Scalable Attestation: A Step toward Secure and Trusted Clouds

Stefan Berger, Kenneth Goldman, Dimitrios Pendarakis, David Safford, Enriquillo Valdez, and Mimi Zohar
IBM T.J. Watson Research Center

To detect and prevent integrity attacks, scalable attestation extends secure boot and trusted boot technologies into the host, its programs, and a guest's operating system and workloads.

As cloud computing architectures and models continue to gain popularity due to their low cost, flexibility, and elasticity, critical security challenges remain. Customers often cite security as a top concern regarding adoption of cloud computing. Of particular concern is the ability to monitor and verify the integrity of sensitive resources such as physical server firmware, hypervi-

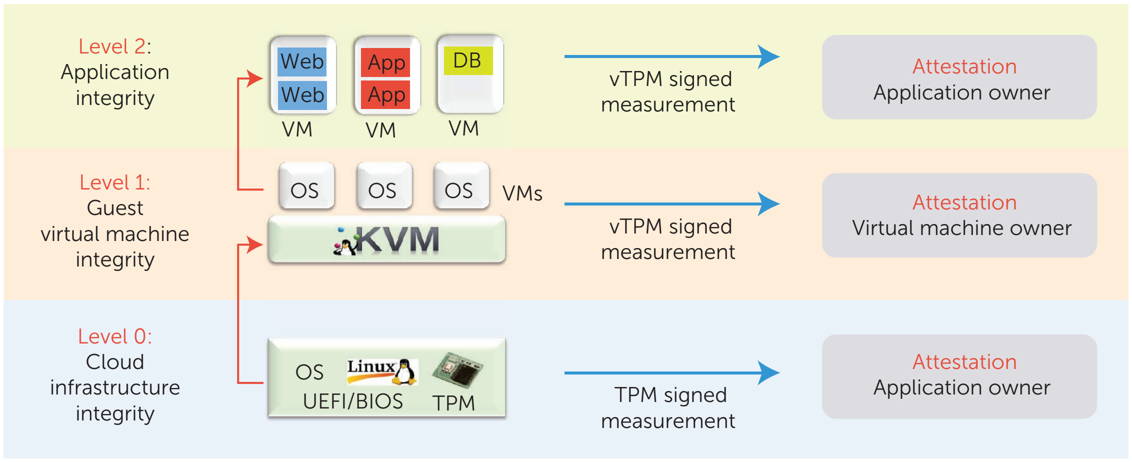

FIGURE 1. The measurement, attestation, and appraisal integrity concepts can be applied at all layers of the cloud. (vTPM: Virtual trusted platform module)

sors, guest operating systems, and key applications. Users of cloud infrastructures want assurance that the environment is running what's expected and that it hasn't been tampered with.

These integrity goals are to detect and prevent access to any persistent storage (files) that are unapproved or have been tampered with, even in the presence of a fully privileged remote attacker. That is, even if an attacker can gain access to a system with full (such as "root") privilege, and thus create or modify any file, we want to be able to detect and prevent access to any such modified files.

There are three related integrity concepts:

- *Measurement* involves creating a hash of a file's contents, such that it would be prohibitively expensive for an attacker to modify the file without changing the measurement.
- *Attestation* is the ability to prove to a third party the measurements of all files accessed on a system, allowing the third party to detect if any unauthorized or tampered with files have been run.
- *Appraisal* involves locally evaluating the measurement for validity, such as with the use of digital signatures on the files. In this case, the local system can refuse access to any file whose appraisal fails. Because it's enforced locally, appraisal is subject to local compromise.

These concepts can be applied at all cloud layers (hypervisor, guest operating system, and workloads), as Figure 1 shows, and on all types of clouds. We implemented a proof-of-concept on a bare metal infrastructure-as-a-service (IaaS) cloud provided by IBM SoftLayer for simplicity. In such a bare metal cloud, both the native and the guest operating systems are under the customer's control, so there are no administrative or network barriers to integrity attestation at the native and guest levels.

Existing Integrity Mechanisms

The success of integrity measurement, appraisal, and attestation, even in the presence of a privileged remote attacker, requires adherence to several design principles:

- *Hardware root of trust*. Because the attacker can modify any piece of software in the system, the integrity mechanism must be rooted in hardware or firmware that can't be modified by a remote attacker.
- *Attestation to a trusted third party*. A system can't be trusted to monitor itself, as any self-monitoring software can be turned off.
- *Complete monitoring*. It isn't sufficient to monitor just boot-time files. All security-sensitive files, both boot time and runtime, and at all levels, from boot to workload, must be monitored.

Much prior work has attempted to ensure the integrity and authenticity of platforms against remote attack, based on hardware roots of trust. The main approaches include trusted boot,[1] secure boot,[2,3] and the integrity measurement architecture (IMA) module[4] (http://sourceforge.net/p/linux-ima/wiki/Home). A broad overview of the concept of trusted clouds is available elsewhere.[5]

Trusted Boot

In trusted boot, a core root of trust for measurement (CRTM) measures (hashes) the initial boot code and commits the measurement to a trusted platform module (TPM) chip.[6] The TPM stores the measurement in a secure storage register called a platform configuration register (PCR). It has a minimum of 24 PCRs for storing measurements. The TPM can attest (prove) this measurement to a third party by signing the measurement with a private key known only inside the TPM. The corresponding public key can be used by the third party to verify the signature, and thus the measurement of the boot code by a valid TPM chip. The TPM is designed so that once a measurement is added to a PCR, no software can reset or remove the value; only a hardware reboot can reset the chip, and thus restart the trusted CRTM. The CRTM function must be hardware protected against modification.

Trusted boot can be extended to measure all subsequent software to form a chain of trust. In this mode, each program measures the next program and adds this measurement to the TPM before reading or executing that next program. If this chain of measurements is kept in a software-maintained list, the list of measurements can be attested to and verified by a third party against the signed PCR value.

If at any point a piece of code is malicious, the model guarantees that the measurement of the malicious code is included in the list. Any attempt by malicious code to modify the measurement list would cause the validation to fail. Note that the local system doesn't assess whether a measurement is good or bad, and doesn't deny access to any files. Only the trusted third party knows if measurements are good or bad.

One of the main problems with the trusted boot model is that validation of the measurement lists doesn't scale well. There are problems with collecting and maintaining a list of hashes of all known "good" files, as these very large lists are constantly changing and are difficult, if not impossible, to keep current. For example, our Fedora 19 collection has on the order of 36,000 distinct packages, with more than 15 million file hashes. Our collection of Red Hat Enterprise Linux 6 packages consists of more than 5,000 distinct packages, with more than 1.6 million file hashes. In addition, in many cases, it's impossible to construct a complete white list, as intermediate package versions aren't available in repositories.

Secure Boot

The second integrity model, secure boot, stores one or more public keys in secure hardware storage that can't be changed by remote attack. These keys are used to appraise (verify) signatures on the boot code; if the signatures aren't correct, the boot is stopped. As with trusted boot, secure boot can also anchor a chain of trust, in which each stage verifies the signature on the next stage.

Digital signatures provide both integrity and authenticity, and in a much more scalable way than simple measurements. If all files on a system are signed by a single distribution key, then they all can be appraised with just this single public key.

However, the secure boot model doesn't provide a hardware-anchored attestation to a third party, so a centralized management system can't tell if a system has been compromised.

Integrity Measurement Architecture

Trusted and secure boot provide measurement, attestation, and appraisal only through boot time of the host's Linux kernel. IMA, a component of the Linux kernel's integrity subsystem, extends the trusted and secure boot chains of trust to cover all files as they're accessed in real time. IMA intercepts all attempts to access files, and implements policy-based measurement, appraisal, and PCR extension, keeping all measurements in a kernel resident list.

OpenAttestation (OAT, https://github.com/OpenAttestation/OpenAttestation/wiki) is an open source client-server application for monitoring the integrity of a set of hosts in a cloud environment. An OAT server can retrieve PCRs and measurement lists from multiple clients. Web-based display is via a REST query API. The OAT server uses a client TPM quote, a signature over the PCR values, to validate the list. One limitation is that OAT currently verifies and displays only the boot-time measurements: it doesn't display or verify the Linux kernel's IMA measurement list.

Scalable Attestation

Some systems come with a TPM, some with secure boot, and others with both. Given the advantages, disadvantages, and availability issues of the two integrity models, it's highly desirable for the overall integrity model to work on top of either, or both, of the hardware roots of trust, and for the kernel to provide both appraisal and attestation services. By combining the models, we get the scalability and authenticity of the secure boot model and the hardware-anchored attestation of the trusted computing model.

Our scalable attestation design integrates the secure boot and trusted boot models, providing roots of trust for measuring and appraising software. Using hardware-anchored public keys, secure boot verifies the boot code, kernel, and subsequent public key certificates used for appraising files in the native operating system. Trusted boot measures software appraised by the secure boot and provides remote verification of the measured software.

Figure 2 gives an overview of the design, including a host with a TPM and secure and trusted boot, an OAT client, and an OAT appraiser.

Our design builds on the existing IMA measurement list and leverages IMA-appraisal's file digital signatures, which are stored as extended attributes. We extend OAT with support for file signature verification, and create a cloud attestation service using OAT for monitoring the integrity of the native hypervisor. OAT also reports the integrity status to a customized cloud controller that affects the management of host systems and placement of virtual machines in OpenStack. Scalable attestation components execute in an external attestation server and on host systems, ideally with both TPMs and secure boot enabled.

Using the extensible IMA template architecture,[7] we define an "ima-sig" template that includes file signatures in the measurement list. A file's entry in the measurement list includes the following parameters: template name (ima-sig), hash, absolute pathname, and signature. Excluding the template name, these parameters are concatenated and hashed. The resulting hash is extended into PCR 10.

IMA-appraisal validates a file by verifying its signature. If a file fails IMA-appraisal, it isn't executed, but its measurement is still recorded in the measurement list. The public key certificates used for verifying signatures are loaded into the IMA keyring during Linux startup by the initramfs. These certificates are validated using a built-in kernel key or secure boot key at their installation time.

The basic OAT functionality validates the lower numbered PCRs 0–7, which are reserved and correspond to the pre-operating-system components measured during boot. OAT compares the current PCRs 0–7 against expected values, acquired when a host system is initially registered with the service.

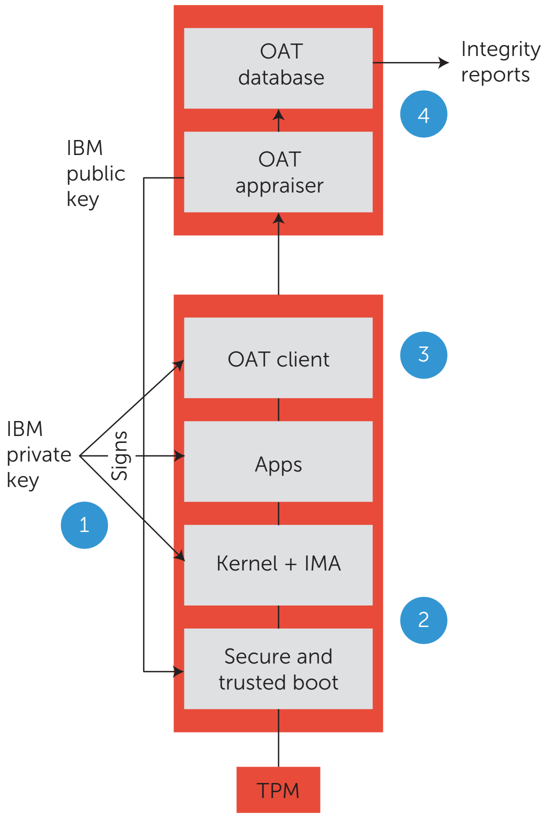

FIGURE 2. High-level view of the scalable attestation design. At installation time (1), the kernel and all applications are signed with a private key, and the corresponding public key is registered with secure boot and the OpenAttestation (OAT) appraiser. At boot time (2), the secure boot code verifies the signature on the kernel. During runtime, the integrity measurement architecture (IMA) verifies the signatures on all applications, and at attestation time (3), the OAT client sends the measurement list and signed quote to the OAT appraiser. The OAT appraiser validates the quote and list, and outputs an integrity report on request (4).

We extended OAT with support for validating IMA measurement lists with file signatures. OAT recomputes the value for PCR 10 by replaying the entries of the measurement list. As it replays the list, each entry is checked for having a known public key and a valid file signature. This validation can fail if the recomputed PCR value doesn't match the actual PCR 10 value, if a file has an invalid signature or no signature, or if a file is signed with an unknown or untrusted key. OAT is preconfigured with the public keys for verifying the file signatures in the measurement list. Any validation failure triggers a notification to the OpenStack host management and virtual machine placement component.

We extend our hybrid integrity architecture to guest machines by providing them with a virtual TPM device.[8,9] The virtual TPM device is a software emulation running as a host-level process and is made available transparently to the guest through integration with the hypervisor.

Implementation

We implemented the scalable attestation architecture in two cloud environments: one in a large internal development testbed using IBM Intel-based servers and one in a commercial IBM SoftLayer bare-metal cloud environment.

In the first implementation, we extended a larger IBM internal DevOps environment consisting of bare-metal Intel-based servers running OpenStack on Ubuntu Linux. We used the Quick Emulator (QEMU, www.qemu.org)/kernel-based virtual machine (KVM, www.linux-kvm.org) as the hypervisor for OpenStack and installed it on two nodes in the testbed. This DevOps environment required that all of our components were packaged and scripted for an unattended installation.

On the SoftLayer systems, we implemented an attestation client on a system with a TPM and the attestation appraiser on a second node. Both systems were running Red Hat Enterprise Linux (RHEL) 6.4.

Neither of the test environments had support for secure boot in their hardware. Because both had hardware TPMs, standard OAT measurement and attestation were sufficient to validate the booted kernels. Once the kernels were booted, IMA provided the measurement, appraisal, and combined attestation integrity model for all subsequent file accesses.

Host Integrity

Because file appraisal is only implemented in recent Linux kernels, we built a current upstream version of the Linux kernel with IMA-appraisal and template support enabled, including the ima-sig template. We replaced the 3.2 kernel of Ubuntu 12.04 and the 2.6.32 kernel of RHEL 6.4 with this updated kernel. When booting Linux with IMA-appraisal enabled, we must load public key certificates into the IMA keyring early in the boot process. IMA-appraisal uses the public keys to verify file signatures and subsequently determine whether an executable is allowed to run. At boot time, the Linux kernel depends on a temporary file system (the initial RAM disk) to mount the root file system. Consequently, we modified Ubuntu's mkinitrd and RHEL's dracut to build initial RAM disks with these necessary certificates.

Because current distribution package installation tools (dpkg, rpm) lack signature support, we extended our DevOps environment to sign executable files—that is, Executable and Linkable Format (ELF) files and interpreted executables—using private keys whose public keys are accepted by IMA-appraisal. The RSA signatures are stored in the files' extended attributes (xattrs). This signing process occurs before machines are rebooted and file signatures are enforced. To simulate a system configured with packages from different software vendors, we use several keys for signing.

We established trust in these keys by signing their corresponding public keys and generating certificates using a local certificate authority. These certificates are used by IMA-appraisal for signature verification and are sent to the OAT verifier during the normal OAT client registration.

Extending OAT

The basic OAT client runs at boot time, sending the boot-time attestation (TPM signed boot-time PCR values) to the OAT appraiser, which verifies the signature and adds the data into its MySQL database for later viewing. For the IMA runtime measurement lists, we extended the OpenAttestation version 1.6 Web portal with a new menu entry that pulls real-time IMA lists and TPM attestation from selected clients. In particular, it launches a script that gathers the measurement lists from all native hosts.

Figure 3 shows an example real-time polled IMA integrity report. First, it verifies that the entire measurement list reduces to a value for PCR 10, which matches the actual (signed) PCR.

Second, it summaries all the files in the measurement list, indicating how many were in the list, how many were signed by each of the registered public keys for that host, and how many (if any) were unsigned.

Third, it lists details on all files that failed signature verification, files that weren't signed, files whose signature failed verification, and files that were signed by an unregistered public key.

Key Management

There are two use cases for public key management: managing public keys for file appraisal, and managing the TPM attestation keys (AIKs).

For file appraisal, the files can be signed locally, by a distribution, or by a third-party software provider. The respective verification public keys need to be loaded into the IMA keyring at boot time in a trusted way. In our prototype, we used a local certificate authority to sign all of the public key certificates we

```
IMA SUMMARY REPORT

Verifying IMA Measurement List:
    Calculated   PCRAggr: 13 63 CD 84 BD 64 70 D0 C0 9C E6 0F 8C 40 34 2D 2D 64 32 92
    Actual       PCR-10: 13 63 CD 84 BD 64 70 D0 C0 9C E6 0F 8C 40 34 2D 2D 64 32 92

Total Measured Files: 479
    /etc/keys/ibm_signed.der 28
    /etc/keys/ubuntu_signed.der 0
    /etc/keys/cacert.x509 0
    /etc/keys/redhat_signed.der 450
    /etc/keys/google_signed.der 0
    unsigned 1

Signature Errors:

    Unsigned 1
412 010 c9ae7bcd026ccf74877635375b176952ed9066dc ima-sig sha1 : 7cf68808adee669f19
7039d2fbfbbf4cae79c4ed /bin/Trojan_nosig

    Invalid signature:  0
    Missing key:        0
```

FIGURE 3. An OAT portal IMA runtime integrity summary showing the computed and actual PCR 10 hash values, number of files measured, and signature verification results.

generated, to simulate the different software providers. We compiled this local certificate authority public key into the kernel and used it to verify certificates for all IMA-appraisal verification public keys. (This depended on the trusted IMA keyring support upstreamed in Linux 3.17.)

In our prototype, we didn't need to update any of the TPM or distribution keys. In the future, distributions may change keys more frequently to limit replay attacks, but this is easily handled by creating a new local certificate authority key and signing all the software validation keys.

For the OAT key management, the OAT-appraiser already comes with a local privacy certificate authority, which certifies the client TPM AIK certificates.

Guest Support

Guests in our virtualized environment run on the Linux QEMU/KVM hypervisor. The hypervisor can be regarded as a thin layer underneath the guests. In addition to splitting up the host's physical memory and providing chunks of it to each guest, the hypervisor emulates hardware devices.

To enable monitoring of a virtual machine's integrity using OAT, we extended the hypervisor with an emulated TPM called a virtual TPM (vTPM; see http://sourceforge.net/projects/ibmswtpm and https://github.com/stefanberger/libtpms). The vTPM provides TPM functionality for a guest operating system. Figure 4 shows the hypervisor's emulator component, QEMU, with a vTPM. Each guest that requires a TPM will now be able to get its own private vTPM.

Additionally, we extended the virtual basic input/output system (vBIOS) running inside the guest with TPM management functionality, shown in red in the figure. It also starts measuring certain parts of its code and data, extending PCRs 0–7 with those measurements, and logging each measurement in a table that can later be accessed by the guest operating system.

Results

The inclusion of file signatures in the measurement list provided a scalable and elegant method for detecting persistent file modifications, without the complexity of maintaining white or black lists of file measurements.

On the host systems, IMA and IMA-appraisal together measured and enforced local file integrity,

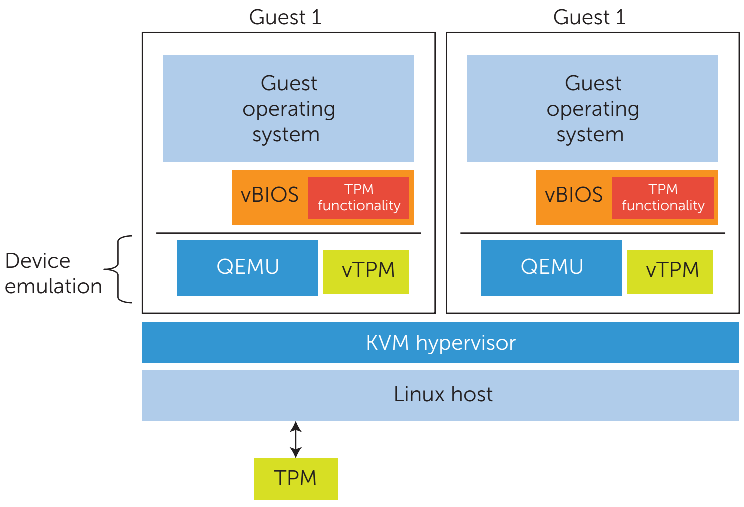

FIGURE 4. The Quick Emulator (QEMU)/Kernel-based Virtual Machine (KVM) virtualization architecture with hardware and vTPMs. The Linux host is assigned the hardware TPM, and each guest is allocated its own vTPM.

Table 1. Host benchmark results when integrity measurement architecture (IMA) measurement and appraisal are turned off and on.

Benchmark	IMA off	IMA on	Overhead (%)
Boot time	45 s	50 s	+11
Sysbench	21.83 s	21.68 s	−0.7
Make kernel	7 m 50 s	7 m 38 s	–3.6

using only keys signed by the built-in local CA kernel key.[10]

We deliberately created the three possible types of malicious files that an intruder can create without an authorized private key: an unsigned file, a signed file with an unknown key, and a modified file with an invalid signature. Figure 3 shows the integrity summary page in the extended OAT Web portal. The files without signatures, invalid signatures, or unknown keys were prevented from executing on the local filesystem, even when attempted by a root-privileged attacker.

The combination of the signature and attestation models makes the integrity design highly scalable. Whereas our collection efforts resulted in an incomplete whitelist database of more than 16.5 million hashes, the extended OAT appraiser required a list of just four public keys to validate the signatures on all appraised host and guest files.

We performed benchmarks on the host, a virtual machine, and the appraiser to understand the performance costs involved in measuring, appraising, and enforcing local file integrity on the host system, and the scalability of verifying the measurement list on the attestation server.

Immutable files were signed during the last stage of the DevOps process using a local private key. This signing was by far the greatest overhead, taking around 10 minutes. Fortunately, this was done only once at installation time. In the future, packages will come with signed files, so this overhead will go away.

We performed three host-level benchmarks with measurement and appraisal turned on and off. First, we measured boot times; second, we performed sysbench; and third, we measured kernel compile times. We ran all tests on a Thinkpad W540 (I7 CPU, 16 Gbytes RAM) after fresh reboots to eliminate cache effects. The kernel was compiled with the default Fedora kernel config file. Table 1 shows the results.

We anticipated that the boot-time measurement would be the worst-case benchmark, given that IMA measures and appraises some 900 files during boot. We found that IMA did increase the boot time by 5 seconds (11 percent).

We anticipated that the sysbench CPU test would be largely independent of the IMA state because it loads once and iterates the test 20,000 times. Compiling the Linux kernel uses a mix of both small file I/O and CPU, and should have a small overhead: all the files have to at least be checked against the IMA policy. Both of these tests actually showed consistent but small performance improvements. We believe that the small improvements were due to IMA's full prefetching of all referenced files, particularly files first accessed during boot.

Runtime performance impacts due to measurement and appraisal were similar to those found on the host. There were some differences in guest boot times due to the vTPM. On a guest's first boot, the vTPM software has to simulate the manufacturing initialization step (as described earlier), which might take several seconds before the guest can be started. Subsequent restarts of the same guest don't incur this overhead again.

On the OAT server, when an integrity report is requested, the measurement list and all the contained signatures have to be verified. Validating the measurement list entails recalculating an aggregate template data hash value, comparing it with the signed PCR value, and verifying each measurement entry template signature and hash. Verifying the ima-sig measurement list with 800 entries took roughly 20 milliseconds on one core, so the server can process roughly 50 such reports per second per core.

This is already quite scalable, and the verification is parallelizable across reports to scale arbitrarily.

Lessons Learned

We encountered an issue when DevOps reprovisioned the TPM. During reprovisioning, the TPM's owner must be cleared to protect any former customer's secrets. There are two ways to clear a TPM owner: physical presence or knowledge of the owner's password. Because of security concerns, we didn't want DevOps to know the highly sensitive owner password, so this clearing requires physical presence. On our systems, physical presence is indicated by changing a motherboard jumper, so this can't be done remotely, and required the help of a DevOps administrator at the test lab site. In the SoftLayer bare-metal environment, clearing the owner similarly requires physical presence, which is done automatically when a host is reprovisioned or by a standard ticket request system.

Recent firmware and operating system extensions may provide another alternative by supporting the Trusted Computing Groups Physical Presence Interface (PPI) specification.[11] Using PPI, an administrator can choose a sequence of TPM operations the firmware is supposed to execute upon reboot, including giving up ownership of the TPM. To enable this, an administrator might need to set up the firmware so even sensitive operations that typically require physical presence can be executed unattended.

The lesson learned is that TPM reprovisioning requires a process for easy and rapid clearing of a TPM owner. The methods we've discussed provide ways to do this securely.

Scalable attestation combines the best aspects of secure and trusted boot with the trust chains extended to include files up through the QEMU/KVM hypervisor, the guest, and its workloads.

Future work items we're pursuing include the following:

- The prototype measured and appraised all executables in the KVM hypervisor. We plan on extending this to include other security-critical files, such as policy and configuration files.
- The prototype used test keys for signing files after installation. We're working to allow software providers to distribute their files already signed with production keys.
- The vTPM support was implemented on a KVM/x86-based cloud. Future work includes integrating it to work on other hypervisors and platforms.

This future work will make the security more complete, simplify adoption, and cover other hypervisors and platforms.

References

1. *TCG PC Client Specific Implementation Specification for Conventional BIOS*, Trusted Computing Group, 2012; www.trustedcomputinggroup.org/files/resource_files/CB0B2BFA-1A4B-B294-D0C3B9075B5AFF17/TCG_PCClientImplementation_1-21_1_00.pdf.
2. W.A. Arbaugh, D.J. Farber, and J.M. Smith, "Secure and Reliable Bootstrap Architecture," *Proc. IEEE Computer Society Conf. Security and Privacy*, 1997, pp. 65–71.
3. *Unified Extensible Firmware Interface Specification Version 2.4*, UEFI 2013; www.uefi.org/sites/default/files/resources/2_4_Errata_B.pdf.
4. R. Sailer et al., "Design and Implementation of a TCG-Based Integrity Measurement Architecture," *Proc. 13th Usenix Security Symp.*, 2004, pp. 223–238.
5. K.R. Jayaram et al., "Trustworthy Geographically Fenced Hybrid Clouds," *Proc. Middleware*, 2014, pp. 37–48.
6. *TPM Main Specification Level 2, Version 1.2, Revision 116*, Trusted Computing Group, 2011; www.trustedcomputinggroup.org/resources/tpm_main_specification.
7. M. Zohar, "IMA: Larger Digests and Extensible Template Support," LWN.net, 21 Oct. 2013; http://lwn.net/Articles/571221.
8. S. Berger et al., "VTPM: Virtualizing the Trusted Platform Module," *Proc. 15th Usenix Security Symp.*, vol. 15, 2006, article 21.
9. *Virtualized Trusted Platform Architecture Specification, Version 1.0, Revision 26*, Trusted Computing Group, 2011; www.trustedcomputinggroup.org/resources/ virtualized trusted platform architecture specification.
10. M. Zohar, "IMA: Extending Secure Boot Certificate Chain of Trust," patch 0/5, 20 Aug. 2013, LWN.net; http://lwn.net/Articles/564114.
11. *TCG Physical Presence Interface Specification Version 1.30, Revision 00.52*, 2015; www.trustedcomputinggroup.org/files/resource_files/D6850418-1A4B-B294-D0427225D4301E82/Physical Presence Interface_1-30_0-52.pdf.

STEFAN BERGER *is a researcher at the IBM T.J. Watson Research Center in Yorktown Heights, New York. His research interests include cloud and virtualization security and application of trusted computing technologies. Berger has a diploma in electrical* *engineering from Universität at Erlangen-Nürnberg. Contact him at stefanb@us.ibm.com.*

KENNETH GOLDMAN *is a researcher at the IBM T.J. Watson Research Center. His research interests cover trusted computing technology, including standards and applications. Goldman has an MS in electrical engineering from the Massachusetts Institute of Technology. Contact him at kgoldman@us.ibm.com.*

DIMITRIOS PENDARAKIS *is a researcher and senior manager in the Secure Systems Group at the IBM T.J. Watson Research Center. His research interests include security for cloud computing infrastructures, hardware-enabled defense, trusted computing, virtualization security, security analytics, and security for embedded and Internet of Things. Pendarakis has a PhD in electrical engineering from Columbia University. Contact him at dimitris@us.ibm.com.*

DAVID SAFFORD *is a researcher at the IBM T.J. Watson Research Center. His research interests include hardware roots of trust and the Linux Integrity Subsystem. Safford has a PhD in computer science from Texas A&M University. Contact him at safford@us.ibm.com.*

ENRIQUILLO VALDEZ *is a researcher at the IBM T.J. Watson Research Center. His research interests include cloud security and analytics and secure systems. Valdez has a PhD in computer science from Polytechnic University. Contact him at rvaldez@us.ibm.com.*

MIMI ZOHAR *is a researcher at the IBM T.J. Watson Research Center. Her research interests include system security and integrity. She is the Linux-Integrity Subsystem maintainer. Zohar has an MS in computer science from Pace University. Contact her at zohar@us.ibm.com.*