Trusted Virtual Machine Management for Virtualization in Critical Environments

Khan Ferdous Wahid
Fraunhofer SIT
Rheinstraße 75
64295 Darmstadt
Germany
www.sit.fraunhofer.de
khan.wahid@sit.fraunhofer.de

Nicolai Kuntze
Fraunhofer SIT
Rheinstraße 75
64295 Darmstadt
Germany
www.sit.fraunhofer.de
nicolai.kuntze@sit.fraunhofer.de

Carsten Rudolph
Fraunhofer SIT
Rheinstraße 75
64295 Darmstadt
Germany
www.sit.fraunhofer.de
carsten.rudolph@sit.fraunhofer.de

Service providers use virtualization technology to better serve their remote customers and to efficiently use their resources. In particular when virtualization is used within critical infrastructures such as industrial control systems security of the virtual machines is crucial. Creating fully secure systems based on a verified small trusted computing base (TCB) is desirable to minimize the attack surface of the host system. However, attacks can still occur, and sometimes it is not practically possible to provide a small TCB or to completely replace a running system to enforce security. Thus, remote monitoring of the integrity of VMs is desired to confirm their trusted state. In general, it is a complex task to incorporate on-demand system integrity verification into the existing host system to measure a hosted virtual machine (VM) at runtime and to switch back at runtime to the trusted state whenever a change or a manipulation is detected. Also it is necessary to provide the host machine's integrity information along with the VM to remote customers when such status are seeked.
In this paper, we address the problem of securing an existing or new host machine with on-demand integrity measurement solution to offer a fresh and trusted VM whenever some illegitimate changes are detected in the current VM. The solution is targeted at smaller devices with a limited number of VMs and customers per device. It also assumes VMs to be rather stable and does not use virtual TPMs. Thus, it focuses on secure virtualization in critical environments, automation, or industry control systems.

Trusted computing, Virtual machine integrity, Security architectures

1. INTRODUCTION

The virtualization concept was initially introduced by IBM in mid 1960s (Karger (2005)). At that time computers were very expensive, so their proper utilization was necessary. The drop in cost of personal computers with x86 architecture diminishes the justification of virtualization at late 1980s, because multiple separate cheap machines could perform similar tasks more efficiently. Nowadays, the advanced processors bring enormous power in computing platforms and multitasking operating systems allow unlimited services. Due to this advancement in CPU power, cheap memory and storage, computing-as-a-service has been gaining a lot of popularity in recent years. Service providers are focusing on virtualization solutions to improve their resource utilization and management. To offer new services to customers, it is often necessary to utilize remote systems provided by third parties, where each third party can accommodate different virtual machines (VMs) from different service providers. For example, a cloud provider can accommodate dedicated VMs from different software service providers, where customers of those software service providers can access the VMs directly to do their desired operations. Similar scenario can be found in the industrial control system, where the control system provider presents on-site host machine to accommodate dedicated VMs from different service providers, so that the customers of those service providers can access the VMs directly to get necessary functionalities. When remote systems are provided by third parties (i.e., host machine providers), it becomes of the utmost importance to measure the integrity and enforce security at runtime. Otherwise, the privacy and security of the customers are at stake. As we have mentioned, these remote systems can already host different virtual machines from different service providers, so to integrate new security solutions that require modification of the host or guest operating

system will significantly reduce our flexibility or make it impossible to deploy. Hence, a novel Trusted Virtual Machine Management solution inside a host machine that can be deployed in commodity computers (i.e., commodity hardware and software) without replacing the existing system is necessary. Besides, it should support easy installation on a new machine. These also hold for architectures used in industrial systems, where the only extension would be to add a hardware security module. Running prototypes use ARM architectures with a I^2C trusted platform module.

In this paper, we mainly focus on the integrity (or trusted state) of the host machines and their stored virtual machines. We show how easily we can use the commodity computers to build a complete trusted host machine with remote verification capabilities, so that different service providers or customers can check their virtual machines on-demand to get assured of the trusted state. If they detect an anomaly, the host machine can offer a fresh and trusted VM, when approved from the owner of the VM. Here, we use open source technologies to completely rely on commodity computers and to allow our architecture to be integrated flexibly.

2. MOTIVATION

There are a plenty of security solutions available to demonstrate secure environment for Virtual Machines. For example, Terra (Garfinkel et al. (2003)) is a stand-alone hypervisor architecture that allows individual VMs to run securely without interfering each other. It also supports attestation feature to establish secure communication channels with another party. Self-protecting modules (in Strackx et al. (2010)) require hardware modification to support isolation between subsystems in a single system. Overshadow (Chen et al. (2008)), CHAOS (Chen et al. (2007)) and sHype (Sailer et al. (2005)) also provide their own hypervisor to separate processes or virtual machines. To implement any of these solutions, we need to either replace the existing system or change the hardware, which does not fulfill our goal to deploy Trusted Virtual Machine Management solution without replacing the existing system.

On the other hand, the Linux Integrity Measurement Architecture (IMA (n.d.)) was developed by IBM to provide an open source integrity subsystem, which includes trusted computing components, like extending platform configuration registers (PCRs) with file measurement, and remote validation of measurement list with Trusted Platform Module (TPM) signed PCR values etc. The IMA is available in mainstream Linux since Kernel version 2.6.30. So we only need to recompile the Kernel if the existing Linux version is older than 2.6.30 or IMA is not enabled. While in user space, it measures each file before reading or executing, and extends the PCR and measurement list if it detects any change. This mechanism is good for a stand-alone system, but in host machine provider, where many individual virtual machines are present, it can cause a chaotic situation, because the host machine needs to inform all the service providers each time there is a change in file. Also the request of a customer for the remote verification can reach the host machine immediately after a change in file, which triggers remote validation mismatches. In that case, the customer needs to contact the service provider for the correct measurement list. At that time, the service provider may not get informed by the host machine for that specific change of file, which causes unnecessary delays in operation. Moreover, one service provider should not get information about another service provider's virtual machines, because they are totally isolated for privacy, security and rivalry. Linux-IMA's functionalities extend the PCR whenever there is a change in a virtual machine (because it is a device file), so it is totally impossible for other service providers to validate the measurement list without knowing the information of that virtual machine. So we need a novel solution to overcome these issues, and this motivates us to our main contribution of this paper -

Convert an existing or build a new host system to support efficient on-demand remote verification of the host machine plus a specific virtual machine. Also provide mechanism to offer fresh and trusted VM when a single breach is detected.

To support this in our solution, the status of the host machine should be static, so that the integrity measurement list of the host machine does not change (e.g. by dynamic changes in the filesystem such as in /var). This reduces communication overheads and delays. Moreover, our system should not limit any hypervisor functionality, it should only integrate new operations as added features.

3. METHODOLOGY

In this section, we analyze the design requirements, outline the architecture and operation of the proposed Trusted Virtual Machine Management with Commodity Computers, and also present the relevant details.

At first we need to define the roles in the architecture, so that the operations can be divided for different parties. In our scenario, we have a trusted host machine provider (e.g., control system

provider or cloud provider), who provides spaces for different trusted virtual machines from different service providers, and also provides necessary functionalities for attesting the virtual machines to allow for remote verification. There are service providers, who have customers, and also own the virtual machines inside the trusted host machine. Finally there are customers who take services from the service provider, and use the virtual machine of their service provider inside the host machine, as depicted in Figure 1. The service provider itself may also use its own virtual machine as a customer.

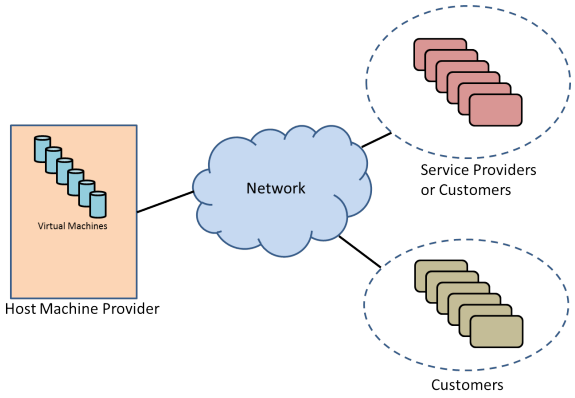

Figure 1: *Different parties in our Architecture*

Clearly, all these actors need to authenticate and authorize each other. One option is to use a public key infrastructure (PKI), but different companies have different requirements about PKI. Hence, setting up the PKIs is out of scope of this paper. We assume the service providers take initiative for the PKI, because they own both the virtual machines and the customers. In our design, a service provider provides the certificate authority (CA) role, but they can choose a trusted third party to provide such operations.

In a trusted system, the system hardware contains a secure co-processor (i.e., TPM), and Core Root of Trust for Measurement (CRTM). The CRTM can be hardly manipulated without physical access and is considered Trusted. On system boot, the CRTM measures (i.e., check the integrity by hashing) the BIOS and necessary firmwares, and gives control to the BIOS. The BIOS measures the bootloader and gives control to it. The bootloader then measures the OS kernel, and gives control to the OS boot procedure. Thus, the measurements goes on in step by step until the user space is ready to use, and each measurement is immediately stored in TPM platform configuration registers (PCRs) before the control goes to the next step. There are two types of PCRs- non-resettable PCRs and resettable PCRs. After booting, non-resettable PCRs can only be extended with a new measurement, whereas resettable PCRs can be cleared at any time. Only PCR 16 and 23 are resettable from the user space. A complete measurement list stores the integrity state of the running system (in non-resettable PCRs) since the last boot of the OS. Therefore, at first, we need a trusted bootloader that can handle the trust chain from the CRTM until the launch of host operating system kernel. For our solution, we assume that the host machine contains a valid TPM, because now-a-days commodity PCs come with a TPM.

Open source trusted bootloaders take advantage of Dynamic Root of Trust for Measurement (DRTM) or Static Root of Trust for Measurement (SRTM). DRTM based bootloaders are completely depended on hardware supported virtualization, and SRTM based bootloaders use only general TPM functionalities. OSLO (Kauer (2007)) and Trusted Boot (tboot (2012)) are DRTM based bootloaders, where OSLO supports AMD Secure Virtual Machine (SVM), and tboot supports Intel Trusted Execution Technology (TXT) technology. Existing embedded processors do not provide DRTM facilities yet, so to develop a general solution to support both world (PC and embedded industrial control system) we need SRTM based solutions, because TPM is already available in embedded processors. However, TrustedGRUB (Selhorst and Stueble (2010)) and GRUB-IMA (GRUB-IMA (2009)) are extensions of the most-popular Grub bootloader, and supplies SRTM, but GRUB-IMA uses SHA-1 hash operations of the hardware TPM during boot process, whereas TrustedGRUB uses software based SHA-1 implementation. Hence, the performance of GRUB-IMA is slower compared to TrustedGrub (Selhorst et al. (2008)). So we select TrustedGrub as our bootloader (for efficiency) to launch the Linux kernel in trusted mode.

If we work with newer kernels (kernel 2.6 and upward), then by default kernel boots and mounts a root file system (rootfs) which is a temporary file system (tmpfs). Shortly after that it extracts an empty gzipped cpio archive into there, and tries to run /init. Here, initramfs (Initramfs (n.d.)) can help us to measure the whole system. Initramfs is a gzipped cpio archive, which can create device nodes, mount root devices or partitions, and can run the /init program to enable user space processes. We do not need to compile the kernel or replace an existing system if we put new initramfs archive. We simply need to update Grub. Initramfs image is presented with the kernel image inside *menu.lst* file of TrustedGRUB. So TrustedGrub can measure the kernel, multiboot modules and any configuration information and extend the PCRs. Later, initramfs can mount and measure the actual root filesystem and /init, and then pass the control to /init. Hence,

our chain of trust can be easily created from BIOS to user space.

The complex part of the boot process is the 'init' program, which dynamically loads modules and files depending on the situation. Hence, static measurement of the boot process is very hard to get. Recent Linux systems use readahead functions to speed up the process, where the first boot generates a log of the executed files during startup. In next boot, the readahead function preloads the files in the memory to speed up the boot process, but it does not enforce the Linux system to sequentially execute the files as they are ordered in the log. So there is no guarantee that Linux systems execute them sequentially in the correct order. This fact can be examined by renaming the log to something else, and then reboot the machine. The new log and the old log always have different ordered list, and therefore, dissimilar measurement lists.

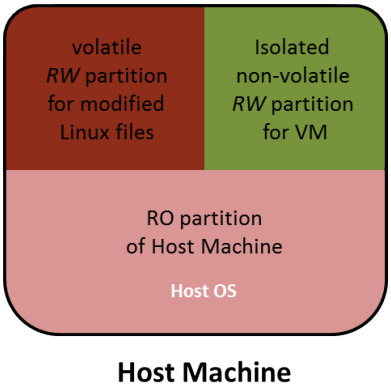

Figure 2: *Partitions of Host Machine*

There is another issue that needs to be considered during boot. Many files are dynamically changed (e.g., files in /var, /run, /random etc.) in each boot, so the measurement values from one boot to another boot always differ. For this reason, the system should transfer the knowledge including the changes to the remote party for complete trust relationship, but it is not convenient to transfer huge amount of data to several remote parties. Obviously, we can build a static host machine if we boot in read only (RO) mode, but to support dynamically changed files, we need a temporary read write (volatile RW) partition as well, and finally to store virtual machines from the service providers, we need a permanent read write partition (non-volatile RW) as depicted in Figure 2. This RO mode of the host operating system also provides us a non-modifiable secure system, which significantly reduces the attack surface on the base and crucial part. In this way, the RO part can be measured, and immediately stored in one PCR (PCR 13), the volatile RW part or a virtual machine can be measured and stored on-demand in a resettable PCR. We choose a resettable PCR for on-demand verification of volatile RW partition or a virtual machine, because their states are not fixed in their lifetime, and customers are not aware of other virtual machines in the host system. Moreover, the measurement of the host operating system should not be tampered at any time, so we plan to extend the complete measurement list in non-resettable PCRs. Because, any malicious software code hidden inside a legitimate code can be hashed and stored in PCR, so it is not possible to disguise without detection. Mounting all partitions and measuring the whole system can be done from initramfs, which keep our operations in the measurement list.

The virtual machine manager (VMM) is located in RO part in our solution, which allows us to measure the VMM with the host operating system, and also presents us a non-modifiable secure VMM. When a service provider hosts a new VM, the VMM can create a unique attestation identity key (AIK) pair (private and public key) using the TPM for that specific VM. This AIK key pair is used to attest the measurement of the VM image in future.

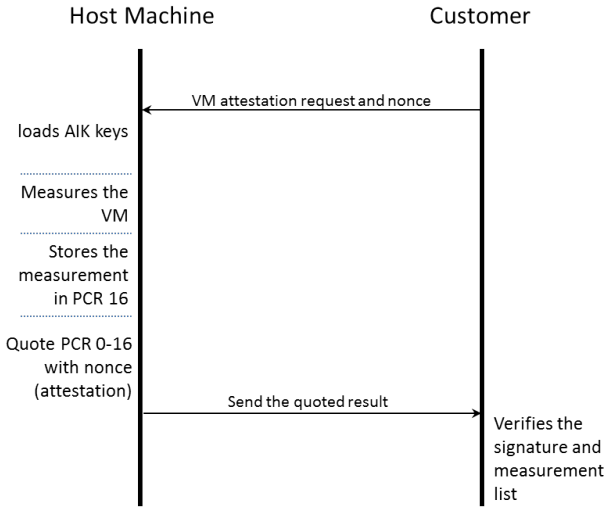

Figure 3: *Remote attestation by a customer*

Our VMM can launch and operate the VM in RO, volatile RW and non-volatile RW partition, similar to its own partitions, and it can measure the complete VM's current state on-demand whenever a remote party wants to verify the state. Upon receiving the request from a remote party, our VMM will launch the VM in read-only mode in a separate process without interfering the running process of the VM. It then measures each and every file of the VM and stores the measurement in PCR 16. Finally, it attests the PCR values of PCR 0-16 by TPM based attestation using the TPM quote operation and the relevant AIK. Then it replies back with the quoted result to the remote party. The remote party should

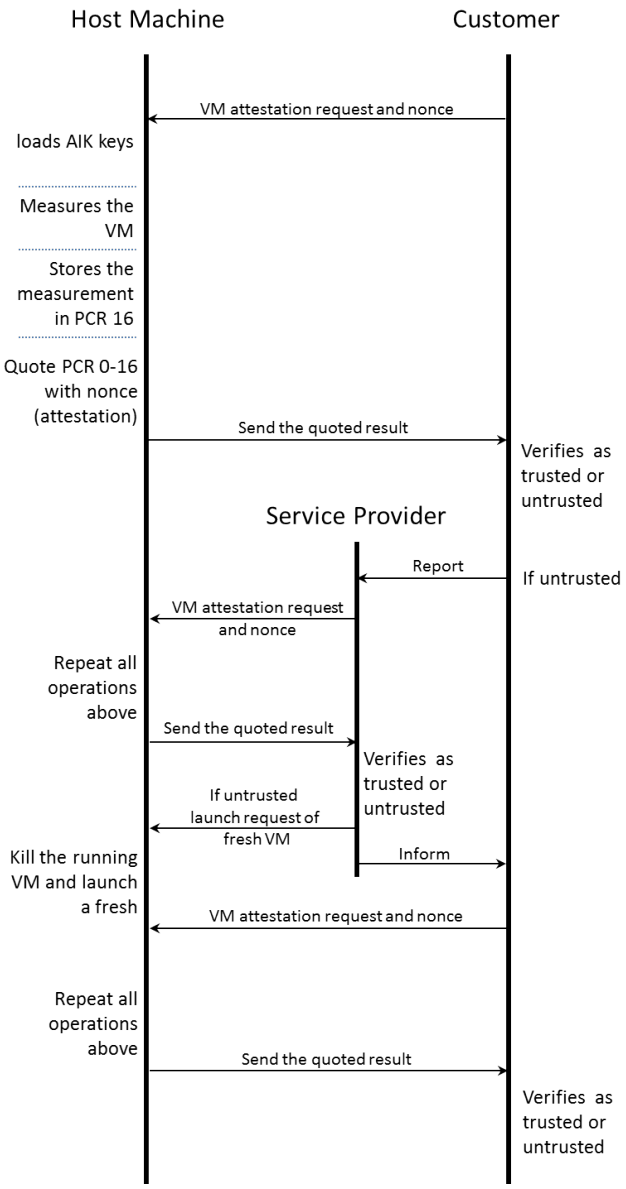

Figure 4: *Complete operation of our solution*

know the public part of the AIK in advance, and it can verify the quoted result as trusted or untrusted, as described in Figure 3, because when we attest common PCRs: 0 to 9 (bios, TrustedGRUB stage 1 and 2), 12 (any command line parameters in TrustedGRUB), 13 (RO part of Host OS) with the specific resettable PCR 16 (VM measurement), the remote party could easily verify the system status and its requested VM image. If the quoted result is untrusted, the remote party can deny to work with that VM or can send request to service provider about the incident. Whenever a report is submitted by a customer, the service provider immediately checks the state of the VM. This is required to prevent any false request. If the recent measurement list is trusted to the service provider, it can inform the customer about it. Otherwise, it initiates a request to the host machine to launch a fresh and trusted VM. Our VMM in the host machine, upon reception of the request from service provider, can kill the malicious VM and re-launch a fresh one easily, because the base part of the VM is always run in non-modifiable RO mode. In the meantime, the service provider can inform its customer about the new launch. Finally, the customer can check again the VM before doing any operation on it. Such operation helps host machine provider (i.e., control system provider or cloud provider) to run the VMM without any disruption, only the operations of VM in question are disrupted. Hence, other VMs from other service providers can operate properly. Figure 4 depicts the complete operation of our solution. Individual AIK for each VM will help us to attest that VM without any dependency to other service provider.

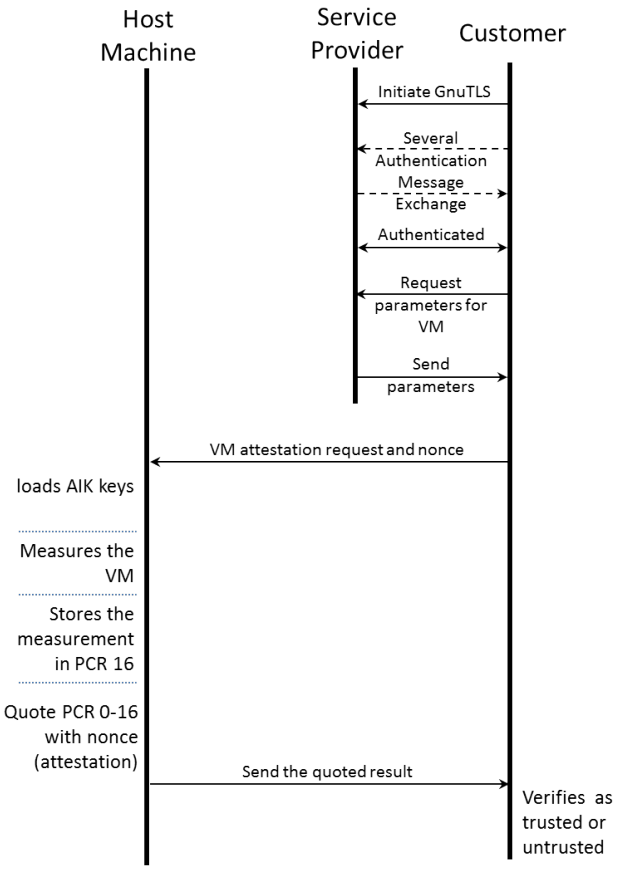

Figure 5: *Remote attestation by a customer over GnuTLS*

4. IMPLEMENTATION

At the beginning, the service provider acts as a CA to generate necessary keys and certificates. We assume that the host machine and the customers get the necessary keys and certificates through a PKI system. Then, in our first phase operation, we use server and client functionalities of GnuTLS (GnuTLS (n.d.)) with X.509 authentication between the host machine (client) and service provider (server). When both are authenticated to each other, the service provider sends a VM to the host machine. Upon reception of the VM, the VMM inside host machine generates an AIK for the VM for future use. Later, in second phase, when a customer wants to use a VM, he/she communicates to the service provider to get necessary parameters to access and verify the VM inside host machine. Again we use GnuTLS with X.509 authentication between the customer (client) and service provider (server) to exchange the parameters. Finally, the customer communicates to the VMM in host machine to verify the VM before using it. For this, we extend open source GnuTLS for our secure communication protocol to give better flexibility to users, because we need to follow strict communication sequence (along with critical parameters) for attesting the virtual machine. Hence, integrating our protocol inside GnuTLS allows us to help users only to initiate normal X.509 authenticated communication without knowing the internal complexity. When the customer sends the nonce in Figure 5, our GnuTLS handshake extension conveys the nonce to the host machine. When the host machine (or VMM) replies back, our GnuTLS supplemental data extension conveys quoted result to the customer. If untrusted, the customer reports to the service provider over a GnuTLS X.509 authentication channel, and then the service provider authenticates the host machine using GnuTLS X.509 authentication, and later use our GnuTLS extension to send attestation request to measure the suspicious VM, as described in Figure 6.

Decoupling the host OS, dynamically changing files and all VMs in separate partitions help us to control them or measure them on-demand. We access the virtual machine in read-only mode to measure its state. For this purpose, we use libguestfs library (libguestfs (n.d.)), which helps us to access the VM without intervening the normal operation of the VM. Moreover, it supports any disk image, for example disks created from scratch or created in VMware, KVM, qemu, VirtualBox, Xen etc. So our solution

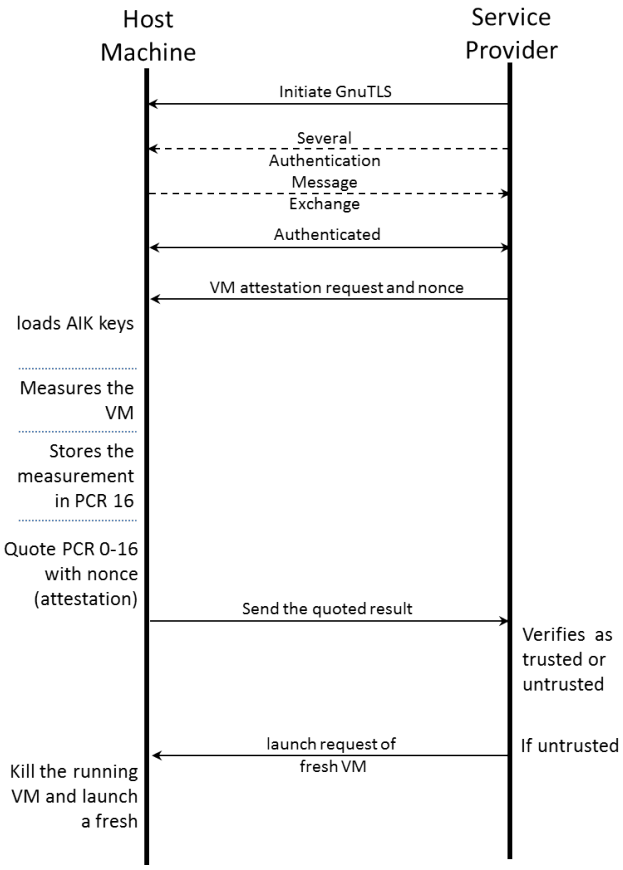

Figure 6: *Launch request for a fresh VM*

supports any of these renowned hypervisors inside the host machine. We have tested our solution on Debian and Ubuntu Linux host machine, but other Linux versions can be configured using their relevant package managers.

5. EVALUATION

We have already mentioned that the Linux-IMA extends the PCR whenever there is a change in the system. Such operations are totally feasible when the system is operated by a single entity. But, in our scenario all VMs are operated by remote machines. Hence, change in one VM should not reflect change in PCR value for another VM, and this is why we invent our novel solution. Although our system is quite different in operation compared to Linux-IMA, here we try to compare the performance of our solution with this standard architecture (Linux-IMA), so that the readers can get some idea about our system.

Linux-IMA in a 8GB CompactFlash card running on an Intel Dual Core 1.6GHz with 2GB Memory requires approximately 4 minutes from system boot to user space, when the footprint of the Linux-IMA is 3.99GB. On the other hand, our system requires more than 6 minutes with the same configuration. At a first glance, it seems huge difference, but we can explain why it is not a big difference. The Linux-IMA measures only the files that are executed, so when it boots the system, it only measures the files that are required to boot the system, whereas our solution measures the complete system. We cannot avoid such measurement, because our base system is read-only, and it does not allow any modification to the system, but Linux-IMA allows all modification with access-time measurement. Moreover, our solution can separately measure the VMs and extend the PCR for a single VM, whereas Linux-IMA measures all VMs and extend them in a single PCR.

6. CONCLUSION

Here, we show how easily we can convert an existing normal Linux operating system with commodity hardware in a Secure Trusted Virtual Machine Manager, which also provides remote attestation features to verify a hosted virtual machine's trusted state. This general methods and architecture can also help us to integrate our solution into a practical system where replacing the current OS is a huge burden from the managerial or technical viewpoint.

We also plan to implement our solution on other versions of Linux in future to estimate the flexibility, compatibility, reliability and performance. We also hope it will encourage many organizations to test their system for VM suitability.

The solution is targeted at smaller systems in critical environments. However, in devices with only one single hardware security module only one VM can be attested at a time. Further, the attestation process is relatively slow, because it depends on the TPM interactions. Thus, the solution as it is will be suitable for virtualization in restricted (but critical) environments, but does not scale for large servers with a high number of users and VMs. Nevertheless, several options for increasing efficiency and scalability are possible. The attestation process itself could be divided into steps, where the actual TPM-based attestation result for the base system is re-usable and the VM attestation is software-based. Finally, large servers could use a cluster of TPMs to provide separate attestation services for different customers.

REFERENCES

Karger, P. A. (2005) *Multi-Level Security Requirements for Hypervisors*, In Proceedings of the 21st Annual Computer Security Applications Conference.

Garfinkel, T. and Pfaff, B. and Chow, J. and Rosenblum, M. and Boneh, D. (2003) *Terra: A virtual machine-based platform for trusted computing*. In Proc. ACM Symposium on Operating System Principles (SOSP).

Strackx, R. and Piessens, F. and Preneel, B. (2010) *Efficient Isolation of Trusted Subsystems in Embedded Systems*. In 6th Iternational ICST Conference, SecureComm.

Linux-IMA, Linux Integrity Subsystem, `http://linux-ima.sourceforge.net/`

Chen, H. and Zhang, F. and Chen, C. and Yang, Z. and Chen, R. and Zang, B. and Yew, P. and Mao. W. (2007) *Tamper-resistant execution in an untrusted operating system using a VMM*. Technical Report FDUPPITR-2007-0801, Fudan University.

Chen, X. and Garfinkel, T. and Lewis, E. C. and Subrahmanyam, P. and Waldspurger, C. A. and Boneh, D. and Dwoskin, J. and Ports, D. R. (2008) *Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems*. In Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems.

Sailer, R. and Valdez, E. and Jaeger, T. and Perez, R. and Doorn, L-v. and Griffin, J. L. and Berger, S. (2005) *sHype: Secure Hypervisor Approach to Trusted Virtualized Systems*. IBM Research Report, RC23511.

Kauer, B. (2007) *OSLO: Improving the security of Trusted Computing*. In 16th USENIX Security Symposium.

Trusted Boot, tboot, `http://tboot.sourceforge.net/`

Selhorst, M. and Stueble, C. (2010) *TrustedGRUB*, http://sourceforge.net/projects/trustedgrub/

GRUB-IMA, `http://ko.sourceforge.jp/projects/openpts/wiki/`

Initramfs, `https://wiki.ubuntu.com/Initramfs`

Selhorst, M. and Stueble, C. and Teerkorn, F. (2008) *TSS Study, Introduction and Analysis of the Open Source TCG Software Stack TrouSerS and Tools in its Environment*.

The GnuTLS Transport Layer Security Library, `http://www.gnutls.org/`

libguestfs, `http://libguestfs.org/guestfs-faq.1.html`