Implementing Trust in Cloud Infrastructures

Ricardo Neisse
Fraunhofer IESE, Germany
ricardo.neisse@iese.fraunhofer.de

Dominik Holling
TU Kaiserslautern, Germany
dominik.holling@holkom.de

Alexander Pretschner
Karlsruhe Institute of Technology, Germany
pretschner@kit.edu

Abstract—**Today's cloud computing infrastructures usually require customers who transfer data into the cloud to trust the providers of the cloud infrastructure. Not every customer is willing to grant this trust without justification. It should be possible to detect that at least the configuration of the cloud infrastructure—as provided in the form of a hypervisor and administrative domain software—has not been changed without the customer's consent. We present a system that enables periodical and necessity-driven integrity measurements and remote attestations of vital parts of cloud computing infrastructures. Building on the analysis of several relevant attack scenarios, our system is implemented on top of the Xen Cloud Platform and makes use of trusted computing technology to provide security guarantees. We evaluate both security and performance of this system. We show how our system attests the integrity of a cloud infrastructure and detects all changes performed by system administrators in a typical software configuration, even in the presence of a simulated denial-of-service attack.**

I. Introduction

In cloud computing, resources are outsourced to service providers on a pay-per-use basis. This model provides advantages both from the economic and the scalability point of view because additional computing resources can be allocated when needed. Today's insufficient solutions to the problem of data confidentiality and integrity, however, prevent some companies from moving their services and data to the cloud. Companies with strict data protection policies miss transparency and security guarantees of the cloud infrastructures. Moreover, they are unable to determine where their data is stored, or do not understand how the cloud infrastructure is managed.

One common approach to improve data protection is encryption of virtual hard disks and network traffic. However, this does not guarantee data protection from malicious or negligent infrastructure providers as these have full control of the cloud infrastructure and are able to configure or modify it in a way that allows them unrestricted access to the data. Thus, in current cloud computing scenarios, service providers have no alternative to trusting infrastructure providers to keep their data secure, and there is no way to verify the integrity of the cloud's hardware/software configuration.

Existing cloud infrastructures use virtualization techniques with hypervisors to transparently allocate resources of physical hosts for a service provider's virtual machines (VMs). In theory, security in virtualization solutions is provided by design. This is because VMs are completely isolated from one another by the hypervisor. In practice, cloud administrators are still able to load a malicious hypervisor module and access the memory and disk content of the service provider's VMs. Furthermore, cloud infrastructure customers do not know who has physical or virtual access to their data by using remote management tools. Therefore, malicious infrastructure providers or one of their internal administrators can manipulate the infrastructure to steal or alter customer data. To illustrate the usefulness of our system, suppose a service consumer C wants to use a cloud service, say, a special file sharing service, provided by a service provider S. S does not provide the physical infrastructure itself but rather runs it on hardware of a third company, I, the infrastructure provider. In principle, I has access to all data stored by S, and hence also the data that C has shipped to S. With our system, we make it possible for S (and also C) to obtain runtime integrity information about crucial system components used by I (e.g., kernel modules, configuration files). Provided that I's system has been set up adequately—which of course has to be ensured by a third party—this in turn provides information about potential leakage of S's (or C's) data. Our approach does not prevent I from altering crucial components and subsequently stealing data, but these activities will at least be detected.

Research Problem. We tackle the problem of protecting entities *using* the cloud from malicious or negligent entities *providing* the cloud infrastructure.

Solution. We present the BonaFides system for remote attestations of security-relevant parts of the cloud infrastructure. It guarantees to service providers at runtime the detection of unintended or malicious modifications of cloud infrastructure configurations. Using trusted computing technology, we protect the BonaFides system itself from tampering. We put special emphasis on the defense against a denial-of-service attack and provide performance and security analyses.

Contribution. We are not aware of implementations that provide guarantees through runtime monitoring of hardware and software integrity. Existing approaches for integrity verification of VMs using trusted computing technology perform integrity measurements at boot time, when a library is loaded, when a program is executed, or when a VM is migrated (§VIII). This makes it possible for an attacker to modify the system in-between integrity measurements in an imperceptible manner. We go one step further and propose a solution to verify the integrity of the hardware and software at runtime whenever changes in the cloud infrastructure are performed.

Organization. §II sets the stage. §III and §IV describe design and implementation of the BonaFides system. In §V, we revisit the above example and show how concretely BonaFides helps protect data. §VI and §VII contain performance and security analyses and highlight our assumptions. §VIII describes related work, and §IX concludes.

II. Cloud Computing

Cloud Computing Model. Cloud computing enables on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort [15]. We distinguish four different roles in a cloud computing scenario: the *infrastructure provider*, the *service provider*, the *service consumer*, and the *cloud certifier*. The entity that coordinates and implements the resource sharing using a cloud computing model is called the infrastructure provider. The service provider is the party that contracts (cloud) outsourcing services from the infrastructure provider. The service provider deploys a platform or software that is used by the service consumers. Service consumers may be the end-users of the service or application deployed in the cloud. Service consumers may also assume the role of service providers if they provide a platform or service in the cloud that is used by someone else. Therefore, it is possible to have a chain of service consumers that also act as service providers. The cloud certifier is a party capable of inspecting the infrastructure provider's infrastructure and retrieving information about security and integrity of the infrastructure used by the service provider. This role might be assigned to any organization that is also a service provider, or to a specialized trusted third party.

Reflecting the above roles, in cloud computing, resources can be outsourced at different layers of abstraction [15], namely software, platform, or infrastructure. In this paper, we are concerned with infrastructure outsourcing where infrastructure providers lease resources at the VM level (processing, storage, and network). Note, however, that software and platform outsourcing models also profit from our approach, since the respective service providers, and consequently, also the service consumers, can rely on the provided guarantees.

Xen Cloud Platform. We chose the open source Xen Cloud Platform (XCP) [31] as the basis for our research, a popular infrastructure that is, among others, used by the Amazon ElasticCloud and CloudCentral. However, we are confident that other virtualization solutions, e.g., VMWare [30], suffer from similar security problems and could equally benefit from the analysis and solution in this paper. A typical scenario of an infrastructure provider running XCP is as follows. The infrastructure provider owns physical hosts that run an instance of XCP each. Every XCP instance has a hypervisor layer using a specific version of the Xen hypervisor that manages access to the physical host's resources, an administrative domain called Dom0 (Domain Zero), and a set of unprivileged domains called DomUs (Domain Unprivileged). The physical hosts are connected to a network infrastructure and may access an external storage device. By design, the Xen hypervisor forbids VMs to access or modify data in the memory of other VMs. The service provider's VMs (DomUs) can only access the CPU and their own predefined memory area, and do not

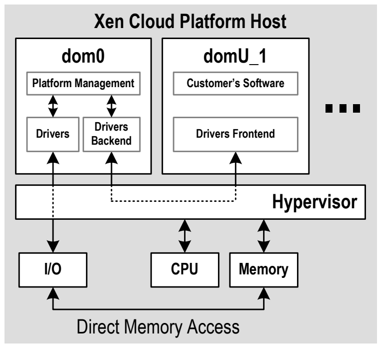

Fig. 1. Physical host running XCP

have direct access to other hardware devices. Furthermore, the Xen hypervisor version adopted by XCP does not include management utilities for memory dumping. This prevents Dom0 from taking a snapshot of the running DomUs' memory content. The Dom0 is a special VM instance with special privileges for hardware access. This is required because, in the Xen architecture, the hypervisor relies on the Dom0 for the mapping and management of all virtual disks, network interfaces, and other device drivers required for hardware access by the DomUs. For this reason, all the network and disk traffic is redirected by the Xen hypervisor through Dom0 backend drivers that interact with devices capable of Direct Memory Access (DMA). Figure 1 depicts the communication between Dom0, DomU, and the Xen hypervisor.

The Xen hypervisor manages the communication between the driver frontend in the DomUs and the drivers' backend in Dom0 as well as the access to the hardware devices of the physical host from Dom0. The Dom0 is also home to other important files including the hypervisor, the kernel, the drivers (virtual disk and network) and the Xen management tools. The infrastructure providers' administrators need full access to Dom0 for system updates and maintenance of the infrastructure. This also allows them to modify the Xen hypervisor including its utilities if required.

Security Issues. Infrastructure providers can perform various attacks to access or modify the service providers' (thus, also service consumers') data. They can access data directly in the physical hosts, exploit vulnerabilities in the cloud customers' VMs, access data directly in the data storage, capture data using network sniffers, or use backup images to retrieve data. We focus on security vulnerabilities in the physical host where XCP is run by an infrastructure provider.

By directly accessing the physical host, an infrastructure provider can install malicious hardware, perform a side channel attack, or modify the system software running on it (Dom0 or DomUs). Modifications of the system software include the possibility to reset the machine and boot with a modified hypervisor or operating system tailored to enable silent modification of access to the service provider's data [23].

Modifications of hardware devices and drivers that perform DMA allow access or modifications to the memory without hypervisor control. DMA allows reads and writes to all the

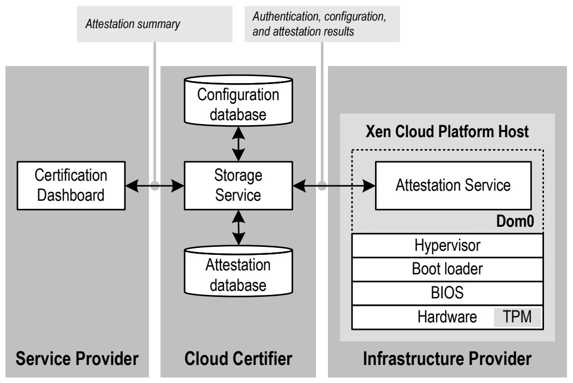

Fig. 2. BonaFides Architecture

memory and can be done by inserting a new hardware device, for example, a specially designed PCI card, or by misusing already available components (e.g. network cards). It is also possible to manipulate devices by modifications of the respective drivers. Considering that infrastructure providers have full control over Dom0 and are able to use specially tailored drivers, this type of attack allows them to capture disk content and network traffic of the DomUs.

Attacks on the system software (Dom0) include modifications of the hypervisor, kernel, or kernel modules. This potentially allows access to the memory of all VMs from cloud customers running on the physical host. Infrastructure providers can also modify the disk utilities and configurations to create a duplicate of the data stored by the service providers' VMs. Finally, modifications of the network utilities or configurations could allow redirection of all network traffic to a file.

While protection guarantees with respect to side channels, physical attacks to the infrastructure provider, and detection of vulnerabilities in the software and hardware components (e.g. using security testing) are important as well, we focus on attacks targeting modifications of the hardware and software that allow access or modification to service provider's data.

III. Design

Our goal is the remote assessment of the cloud infrastructure's integrity by a cloud certifier. We hence need to detect all changes in the remote system that can possibly compromise security. All changes in the hardware or software should be reported to the cloud certifier, even if the infrastructure provider has superuser access to the machine.

Our BonaFides system monitors the infrastructure provider's physical hosts by observing file modifications on a low level and persistently stores the history of these integrity measurements and file changes. Files are measured at regular intervals and whenever changes in the files are detected. BonaFides measures the hypervisor, kernel, kernel modules, disk and network utilities, and system configuration files in the Dom0. Depending on whether or not the BonaFides system is caught in an assumed DoS attack (§IV-C), we perform the measurements at one of two levels of granularity. In normal operation, we compute a hash value of the file after its

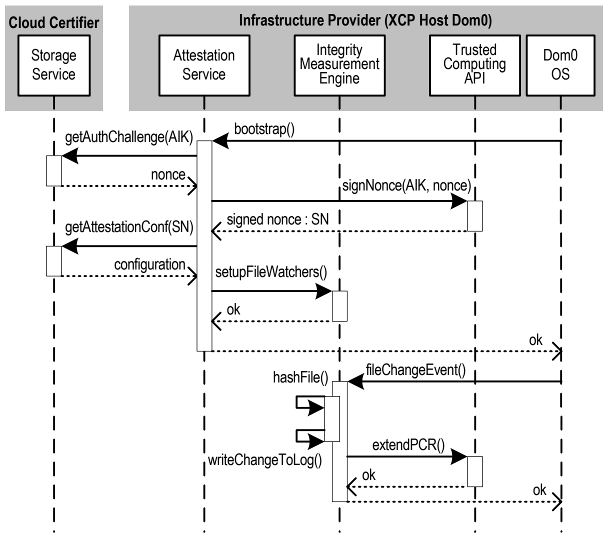

Fig. 3. Authentication, configuration, file changes

modification and store this value persistently. This information can later be used by the infrastructure provider to prove that the change has been legitimate and was in the best interest of the customer. This, for instance, is the case if updates were installed. In contrast, if the system is under heavy load, we may not be able to perform these computations of hashes for performance reasons. In this case, we persistently record only the fact *that* a file or a set of files have been changed. Upon forensic analysis, it is then up to the infrastructure provider to prove that the respective change events have been legitimate.

One problem when designing an integrity measurement and remote attestation solution is trusted boot as well as secure storage of authentication keys and measurement logs. In BonaFides, we make use of the Trusted Platform Module (TPM) proposed by the Trusted Computing Group (TCG) standard [29], as explained in §IV.

Figure 2 sketches the system structure. The Storage Service (SS) informs the Attestation Service (AS) about the hardware and software components of the XCP infrastructure (run by the infrastructure provider) to be attested. The measurements are the task of the AS. To do so, it configures the Integrity Measurement Engine (IME) using the input from the SS. The IME is an internal component in the AS that performs integrity measurements when file changes are detected (the IME is not shown in the figure). The SS retrieves the integrity measurements by querying the AS on a regular basis, or whenever changes are detected and signaled by the IME.

The SS stores the attestations in a database for auditing purposes. It notifies the Certification Dashboard when undesired changes in the hardware or software are detected. Undesired changes in the XCP are changes of the hardware/software that possibly allow unauthorized access to the service provider's data. The cloud certifier must specify the requirements for the cloud infrastructure, for instance, that a specific version of the hypervisor must be in place or that access to DMA devices should be made only trough certified drivers.

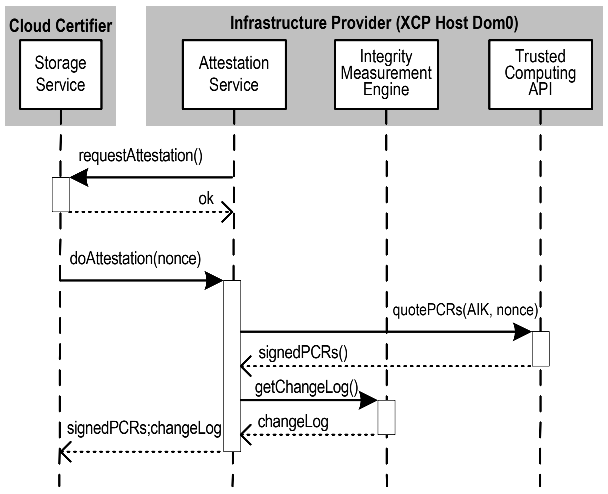

Fig. 4. Attestation request and execution

Figure 3 shows the bootstrapping of BonaFides and the operations executed when a file change is detected. Bootstrapping the AS component is triggered by the Dom0 operating system (OS); this also initiates the authentication and configuration request operations with the SS component. The authentication is a public key authentication where the AS requests a nonce that is signed with a key protected by the TPM chip. The signed nonce guarantees that replay attacks are not possible and is used by the AS to request the attestation configuration from the SS. Upon reception of the attestation configuration, the AS checks the integrity of all files and configures the IME, which sets up file change watchers. These file change watchers are configured using an OS kernel module in Dom0 that notifies the IME whenever a watched file is modified. Upon detection of a file modification, the IME receives a respective event and initiates the integrity measurement operation. This operation consists of (1) generating a log entry that is hashed and (2) updating an in-memory log file that records the changes. The IME stores the hash of the log entry in the TPM chip (§IV-A), which allows for future integrity checks.

Figure 4 shows the attestation request and execution operations. The SS performs an attestation execution operation at configurable periodical attestation intervals. The attestation operation consists of retrieving integrity measurements from the TPM chip and the change log (which is cleared after each attestation). The SS uses the integrity measurements from the TPM to verify that the change log has not been tampered with. The attestation request operation shown in the upper part of the diagram is optional. It can also be triggered by the AS when a threshold of *N* file changes is reached. We call this an *alarm group*, which is a parameter that can be configured by the cloud certifier for each file being monitored. Files in alarm group *zero* do not trigger an attestation request by the AS; files in alarm group *one* trigger an attestation request whenever one file change in any of the files of the group is detected; files in alarm group *N* trigger an attestation request whenever N changes are detected. The alarm group option increases the response time of the cloud certifier if undesired changes happen between attestations. Typical files to be configured as group *one* include the kernel modules and device drivers that can be loaded at runtime and possibly allow instant direct memory access. Changes in files such as the kernel image and disk configuration files that are effective upon the next booting of the system can be part of a different alarm group.

We expect BonaFides to be implemented and configured by the cloud certifier introduced in §II. In specific scenarios, BonaFides can also be implemented and used directly by service providers. However, we foresee cloud certifiers to be globally trusted public key infrastructure providers or government agencies specialized in market protection.

IV. Implementation

A. Trusted Boot

We implemented authentication and trusted boot support on the grounds of the TCG proposal [29]. The TCG specifies a trusted boot procedure where the different software components that assume control during the boot process are measured using a hash algorithm. The hash values of these components are stored in a tamper-proof chip called the Trusted Platform Module, or TPM. TPMs store hash values in so-called Platform Configuration Registers, or PCRs. During the boot process, PCRs are *extended* and can not be directly written to or reverted to a previous state. The extension operation consists of the concatenation of the current content of a PCR with a TPM-external given hash value, computation of the hash (SHA-1) of the concatenated value, and storage of the resulting hash value instead of the old PCR value. This feature guarantees that once a hash value of a software component has been extended in a PCR, it is impossible to revert it to the previous value. The values stored in a PCR can be retrieved together with a signature that proves the PCR values are in fact the values stored in the TPM chip. The authenticity of the TPM chip is guaranteed through a certificate provided by the TPM manufacturers (that we assume to be trusted).

Trusted boot and remote attestation support of the TCG also includes pre-OS integrity measurements that are performed by the BIOS and boot loader of the OS. This makes it possible to verify that hardware and system software configuration have not been modified. The boot loader and its configuration are verified prior to execution of the BIOS. The boot loader then verifies if the Xen hypervisor, XCP kernel, kernel modules, and the binaries and configuration files required to start the AS including the libraries the AS depends on (especially the trusted software stack to communicate with the TPM) are correctly configured. The key used for the authentication is sealed in the TPM using the values of the PCRs from the pre-OS environment. By sealing the authentication key, we guarantee that the attestation service is not able to retrieve the authentication configuration if the PCR values extended by the pre-boot environment do not contain the exact same values. The values may not match if the pre-OS is modified, for example, by updating the BIOS in the physical host. In case the values do not match, the storage server needs to perform an attestation and analyze if the new PCR values correspond to a trustworthy configuration. If the new PCR values are legitimate, the attestation service generates a new key that will be accepted from that point on.

B. Attestation Configuration

After trusted boot, the AS requests the attestation configuration. This configuration consists of a list of files and respective alarm groups that are to be attested and monitored by the IME. It also includes several configuration parameters that we explain as we introduce them. The list of files contains paths to standard files, folders, or executable commands. The information that is monitored differs depending on the type of the configuration entry. For standard files, the file content is monitored; for folders the folder listing; for commands the output of the command execution as well as the code of the commands themselves (see §IV-C). Commands come as binaries or scripts; examples of command executions include diagnosis tasks, e.g., to list the loaded kernel modules or to list the active network file systems. Execution and hashing of the command output is done by the AS only when an attestation is executed. We limit the list of commands that can be executed by the AS using a whitelist-based approach, thus preventing a corrupt SS from executing arbitrary security-critical commands on the machine that runs the AS.

C. Integrity Measurement Engine

The IME monitors the integrity of files and guarantees that all changes to relevant files in the system are detected and reported. To this end, it subscribes to a special kernel module, the inotify module [14]. The inotify module notifies the IME about all file change events related to files, folders, and commands listed in the configuration. The IME also subscribes to change events for commands because we need to monitor changes to the command scripts or binary files. This is necessary because, if we attest the output of the command, we must also ensure that the integrity of the command itself is respected. For each received file change event, the IME generates a log entry that is hashed and extended in a PCR in the TPM module. A log entry consists of the index of the file changed, a change counter, the timestamp when the change was received, the timestamp when the change event was processed, and the hash of the changed file's content.

Denial of Service. One limitation of the inotify kernel module is that the event buffer has a limited size. In case this buffer is full, file change events might be overwritten. This gives rise to a Denial of Service (DoS) attack: The infrastructure provider's administrator floods the inotify kernel module with file change events, thus becoming able to modify a set of files without being noticed. A possible attack is a superuser script that overwrites one byte in each of the monitored files with its original content, hence not changing the hash of the file. If a malicious infrastructure provider manages to write and execute this script, and overloads the inotify file change queue, the IME might use too much system memory and CPU to hash files and keep the log of file changes. Such an attack can be detected by counting the number of file change events. Since too many file change events should not occur in normal operation, we could simply scramble the PCRs by extending zeros and leave it to the infrastructure provider to justify these changes to the cloud certifier. This simple approach, however, makes later forensic analysis difficult because no further useful information is stored in the PCRs. In the remainder of this paragraph, we describe a solution that records some more useful forensic information when a DoS attack occurs. We consider this kind of DoS attacks particularly relevant in our approach because it is the only possibility for an attacker to modify files in Dom0 without being noticed.

Internal Queue. To counter this threat, we have implemented an internal buffer in the IME that directly dequeues all elements of the inotify queue and thus allows us to capture all change events. If two events for the same file are received, only the newest event is kept in the queue, and a counter is incremented for that specific file. This reduces memory overhead and also enables detection of unsafe states: If a file change event is in the queue to be processed and one more change happens, this indicates that the file was in a non-measured state during a specific time interval. The time difference between the first event received and the time the change event is processed represents the length of the time interval the watched file was in a non-measured state. This time interval is also written to the log. Since we do not enqueue changes twice for the same file, the size of the internal buffer is equal to the number of files being monitored.

If a change event is in the queue and the change counter is not greater than one, it is of no significance how long it takes to process the file change event. In this situation, we are sure that we are hashing the one and only version of the changed file. However, if the change counter is greater than one for unprocessed change events—meaning that two modifications have been performed in a small time interval—we can not be sure about the previous states of the file when the change event is processed. Our approach is to record this fact in the log for further investigation; the infrastructure provider will have to provide evidence that these changes have been legitimate.

Virtual PCR. In addition to a possible overload in terms of processing power and memory, the IME is not able to hash and extend a PCR in the TPM if the number of change events is too large. We have done performance measurements on three different systems to assess the throughput of a TPM to extend PCRs. The average time for one TPM PCR extend operation computed over 100 executions is .49 seconds for a Windbond TPM 1.1; .084 seconds for an Infineon (IFX) TPM 1.2; and .04 seconds for an Atmel TPM 1.2. If we ignore the time it takes to detect and process a file change (e.g. computing a file hash), the maximum throughput we can achieve with the fastest TPM we have measured is hence roughly 28 PCR extend operations per second. An attacker is able to generate file change events at a much faster rate (§VI). If too many changes are detected, the queue will hence take a long time to be processed. However, when file change events are not processed sufficiently fast, an attacker has a rather large time window to possibly circumvent the system by corrupting the in-memory log of file changes. Fortunately, as long as there is no vulnerability in the system that would explicitly make this possible—of course a strong assumption—this can be excluded: it is possible to ensure that only the process which owns the memory region, and not even the kernel, can modify it. We discuss this in §VII. Because the length of the internal queue is bounded by the number of files to be measured, it is unlikely that the queue uses too much physical memory in the machine to store the change events.

The IME strategy in case of overload is to continue processing the file change events in the queue while a PCR extension operation is being executed in the TPM. The log entries for the change events continue to be written to the in-memory log. Now, in the overload situation, in order to ensure integrity of the log, we extend a *virtual* PCR image with the hash of the log entry rather than the real PCR [27]. Like the physical PCR extension, the virtual operation consists of overwriting the virtual PCR with the digest (sha1) of its current content appended to the hash to be extended. Because this computation is performed by the CPU and not by the TPM, performance problems are not an issue. The virtual PCR image is an image of the real PCR value received from the TPM in the last extend operation. When the TPM finishes executing the PCR extend operation, the IME extends the real PCR with the updated value of the (virtual) PCR image and overwrites the in-memory PCR image with the new PCR value returned.

This strategy guarantees that all file changes processed (and logged) by the IME while the TPM is busy extending a real PCR are chained using the in-memory PCR image. To corrupt our in-memory log, an attacker has to search in the memory of the IME process for the PCR image value (which cannot be prepared in advance), overload the IME with change events to trigger the in-memory PCR extension, and overwrite the PCR image value at precisely the moment after the last change event has been generated, but before the IME extends this value in the real TPM. Moreover, the TPM device is single-threaded and does not allow access to PCR values by other processes during an extend operation. This gives the IME an advantage in guaranteeing the integrity of our log even if the attacker manages to read and write to the IME's memory log (which can be excluded; see §VII). Note that it is also not possible for an attacker to reproduce the TPM PCR extensions and predict the current value of the PCR in the TPM. This is because the log entry that is hashed and extended includes fine-grained timestamp information that is practically impossible to predict. The attacker would need to know the timestamp with a nanosecond precision when the file change event generated was received and processed by the IME.

The virtual PCR image hence guarantees the integrity of our in-memory log considering the limited throughput of the TPM device. However, if too many file changes are generated, the IME is not able to compute the hash of the file content for all changes in the queue (which is done by the CPU). In these cases, the IME does not hash the file content for each file event in the queue but only logs the number of changes without the hash of the file content. The referenced file is marked with a flag which indicates that the file should be hashed later. When the inotify queue is empty, the IME works on the backlog: it hashes and logs all the files that have been marked, until new file change events are enqueued again. This approach prioritizes the logging of the *number* of file changes over the *content* of these changes, which is useful to indicate that a possible DoS attack took place. We have explained above that these cases must be justified by the infrastructure provider.

Log size. The IME supports a configuration parameter to limit the in-memory size of the change log. If the size of the log is larger than this value, the file change event is not recorded. Instead, the system enters panic mode and the PCR value is scrambled. We consider this a critical situation where nothing can be guaranteed anymore. In this case, BonaFides cannot even provide information about which file was changed but rather that too many changes have happened. This, in turn, will have to be explained by the infrastructure provider.

D. Tamper Detection

The AS can detect malicious operations from infrastructure providers. We have implemented a tampering detection mechanism that detects when the superuser of the Dom0 tries to sandbox the AS using *ptrace* [17], reduce the priority of the process, or change the apparent disk root of the process (chroot). If this is detected, BonaFides scrambles the PCR values using random numbers. This tells the cloud certifier that something went wrong. The only way of having the AS operational again is to reboot the physical host.

V. Example Revisited

To illustrate how BonaFides helps secure a cloud infrastructure, we revisit the example described in §I where a service provider *S* runs its service on the hardware of an infrastructure provider *I*. Now we present four examples in which *I* wants to steal the data of *S*'s customers (i.e. *C*'s data).

Modified hypervisor or hardware. In the first example, *I* changes the master boot record (MBR) to load a malicious hypervisor like the Blue Pill hypervisor [22]. This hypervisor allows arbitrary modifications of the system's memory and raw access to hardware devices such as hard disks and network cards. It is very hard to detect, but to load it during system startup, the MBR must be modified. This MBR modification is detected by the SS as the BIOS measures the MBR and extends this value into the PCRs. The SS immediately learns that the system has been tampered with because the AS is unable to use the authentication sealed to the pre-OS PCRs. In the second example *I* exchanges the file containing the hypervisor with a version that allows *I* to read arbitrary memory addresses and thus gains direct access to the VM's memory. During the boot process the boot loader measures the file of the hypervisor and extends the measurement into the PCRs. The SS immediately knows that the system has been tampered with because the sealed authentication key can not be used. The same tampering detection can be used when inserting a hardware device to access arbitrary memory locations and thus the VM's memory;

possibly also changing the kernel or one of its modules in order to gain access to the stream of data to/from the disk.

Modified configuration. Our third example consists of *I* changing the configuration file of a common virtual switch, Open vSwitch, that allows *I* to mirror a VM's network port in the internal network bridge. *I* can thus capture and inject data in this channel. The Open vSwitch configuration file is in the highest alarm group because there is the direct possibility of a data leak if this single file is changed. Once *I* changes the Open vSwitch configuration file, the AS is notified by the inotify kernel module, and processes the change by measuring the file. It logs the change and extends the hash of the log entry to the PCR. An attestation is immediately requested from the SS because of the highest alarm group. Moreover, the cloud certifier is notified by the SS as the configuration of Open vSwitch does not correspond to the expected configuration.

Modified commands. The last example shows how *I* can modify the XCP to steal *C*'s data by changing the binary of the disk utility tapdisk. By doing so, *I* can duplicate the stream of data to/from the disk as tapdisk manages the requests of the VMs for data to be read from/written to the storage. Because of its criticality, the cloud certifier is likely to include this file into the configuration of the AS in the highest alarm group. Once *I* modifies the binary, the AS is immediately notified of the change by the inotify kernel module and processes it by measuring the file and creating a log entry, which is once more hashed and extended to a PCR. An attestation is immediately requested from the SS, which is able to determine that the binary has been tampered with.

Conclusions. By adopting BonaFides, a cloud certifier *C* can provide a service provider *S* with guarantees that the hypervisor, Dom0 kernel, and Dom0 kernel modules are indeed the ones expected to be running in *I*'s physical hosts. Furthermore, all changes in *I*'s software infrastructure are detected and reported to *C* through the remote attestation procedure. Consequently, *C* informs *S* about changes that might have had an impact on data protection policies required by *S*. We emphasize that we do not protect against stealing or modification of data itself but make it possible to detect this.

VI. Performance

BonaFides impacts resource consumption for hashing files and resource consumption for the IME to process file change events. The communication overhead is negligible. To assess the impact, we performed a set of experiments on an Intel Core 2 Quad Q9650 processor with 8GB of system memory, a TPM chip, and a WD Velociraptor hard disk running Ubuntu with a 2.6.32 Linux kernel.

Computation of Hashes. Our experiments suggest that, not surprisingly, the hashing time is linear in the size of the files and that caching has an impact on hashing. In our prototypical implementation, we manage to hash 40 files with 40 MB each in about 5 seconds CPU time, 160 files with 40 MB each in about 20 seconds. At this point, in-memory caching becomes impossible; 210 files with 40 MB each require 40 seconds, 300 files with 40MB each require 60 seconds. Depending on the size of the attested files, this linear overhead may turn out to be problematic. For directories we measured the attestation of 1 to 300 directories containing 40 files and found a linear correlation with respect to CPU usage, but had fluctuations in the CPU time, which we also attribute to caching and recursive reading of directory contents. The attestation of 300 directories consumed 180 ms. We observed the same results when measuring the time it takes to execute an attestation of 300 commands each producing 400 bytes of output.

In sum, measuring files is the main part of the CPU usage when performing an attestation. The time to measure directories and commands is negligible. Thus, when setting up BonaFides, the overall size of all files to be attested is relevant. Similarly, it is important to know the time it will take for an attestation. We conclude that the AS memory requirement is equal to the overall estimated size of all files to be attested.

We illustrate the performance impact with an example. Suppose that a cloud certifier wants to attest an XCP Dom0 physical host and the installed Oracle Java Runtime Environment. The disk size for a standard XCP installation is approximately 785 MB, with the average file size below 1 MB. The complete JRE consists of 700 files, also with less then 1 MB average size per file. The XCP boot loader, boot configuration, hypervisor, kernel, kernel modules, and the JRE system library dependencies consist of approximately 2300 files. In our experiment, hashing and attestation of all these 3000 files, with a total size of around 300 MB, does not take more than 2 seconds. We may conclude that our system can be used to remotely attest the integrity of realistic infrastructures.

Event Processing Overhead. To examine the implications of the DoS attack described in §IV-C, we created 700 files with 1 MB each in one folder. In the experiment, we simulate an attacker who keeps the system as busy as possible. To do so, we apply the smallest possible change—one byte—ten times to every file, one after the other, and restart after all 700 files have been changed. In two separate experiments, we selected this byte randomly, or always picked the first byte. The results confirm the intuition that when always changing the first byte, the system caches this byte after the first loop. In contrast, when changing a random byte, the system needs to re-access the disk in every loop, thus making the attack significantly slower. To double check, we also performed this experiment on a RAM disk. As expected, in this case there is no difference when using either method since both the random and the first byte are already in the system memory.

For 700000 file changes (100 loops), Figure 5 shows the length of the queue over time when using a RAM disk as underlying storage medium. The queue never exceeds 700 entries which is the number of files in the configuration. In the beginning, the queue is empty, then quickly fills up to the maximum. This is because the changes are coming in at a fast and steady rate. Then, processing of events (that is, PCR extension) starts. Execution of this thread, which is dequeuing the events, causes the fluctuations visible in the figure. The burst of file changes ends after 1.5 seconds, causing the queue to be completely processed shortly afterwards, and starting to

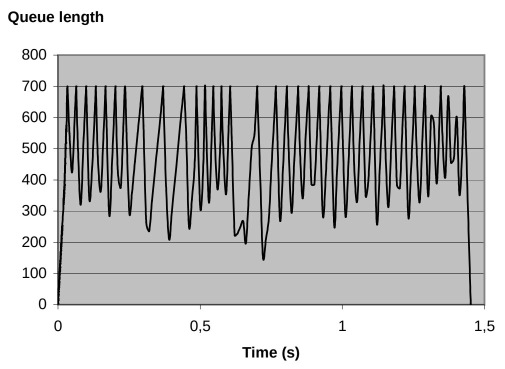

Fig. 5. Queue length, DoS attack on RAM disk

hash all files that were changed. The same experiment on a physical hard disk yielded qualitatively identical results.

Conclusions. We conclude that BonaFides can handle DoS attacks both on RAM and hard disks with file changes at a maximum rate. BonaFides is able to capture the number of changes as well as the final state and extend the PCRs. Space restrictions forbid a detailed description, but we have conducted experiments to find out that we would still need to use our approach with the internal queue to capture all file change events if the TPM chip was 1000 times faster.

VII. Security

We can guarantee that BonaFides monitors and reports all changes to a set of files, folders, and commands to the cloud certifier. We do not overload the system memory in case of a DoS attack. Even in the presence of this type of attack, all changes in the monitored files are detected and reported. The load of the system depends on the configuration parameters and on the size of the attested files.

By monitoring the kernel, kernel modules, network and disk utilities and configurations we guarantee that the infrastructure provider is not able to perform raw access to the memory through software, and to access the disk and network data. Considering that BonaFides relies on trusted computing support to verify the integrity of hardware devices and firmware capable of DMA, BonaFides inherently provides this guarantee. From our performance measurements in §VI, we may conclude that this can be done for realistic infrastructures.

In order to provide these guarantees we must assume that the infrastructure provider will not attack the machine physically using side channels, cold boot, or platform reset attacks. A malicious infrastructure provider is able to physically remove a memory card from the physical host and read information from it in another computer [9]. The TPM chip is also vulnerable to a platform reset attack where the TPM chip is cleared without resetting the whole machine; a skilled attacker could theoretically manipulate the PCR values [4].

We need to assume that if external storage is used by the infrastructure provider, the data is encrypted with keys that are stored in the service providers' VMs. In the trusted boot process we must assume that BIOS and boot loader are configured to verify the pre-OS boot environment (which is part of the BonaFides system). We have to assume that none of the hardware and software components contain vulnerabilities that would allow the infrastructure provider unauthorized access to the service providers' data (a problem shared by all solutions that use the TPM). We have to assume that the TPM chip has been initialized, that ownership has been taken by the cloud certifier, and that we can trust the TPM manufacturer.

To avoid direct memory access by the superuser from user space, and consequently, modifications to the running BonaFides system including the in-memory log, we assume that the Dom0 kernel is properly configured. We assume it is compiled with option NONPROMISC_DEVMEM enabled which restricts direct access to the physical memory from user space through /dev/mem. A Linux kernel compiled with this option enabled only allows user space access to specific memory areas used by PCI devices and the BIOS code, which is required by some specific applications (e.g. X server). Access to all other physical memory areas is forbidden.

Using loadable kernel modules (LKM), the superuser can overwrite the interrupt table; overwrite the system calls table; patch the running kernel code or any memory region; modify the network stack; and manipulate kernel internal structures [18], thus circumventing BonaFides. To prevent LKM attacks we assume that Dom0 does not allow loading of dynamic kernel modules or that a control mechanism for this functionality is in place. An example control mechanism available for the Fedora Linux distribution is a kernel module that implements a white list functionality [28]. Only the kernel modules that are in the list of names of allowed kernel modules can be loaded. However, no signatures or hashes for integrity checks are used. If this control mechanism is in place, BonaFides can monitor this list and the kernel modules themselves to guarantee that only trustworthy kernel modules are loaded.

To avoid unloading of the inotify and TPM device driver kernel modules, which are used by the attestation service, we assume that the Dom0 kernel is compiled with the option MODULE_FORCE_UNLOAD disabled. This prevents the superuser from removing kernel modules that are in use. We also assume that a guardian kernel module is loaded at startup to prevent the Trusted Computing Daemon and AS from being terminated or stopped, or have their priority reduced (nice value) by the superuser [19], [21]. The AS component has built-in functionality to detect its process priority.

Infrastructure providers are likely to create backups of the customers' VMs (which itself must be allowed by BonaFides). Using customer encryption keys, the storage areas of these VMs can be considered secure. We have to assume that the backup servers are recursively secured by BonaFides. Infrastructure providers can also suspend consumer VMs and write them to disk. BonaFides can monitor the (by design fixed) folder in which the suspended VMs are stored. If the VM images are copied from this location and thus leave the realm of BonaFides, no guarantees can be provided; but the infrastructure provider will have to justify where the copies went. Akin to suspension, VM images are sent to a network socket whenever the VM is migrated. In this case, we have of course to assume that the receiving end is protected by BonaFides as well; once more, we can detect *that* a migration has taken place. This puts the infrastructure provider in a position to prove that the migration was legitimate.

If BonaFides itself is not implemented correctly or exhibits vulnerabilities, we cannot give any guarantees. We hence assume that our implementation is correct and not vulnerable. Moreover, we assume that the complete software stack on which BonaFides relies (e.g. C compiler and libraries) does not contain vulnerabilities, and that all communication between storage server and attestation service is encrypted and the keys are protected using the TPM chip. Note that integrity-based security measures, by definition, do not make any statement about the functional correctness of the measured artifacts.

BonaFides does not provide guarantees with respect to the behavior of the attested system. It is possible to guarantee that a binary is present, and that an initialization script executes this binary file at boot time. However, at the current stage of development, it is not possible to guarantee that the binary file will successfully start and continuously run when executed by the initialization script. A primitive support to check if a specific process is running could be added, using a command configuration option that attests the output of the *ps* command.

VIII. Related Work

Schiffman et al. [26] describe a system for integrity measurements of VMs. In their work, the trustworthiness of the machines depends on the input given to the VMs for processing, not on actions taken on the physical host. Thus, the possibility of a malicious insider modifying the physical host is not addressed. Quynh and Takefuji [20] describe a real-time integrity monitor for Xen VMs addressing functionalities that are not covered by existing file system integrity measurement tools. In their work the assumption is that Dom0 is trusted to perform and store integrity measurement reports. Jansen et al. [10] present a solution to the problem of attesting VMs by deploying security services in Dom0. This enables the secure creation and execution of VMs, including the ability to attest VMs at runtime. An essential part of the solution is the TPM, which enables secure storage of security policies in Dom0.

Santos et al. [16] focus on integrity measurements and attestation of physical hosts when VMs are migrated, which guarantees that physical hosts are trustworthy only at the time of migration. Furthermore, by adopting Terra [7], attestations of the physical host are also made upon request by using a trusted VM monitor (TVMM) as a hypervisor. The TVMM provides isolation and separation of VMs plus additional sealed storage for the attestation values using the TPM. The hypervisor has the responsibility to provide the hardware level attestations of the system in a trusted way to software running in one of the VMs.

Descher et. al. [5] proposes to use encrypted VMs in encrypted partitions (EP) to increase security. Upon execution on a physical host, the VM owner is given access to the boot system (BS) and has to provide a decryption key for the EP in order to start the VM. The access to BS is protected using public-key cryptography where the physical host is only given a public key to install in the BS. All other keys received by the BS (including the key to decrypt EP) are only stored in volatile memory thus ensuring that potential intruders need "substantial control over the infrastructure."

The Integrity Measurement Architecture [25] is a general purpose solution that checks the integrity of system components (kernel, kernel modules, system libraries, etc.) by computing hashes of system files at boot time. In the context of grid solutions that rely on virtualization, Löhr et al. also measure the integrity of system components at boot time [13] to guarantee that only nodes in a safe state are able to decrypt jobs submitted to them. Haldar et al. [8] and Sadeghi et al. [24] propose to attest system properties instead of hashing binary system files. The attestation of system properties is possibly better in case of system updates, however, the problem of specifying and assessing sound and relevant system properties is not extensively addressed in their work. In contrast to all existing approaches, we propose to measure the integrity of the system components not only at boot time but also at runtime. In a recent overview of attestation in trusted computing, Lee-Thorpe identifies this as one of the open problems [12].

Berger et al. [1] and Krautheim et al. [11] propose virtual TPM (vTPM) devices to prevent virtualization customers from sharing the same TPM in a physical host. England and Jork [6] propose a para-virtualized solution where one real TPM device is shared among virtualized hosts. According to Krautheim et al. their solution is preferable because it is the only one that provides complete support to all TPM functionality. These approaches to TPM virtualization and sharing are fundamentally different from the work described in this paper. Their focus is on the provisioning of trusted computing services to guest VMs rather than on attestations of the host platform.

Some companies and research studies [2] have indicated their intent to follow a similar direction as we have done to enable trust in cloud infrastructures. Intel, VMWare and RSA announced [3] that they are collaborating to build the Data Loss Prevention Enterprise Manager to measure and monitor cloud infrastructure security. This system uses the Intel trusted execution technology CPU extension, RSA software components to collect the data, a dashboard for security evaluation, and a component to prevent data loss. We are not aware of technical details or implementations of this product.

Delta. In contrast to these approaches, our system is designed to be flexible and independent from operations taking place in the system. It records the trustworthiness of the host system by performing measurements at boot time, and also afterwards whenever a preconfigured change event happens including its measurement in a trustworthy log. Attestations are performed periodically, on demand, or after a specified number of files has been changed. Thus our solution works with a standard hypervisor and kernel and does not require a special implementation of the hypervisor or changes of the kernel. Moreover, our approach works in a distributed scenario and needs only one service running in the administrative domain of each physical host, plus a central service for the attestation collection that runs separated from the cloud.

IX. Conclusions

We provide a solution to the problem of malicious infrastructure providers stealing or modifying the service providers' or service consumers' data. Our system can remotely monitor and attest the integrity of crucial system files, thus filling a gap in the Xen Cloud Platform and other Linux-based cloud operating systems. We have discussed assumptions and security guarantees that we can provide.

Our system does not ensure data protection. Instead, it serves as a tool to assess the integrity of the infrastructure provider's hardware and software. The data that it creates in a tamper-proof way is supposed to be used for forensic purposes: the infrastructure provider must—and, sometimes more importantly, can—prove that all recorded change events have been legitimate. It is up to the cloud certifier to identify the specific files and configuration parameters that must be in place and that guarantee specific security properties.

BonaFides complements existing intrusion detection systems of infrastructure providers because it yields information on whether or not systems are compromised, and which sensitive files were modified. This strengthens protection for the service provider because an attack not originating from the infrastructure provider can now also be detected.

While our work is based on the XCP, BonaFides can be used in other environments such as VMWare ESXi without major changes in the general architecture and concepts. For example, the SS component provides a generic database for specification of integrity measurement configurations considering that different cloud solutions could be used.

In addition to whether or not the assumptions in §VII are realistic, there is a number of limitations and open questions. We have not yet implemented a solution for the situation when backups or migrations within the infrastructure providers' cloud take place (§VII). Moreover, we have not yet investigated possible approaches to remotely deploying BonaFides. We believe that new infrastructure providers can ship BonaFides with new machines and pre-owned TPMs. In existing infrastructure providers the installation and ownership procedure of the TPM device would require physical presence or remote serial access. Integrity can only be guaranteed in this case if the existing machines are inspected to ensure that no malicious software is running when BonaFides is installed.

Acknowledgment. C. Moucha provided valuable technical advice. This work was supported by FhG Internal Programs Attract 692166 as well by the EU-funded IP MASTER.

References

[1] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn. vtpm: virtualizing the trusted platform module. In *Proceedings of the 15th conference on USENIX Security Symposium*, 2006.

[2] R. Chow, , P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW, 2009.

[3] S. Curry et al. Infrastructure security: Getting to the bottom of compliance in the cloud. http://www.rsa.com/innovation/docs/CCOM_BRF_0310.pdf, Mar 2010.

[4] Dartmouth College PKI/Trust Lab. Tpm reset attack. http://www.cs.dartmouth.edu/~pkilab/sparks/, Sep 2010.

[5] M. Descher, P. Masser, T. Feilhauer, A. M. Tjoa, and D. Huemer. Retaining data control to the client in infrastructure clouds. *International Conference on Availability, Reliability and Security*, 2009.

[6] P. England and J. Loeser. Para-virtualized tpm sharing. In *Proceedings of the 1st international conference on Trusted Computing and Trust in Information Technologies*, Trust '08. Springer-Verlag, 2008.

[7] T. Garfinkel, , B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: a virtual machine-based platform for trusted computing. *SIGOPS Oper. Syst. Rev.*, 37(5), 2003.

[8] V. Haldar, D. Chandra, and M. Franz. Semantic remote attestation - a virtual machine directed approach to trusted computing. In *USENIX VM Research and Technology Symposium*, 2004.

[9] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *Proceedings 17th USENIX Security Symposium*, Jul 2008.

[10] B. Jansen, H. Ramasamy, and M. Schunter. Flexible integrity protection and verification architecture for virtual machine monitors. In *Proceedings Second Workshop on Advances in Trusted Computing*, August 2006.

[11] F. Krautheim, D. Phatak, and A. Sherman. Introducing the trusted virtual environment module: A new mechanism for rooting trust in cloud computing. In *Trust and Trustworthy Computing*, volume 4734 of *LNCS*. Springer Berlin / Heidelberg, 2010.

[12] A. Lee-Thorp. Attestation in Trusted Computing: Challenges and Potential Solutions. Technical report, Royal Holloway, University of London, 2010.

[13] H. Löhr, H. Ramasamy, A. Sadeghi, S. Schulz, M. Schunter, and C. Stüble. Enhancing grid security using trusted virtualization. In *Autonomic and Trusted Computing*, volume 4610 of *LNCS*. 2007.

[14] R. Love. Kernel korner - intro to inotify. *Linux Journal*, Sep 2005.

[15] P. Mell and T. Grance. The nist definition of cloud computing. Technical report, National Institute of Standards and Technology, 2009.

[16] K. P. G. Nuno Santo and R. Rodrigues. Towards trusted cloud computing. In *Proceedings of the Workshop On Hot Topics in Cloud Computing (HotCloud), San Diego, CA*. MPI-SWS, October 2009.

[17] P. Padala. Playing with ptrace. *Linux Journal*, 103, November 2001.

[18] R. S. Pelaez. Linux kernel rootkits: protecting the system's "ring-zero". http://www.sans.org/reading_room/whitepapers/honors/linux-kernel-rootkits-protecting-systems_1500, Sep 2010.

[19] Pragmatic and THC. Complete linux loadable kernel modules. http://packetstormsecurity.org/docs/hack/LKM_HACKING.html, March 1999.

[20] N. A. Quynh and Y. Takefuji. A real-time integrity monitor for xen virtual machine. In *Proceedings Intl. Conf. on Networking and Services*, page 90, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[21] M. Reiter. Exploiting loadable kernel modules. http://www.thedarkside.nl/honeypot/extra/expl_lkm.html, September 2010.

[22] J. Rutkowska and R. Wojtczuk. Bluepilling the xen hypervisor. http://invisiblethingslab.com/resources/bh08/part3.pdf, August 2008.

[23] J. Rutkowska and R. Wojtczuk. Virtualization (in)security. http://www.blackhat.com/html/bh-us-10/training/bh-us-10-training_jrk-virt.html, Sep 2010. Black Hat USA 2010 Weekday Training Session.

[24] A. Sadeghi, C., Stüble, and M. Winandy. Property-based tpm virtualization. In *Proceedings of the 11th international conference on Information Security*, ISC '08. Springer-Verlag, 2008.

[25] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *Proc. of the 13th conference on USENIX Security Symposium*, 2004.

[26] J. Schiffman, , T. Moyer, C. Shal, T. Jaeger, and P. D. McDaniel. Justifying integrity using a virtual machine verifier. In *IEEE ACSAC*, 2009.

[27] B. Schneier and J. Kelsey. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur.*, 2(2), 1999.

[28] M. Trmac. Modprobe whitelist. http://fedoraproject.org/wiki/Features/ModprobeWhitelist, Sep 2010.

[29] Trusted Computing Group. Tpm main specification 1.2, July 2007.

[30] VMWare. Vmware to collaborate with google on cloud computing. http://www.vmware.com/company/news/releases/vmware-google.html, May 2010.

[31] XenProject. Xen cloud platform. http://www.xen.org/products/cloudxen.html, May 2010.