how customer data can be protected in the face on these attacks using the proposed architecture.

Keywords: Trusting the cloud, SELinux, TPM, SSL/TLS, symmetric encryption/integrity verification.

1 Introduction

According to a Berkeley publication [12], all the top five software companies by sales revenue have their own cloud offerings. Popular sites such as reddit.com and quora.com run completely off the cloud. Merrill Lynch estimates the market capitalization of the ”Cloud Computing” sector to be as large as 16B in 2011. Inspite of such impressive growth, a Morgan-Stanley survey [8] says that only 28% respondents of the survey use public cloud of some form in their daily operations whereas only 14% use IaaS services of any kind. Various publications [12, 11] clearly identify the risks associated with security of the storing data in the cloud. Fear of leakage of sensitive data from the cloud is a common thread in these publications. These risks induce a ”Fear of the Cloud”, thus, slowing down adoption. Vulnerabilities discovered in popular cloud services like Amazon EC2 [24] and Dropbox [23] exacerbates the fear.

For corporations to offload their secure data to the cloud, they need to have guarantees against leakage of their data. Such guarantees can come only from a huge audit of the cloud provider, which might be unfeasible and time consuming considering the size of the cloud providers, or getting themselves involved in designing of the security on the cloud provider side. The cloud can thus be viewed, from the perspective of our project, as a provider of infrastructure and a Trusted Computing Base (TCB) which can be utilized by the cloud customers to enforce a security policy.

Various methods have been proposed in the past to avoid data leakage from the cloud. Simply encrypting data in the cloud (without any trusted software) has been discussed in literature sparingly [19, 32]. The problem with just encrypting the data, without attestation of the underlying software, is that the underlying software could have been maliciously modified by an attacker who has broken into the cloud provider. A modified guest kernel would definitely be able to steal all the encryption keys. So, even though encryption is an essential component of securing the data, a trusted computing base is essential too.

Setting up a Trusted Computing Base (TCB) for the cloud has been more widely discussed [28, 30, 16, 18]. A TCB is setup with an established root of trust. The root of trust is usually a TPM (Trusted Platform Module). Beginning with the TPM, the BIOS, bootloader and the Operating System are verified before passing control to them. Any modification to any of the above components would lead to failure of Remote Attestation. The problem with such an approach is that, it provides security only in static aspects. Any attack on the runtime system (process, memory, etc) would go undetected.

Literature also talks about runtime integrity verification [25, 31, 20, 29]. The problem with runtime integrity verification techniques is that they are very difficult to get right in practice. No cloud in the current market offers runtime integrity verification [30].

Without runtime integrity protection, the data on the cloud will still be vulnerable to attacks of the dynamic nature. A practical alternative to runtime integrity verification is to sandbox the trusted VM launcher process, minimize its interaction with the outside world and audit the small external facing code thoroughly for security vulnerabilities. Thus, in addition to encryption and trusted computing, we propose to use strong Mandatory Access Control (MAC) based confinement techniques to sandbox the trusted VM launcher and the kernel images. This can be viewed as a light-weight and practical alternative to the complex runtime integrity verification techniques. SELinux [21] is an implementation of MAC on Linux-based systems. We use SELinux to sandbox the launcher and thus, protect it from dynamic attacks. Hardening the core virtual infrastructure of a cloud has been discussed in atleast one paper before [33].

We propose that the cloud customer encrypt and integrity protect the kernel used to boot his VM image. Since the threat model is attack on the cloud, we propose that the keys for encryption/decryption of kernel and the data (two separate keys) be stored on the customer side. The keys are revealed only to an attested and authenticated (via SSL [35]) cloud process. The verified cloud process then goes on to provision the guest image on one of its nodes.

Note that the core of the idea is protection of business critical data *in the face of* attacks on the cloud infrastructure.

Section 2 explains some terminology used widely in the paper. Section 3 further explains the problem statement and the threat model that we protect against. Section 4 explains and justifies the solution in detail. Section 5 lists the types of possible attacks that we envision and defences in place against them. Section 6 lists some related work we are aware of and lastly section 7 concludes the paper.

2 Background

In this section we will give a background of some of the technologies employed in the project.

2.1 Trusted Platform Module

The Trusted Computing Group (TCG) [6] was formed with a goal of creating a Trusted Platform Module (TPM). A TPM is a secure cryptoprocessor which is activated from very early in the boot process. The property of TPM which makes it useful is that it can seal data inside it and reveal it only to a verified authority. Since, the data inside the TPM is unforgeable, TPM can be used as a root of trust. The idea is to start with this root and expand the perimeters of trust one-by-one from BIOS to the OS to the applications.

A TPM usually has a small processor which can do symmetric as well as asymmetric cryptography. An asymmetric key called the SRK (Storage Root Key) is burnt into the TPM, unreadable by the external world. The SRK is used to create a tree of keys used to perform encryption/decryption. This makes sure that any encrypt/decrypt operation needs SRK and hence, an involvement of the TPM.

There is also some permanent storage available in the form of Platform Configuration Registers (PCRs). There can be upto 32 such PCRs according to the TCG 1.2 specification. It is not possible to directly write to a PCR. A TPM supports only two operations to a PCR: Read and Extend. The following paragraph describes how to establish trust using a TPM.

When the machine is reset, TPM is booted with all PCRs reset. The hardware then calculates the hash of the BIOS code and extends it to a PCR in the TPM, say, PCR#16 and the control is passed to the BIOS. The BIOS calculates hash of the boot loader and extends it to PCR#16 again and passes control to the bootloader. The bootloader does the same for the kernel. Thus, after the kernel is booted, the value in PCR#16 is a unique summary of the {BIOS, bootloader, kernel} tuple. Any modification in either of these will change the value in PCR#16. Now keys can be created and data encrypted against PCR#16. Only an unmodified kernel can then unseal the secret values. Thus, a TPM can be used to reveal secrets only to trusted parties.

Note that to perform these operations on a TPM, AuthData is required. AuthData can be thought of as the key to unlock the operations of the TPM. It could be as simple as a password or a complex derivation (salted hash, etc) of a password.

2.2 SELinux

Security Enhanced Linux (SELinux) [21] is an implementation of Mandatory Access Control (MAC) over a traditionally DAC (Discretionary Access Control) implementation on Linux Systems. The advantage of MAC over DAC is that the security policy is centrally administered from a policy database and cannot be modified by the users. MAC also offers a more fine-grained control on the resources that can be accessed by a subject. For example, it is possible to disallow Apache httpd to send emails, but allow to write files in a certain directory.

SELinux bases it decisions on the tuple {user, role, type}. The *user* is the SELinux user and *not* the Linux user. Roles can be used to implement RBAC (Role Based Access Control). *type* is the field on which access control decisions are based. This 3-tuple is known as the *context*. Each subject and object has a context. To execute a file from a different context, a transition rule must explicitly enable the process to transition from one context to the other. Execution then continues on the new context. Read and write accesses can be granted to other types. For example,

```
# allow user_t other_t:file create
{read}
```

The rule states that *user_t* type is allowed to only read the files on the type *other_t*.

We use SELinux to sandbox the launcher process and to confine the root user.

2.3 SSL/TLS

Transport Layer Security or its predecessor Secure Socket Layer is a cryptographic suite used to provide encrypted and authenticated communication between two parties who trust a third-party called the Certificate Authority (CA). SSL/TLS is currently deployed on all major websites including Google, Facebook, Twitter, e.t.c. to protect users from man-in-the-middle and DNS attacks.

We use SSL/TLS to mutually authenticate the cloud provider and the customer site. We also use encrypted communication to transfer sensitive data from the customer site to the cloud.

2.4 Miscellaneous

This subsection will explain terminology used throughout the paper. We use the term *Management node* to refer to a node in the cloud which is exposed to the outside world as well as the VM image store, which makes it vulnerable to attacks. The term *Launcher* is used to denote the process which launches and provisions the guest image upon a trigger from another process (like the webserver. The webserver accepts launch requests from the customer and triggers the *launcher*).

3 Problem Statement

Our work deals with securing the process of image loading done by the Management node and protecting against tamper of images in the image library. Since the architectures of popular IaaS providers like Amazon are not in the public domain, all assumptions about the cloud architecture are based on our study of VCL. We begin by briefly explaining the VCL architecture and following it up with threats and risks in this setup.

3.1 VCL Architecture

The VCL cloud architecture [27] associates resources in the cloud with different levels of abstraction. Applications are associated with the use of custom images and OSes that run on hypervisors or bare metal nodes. While computing resources and hardware resources are handled by the VCL Manager Software, network resources are managed by the virtual networks, VLANs and VPNs.

The VCL Manager Software runs on a Management node that is responsible for hardware resource allocation, provisioning, optimal scheduling, launching and scheduling image instances. User requests are mapped by the VCL Manager software onto available software application images and available hardware resources. It is then scheduled for use by the customer. The Management node needs access to the image library for these tasks.

The VCL node manager handles both bare-metal images - that run directly over a machine - and virtual images that run on a hypervisor. It is imperative to know that, the Management node has three interfaces: load images, connect to the management module on a chassis, and for incoming connections from the public network. The VCL architecture has been illustrated in Figure **??**.

The Management node is vulnerable to exploits in its OS, middleware and other software components. The risk is higher owing to the fact that it accepts connection over an external network. Misconfigurations and vulnerabilities in the httpd and ssh daemon or other system software may be exploited to take control of the node.

3.2 Possible Attacks

We now describe a few attack scenarios and try to assess the magnitude of the loss.

As mentioned earlier, the Management node has a processing engine to process reservations obtained from a web portal. It also has access to the image library. An adversary who has attained sufficient privileges on the Management node can thus create havoc by loading images that were never requested. Access to the image store is more lethal if, the adversary has also gained modification rights for storage. If there is no mechanism in place to verify the integrity of images before a launch, the attacker may launch malicious image instances. These instances may then be used to capture user sensitive information transferred usually during a connection request. In another case, an attacker can make the image unavailable

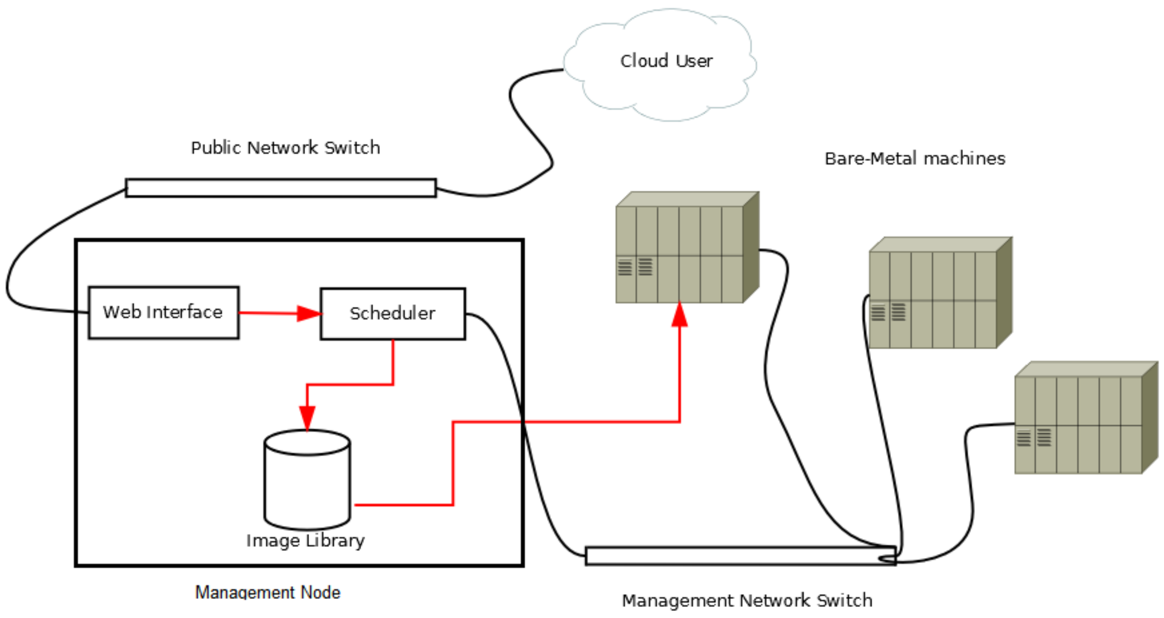

Figure 1: *NCSU VCL cloud architecture. The red lines denote the control flow in the cloud. The Management node is shown connected to a public network switch for incoming public connections and a management network switch to perform administrative actions and management.*

thereby launching a DoS attack. The use of a malicious modified image by the attacker can also be used to capture/manipulate data generated by those instances.

An adversary who has gained access of the Management node may also tamper with out-of-band management features, central to proper functioning of the infrastructure itself. With control features enabled for bare-metal machines and images, from the Management node, an adversary gains access to all these resources in the event of a compromise.

The Management node also has remote shell access over the management network to the bare-metal machines and virtual machines that it has launched. Root access to the attacker on the Management node further can be used to take control of these instances via remote shell.

The Management node is responsible for provisioning and scheduling. Thus, on a compromise, the attacker can use these capabilities to gain control over the computing power available on the cloud to launch attacks on other instances running in the cloud.

Access to the data store is another concern. In the event of a storage compromise, without encryption to protect the confidentiality of user data, user's data on the cloud becomes accessible to the attacker.

Thus, we conclude that the Management node is central in a cloud setup to manage user requests, schedule resources, security and for multi-site coordination.

We expect most of the cloud environments to have such a node. Through our design we secure the cloud's Management node from attacks that try to launch maliciously modified components with the intent of stealing sensitive data. We try to address the problem of being able to use the cloud with an assurance of security and devise a solution whereby, the customer is also made part of this process.

In the next section we describe how we take on the above challenges.

4 Approach

As we saw in the last section, obtaining root access on the Management node can be catastrophic. It is never possible to claim with a 100% guarantee that a particular software is free of bugs. Thus, putting in measures to check the damage in case of a compromise is as necessary as following secure coding practices. The next subsection attempts to summarize the salient points of the design.

4.1 Design Choices

4.1.1 Store Keys on the Customer Site

The fundamental premise of the project is to enable the customer to trust the cloud. Thus, we choose to store the data and kernel encryption/decryption keys on the customer site. The customer site reveals the keys only to a *secure launcher daemon* on the cloud side, who can attest its state and authenticate via SSL. (Note: Workings of the *secure launcher daemon* is discussed in the later section)

4.1.2 Integrity Protect the Kernel

The kernel which the guest VM uses to boot should run unmodified. Any malicious alteration in the kernel can lead to leakage of data. Thus, the customer site stores an HMAC of the kernel alongwith the kernel decryption key. This, HMAC is passed along to the *secure launcher daemon*. Only if this HMAC matches the one obtained from the kernel, is the kernel booted. Note that we also store the kernel encrypted on the cloud side (although one might argue that it isn't really necessary).

4.1.3 SSL Private Key Sealed by the TPM

To enable mutual authentication between the cloud provider and the customer site, they communicate via SSL. If an attacker has to pose as the cloud provider, he will need the SSL private key of the cloud provider. To avoid imposters, we seal the SSL private key in the TPM. This private key can now only be accessed if the TCB is established and the *secure launcher daemon* is unmodified.

4.1.4 TPM Authdata Management

To perform a TPM seal/unseal operation, it is necessary for the requesting entity to present Authdata: an entry ticket into the TPM. Authdata can be as simple as passwords or a complex derivation of passwords. We let the cloud customer choose the Authdata type. The Authdata can either be entered manually on a terminal when launching a guest VM or can be passed on from the customer site with the launch request or stored in a file with very strict SELinux confinements. The most preferable and secure way is to enter it manually on the terminal (might not be the most practical).

Next we provide an overview of how the solution works.

4.2 Initial Setup

For the success of this scheme, the customer's participation is required in setting up the initial security of the system. In this section, we cover how the initial setup is done.

The customer first sets up his separate keys for encryption/decryption of the kernel and the data. The cloud provider either sends a copy of the kernel to be authenticated by the customer or the customer compiles his own copy. In the former case, the unencrypted kernel is sent via snail-mail on a harddrive using a proper digital signature. The customer verifies the digital signature, creates an HMAC with his kernel signing key and encrypts the kernel. The encrypted kernel is then returned via snail-mail, again, with proper digital signature. The customer keeps the HMAC. The cloud provider installs this encrypted kernel on his side. Note that the snail-mail is only to provide a secure out-of-band communication mechanism.

The customer then sets up an SSL-enabled server to serve the keys and HMAC to a properly authenticated cloud entity.

On the cloud side, the customer takes part in setting up the TCB. The customer can, optionally, provide a TPM. The TPM setup procedure is fully audited by the customer to ensure security (core assumption is that the cloud provider is trusted and the setup procedure won't be botched. The audit is to make sure that the proper security policies are followed). The setup of the management node occurs behind closed doors with no access to the external world, so it is assumed that it cannot be compromised by an external entity.

A proper hash of BIOS, bootloader, kernel, secure launcher daemon is extended in on the PCRs of the TPM. The SSL private key of the cloud used for authentication with the customer is then sealed off againt the said PCR. Proper SELinux policies are then put in place to sandbox the secure launcher daemon.

The next section describes the operation of the setup.

4.3 Solution

The very first step to secure data in the cloud is to encrypt it. The cloud provider can provide facilities that allow encryption just before storing to the persistent storage at the cloud site. However, based on the end-to-end security argument [26], if the cloud customer needs guarantees, the encryption has to be done at the application layer by the customer software. The customer may choose to have a full-disk encryption (encrypts the whole filesystem) [15, 14] or a selective file-encryption [4]. In either method, for efficiency reasons, the encryption technique chosen would probably be some kind of symmetric encryption (like AES) and will need a key.

This key cannot be stored on the guest image in plaintext as it will be visible to everyone. In case of a security breach, as the attacker will have access to the guest image from the Management node this key becomes accessible

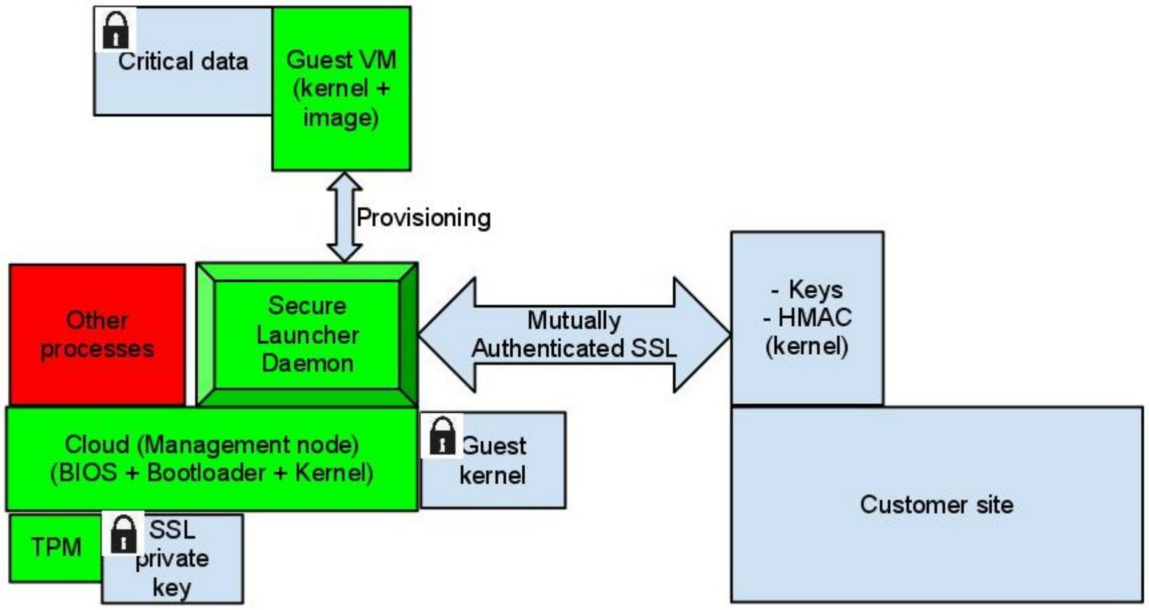

Figure 2: *The architecture of the solution. The green parts denote trusted entities. Red denotes entities not in the TCB. The SSL private key is shown near the TPM block to signify that it is sealed in the TPM. The locked entities can only be unlocked by the trusted Secure Launcher Daemon. The guest VM resides on a different node.*

at no effort. As described above, keys will be revealed only to a cloud process who can provide attestation (using TCB) and authenticate itself using SSL. Next we describe ways in which it will be impossible (or atleast hard) for an attacker to falsely pose to the customer as an attested cloud process.

Figure 2 gives an overview of the architecture. A TPM is used to establish a TCB, as mentioned in the ”Background” section. The TPM verifies the BIOS, bootloader and the kernel. Eventually, the kernel extends the hash of a *secure launcher daemon* in the TPM and starts the daemon. The function of this daemon is to listen for requests to launch a VM. When a request is received, it unseals the SSL private key from the TPM. This unseal operation only works if neither of the BIOS, bootloader, kernel or secure launch daemon have been modified.

After the SSL private key is unsealed, the daemon can authenticate to the customer site. Once the customer site authenticates the cloud provider, it releases the following objects: kernel decryption key, data decryption key and an HMAC of the kernel image.

The *secure launcher daemon* decrypts and verifies the integrity of the kernel. It then reads the actual launcher binary (usually a provisioning engine like xCAT [2]) into memory, calculates its hash and compares against the expected hash. If both the kernel and the launcher binary are verified correctly, then the daemon forks off the launcher process which does the appropriate provisioning.

The data decryption keys are passed on to the guest kernel via a shared page in memory or as a provisioning parameter.

Note that the *secure launcher daemon* is confined using SELinux. The SELinux policy recommendations and how to sandbox the daemon is described in detail in the next section.

4.4 SELinux Policy

An introduction to SELinux can be found in section 2.2. Confining the secure launcher daemon to a sandbox is a critical part of protecting the data. The SELinux policy should be able to protect the sandboxed daemon from the following intrusions: root access gained by a compromised server process (httpd, sshd, e.t.c.) and guessing password for one of the users for sshd eventually giving root access on the Management node to the attacker.

Before discussing the SELinux policy, it would be worthwhile to discuss some SELinux quirks.

Three SELinux usertypes we need to know about are: `user_u, system_u and unconfined_u`. The `user_u` type is the most restrictive among the three. It allows the user only to access files which are allowed to the `user_t` type and the types it can transition to. The user type `system_u` is reserved for system operations and SELinux commands. The `unconfined_u` user is a special user category which reduces the user to DAC privileges i.e. no MAC. Clearly, `unconfined_u` and `system_u` user types are potential threats to the security of the system and need to be well-monitored.

A Management node on the cloud has a defined set of responsibilities. Thus, it has a known set of software installed and running on it, making it feasible to write *strong* SELinux policies for it. Each server/daemon is constrained to its own domain and given access to just the files it absolutely needs. This is not an unruly demand. Fedora already comes with a Targeted SELinux policy that has known servers (httpd, sshd, e.t.c.) appropriately confined. For example, httpd subjects and objects are labelled with the type `httpd_t`.

Writing a new policy from scratch for the Management node is a tough job. One needs to know a lot about the internal workings of a program to be able to write a correct policy for it. Consequently, a better approach is to start with a stock policy like Fedora's targeted policy and rip-off the potentially insecure parts. Our focus on writing the SELinux policy is two fold: to remove the "unconfined" privileges of Linux root user and confine our secure launcher daemon. All other servers and daemons are assumed to be already properly confined for our purpose.

4.4.1 Part one: Remove unconfined privileges of root

The targeted policy maps each Linux user to the SELinux `unconfined_u` user. As we have seen above, such a mapping reduces the system to a DAC system. This gives the root user ultimate privileges even with SELinux enabled. In order to confine root, it is necessary to remove the `unconfined_u` SELinux user totally from the system. This can be done by the following set of commands:

```
# sudo semanage login -m -s "user_u"
-r s0 __default__
# sudo semanage login -m -s "user_u"
-r s0 root
# sudo semodule -r unconfined
```

The first command maps all the Linux usernames to `user_u` SELinux user. The second command maps the Linux root user to `user_u` SELinux user. The third command (optional) removes the unconfined module. Removing the unconfined module greatly adds to the security. The Linux root user can now be confined and hence, damage control is possible.

4.4.2 Part two: Sandboxing the secure launcher daemon

A new SELinux user is now defined for the secure launcher daemon, say, `secure_u`. All the files needed by the daemon are marked with the type `secure_t`. Thus, only the secure daemon can access the files. No transition is allowed to or from `secure_t` type. This provides appropriate sandboxing. The following commands show how to add a new user and label files:

```
# sudo semanage user --add --level s0
-r s0-s0 --role secure_r --prefix user
secure_u
# sudo semanage fcontext --add
--seuser secure_u --type secure_t
/dev/tpm
```

The first command adds a new SELinux user called `secure_u`. The second command shows how a file pertinent to the secure launcher daemon can be labelled with the type `secure_t`.

4.4.3 Unconfined privileges to root in limited cases

Mapping the Linux root user to the SELinux `user_u` user already took off many of the privileges of the root user. Adding "PermitRootLogin=no" to the sshd config disallows root login over ssh. But, it would be truly wishful thinking to assume that a root login will never be needed. Upgrading the guest machine, adding new software, reconfiguration would be impossible without root. The trick then is to permit a transition to the unconfined domain ONLY if the user can prove physical access. Physical access means the requesting process should have a tty. Adding the following line to the `/etc/sudoers` will enable one to sudo to root user with unconfined access:

```
  root ALL=(ALL) TYPE=unconfined_t
ROLE=unconfined_r ALL
```

A minor patch to "sudo" will check for the presence of tty (a call to `isatty()`) before granting the root access.

Since a remote attacker can never have a physical tty, he can never have an unconfined root access to the Management node.

5 Experimental Evaluation

5.1 Tools Used

Since none of us had access to a real TPM, we used a TPM emulator software from BerliOS (`tpm_emulator-0.7.2`). The emulator exports a device `/dev/tpm` and a user-space daemon `tpmd` which emulates the TPM hardware. Any requests on the TPM device file are forwarded to the daemon for execution. We used the `TrouSerS` library to interface with the TPM in userspace (`trousers-0.3.6`). We also use a patched version of `tpm-tools-1.3.5` package for the TPM unseal operation.

We used the `OpenSSL` library for SSL communication and symmetric key encryption/decryption/integrity verification. Specifically `openssl-1.0.0e-1`.

Since the actual implementation of a cloud with a provisioning engine is infeasible on a local machine, we used a poor man's cloud: a virtual machine. We chose KVM [3], since it is popular and opensource. Overall, to come up with a proof of concept, we used the following tools: Host machine as Management node. `qemu-kvm` acts as the provisioning engine (It is actually an emulator, but at a higher level, given a kernel image, it first does the appropriate provisioning on the host machine and then boots the image). Specifically we used `qemu-kvm-0.14.0` as our ”launcher”.

All the experiments were performed on a Lenovo SL510 laptop running a Core 2 Duo @ 2.1GHz processor with 1GB of RAM.

5.2 Attack Vectors

In this section we describe the various attacks that an attacker can launch and our defences against them. The premise of an attack is that an attacker can obtain *remote root shell access* on the Management node. This maybe obtained by compromising one of the servers running on the Management node or guessing a password of one of the users over sshd (either root user or other non-root user). In the presence of this shell access, the problem definition is to protect the confidentiality of customer data.

5.2.1 Re-program the TPM

Since TPM is our root of trust, a malicious attacker can try to reset the TPM to cause a DoS attack or the attacker can try to unseal the SSL private key using PCR#16 which already has the correct extended hash. There are two defences in place against this attack:

1. Authdata is required to seal/unseal data from the TPM or program the TPM.

2. The TPM device, `/dev/tpm` is labelled as `secure_t` and hence, can be accessed only by processes who can transition to that type. Thus, only processes part of TCB i.e. the secure launcher daemon and the actual launcher is allowed to access the TPM device.

5.2.2 Steal kernel/data encryption keys

The attacker can try to steal the kernel encryption key (to create a malicious kernel and a valid HMAC for it) or the data encryption key (to compromise the actual customer data). The defences against these attacks are:

1. The customer site reveals the keys only to a properly attested and authenticated cloud process. To authenticate to the customer site, the attacker will need the SSL private key of the cloud provider. This has been sealed off in the TPM and can be unsealed only by processes in the TCB as explained earlier.

2. SELinux confinements make sure that no user or server gets unconfined access to the Management node. Unconfined access can only be obtained via a physical terminal. Thus, privileged operations like inserting a kernel module in the kernel are not possible for a remote attacker, thus, disallowing the attacker to sniff memory after SSL has decrypted the keys into memory.

5.2.3 Modify the kernel

The attacker may try to maliciously modify the guest kernel in order to steal the customer data. However, the kernel is stored encrypted and integrity-protected. Thus, any modification in the kernel will be caught and the secure launcher daemon would refuse to boot the kernel. Also, SELinux confinements make sure that the kernel image is marked with the type `secure_t` and thus, can only be accessed by the processes in the TCB.

5.2.4 Use a customized kernel

The attacker may just try to get his own kernel copy and a launcher binary and try to boot an image completely outside of the TCB. Although nothing will prevent such a image to boot, the attacker doesn't have access to the data decryption keys and hence, cannot decrypt data on the guest image. The data decryption keys can only be obtained by processes in the TCB.

5.2.5 Attack launcher process

There are two possible places where the secure launcher daemon interacts with the external world: To get a launch request from the httpd server (via a Unix socket, in our implementation) and the SSL connection to the customer site. The launch request is merely a string with appropriate parameters to boot the VM and optionally, Authdata. This string is fed into the Unix socket that the secure launch daemon reads. The customer audit should make sure that the string is properly checked for possible buffer overflow attack and XSS attacks. The SSL connection merely transfers a known number of bytes (two keys and an HMAC) from customer site to the cloud site and hence, the code to parse them can be easily sanitized. The only real point of attack is the OpenSSL library. A vulnerability in the OpenSSL library could indeed compromise the secure launcher daemon and hence, the whole scheme. Fortunately, OpenSSL is security-sensitive code and millions of eyes around the world audit OpenSSL for possible vulnerabilities. It is recommended to keep a watch on the OpenSSL vulnerabilities page [5] and update as soon as a new version is available (New versions usually fix bugs). SELinux confinements on the Management node (disallowing remote unconfined users, `secure_t` labelling, etc) leave only the above two possible attack alleys on the secure launcher daemon.

5.2.6 Access running guest VM

Although in our test case, we run the Management node *and* the guest machine on the same physical machine, in the real cloud, the guest VM will be provisioned on a different physical node and hence, its memory and other dynamic characteristics cannot be directly accessed from the Management node by the attacker.

5.2.7 Cold boot attacks

The attacker might try to start a malicious kernel on the same physical node that a customer VM was running on, and try to read the memory in the hope of catching stray sensitive data. The provisioning software or the customer kernel should zero out the physical memory before allowing another kernel to execute in the same physical space. Such a requirement should be checked as a part of the security audit done by the customer.

Finally from the practical perspective, no security system can be perfect and we don't claim that ours is perfect either. Writing a SELinux policy is a complex task and prone to errors, hence, we have included as much fallback security as possible. Some core areas do need to be protected with SELinux without an option. Those are: disallow unconfined access, protect `/dev/tpm` with `secure_t` label, label the secure launcher daemon the files it needs as `secure_t`. We believe that implementing the methods described above would raise the bar of security on the cloud significantly.

5.2.8 Performance overhead

The performance overhead is measured as the delay between arrival of a request to launch a customer VM and the actual booting of the VM.

Since we didn't have a real cloud environment installed, the delay for the normal case (i.e. just entering the launch command on the commandline) is zero seconds.

After the complete implementation, the delay between the trigger and the actual boot on the VM is around 2 seconds. This time includes time for SSL connection, decryption of the kernel (4MB encrypted with AES in CBC using using a 128-bit key) and TPM unseal operation to unseal the private SSL key.

The caveat in the measurement is that SSL connection was done on a fast a ethernet connection (LAN) and a loopback interface. The real SSL latencies might be a bit higher. Also, the TPM operations were done with a TPM emulator which doesn't emulate the real TPM times. Real latencies can be calculated very easily with access to the real hardware.

6 Related work

Owing to the trusted computing initiative we have seen increasing research in attestation systems [13] and their use in verification of application integrity.

Work has been carried out earlier with the use of a cloud verifier [30] to isolate users and protect them from attacks in the cloud. It clearly summarizes the customer requirements from a cloud platform and provides a cloud verifier implementation that works to protect the integrity of customer VMs through attestations and enforcement of access-control mechanisms.

At NC State University under the the Secure Open Systems Initiative (SOSI), VCL developers continue to work [34] towards a comprehensive security solution for encryption of data on VCL images for VCL units. Within the same context work is being carried out to use watermarking and image security certification for images stored in the VCL image library. The VCL team is also working on other security aspects, like IP-locks, End-to-end isolation to further enhance security in the cloud.

There have been numerous one-off works for providing a TCB similar to the one we setup. One such article outlines a trusted boot method for standalone IBM machines using PKI that successively verifies each component before transferring control over to it [9]. Alternatives like Trusted Platform Module (TPM) [6] based solutions have also been explored widely in papers[28, 16, 18, 17]. These and additional requirements of a trusted cloud platform have been well-summarized in [30]

7 Conclusion

IaaS (Infrastructure as a Service) providers provide a low-cost outsourcing of infrastructure to the Cloud. However, to enable customers with business critical data to offload their data and computations to the cloud, the cloud provider must work with the customer to establish a strong Trust Computing Base which can protect the customer data in the face of compromise on the cloud provider side. To enable such a collaboration, we recognize the minimum components required for a Trusted Computing Base to be provided, in an auditable way, to the cloud customer. We also show how the customer, in partnership with the cloud provider, can build a secure system on top of this TCB. We recommend using SELinux to confine each server/daemon to its own domain. In the future, SELinux may be replaced by a reasonable Dynamic Integrity Attestation technique, which provides a stronger guarantee of secure computation.

The performance overhead of such a system is a little extra time needed during launch of a VM. This time is typically close to about 2s, but it will vary depending on the Round-Trip time between cloud provider and the customer's servers. Such an overhead during launch of a VM, which is a sufficiently non-frequent event, will be acceptable for the security benefits it provides.

Currently, due to lack of information in the public domain about the internals of popular IaaS providers, we had to base our design on NCSU's VCL system. This lack of information about real clouds, can be seen as a limitation of the project. But, we are confident that most of our concepts can be easily applied to most of the clouds.

References

[1] Cloud computing adoption survey results. `http://www.thehostingnews.com/cloud-computing-adoption-survey-results-released-12517.html`.

[2] Extreme cloud administration toolkit. `http://xcat.sourceforge.net`. [Online; Accessed: December 1, 2011].

[3] Kernel-based virtual machine. `http://www.linux-kvm.org/`. [Online; Accessed: December 1, 2011].

[4] Network security services. `http://www.mozilla.org/projects/security/pki/nss`. [Online; Accessed: December 1, 2011].

[5] Openssl vulnerability news. `http://www.openssl.org/news/vulnerabilities.html`. [Online; Accessed: December 1, 2011].

[6] Trusted computing group. http://www.trustedcomputinggroup.org, 2010. [Online; Accessed: 28 Sept, 2011].

[7] Amazon web services: Overview of security processes, May 2011.

[8] Cloud computing takes off. `http://www.morganstanley.com/views/perspectives/cloud_computing.pdf`, May 2011. [Online; Accessed: December 1, 2011].

[9] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A secure and reliable bootstrap architecture. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP '97, pages 65–, Washington, DC, USA, 1997. IEEE Computer Society.

[10] Frederik Armknecht, Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, Gianluca Ramunno, and Davide Vernizzi. An efficient implementation of trusted channels based on openssl. In *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, STC '08, pages 41–50, New York, NY, USA, 2008. ACM.

[11] D. Catteddu and G. Hogben. Cloud Computing: benefits, risks and recommendations for information security. Technical report, Technical Report. European Network and Information Security Agency, 2009.

[12] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pages 85–90, New York, NY, USA, 2009. ACM.

[13] George Coker, Joshua Guttman, Peter Loscocco, Amy Herzog, Jonathan Millen, Brian OHanlon, John Ramsdell, Ariel Segall, Justin Sheehy, and Brian Sniffen. Principles of remote attestation. *International Journal of Information Security*, 10:63–81, 2011. 10.1007/s10207-011-0124-7.

[14] Alexei Czeskis, David J. St. Hilaire, Karl Koscher, Steven D. Gribble, Tadayoshi Kohno, and Bruce Schneier. Defeating encrypted and deniable file systems: Truecrypt v5.1a and the case of the tattling os and applications. In *Proceedings of the 3rd conference on Hot topics in security*, pages 7:1–7:7, Berkeley, CA, USA, 2008. USENIX Association.

[15] Clemens Fruhwirth. New methods in hard disk encryption. Technical report, 2005.

[16] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. pages 193–206. ACM Press, 2003.

[17] Tal Garfinkel, Mendel Rosenblum, and Dan Boneh. Flexible os support and applications for trusted computing. In *IN 9TH HOT TOPICS IN OPERATING SYSTEMS (HOTOS-IX)*, pages 145–150, 2003.

[18] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation - a virtual machine directed approach to trusted computing. In *USENIX Virtual Machine Research and Technology Symposium*, pages 29–41, 2004.

[19] Seny Kamara and Kristin Lauter. Cryptographic cloud storage. In *Proceedings of the 14th international conference on Financial cryptograpy and data security*, FC'10, pages 136–149, Berlin, Heidelberg, 2010. Springer-Verlag.

[20] Chongkyung Kil, Emre C. Sezer, Ahmed M. Azab, Peng Ning, and Xiaolan Zhang. Redas: A remote dynamic attestation system based on constraint verification. In *Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009).*

[21] Peter A. Loscocco and Stephen D. Smalley. Meeting critical security objectives with Security-Enhanced linux. In *Proceedings of the 2001 Ottawa Linux Symposium*, 2001.

[22] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. Safe passage for passwords and other sensitive data. In *NDSS*. The Internet Society, 2009.

[23] Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, Markus Huber, and Edgar Weippl. Dark clouds on the horizon: using cloud storage as attack vector and online slack space. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 5–5, Berkeley, CA, USA, 2011. USENIX Association.

[24] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.

[25] 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.

[26] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2:277–288, November 1984.

[27] Aaron Peeler Henry Shaffer Eric Sills Sarah Stein Josh Thompson Mladen Vouk Sam Averitt, Michael Bugaev. Virtual computing laboratory (vcl). `http://renoir.csc.ncsu.edu/Faculty/Vouk/Papers-Public/VCL_ICVCI_May07.pdf`, May 2007.

[28] Nuno Santos, Krishna P. Gummadi, and Rodrigo Rodrigues. Towards trusted cloud computing. In *HOTCLOUD*. USENIX, 2009.

[29] Joshua Schiffman, Thomas Moyer, Christopher Shal, Trent Jaeger, and Patrick McDaniel. Justifying integrity using a virtual machine verifier. *Computer Security Applications Conference, Annual*, 0:83–92, 2009.

[30] Joshua Schiffman, Thomas Moyer, Hayawardh Vijayakumar, Trent Jaeger, and Patrick McDaniel. Seeding clouds with trust anchors. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, CCSW '10, pages 43–46, New York, NY, USA, 2010. ACM.

[31] Umesh Shankar. Toward automated information-flow integrity verification for security-critical applications. In *In Proceedings of the 2006 ISOC Networked and Distributed Systems Security Symposium (NDSS06)*, 2006.

[32] S H Shin and Kazukuni Kobara. Towards secure cloud storage. *Demo for CloudCom2010*, 2010.

[33] A. Tolnai and S.H. von Solms. Securing the cloud's core virtual infrastructure. In *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on*, pages 447 –452, nov. 2010.

[34] M. A. Vouk, A. Rindos, S. F. Averitt, J. Bass, M. Bugaev, A. Kurth, A. Peeler, H. E. Schaffer, E. D. Sills, S. Stein, J. Thompson, and M. Valenzisi. Using vcl technology to implement distributed reconfigurable data centers and computational services for educational institutions. *IBM Journal of Research and Development*, 53(4):2:1 –2:18, july 2009.

[35] David Wagner and Bruce Schneier. Analysis of the ssl 3.0 protocol. In *Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, pages 4–4, Berkeley, CA, USA, 1996. USENIX Association.