Tenants attested Trusted Cloud Service

(+) Jiangchun Ren, (*) Ling Liu, (+) Da Zhang, (*) Qi Zhang, (+)Haihe Ba
(+) School of Computer, National University of Defense Technology, Changsha, China
(*) School of Computer Science, College of Computing, Georgia Institute of Technology, Atlanta, USA
wwwrjc@163.com, lingliu@cc.gatech.edu, dadabop2003@163.com, qzhang90@gatech.edu, haiheba@nudt.edu.cn

Abstract—**Cloud computing has successfully enabled large scale computing to be offered as pay-as-you-go services to many enterprise and individual tenants. However, the trust on public cloud services has been a sensitive issue for both cloud tenants and cloud service providers (CSPs). Tenants tend to worry about losing the total control over their codes and data hosted on remote servers. Public cloud providers often fear that the applications uploaded by their tenants may carry vicious codes, which may cause serious violations of security and privacy on their cloud platforms. This trust issue has slowed down the wide deployment of public clouds and hindered the promises of cloud computing for both CSPs and Cloud consumers. In this paper, we present Ta-TCS, a novel system framework for two-phase tenants attested trust validation and trust management over their remote VMs and cloud service executions. At the CSP end, we build a Minimal Trusted Environment (MTE) in VMM and an Integrity Verification & Report Service (IVRS) hosted in the control domain Dom0. At the tenant end, we deploy an Integrity Configuration and Attestation Service (ICAS) in new framework. With Ta-TCS, tenants can configure and attest the integrity of their services, and Cloud providers can verify codes running on a guest VM by introspection. Tenants can also check whether the basic platform of Dom0 is trusted or not. This two phase trust establishment increases the level of mutual trust between tenants and its CSP. We implement the first prototype system of Ta-TCS on Xen platform, and most of our implementation mechanisms can be deployed to some open-source virtualization platforms such as KVM. Our evaluation results show that Ta-TCS is effective with negligible performance overhead.**

Keywords—cloud; trust; remote attestation; VM introspection

I. Introduction

Cloud computing has revolutionized how large scale computing is delivered to both enterprise organizations and individual business owners and scientists. The promise of economics of scale and the utility based computing have fueled or transformed information technology and everything in computing into pay-as-you-go services. We have witnessed three types of Public cloud services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS). IaaS has been the most popular platform for public cloud services. The typical use case for IaaS is the virtualize cloud computing environment in which Cloud Service Providers (CSPs) partitions their physical host machines into Virtual Machines (VMs) and provide VM rental services to the enterprises or individual consumers and we call them the tenants. Tenants can use the VMs as if these VMs are their own physical servers and they can install their preferred operating system (OS), web server and middleware and the only difference is that these VMs are running remotely in a third party CSP site.

However, the trust on public cloud services has been a sensitive issue for both cloud tenants and CSPs. Tenants tend to worry about losing the total control over their codes and data hosted at remote servers. Even though tenants can remotely manage how their data is stored and accessed, restrictions of access control, and running/stop state of their services on the VMs they rented, they often worry about that the remote VMs might be spied, tampered or wrongly configured behind their back. They do not trust CSPs and fear insider attacks within CSPs [1]. Consequently, many tenants hesitate to use public clouds to host and analyze their confidential and sensitive data.

At the same time, CSPs cannot trust tenants too. They fear that the application and system programs uploaded by their tenants may carry malicious codes, which can cause serious security and privacy violations, such as attack and compromise other tenants' VMs, break down their physical platforms. Furthermore, the VMs may be less aware of the abnormality of other VMs and thus less capable of preventing virus and hacker's attacks than physical servers. Also packages communicating between different VMs within a cloud data center perimeter may not pass through an outer firewall or intrusion detection system. This may also make the cloud services more vulnerable to other VM viruses and attacks in the clouds. Therefore, both cloud service providers and their tenants are demanding trust management mechanisms that can improve the level of mutual trust between tenants and CSPs.

A fair amount of research efforts in recent years have been dedicated to trust evaluation and integrity verification in the Clouds. A majority of existing efforts have been centered along two directions: one is to help tenants to choose the most trusted CSPs based on historical experience records [2~3], and another is to help CSPs to improve the capabilities of detecting attacks [4~11]. An overview of this line of work will be provided in the next section on related work. In this paper, we argue that relying solely on historical experiences of tenants can be error prone, especially for new cloud providers or new tenants or for tenants who have only limited experiences with a small number of CSPs. We argue that an effective alternative mechanism is to develop efficient and yet secure mechanisms that can help tenants attest the trustworthiness of their cloud services running in a third party cloud datacenter. Furthermore, we also argue that the tenant-attested trusted cloud services should raise the level of mutual trust between a CSP and its tenants. .

In this paper, we proposed our tenant-attested trusted cloud service, called Ta-TCS, which presents a novel two-phase framework of tenants-attested trustworthiness for cloud service. It provides functional capabilities for configuring and attesting remote services to tenants. At the cloud provider end, we build a Minimal Trusted Environment (MTE) in VMM and host an

Integrity Verification & Report Service (IVRS) in the control domain Dom0. At the tenant end, we provide an Integrity Configuration and Attestation Service (ICAS) in the tenant's private cloud or an independent Trusted Third Party (TTP). ICAS collaborates with MTE and IVRS to enable tenants to manage and attest their remote services conveniently. We also allow CSPs to check tenants' codes in a non-intrusive manner. These capabilities can increase the level of mutual trusts between CSPs and their tenants, encouraging to establish trusted behavior through collaborative agreements. This paper makes three original contributions. Firstly, Ta-TCS can help tenants to manage and attest service in remote VMs, by establishing a better trust relation between CSPs and tenants by technical means. Secondly, Ta-TCS can measure dynamic integrity of cloud services at runtime based on some measurement lists through VM introspection. It can implement efficient integrity verification for all types of VMs. Finally, Ta-TCS provides a valuable reference to build trust chains between remote VMs and tenants in clouds.

The rest of this paper is organized as follows: section II describes related work. Section III describes the architecture and design of our system; section IV shows the implementation and evaluations; section V provides discussions and concludes this paper. We summarize major acronyms used in the paper in TABLE I.

TABLE I. ACRONYMS (IN ALPHABETICAL ORDER)

AIK	Attestation Identity Key
DRTM	Dynamic Root of Trust for Measurement
ICAS	Integrity Configuration and Attestation Service
IM	Integrity Management
IMA	Integrity Measurement Architecture
IVRS	Integrity Verification & Report Service
ML	Measurement List
MTE	Minimal Trusted Environment
PCR	Platform Configuration Register
PTS	Platform Trust Services
SRTM	Static Root of Trust for Measurement
TCB	Trusted Computing Base
TCCP	Trusted Cloud Computing Platform
TCG	Trusted Computing Group
TPM	Trusted Platform Module
TSS	TCG Software Stack
VMI	Virtual Machine Introspection
vTPM	Virtual TPM

II. BACKGROUND AND RELATED WORK

2.1 Trusted computing and Integrity measurement

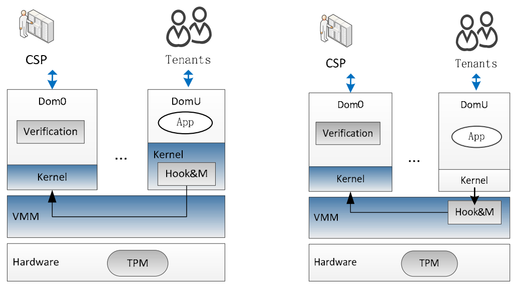
Trusted Computing Group (TCG) [12] defines Trusted Platform Module (TPM) as a root of trust in hardware for measurement, storage and report the integrity of computing platform. Based on TPM and TCG Software Stack (TSS), security and trust related services like remote attestation, sealing or binding are defined. Remote attestation is a method that proves the trusted state and configuration of a terminal computing platform environment with respect to the remote platform. In TPM, a special purpose signature key named Attestation Identity Key (AIK) is used for platform authentication, platform attestation and certification of keys. To enable trusted computing for an unlimited number of VMs on a single hardware platform, vTPM [13] was written for the Xen hypervisor, which can virtualize the physical TPM, manage simultaneous instances of the TPM emulator, and connect each VM to its dedicated TPM emulator instance.

By TCG specifications, integrity measurement is used to attest the trustworthiness of a system. Concretely, the stored Measurement List (ML) is a list that contains the sequence of measurement records, each of which typically includes the Platform Configuration Register (PCR) number, the filename, and its hash digest. According to the records in the SML, the challenger can re-compute the expect digest and compare it with the PCR value to verify the integrity of the system. Integrity Measurement Architecture (IMA) [5] is a secure integrity measurement system for Linux, which can measure the integrity of the runtime executable programs, shared dynamic libraries, and kernel modules. After calculating the hash digest of a file, IMA extends the digest to the predefined PCR and adds it to a list. The list of measurement is kept inside the kernel space and can be used in the verification of the system integrity during the remote attestation process. OpenPTS is an open-source package of Platform Trust Services (PTS) defined by TCG, which offers the command interface to allow the system (the verifier) to connect to a remote host (the collector) to determine whether the collector has performed a trusted boot. OpenPTS can work with other trusted computing components, e.g., BIOS, GRUB-IMA, Linux-IMA, TrouSerS.

2.2 Virtual Machine Introspection

Virtual Machine Introspection (VMI) allows security applications to have complete visibility over another VM's raw memory state, higher-level code and data structures. VMI has been used to support a wide range of use cases including: intrusion detection [14], digital forensics [15] and so forth. LibVMI [16] is an open-source implementation of VMI supporting commodity hypervisors such as Xen and KVM. It is a C library with Python bindings that makes it easy to monitor the low-level details of a running virtual machine. Volatility [17] is the most popular open-source memory forensics tool, implemented in Python, for the extraction of digital artifacts from volatile memory samples. The extraction techniques are performed completely independent of the system being investigated but offer visibility into the runtime state of the system.

2.3 Reputation Trust vs. Trust Attestation of cloud service



(a) In-the-box (b) Out-of-the-box

Fig.1. Integrity measurement for VMs in clouds

Reputation-based trust evaluation and trust attestation are the two main categories of efforts for trusts in the context cloud services. With reputation trust based trust evaluation, tenants can search and select the potential trustworthy CSPs whose functions satisfy their specific requirements. Similarly, CSPs can choose to provide services to credible tenants based on their reputation trust. Trust evaluation is a mechanism to help tenants to decide if it is worth to obtain services from a certain CSP or not, and it also help CSPs to decide if or not they will provide services to a given tenant. For example: [2] presents a trust evaluation mechanism that utilizes both direct and indirect experiences; [18] proposes a trust management system for a cloud computing marketplace, which identifies the trustworthy CSPs in terms of a subset of attributes.

Research on trust attestation focuses on reliable technical mechanisms to attest trust of remote services using integrity verification methods. The integrity verification often includes files, memory and control flow integrity [19]. Two popular models are used to measure the integrity of remote services: the *in-the-box* methods and the *out-of-the-box* methods [4, 20]. Fig.1 illustrates these two types of integrity measurement approaches. The in-the-box methods typically rely on the security hooks provided by the OS kernel (Fig.1.a). A classical framework of in-the-box methods is the IMA [5], which defines the process to carry out the integrity measurement for executable code in application layer, dynamic linking base, script and kernel module of Linux. PRIMA [6] is another approach that extends IMA based on information flow integrity. [21] presents a security-measurement framework for SaaS, which guarantees that the measurement process in VM is trusted. The in-the-box methods have the benefit of straightforward implementation. However, they are vulnerable to kernel rootkits and modifications must be made to the guest OS. The out-of-the-box methods can monitor or analyze program integrity [22], file-system integrity [23], rootkit activities [24], malware [4], and so on [9] in a virtualized environment. As shown in Fig.1.b, the out-of-the-box methods can hook some syscalls or dump guest VM's memory in VMM and check program integrity by computing their checksums [11]. But the out-of-the-box approach can only observe the memory, the register, the disk image, and so on, from the outside. It has no direct access to the internal context of the guest OS, such as processes, modules, and files.

Both methods assume that some components such as VMM and Hook/Measure are trustworthy, as they are critical for system security. For example, [25] presents a TPM based solution for commodity computers. [26] gives a TPM based framework which lets applications run on a separate, dedicated, tamper-resistant hardware platform. [27] proposes the Credo architecture to enable trustworthy virtualization based cloud platforms with a small trusted computing base. [28] proposes an architecture for remote platform attestation, allowing applications to establish trust based on their current configuration. [3] designs a Trusted Cloud Computing Platform (TCCP) for IaaS to offer a closed box execution environment with confidentiality protection.

Surprisingly, existing solutions have primarily centered on addressing the security concerns for CSPs. In comparison, little work has dedicated to provide a complete solution to addressing the integrity verification for tenants. Although CSP provides performance assurance in the Service Level Agreements (SLAs) [29], SLAs do not address the problem of tenants about whether remote cloud services can be trusted or not. In recent years, independent research projects have worked on either VM verification or cloud infrastructure verification at a specific software stack. For example, [30] describes a Virtual Machine Verifier system, in which the trustworthiness of the computing platform depends solely on the input given to the VMs running on top of the host platform, instead of the actions taken on the physical host. Thus, it fails to address the threat by a malicious insider modifying physical host. CERTICLOUD [31] describes two protocols to verify the trustworthiness of cloud: TCRR and VerifyMyVM. However, this work focuses primarily on the authentication of user deployed environment with attention to protocols in an IaaS cloud. It provides no system design for implementing the tenants-attestation of trust on cloud services. [32] presents a monitoring system that enables periodical and necessity-driven integrity measurements of some vital parts of cloud computing infrastructure. But this work only monitored files in low level, and cannot detect memory tampering.

In this paper we argue that integrity attestations should be provided for both CSP and tenants with respect to cloud services. Additional challenges for tenants-attested trusted cloud services should be explored and studied. For example, in contrast to integrity attestation for CSP, trust attestation from tenants not only need to verify the trust of service execution at runtime, but also need to verify the trust of remote basic computing platform and VMs. At the same time, the performance overhead for these additional security measures should be minimized.

III. Solution Approach

In this section we describe our solution approach to tenants-attested trusted cloud services. We first discuss the design objectives and challenges and then give an overview of the Ta-TCS system architecture and describe the design of its core components.

3.1 Design Objectives

The design of the Ta-TCS system aims at achieving the following objectives: (1) providing a system-level tool for tenants to manage and attest their remote services in VMs, attestation includes VM identification, bootstrap and runtime state; (2) developing an efficient method to verify remote execution environment at runtime for all kinds of OSs, targeting at eliminating spyware installation, code modification by Trojans or tempered by hackers; (3) constructing a complete trusted chain which involves physical machine, VMM bootstrap, Dom0 bootstrap and DomU bootstrap.

To address the above problems within a unified framework, we present the design of a tenants-aware integrity attestation framework. Our first prototype is developed with the following assumptions: First, the servers have adopted the latest software, and equipped with TPMs, which helps to build the Trust Computing Bases (TCBs). Second, the tenants' application data are encrypted in the cloud storage and decrypted in fragments and protected by applications in the memory of VM [31] or applications can run directly over encrypted data.

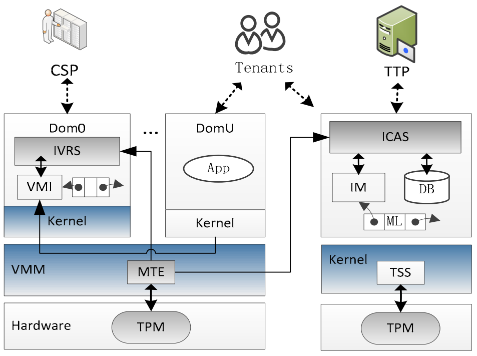


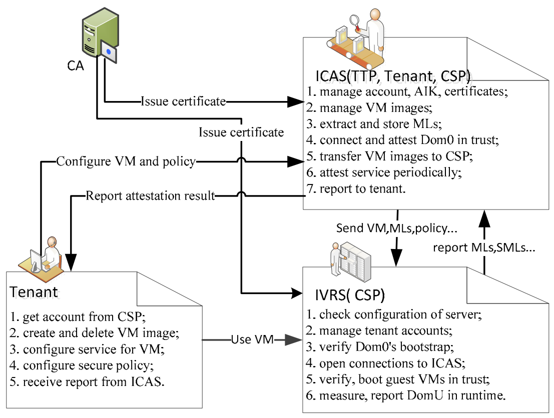
Fig.2. Ta-TCS system architecture

Thus the first prototype implementation of Ta-TCS focuses on code protection in VMs. Third, we assume that the TCB is small and includes the TPM, bootloader and VMM, but not Dom0 and any guest VMs. Attackers may compromise or take over Dom0 or a guest VM. Though tenants do not trust CSPs, they trust the TTP and also trust the technical attestation based TPM and cryptology. Finally, we focus on attentions to the integrity and trustworthiness of VM and software in the Ta-TCS design. The problems of bugs and information leakage are not considered, such as hardware level side channel attacks and the attacks that modify the contents of executables at runtime. Our goal is to enable tenants to attest their remote service execution at runtime by themselves, in addition to configure their VMs, and define access control policies, and ultimately improve the trustworthiness of cloud execution environments for both CSPs and their tenants.

3.2 Ta-TCS System Architecture

Fig.2 gives a sketch of the Ta-TCS framework, with three main functional components: (i) Minimal Trusted Environment (MTE) hosted in VMM, (ii) Integrity Verification & Report Service (IVRS) hosted in Dom0, and (iii) Integrity Configuration & Attestation Service (ICAS) hosted in a TTP server, or the tenant's private cloud.

The role of MTE is to build a minimal trusted environment based on TPM for CSP and tenants to support trust attestation. TPM acts as the trust root for the whole system and the remote tenants/users [27, 28]. Given that cloud services are often running continuously, we cannot frequently reboot the VMs to attest their trustworthiness. Thus, the Static Root of Trust for Measurement (SRTM) technology cannot be used. According to solutions of Trusted Boot (tboot) [33], we adopt the new technology of Dynamic Root of Trust for Measurement (DRTM) and use the DRTM based TPM to build the MTE in VMM. DRTM can allow untrusted code to establish the state of a platform, reset the platform with the DRTM event using a special CPU instruction from MTE. This removes a host of pre-boot software, such as Bootloader and firmware from the platform TCB. MTE also stores and checks the stored MLs in TPM, transfers trust to other components in the TSS such that both CSP and tenants can verify the trustworthiness of the system.


CA
Issue certificate
Issue certificate
Configure VM and policy
Report attestation result
ICAS(TTP, Tenant, CSP)
1. manage account, AIK, certificates;
2. manage VM images;
3. extract and store MLs;
4. connect and attest Dom0 in trust;
5. transfer VM images to CSP;
6. attest service periodically;
7. report to tenant.
Send VM,MLs,policy...
report MLs,SMLs...
Tenant
1. get account from CSP;
2. create and delete VM image;
3. configure service for VM;
4. configure secure policy;
5. receive report from ICAS.
Use VM
IVRS(CSP)
1. check configuration of server;
2. manage tenant accounts;
3. verify Dom0's bootstrap;
4. open connections to ICAS;
5. verify, boot guest VMs in trust;
6. measure, report DomU in runtime.
Fig.3. Key roles and functions in the Ta-TCS system

IVRS is responsible for communicating with MTE, opens trusted connections to ICAS, gets MLs from ICAS, and calls sub-component of VMI to measure the integrity of specific VMs. It also reports whether Dom0 has booted in a trust state.

ICAS provides either GUI or command-line interface to allow tenants configure their remote services and security policies, and acts as a proxy to attest the integrity of remote services. An important component of ICAS is the Integrity Management (IM), which manages the certifications of CSP, the Attestation Identity Key (AIK) of TPM, the images of VMs and SMLs for all kernel components and user programs inside VMs. In Ta-TCS, ICAS acts as an important auditing agent for tenants, which monitors and audits the guest VMs on which their applications are running. ICAS is also used by the administrators of CSPs to check and verify specific VMs from IaaS perspective. Fig.3 illustrates some of the different roles and functions of ICAS and IVRS in our Ta-TCS system. After the tenant created and configured his VMs in ICAS, ICAS can boot up his VMs in native server and produce the MLs automatically, then it sent the VM image and corresponding MLs to CSP via the IVRS in Dom0. CSP can also boot, measure and verify the VM according to tenant's policy and MLs. Final results of attestation will be reported by IVRS to ICAS and by ICAS to the tenant.

Building of Trusted Chain. We need to construct a complete trusted chain, which involved physical machine, VMM bootstrap, Dom0 bootstrap and DomU bootstrap. Fig.4 gives the system architecture for trust chain construction of CSP. The trust of bootstrap for VMM is guaranteed by MTE (Tboot integrated in it) and TPM. We also put the IVRS in the trusted chain. So the critical trusted chain in Dom0 is: TPM→MTE→OS→TSS→IVRS. Given that Dom0 has a virtual TPM (vTPM) manager, tenants can build trusted chain in guest VMs based on vTPM too.

3.3 Design of IVRS

As shown in Fig.4, IVRS mainly has three components: the VM introspection (VMI), the Platform Trust Service (PTS) collector and vTPM manager. The VMI is used to measure guest VM by the out-of-the-box method. The PTS collector is used to collect all kinds of information to attest the trust of platform. The vTPM manager manages all virtual TPM instances in guest VMs. The integrity verification is the most important function in IVRS with three main tasks:

(1) Verification for Dom0: Get SML of bootstrap in Dom0, quote corresponding Platform Configuration Register (PCR) value, and attest them by native TPM in CSP or ICAS by tenants, which can be easily implemented by integrating open-source packages of Tboot and OpenPTS. To prevent critical files from tampering at runtime, OpenPTS also collects integrity measurement logs from IMA in the guest kernel.

(2) Verification for VM image: when CSP received the tenant's VM image (often encrypted and encoded by a hash), IVRS will decrypt it and re-compute the hash to verify its integrity.

(3) Verification for services in guest VMs: The methods for files integrity measurement are not effective to verify the integrity of service at runtime, because malware often tamper memory directly. We can use VMI to check details in virtual memory and disk from the outside of VM, but we need an efficient solution to measure, verify and report integrity of the

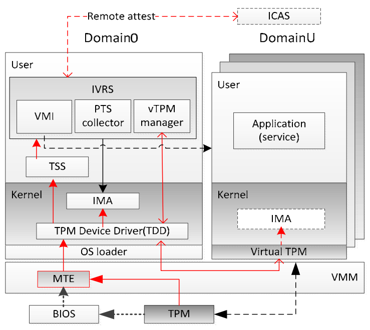


Fig.4. Trust Chain in a virtualized platform

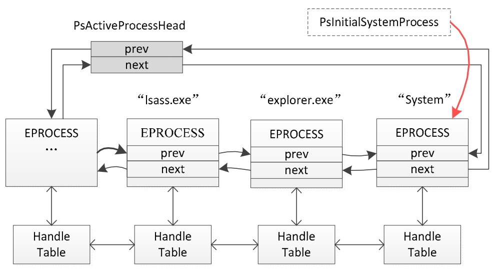


Fig.5. Locating a process in memory

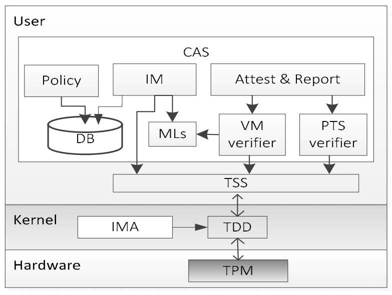


Fig.6. the ICAS subsystem architecture

services running on VMs.

3.3.1 Integrity Measurement for Services Running on VM

In Ta-TCS, we implement integrity measurement on top of LibVMI and Volatility. Recall Section II, LibVMI provides an application programming interface (API) for reading to and writing from a VM's memory. LibVMI API is accessible via a Python interface, which is extended by Volatility to read, interpret context in memory of guest VM easily and efficiently. LibVMI and Volatility use some kernel data structures to access the footprints of VM memory.

For Window OS, the kernel maintains a circular linked-list of all active processes, i.e., the ActiveProcessLinks. Each active process is represented in this linked-list with an EPROCESS structure. We can locate this list indirectly using the exported PsInitialSystemProcess symbol, which points to the EPROCESS structure of the "System" process. Next, within the executive process structure, there is a HANDLE_TABLE structure containing information about that process's files, devices, ports, and other similar handle objects. This structure contains a HandleTableList, consisting of backward and forward links to handle tables of other processes. We can traverse each handle table for all active processes in the system.

Linux offers a much simpler process management scheme. With circular linked list of Task_Struct structures, each of which contains information about processes running in the system. By leveraging the kernel-exported symbol information we can find a starting task_struct and then traverse all of the processes. The virtual address of the init_task pointer can be found from the System.map file in the /boot directory.

To measure the integrity of a process in memory, we compute a checksum for its static codes and then compare that with the value recorded while verifying. The work of dumping codes in process can easily be done by the Volatility. Volatility can list all executable entities, such as process, DLL and kernel module in a specific VM and dump their codes to compute checksums. Section V will detail the implementation. We use three MLs named 'ProcList', 'ModList' and 'DllList' to record information for all processes, modules and libraries. Values of some members in the structure are assigned initially by ICAS (e.g. 'bPass' is set -1). In Ta-TCS, integrity measurement is done periodically or on-demand. It only checks the static codes in memory, not including any dynamic data.

Algorithm 1 Process Measurement and Verification

```
1:  Input: vmID, ProcList
2:  Output: measured ProcList
3:  Load a copy of ProcList for VM using vmID
4:  Get double linked blocks of processes in memory
5:  for each process in blocks do
6:      Get the ProcName
7:      Retrieve ProcList to find process by ProcName
8:      if not found then
9:          create and add a TmpProc node to tail of ProcList
10:         set TmpProc.chName as ProcName
11:     else
12:         Get current process's code buffers in memory
13:         Set S = 0
14:         for each buffer in buffers do
15:             count code buffer's checksum into C
16:             count S = SHA1( $S|C$)
17:         end for
18:         if $pProc->CkSum == S$ then
19:             set $pProc->bPass$ as 1
20:         else
21:             set $pProc->bPass$ as 0
22:         end if
23:     end if
24:     set pProc.CkSum as S
25: end for
```

3.3.2 Integrity Verification

Integrity verification is ongoing while integrity measurement is triggered periodically. We use a thread to load MLs and perform introspection for guest VMs one by one. Algorithm 1 shows the steps of measurement and verification for processes. The steps of measurement and verification for modules and DLLs are similar. Our method measures integrity of processes and modules according to the predefined whitelists. Thus it can find out unauthorized, wrong configured, tampered or missing process. It can also find which DLL in a process is tampered.

3.3.3 Integrity Report

In order to report the integrity verification results, IVRS checks the measured MLs to find out all nodes whose 'bPass' is tagged as -1, 0, or 2. For example: in ProcList, a node with tag -1 means that the process is missing or disabled; a node with tag 0 means that the process has been tempered; and a node with tag 2 means that an unauthorized entity has appeared. All the nodes with exception tags will produce an alert to CSP or tenant depending on whether CSP or tenant administers the process. Upon receiving an alert on some critical process, modules or DLLs, Dom0 will suspend the current VM and send a message to ICAS immediately.

3.4. Design of ICAS

In the first prototype of Ta-TCS, we implement the ICAS subsystem on top of a Linux server equipped with TPM, tboot and IMA. Fig.6 sketches the architecture of ICAS with three main functional components: (1) integrity management, (2) policy configuration, and (3) attestation and report.

Integrity management. The IM component helps the tenant to manage all of his VMs in clouds. When the tenant finished configuring his VM, ICAS can boot up his VM in a native environment. Then, the MLs and corresponding checksums will be extracted from memory. The hash of whole VM image will be computed too. All these records will be sent to IVRS to support verification in Dom0. In addition, the SML and extended PCR value of bootstrap for this VM are recorded, such that the tenant who deployed the Ta-TCS can use these records for VM attestation.

Policy configuration. Ta-TCS offers a number of configuration options to help tenants to attest her remote VMs and services, including attestation for remote platform, VM images, bootstrap and service in runtime. For measurement using the out-of-the-box method, the system supports the VM introspection periodically or on demand. A tenant can set the periodic time interval size.

Attestation and Report. Initially, IVRS and ICAS need to authenticate their identification by each other with TPM's AIK certificate. Next, ICAS will attest the trustworthiness of Dom0. We called this the platform attestation. Concretely, the ICAS receives a connection request from a tenant and starts to open a channel to IVRS. To reflect the state of system in Dom0 in the PCR, the IVRS proves to the ICAS that the system is running in a desired trusted configuration by running the special TPM quote operation. It reports the content of a selected set of PCR registers and signs this information with AIK. IVRS sends out the following three items: the quote blob, AIK certificate and SML. The ICAS is now able to determine the state of remote Dom0 by evaluating the quote blob and ML. The SML contains a list of all software components that have been loaded on the remote platform, including their hashes. By recalculating the hashes and comparing them with the hashes from the quote blob, the ICAS has the evidence of the remote platform state. Furthermore, the signature on the quote blob is verified with the AIK certificate. If required, the CA (central authority) is contacted for verification of the certificate itself. Then, ICAS will attest VM and remote service running in the VM. As Fig.7 shows, ICAS first generates a nonce, sends it to

IVRS. Upon receiving the nonce, IVRS loads it into TPM, and uses TPM to quote the current PCR value and nonce; then it returns the TPM signed hash of initial VM image and nonce, extended PCR value to ICAS. When ICAS receives the evidence information, it checks the received nonce with the one sent before, ensuring that the received verification information is in the same session. Then it checks the signature for initial VM image to identify the exact VM. ICAS then uses local MLs to extend operation to calculate the PCR value: TestPCR = SHA1 (TestPCR||m), m is one of the measurements in MLs, TestPCR is the final value that needs to be compared with the PCR value. If the calculated PCR value is consistent with the returned PCR value, then it indicates that the ML in IVRS has not been modified. Finally, ICAS send another nonce to IARS to launch a command triggering remote integrity verification and reporting. ICAS receives the results of report and verifies it. If any problem is detected, ICAS sends out an alert to the respective tenant or CSP.

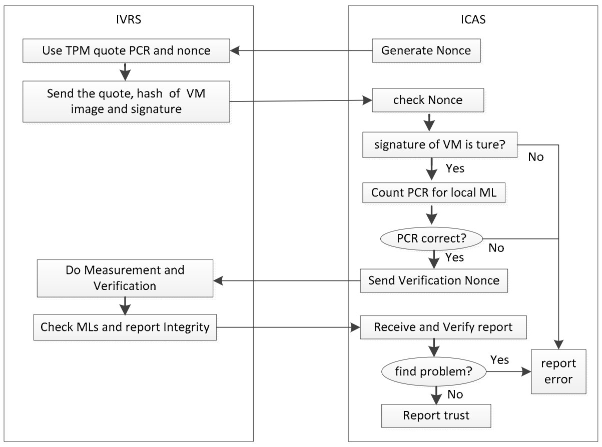


Fig.7. Remote attestation for VM and VM services

IV. Implementation and Evaluations

We implement a prototype of Ta-TCS by integrating some open-source systems, adding about 3000 lines of codes. Fig.8 shows the configuration of hardware and software. A DELL machine, with Intel Xeon CPU, 8@2.0GHz cores, 20M Cache, 64GB memory, TPM built-in, Xen 4.6 and Ubuntu 15.10, is configured virtual TPM for Xen.

The control domain Dom0 is implemented with vTPM, IMA and OpenPTS, which is used to calculate and attest the integrity of the software and hardware configurations. The guest domain DomU can install any type of OS on the Xen platform. We use another physical server, with 4@3.6G CPU cores, 8M Cache, 8GB memory, TPM built in, and Ubuntu 15.10 installed, to simulate the TTP server to host the ICAS subsystem, a core component of our Ta-TCS for tenants-attested trusted cloud services.

To build and configure Ta-TCS, we first enable TPM, TXT [33] in BIOS. Then we install tboot in host OS, which guarantees the host system to boot in trust. We also install Trousers [34] in Dom0, which realizes the TCG Software Stack (TSS), and provides an API to OS and applications. The Ubuntu 15.10 has a built-in vTPM implementation. Finally we install seaBIOS and GRUB-IMA for guest VMs to help VMs boot up in trust based on vTPM. IMA checks the integrity of important files in Dom0 and basic platform of ICAS while they are running. OpenPTS, an open-source package of Platform Trust Service (PTS), is used to verify integrity by IVRS and ICAS. For IVRS, 'ptscd' command is a daemon process which manages the integrity of trusted VM. For ICAS, 'openpts' command is used to validate the target platform by remote attestation. The protocol between ptscd and openPTS is based on TCG IF-M protocol [35]. OpenPTS setup the SSH tunnel between IVRS (collector) and ICAS (verifier) to secure the remote attestation.

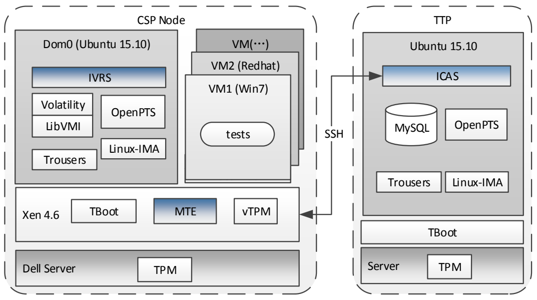


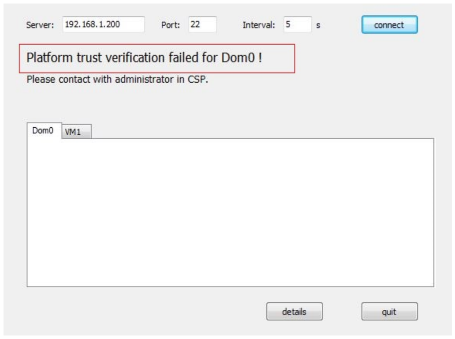
Fig.8. Configurations of Ta-TCS

We use LibVMI and Volatility to read and parse the context of guest VMs' memory in IVRS. To get all active processes and modules (drivers), we use command 'pslist' and 'modules' in Windows OS and command 'Linux_pslist', 'linux_lsmod' in Linux OS. To get DLLs (libraries), we can use the command 'dlllist' in Windows and 'linux_proc_maps' in Linux. To dump the executable codes, we use command 'procexedump', 'moddump', 'dlldump' in Windows and 'linux_memmap', 'linux_dump_map' and 'linux_moddump' in Linux. Next, we adopt the algorithm of SHA-1 to compute a hash as the checksum of process, module or DLLs.

Our test guest VM has 1GB allocation of the host physical memory and 4 GB virtual disk with a standard installation of Windows 7. All software except OS in our test environment is listed in TABLE II. We measured the checksum for every module and every process by ICAS. The result set contained 167 checksums. Then we configured the OpenPTS collector and performed verification for Dom0. Finally, we run some tests to evaluate our system.

TABLE II. Environment of Test VM

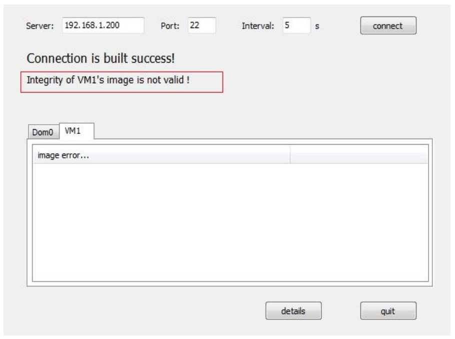
No	Software	Code size
01	Apache Web Server	6.72MB
02	HardInfo	5.61MB
03	IOZone	3.43MB


Server: 192.168.1.200 Port: 22 Interval: 5 s

Platform trust verification failed for Dom0 !

Please contact with administrator in CSP.

Fig.9. Report of Dom0 untrusted

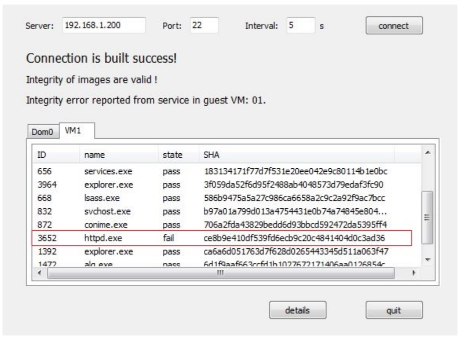

Server: 192.168.1.200 Port: 22 Interval: 5 s

Connection is built success!

Integrity of VM1's image is not valid !

image error...

Fig.10. Report of VM image tampered


Server: 192.168.1.200 Port: 22 Interval: 5 s

Connection is built success!

Integrity of images are valid !

Integrity error reported from service in guest VM: 01.

```
ID      name           state   SHA
656     services.exe   pass    183134171f77d7f531e20ee042e9c80114b1e0bc
3964    explorer.exe   pass    3f059da52f6d95f2488ab4048573d79edaf3fc90
668     lsass.exe      pass    586b9475a5a27c986ca6658a2c9c2a92f9ac7bcc
832     svchost.exe    pass    b97a01a799d013a4754431e0b74a74845e804...
872     conime.exe     pass    706a2fda43829bedd6d93bbcd592472da5395ff4
3652    httpd.exe      fail    ce8b9e410df539fd6ecb9c20c4841404d0c3ad36
1392    explorer.exe   pass    ca6a6d051763d7f628d0265443345d511a063f47
1472    alg.exe        pass    6d1f9aaf663ccfd1b1027672171406aa0126854c
```

Fig.11. Report of Aparch server tampered

5.1 Effectiveness of Ta-TCS Integrity Attestation

First, we replace IVRS with an old version, and use ICAS to build a connection to it. Because the configuration is changed, a report shows that the remote platform is not trusted (Fig.9). Second, we test the effectiveness of IVRS in verifying the image of guest VM before booting it. We replaced the VM with another VM image, which has different configuration. The test shows an alert in ICAS (Fig.10). Next, we test the effectiveness of Ta-TCS in detecting tampered services running on guest VMs. We implemented a tiny tool named 'tamperBuf', which can probe and tamper a process's memory. We run this tool in guest VM to tamper apache web server (httpd.exe). Fig.11 shows the report from ICAS that the apache web server failed the integrity verification. This demonstrates that our system can measure integrity of services from outside of VM effectively.

5.2 Performance of Ta-TCS Attestation

We first measure the overhead of VM introspection on the performance of guest VM. We used command 'xentop' to monitor CPU usage by specific VM while we do introspection on it. Given that the time cost by VMI is sensitive to workloads in VM. We deployed two VMs, both run all the software in Table 1, but we built a heavy workload in VM2 by running 50 copies of IOZone. We set the introspection interval at 15 seconds. Fig.12 shows the measurement of CPU usage in two VMs. We can see that introspection has an obvious impact on the performance of VM, because Xen uses pause/resume model to ensure consistent and coherent memory views in guest VM. We also see that VM2 has higher CPU usage than that of VM1, because the system needs to check more processes in VM2, it takes longer time to read more buffers.

In the next set of experiments, we divide the whole attestation process into five phases: (i) read memory buffer; (ii) compute checksum for codes; (ii) verify and report integrity; (iv) transfer report by network; (v) ICAS verifies the report and sends it to the tenant. Fig.13 shows the time spent in these five phases for two VMs. We observe that much time is spent to compute checksum, and the access to memory and verification are fast. We change the algorithm from SHA-1 to CRC32. Fig.14 shows that the total time of using CRC32 is much less.

To evaluate the performance impact of VM introspection on services in guest VMs, we use the micro-benchmark of 'HardInfo'. HardInfo can run some benchmarks in guest VMs, and calculate the CPU time. We first turn off VM introspection in Dom0, run HardInfo for 100 times to get the average CPU time. Then we set different introspection intervals as 2, 4, 6 seconds and so on. We also run HardInfo for 100 times at each interval setting, measure the average CPU time (shown in Fig.15.). Finally, we measure the increased overhead by percentage. We can see when the interval is larger than 6s, VM introspection has little impact on CPU (<2%).

We also use IOZone file system benchmark 3.4 to test the increased overhead on file reading and writing. The IOZone file system benchmark can do a variety of file operations (e.g., Read, write, fread, fwrite, mmap), and we measure the total completed operations in unit time. We first turn off the VM introspection in Dom0, run IOZone file system benchmark for 100 times to get the average count of operations. Then we calculate the increased overhead on memory access for different interval settings similar to the experiments on CPU cost. Fig.16 shows the results: when the interval is larger than 6s, VM introspection introduces little overhead (<4.4%) w.r.t. memory operations compared to no VM introspection.

Finally, we adopt the real apache web server application benchmark to evaluate our system. The apache web server application benchmark is able to handle approximately 5500 connections per second. The result of this benchmark is the pages served per second by the introspected guest running the apache web server. The VMI mechanism causes performance to drop to one quarter of that rate at the one second period. We also replace sha-1 with crc32 to test. Fig.17 illustrates apache web server application benchmark behavior observed for different introspection and configurations. The curve marked 'None' denotes the test without VMI. We can see the performance of apache web server is near top while the VMI interval is larger than 16 seconds. We think it is acceptable by CSP and tenants.

V. Discussions and Conclusion

We have presented Ta-TCS, a tenants-attested trusted cloud service framework. Different from the state of the art in trusted cloud computing, this tenants-aware integrity verification framework can allow tenants to configure and attest their remote VM and runtime services from the outside of VM. The design of Ta-TCS aims to help both CSP and tenants to verify the trustworthiness of remote services at runtime. Our approach is transparent to the guest OS. Thus, the guest VM that runs Windows, FreeBSD, and other Unix-like systems can also be supported. Furthermore, our approach is mainly implemented

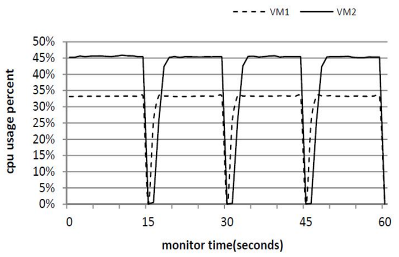


Fig.12. VM's CPU usage while introspection.

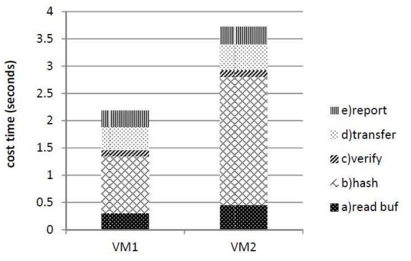


Fig.13. cost time in different phases (using sha-1).

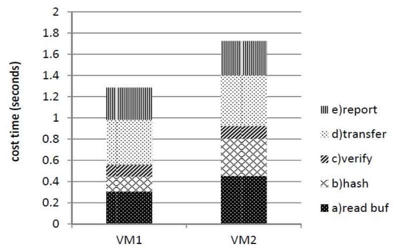


Fig.14. cost time in different phases (using crc32).

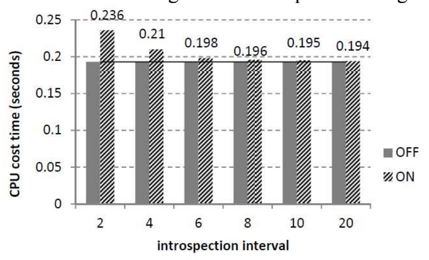


Fig.15. Performance impact on CPU.

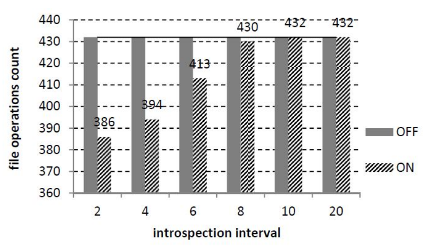


Fig.16. Performance impact on memory.

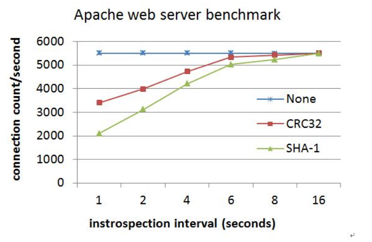


Fig.17. Performance impact on application

by integrating existing open-source tools. It does not require much modification to the VMM or the guest OS kernel. Also our approach allows tenants to configure and attest their remote services and obtain integrity verification report in a timely fashion from ICAS. By implementing the ICAS subsystem in a TTP such as a private cloud, it also enables CSP to monitor and audit cloud services in VMs. Although our approach only checks the static codes in memory, it can guarantee that the processes only run their original codes at runtime. Since the transferred instructions (such as JMP) are also measured, it can detect some attacks of restructuring existing codes too.

The current implementation of our Ta-TCS has some limitations. For example, if a hacker tampers some codes and recovers them within the same VMI interval, it may not detect the change because Dom0 only checks a guest VM once for each interval.

VI. Acknowledgement

This work is carried out when Jiangchun Ren is a visiting scholar in Georgia Institute of Technology. Authors from National University of Defense Technology are partially supported by the National High Technology Research and Development Program of China (863) under Grant No. 2015AA016010. The authors from Georgia Institute of Technology are partially supported by the National Science Foundation under Grants IIS-0905493, CNS-1115375, NSF 1547102, SaTC 1564097, and Intel ISTC on Cloud Computing.

References

[1] T. Ristenpart, E. Tromer, H. Shacham, et al. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds[C]//Proceedings of the 16th ACM conference on Computer and communications security. ACM, 2009: 199-212.

[2] L. Gao, Q. Liu, P. Lou. Trust evaluation in cloud manufacturing environment[C]//The 19th International Conference on Industrial Engineering and Engineering Management. Springer Berlin Heidelberg, 2013: 1173-1180.

[3] S. Habib, S. Ries, M. Mühlhäuser Towards a trust management system for cloud computing[C]//Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on. IEEE, 2011: 933-939.

[4] X. Jiang, X. Wang, D. Xu Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction[C]//Proceedings of the 14th ACM conference on Computer and communications security. ACM, 2007: 128-138.

[5] R. Sailer, X. Zhang, T. Jaeger, et al. Design and Implementation of a TCG-based Integrity Measurement Architecture[C]//USENIX Security Symposium. 2004, 13: 223-238.

[6] T. Jaeger, R. Sailer, U. Shankar. PRIMA: policy-reduced integrity measurement architecture[C]//Proceedings of the eleventh ACM symposium on Access control models and technologies. ACM, 2006: 19-28.

[7] L. Catuogno, I. Visconti. An architecture for kernel-level verification of executables at run time[J]. The Computer Journal, 2004, 47(5): 511-526.

[8] L. Chuanyi, L. Jie, F. Binxing. T-YUN: Trustworthiness Verification and Audit on the Cloud Providers[J]. IEICE TRANSACTIONS on Information and Systems, 2013, 96(11): 2344-2353.

[9] A. Azab, P. Ning, E. Sezer, et al. HIMA: A hypervisor-based integrity measurement agent[C]//Computer Security Applications Conference, 2009. ACSAC'09. Annual. IEEE, 2009: 461-470.

[10] N. Santos, K. Gummadi, R. Rodrigues. Towards Trusted Cloud Computing[J]. HotCloud, 2009, 9: 3-3.

[11] B. Xing, Z. Han, X. Chang, et al. OB - IMA: out - of - the - box integrity measurement approach for guest virtual machines[J]. Concurrency and Computation: Practice and Experience, 2015, 27(5): 1092-1109.

[12] Trusted Computing Group. TCG Specification Architecture Overview, Specification Revision 1.4. (Available from: https://www. trustedcomputinggroup.org/resources/tcg_architecture_overview_versio n_14) [Accessed on 10 Dec. 2015].

[13] R. Perez, R. Sailer, L. Doorn. vTPM: virtualizing the trusted platform module[C]//Proc. 15th Conf. on USENIX Security Symposium. 2006: 305-320.

[14] T. Garfinkel, M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection[C]//NDSS. 2003, 3: 191-206.

[15] A. Shaw, B. Bordbar, J. Saxon, et al. Forensic virtual machines: dynamic defence in the cloud via introspection[C]//Cloud Engineering (IC2E), 2014 IEEE International Conference on. IEEE, 2014: 303-310.

[16] H. Xiong, Z. Liu, W. Xu, et al. Libvmi: a library for bridging the semantic gap between guest OS and VMM[C]//Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on. IEEE, 2012: 549-556.

[17] Volatility Usage. https://github.com/volatilityfoundation/volatility/ wiki/ Volatility-Usage. [Accessed on 10 Dec. 2015].

[18] Z. Wang, X. Jiang. Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity[C]//Security and Privacy (SP), 2010 IEEE Symposium on. IEEE, 2010: 380-395.

[19] S. Suneja, C. Isci, V. Bala, et al. Non-intrusive, Out-of-band and Out-of-the-box Systems Monitoring in the Cloud[C]//ACM SIGMETRICS Performance Evaluation Review. ACM, 2014, 42(1): 249-261.

[20] Q. Liu, C. Weng, M. Li, et al. An In-VM measuring framework for increasing virtual machine security in clouds[J]. Security & Privacy, IEEE, 2010, 8(6): 56-62.

[21] N. Li, B. Li, J. Li, et al. vMON: an efficient out-of-VM process monitor for virtual machines[C]//High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on. IEEE, 2013: 1366-1373.

[22] N. Quynh, Y. Takefuji. A novel approach for a file-system integrity monitor tool of Xen virtual machine[C]//Proceedings of the 2nd ACM symposium on Information, computer and communications security. ACM, 2007: 194-202.

[23] T. Hwang, Y. Shin, K. Son, et al. Design of a hypervisor-based rootkit detection method for virtualized systems in cloud computing environments[C]//Proceedings of the 2013 AASRI Winter International Conference on Engineering and Technology, December 2013. 2013: 27-32.

[24] B. Parno, J. McCune, A. Perrig. Bootstrapping trust in commodity computers[C]//Security and privacy (SP), 2010 IEEE symposium on. IEEE, 2010: 414-429.

[25] T. Garfinkel, B. Pfaff, J. Chow, et al. Terra: A virtual machine-based platform for trusted computing[C]//ACM SIGOPS Operating Systems Review. ACM, 2003, 37(5): 193-206.

[26] H. Raj, D. Robinson, T. Tariq , et al. Credo: Trusted computing for guest VMs with a commodity hypervisor[J]. Tech-nical Report MSR-TR-2011-130, Microsoft Research, 2011.

[27] K. Dietrich, M. Pirker, T. Vejda, et al. A practical approach for establishing trust relationships between remote platforms using trusted computing[M]//Trustworthy Global Computing. Springer Berlin Heidelberg, 2007: 156-168.

[28] M. Alhamad, T. Dillon, E. Chang. Conceptual SLA framework for cloud computing[C]//Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on. IEEE, 2010: 606-610.

[29] J. Schiffman, T. Moyer, C. Shal, et al. Justifying integrity using a virtual machine verifier[C]//Computer Security Applications Conference, 2009. ACSAC'09. Annual. IEEE, 2009: 83-92.

[30] B. Bertholon, S. Varrette, P. Bouvry. Certicloud: a novel tpm-based approach to ensure cloud iaas security[C]//Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE, 2011: 121-130.

[31] R. Neisse, D. Holling, A. Pretschner. Implementing trust in cloud infrastructures[C]//Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE Computer Society, 2011: 524-533. .

[32] Trusted Execution Technology and Tboot Implementation. whitepaper. https://fedoraproject.org/wiki/Tboot. [Accessed on 10 Dec. 2015]

[33] IBM.TrouSerS: The open-source TCG Software Stack. http://trousers. sourceforge.net/[Accessed on 10 Dec. 2015]

[34] Trusted Computing Group. TCG Trusted Network Connect TNC IF-M. Technical report, 2008.