Trusted Tamper-Evident Data Provenance

Mohammad M Bany Taha
Cyber Security Lab
The University of Waikato
Hamilton, New Zealand
mmmb1@students.waikato.ac.nz

Sivadon Chaisiri
Cyber Security Lab
The University of Waikato
Hamilton, New Zealand
chaisiri@waikato.ac.nz

Ryan K. L. Ko
Cyber Security Lab
The University of Waikato
Hamilton, New Zealand
ryan@waikato.ac.nz

***Abstract*—Data provenance, the origin and derivation history of data, is commonly used for security auditing, forensics and data analysis. While provenance loggers provide evidence of data changes, the integrity of the provenance logs is also critical for the integrity of the forensics process. However, to our best knowledge, few solutions are able to fully satisfy this trust requirement. In this paper, we propose a framework to enable tamper-evidence and preserve the confidentiality and integrity of data provenance using the Trusted Platform Module (TPM). Our framework also stores provenance logs in trusted and backup servers to guarantee the availability of data provenance. Tampered provenance logs can be discovered and consequently recovered by retrieving the original logs from the servers. Leveraging on TPM's technical capability, our framework guarantees data provenance collected to be admissible, complete, and confidential. More importantly, this framework can be applied to capture tampering evidence in large-scale cloud environments at system, network, and application granularities. We applied our framework to provide tamper-evidence for Progger, a cloud-based, kernel-space logger. Our results demonstrate the ability to conduct remote attestation of Progger logs' integrity, and uphold the completeness, confidential and admissible requirements.**

***Index Terms*—Trusted Computing; Accountability in Cloud Computing; Data Security; Trusted Platform Module; Data Provenance; Remote Attestation; Tamper Evidence; Cloud Computing**

I. Introduction

Data provenance describes the origins and derivation of the data, starting from its original sources [26], [4]. Data provenance can be used to enhance cloud accountability and trust [12], [11], [8] as evidence for auditing, forensics, and data analysis purposes [9]. Provenance data, for example *provenance logs* can be obtained at the system, network, and application levels [26], [22]. In general, data provenance used as evidence should comply with **five** requirements, namely *admissibility*, *authenticity*, *completeness*, *reliability*, and *believability* [19]. Tampered evidence (data provenance) renders evidence useless.

A. Objectives

Currently, only some provenance logging tools (e.g. Progger [13], and Forensix [6]) provide tamper-evidence. These tools generate provenance logs which include data activities (e.g., reading, modifying a file) along with actors and related identifying entities (e.g., process ID, user ID, and timestamp). While provenance tools such as Progger can be deployed in cloud environments monitoring data activities within physical and virtual machines, these logs may be tampered by malicious users or processes *at-rest* or *in-transit*. In other words, these provenance logs cannot fully meet the five requirements required for use of provenance as evidence.

In this paper, we focus on providing remote attestation and tamper-evidence capabilities to **all** types of data provenance logs generated in distributed systems such as cloud computing. To enable tamper-evidence, a robust mechanism needs to be developed to handle the provenance logs while providing tamper-evidence to the logs. Despite the increased importance of situation awareness in distributed systems, tamper-evidence for provenance is not well-studied.

B. Leveraging TPM capabilities

In this paper, we propose a novel framework for tamper-evident data provenance in cloud computing based on the Trusted Platform Module (TPM). We also demonstrate the framework's capability to perform remote attestation for the integrity of data provenance logs. The TPM is a security microcontroller chip on the motherboard that securely stores artefacts (e.g., hash values and encryption keys) [23]. We will elaborate on the application of TPM features in Section III-C.

Within our framework, the integrity and secrecy of the provenance logs is supported by the TPM. Our framework also stores provenance logs in trusted provenance and backup servers to guarantee the completeness and availability of the data provenance. As a result, the confidentiality along with the privacy of the provenance logs can be preserved while the tampered logs can be detected and recovered.

C. Paper Organisation

This paper is organized as follows: Section II lists the requirements for tamper-evident data provenance. Based on the requirements, we evaluate related work in Section III. In Section IV, we present our proposed framework. Then, Section V discusses the implementation and results of our framework. Next, we discuss the advantages of the proposed framework in Section VI. Finally, the paper is summarized in Section VII.

II. Requirements of Tamper-Evident Provenance

Tampered provenance logs cannot be used as evidence for forensics or investigation purposes. Hence, trusted data provenance needs to be implemented along with the root of

978-1-4673-7952-6/15 $31.00 © 2015 IEEE
DOI 10.1109/Trustcom.2015.430

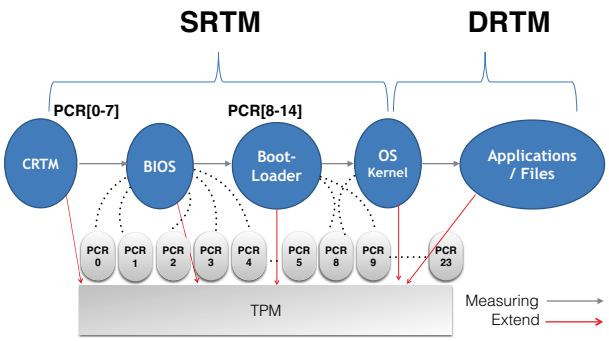

Fig. 1. Chain of Trust

trust. Fig. 1 shows a chain of trust empowered by the TPM. If a component in this chain of trust (e.g., updated OS kernel or bootloader) is tampered, the logs may also be affected and can no longer be useful as tamper-evidence. According to [19], the tamper-evidence must meet all five requirements :

- *Reliability* – The evidence must be reliable, i.e., the evidence can be used without doubt.
- *Authenticity* – The evidence must be authentic, i.e., the evidence can relate to the actual incident as it is.
- *Admissibility* – The evidence must be admissible, i.e., the evidence must be accepted when it is used (e.g., in court).
- *Believability* – The evidence must be believable, i.e., the evidence must be understandable and believable by a jury.
- *Completeness* – The evidence must be complete, i.e., the evidence must be validated from the boot time until the runtime.

III. Related Work and Evaluation

A. Provenance and Tamper-evidence

Data provenance describes the derivation history of data [26]. In cloud computing systems, this includes data actions, actors (e.g., process IDs and user IDs) associated with the data, and timestamps of the actions. However, if no tampering detection mechanisms are in place, malicious users or processes may access and tamper the evidence, resulting in inadmissible evidence. Hence, tamper-evidence is required to provide an indication that the evidence is altered by accidents or malicious actions.

In the past decade, several data provenance tools have been developed. In PASS [19], [16], mechanisms were developed to collect system-level provenance logs from virtual machines. Flogger [10] and S2Logger [21] record file-level and block-level kernel-spaced system calls for cloud virtual machines and physical machines, respectively. In [14], Ling describes methods to collect the provenance from machines, and then uses the audit trail to find the evidence from provenance. In [2], Ansari *et al.*, analysed and measured the performance of efficient file system intrusion detection systems and established a complementary approach for the existing access control mechanism in the Linux kernel 2.6.x.y. They focused on preserving Modification, Access, and Creation Data and Time Stamp (MAC DTS) of files. This mechanism can be used to maintain a log file (e.g., provenance log file) that records the MAC DTS of the files being accessed and changed in any underlying file system that is registered to the Virtual File System (VFS). The mechanism can trap and log activities from system calls into a log file, which will then hide the log from the file system. The administrator can unload the module from the system to use this log for tamper-evidence.

In 2014, Ko and Will developed Progger [13], a kernel-level provenance logging tool that supports tamper-evidence. Progger can capture major data activities such as kernel-level system calls relating to create, read, update and delete actions. On the other hand, this differs from other kernel-level monitoring tools such as Forensix [6], which do not provide tamper-evidence capabilities in their implementation. The Forensix tool generates data logs and collects them in database servers. After which, users can submit SQL queries to retrieve the events that occurred in the machine (e.g., PID, start-time, end- time). The authors did not mention any technology which provides secure, integrity-preserving environments for the database server. This potentially makes it easy for malicious users to mask their trace by modifying the logs in database server or within the server which generate the logs. To our best knowledge, work on Forensix has been discontinued since 2011 with the last source code update not working as expected in terms of providing a secure environment. In [7] [25], the authors developed application-level provenance tools for tracking data writes for applications and providing integrity of the provenance by using checksum-based approaches.

B. Evaluating Provenance Mechanisms with Tamper Evidence

The provenance mechanisms were evaluated in Table I against the tamper-evidence requirements in Section II. It was found that most provenance loggers do not satisfy more than two requirements. Our framework, which will be elaborated in Section IV, complies with all **five** requirments to provide full tamper-evidence.

Hence, to our best knowledge, the provenance tools described in section III-A cannot provide tamper-evidence that is *admissible*, *complete*, *authentic*, *reliable* and *believable*. We believe that this is because they lack a root of trust that can verify changes which occured to the system (i.e., a machine running the provenance tool) from the boot time to the runtime (i.e., the time when the provenance tool is generating logs).

C. Application of TPM to Logs

We propose that the Trusted Platform Module (TPM) can be applied to address this need for full tamper-evidence. Boeck *et al.*, [3] presented a solution using a TPM chip and the AMD Secure Virtual Machine (SVM) technology to provide the root of trust for any log collected by syslog as data provenance.

TABLE I
EVALUATING PROVENANCE LOGGERS AGAINST TAMPER-EVIDENCE REQUIREMENTS

	Forensix	Progger	PASS	[2]	TPM + Progger
Reliable	✓	✓	✗	✗	✓
Authentic	✗	✓	✓	✓	✓
Admissible	✗	✗	✗	✗	✓
Believable	✗	✗	✗	✗	✓
Complete	✗	✗	✗	✗	✓

However, this approach did not address the integrity and availability of the logs.

As shown in Fig. 1 , TPM provides a 'root of trust' for a trusted environment by measuring critical components including CPU instructions (i.e., Core Root of Trust for Measurement (CRTM)), BIOS, boot stages, operating systems, and applications. The TPM measures the status of the environment by storing hashed (extend function) values relating to these components inside *platform configuration registers* (PCRs). Fig. 2 shows the PCRs inside TPM chip. The values of PCRs cannot be tampered by unauthorised actions or software. As a result, any change in these components measured by the TPM can be detected. For example, when the BIOS is updated by a firmware or the bootloader is changed by an 'evil maid attack' [17], the TPM can detect these changes (We will further discuss this in Section VI).

Technically, TPM is a chip that securely stores hash values inside PCRs to authenticate the platform [23]. An authenticated platform means that such a platform can be trusted. For example, Fig. 2 shows a snapshot of values stored inside 24 PCRs. Each PCR consists of 160 bits, containing hash values. Each PCR keeps hash values of particular components (e.g., PCR-0 and PCR-1 are allocated for the BIOS).

The Trusted Computing Group (TCG) defines *Technical Trust* as a state where "the platform always behaves in the expected manner for the intended purpose". In Fig. 1, a chain of trust starts from the Core Root of Trust for Measurement (CRTM), which is a trusted code in the BIOS boot block. It reliably measures integrity values of other entities, and stays unchanged during the lifetime of the platform. CRTM is an extension of normal BIOS which first measures other parts of the BIOS block before passing control back to the BIOS, the BIOS then measures hardware and the bootloader, and then passes control to the bootloader. Next, the bootloader measures the OS kernel image and passes control to the OS. Each step of this boot process extends the appropriate PCR values in the TPM with the measurements taken in that step.

The term *extend* in Fig. 1 means hashing the measurement value and saving it inside PCRs. The extend function can be described by $PCR_{n+1} = SHA1\Big(PCR_n + SHA1(Component)\Big)$ where PCR_{n+1} is the new (expected) value of PCR where PCR_n is the current value of PCR.

If any change happens in Static Root of Trust for Management (SRTM) (as shown in Fig. 1 phase), the value of PCR_{n+1} will indicate the change because the value will be different from the value of PCR_n.

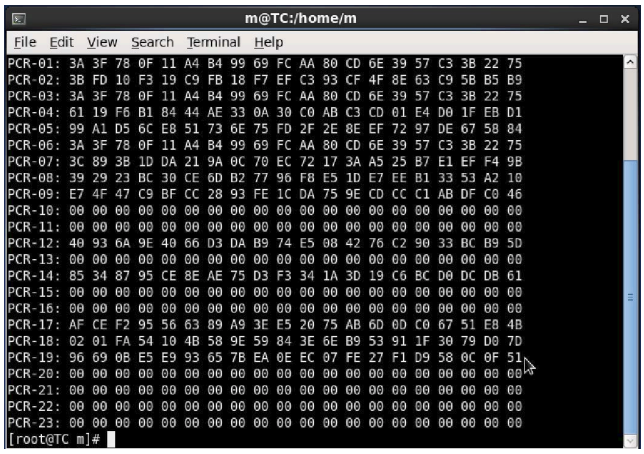

Fig. 2. TPM PCRs

Referring back to Fig. 1, the TCG defines the next phase called Dynamic Root of Trust for Measurements (DRTM) as the measurements of the platform at run time. The dynamic chain of trust starts a request from the operating system (OS) via a special processor instruction. To provide DRTM capabilities, Intel developed the Trusted Execution Technology (TXT), while the AMD equivalent is the Secure Virtual Machine (SVM). The DRTM can provide the trust environment at any time. It may also be interesting to note that IBM also provides a technology called Integrity Measurement Architecture (IMA) [20] to maintain a runtime measurement list (e.g., measurement for a list of sensitive files) and attest to the runtime integrity of the platform.

IV. PROPOSED FRAMEWORK AND TRUST MODEL

In this section, our proposed framework is discussed. All tools used in this framework are either kernel modules or open sourced software, e.g., IMA, TPM-Quote, and Intel-TXT. The components of the framework are discussed as follows (see Fig. 3):

- *Provenance Generator* – Provenance generator is a kernel-space provenance logging tool. In our case, the provenance generator is Progger. The provenance generator in client machines generates provenance logs and sends it to the buffer as shown in Fig.3.
- *TPM* – A TPM chip is required to provide the integrity, confidentiality and reliability for the data provenance logged. The hash value of the program of provenance generator (e.g., Progger) is stored inside a PCR – detecting any change to the code. After which, IMA is used to detect expected events to the provenance log file. TPM stores the hash values of measurement for CRTM, BIOS, bootloader, OS kernel, and OS. During runtime, the Intel-TXT and IMA can detect attacks at the DRTM phase. We assume that the storage used to

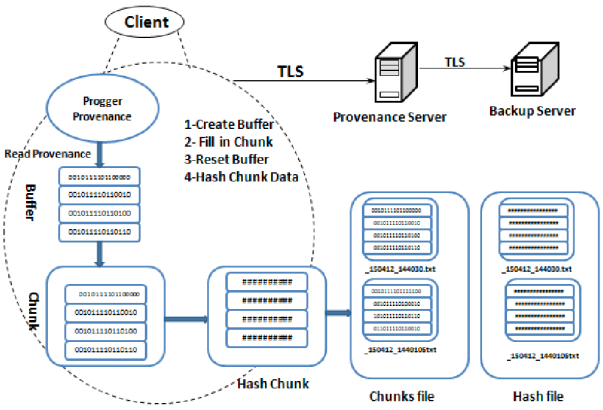

Fig. 3. Framework Design

store data provenance applies the Opal TCG technology to enable full disk encryption [24]. Intel TXT can be configured to only allow the OS to launch if it knows the platform and system software are secure and trusted (as defined by the data centre's policy). This secure launch allows the software to operate as a trusted OS, but only after validating that the platform configuration and system software meet the system administrator policy [5].

- *Client Machine* – In Fig. 3 client machines could be physical or virtual machines. A client machine is the machine where data provenance is deployed for capturing provenance logs. A provenance generator is assumed to be installed in every client machine. It is assumed that the TPM chip is available and active in the client machine. In addition, DRTM and IMA modules are enabled to give the administrator ability to remotely check client machine status (e.g., values of the PCRs) using TPM-Quote or OpenPTS [20].
- *Provenance and Backup Servers* – The provenance and backup servers in Fig.3 are deployed to collect the provenance logs from client machines. The provenance logs consolidated on provenance servers can be used to examine the provenance logs for short-term periods (e.g., two or three months) in provenance servers. For longer terms, the backup server works as an archive server of provenance logs when the amount of provenance logs becomes too huge to be stored in the provenance server. A large amount of provenance logs in the backup server can be also used for data analytics purposes.
  Both the provenance and backup servers must have TPM, DRTM, and IMA implemented. The provenance server is accessed by the administrator who has the ability to examine the data provenance. The administrator must have the TPM-ownership password to gain TPM features. In this framework, pieces of provenance logs from a client machine are grouped as a *chunk*. Every chunk along with it's hashed value will be transferred via the TLS to the provenance server. The administrator can check the hash values from client machines and compare them with the hash values stored in the provenance server. We assume that the hash function (used to hash chunks) applies a keyed-hash message authentication code (HMAC) (e.g., HMAC-SHA1). The secret key used for the HMAC is shared by all the machines with the support of TPM.

V. Framework Implementation and Results

In our implementation, we have client virtual machines and two physical servers as the provenance and backup servers. We used CentOS 6.4 with kernel 2.6.32 for all machines. Progger is deployed in the client machines as the provenance generator. The machines come with TPM v1.2. The Trusted Execution must be active from the BIOS to enable Intel-TXT. Software such as TrouSerS [1] and TPM-tools are installed for the communication with the TPM chip. TPM-Quote also needs to be installed on the provenance server for remote attestation.

The connection between the provenance server, the backup server, and client machines is secured by TLS. The procedure of our implementation is explained step by step in the following subsections. In Algorithm 1 and Fig. 3, we briefly explain the operations on client machines.

Algorithm 1 Client machine

```
void generator (log)
if (buffer.full() || buffer.timeout()) then
    Chunk chunk = new Chunk(buffer.getData())
    chunkStore.add(chunk)
    hashStore.add(hash)
    buffer.flush()
    chunk.getbackup()
    hash.getbackup()
else
    buffer.add(log)
end if
```

A. Fetch Provenance Logs from Provenance Generator

When the client is operating at runtime, Progger (i.e., provenance generator) records provenance logs. Each record of the logs includes an action or event (e.g., create, read, write a file), process ID (PID), user ID (UID), and timestamp of the event.

The hash value of the Progger program is stored and measured its change by PCR-23. Hence, any attacks tampering the Progger code will be detected. A buffer is prepared for new generated provenance logs. A time counter is used to evaluate the age of the buffer. Once the buffer is full or the time counter exceeds a fixed amount of time (i.e., timeout), the provenance logs inside the buffer are fetched into a new created chunk.

After which, the buffer is clear and time counter is reset. The idea of using the time counter together with the buffer size is that the data inside buffer that is not full for a long amount of time (i.e., no arrival of new provenance logs) can be taken out to a chunk in the case that the client machine can be halted anytime (e.g., power disruption).

Fig. 4. Measuring the chunk file by IMA

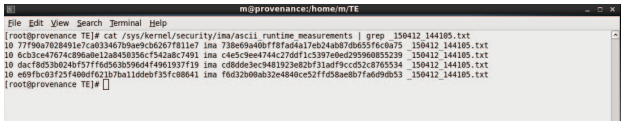

Fig. 5. Detecting tampered chunk file by IMA

B. Create Chunk

Once the buffer is full or when the time counter reaches the timeout, a new chunk is created to store the provenance logs fetched from the buffer. The chunk size is equal to the amount of data fetched from the buffer size. We use the IMA technology to measure the chunk by verifying the PCR-10. Fig. 4 shows that the administrator tried to see the measurement list for the file 150412-144030.txt. As the result shows in Fig. 4, there is no event on this file, i.e., no changes happened for this file.

IMA can help us to track the operations happening to a specific sensitive file (i.e., the chunk). For example, in Fig. 5, the file 150412-144105.txt is a created chunk file. As shown in Fig. 5, this chunk file can be subsequently detected as a tampered chunk. The number '10' located at the beginning of each row in Fig. 5 refers to the PCR-10 that is the default PCR for the IMA measurement. The second and forth columns are hash values. That is, the hash value in the fourth column is the hash value of the chunk file (i.e., *filedata-hash*). The hash value in the second column is called *template-hash* that is the result of concatenating the SHA1 hash value of PCR-0 to PCR-7 (called *boot-aggregation*) with the hash value of the chunk file. This template-hash is calculated by template-hash $= SHA1$(filedata-hash $||$ boot-aggregation). This value will be stored in PCR-10 with boot-aggregation. The file name named by a creation timestamp will help the administrator to find the provenance by specifying the time the administrator is looking for.

We extended our program to utilise an unused/available PCR, i.e., PCR-15 to detect any change that can happen by attackers for the program of Algorithm 1. Hence, any tampering the has occurred to the program will be detected. Meanwhile, the administrator should frequently remotely check PCR-17 for the client machine to make sure that the provenance was created in secure mode. Hence, PCR-17 is related to run time environment used by Intel-TXT technology.

C. Hash Chunk and Transfer the Chunk to the Provenance Server

Hashing the chunk is to provide the integrity for the provenance to help us know whether the chunk is tampered or not. The chunk file is hashed in the client machine once it is created. Periodically, chunks along with their hash values are transferred to the provenance server. The administrator who has access to the provenance server can check the PCR-17 for client machines remotely using TPM-Quote or OpenPTS tool to ensure that the chunks transferred in the trusted environment.

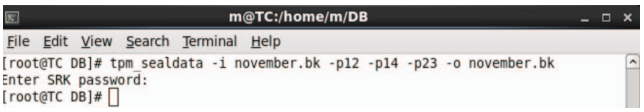

Fig. 6. Seal Data Using TPM

D. Backup Chunks from the Provenance Server in the Backup Server

The provenance server receives data provenance logs from different clients. The aim of this server is to collect the provenance from clients to allow the administrator to examine provenance logs on the clients. All data on provenance and backup servers is encrypted by the Opal technology.

When the administrator wants to check specific provenance data, the administrator will hash the chunk data from the provenance server and compare the hash value with the hash value that is already stored in the server. If the comparison is not matched, a tamper-evidence is found.

The administrator should check the runtime measurement from PCR-17 value frequently on the provenance server. This will allow the administrator to detect any attacks which can happen on the server.

PCR-23 in this provenance server allocated for backup software can ensure that the software works in well-known situations. PCR-10 will help the administrator in this server to know what happened to data provenance files or hash files using IMA features to measure sensitive files like (e.g., Fig. 4).

We transfer the old provenance from the provenance server to the backup server for the longer-term archiving purpose. In the backup server, we use TPM-sealed feature to seal (i.e., encrypt) the backup provenance logs with their hash values using TPM storage root (SRK) key.

This sealed data allows us to protect the data itself by binding the data with PCRs specified (for example., Fig. 6). In Fig.6, if the administrator wants to unseal (decrypt) the data, TPM will unseal it only if the hash values of PCRs-12, PCR-14, and PCR-23 match the same value of these PCRs at the time that the data sealed. This will be secure and helpful for the administrator to secure the data if any changes happened to the data. Since these components or software are measured, hashed, and stored in PCR-12, PCR-14, or PCR-23 in this example.

The backup servers have two databases for chunks and their hash values. Hence, when a change occurs in a chunk or its hash value, the change will be detected by comparing the new hash value of the tampered data with the data stored inside the databases.

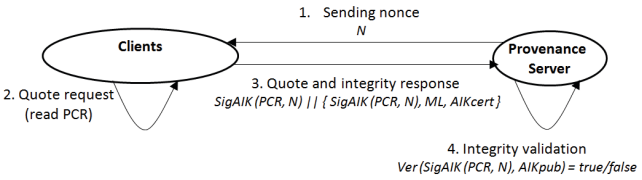

Fig. 7. TPM-Quote operation

E. Remote Attestation

In this subsection, the remote attestation using TPM-Quote is discussed. The TPM-Quote can verify the contents of PCRs remotely. For example, the administrator of the provenance server can remotely check the PCR values on client machines and the backup server (e.g., PCR-8 for bootloader and PCR-23 for the Progger program) and give the tamper-evidence for any changes in the PCRs.

Fig. 7 shows the steps of TPM-Quote recommended by IBM [15]. TPM-Quote allows us to use TPM random number generator to produce a nonce value and Attestation Identity Key (AIK) including the public and private key pair. The provenance server sends the nonce value and AIK to a remote machine (e.g., the backup server and client machine to provide the authenticity between them). Then, the remote machine uses its private key to sign the PCR values which are requested for the attestation by the server, and then replies the signed PCR values to the provenance server. The provenance server verifies the signed PCR values by the public key of the remote machine. This nonce helps prevent the signed PCR values against replay attacks. Using this technique, we achieve the authentication between two machines and ensure that the remote machine runs in a secure mode.

VI. Framework Advantages

With the support of the TPM chip, our framework can provide *admissible*, *complete*, *authentic*, *reliable* and *believable* aspects of tamper-evidence for the data provenance. This section explains advantages of the proposed framework as follows:

A. Data Provenance with Tamper-Evidence

The values of PCRs can obtain the static and dynamic root of trust while the provenance logs are created and stored in a machine, and transferred among machines. If any changes occurred to the data provenance and its related components (e.g., bootloader and OS kernel), the values of the PCRs will change and be provided as the tamper-evidence for the data provenance. Since these PCRs cannot be accessed and tampered by unauthorized users, the provenance logs along with the tamper-evidence can be guaranteed to be admissible, complete, authentic, reliable and believable.

The tamper-evidence can be provided for the following situations:

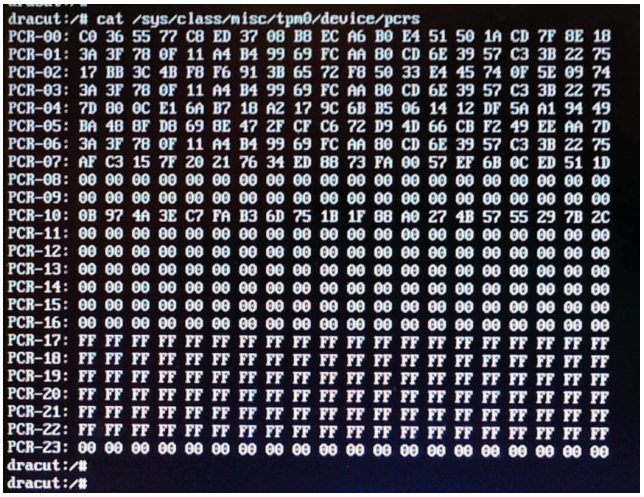

Fig. 8. Screen capture of PCRs value when an attacker changed the default boot

- *Tampered Chunks* – In this framework, chunks represent pieces of provenance logs. Hence, tampered chunks are referred to as tampered provenance logs as well. These tampered chunks can be detected by the IMA module through PCR-10. For example, the chunk file in Fig. 4 is a new created chunk. When an attacker tampers this file (e.g., reading and writing actions), the tamper-evidence can be obtained as shown in Fig. 5.
- *Tampered Provenance Generator* – The program of the provenance generator may also be tampered, but this tampered program can be detected. For example, an attacker changes the Progger code and replaces the new Progger program in a client machine. However, the hash value of the original Progger is stored in PCR-23. Hence, the hash value of the new Progger program is different from the value inside PCR-23 such that the administrator can remotely attest this change.
- *Tampered BIOS Configuration* – An attacker can access the BIOS of a client machine (e.g., by using compromised BIOS password), and then changes the BIOS configuration or update the BIOS firmware. Any changes in the BIOS can be detected by PCR-0 and PCR-1. For example, the BIOS firmware is updated, and then this change of the BIOS version can be detected.
- *Tampered Bootloader* – The attacker can launch boot live attacks e.g., evil maid attack [18]. Any bootloader (including using a changed bootloader) which is not the same as the legitimate bootloader tracked by PCR-4, PCR-5, and PCR-8 will be detected by the TPM chip. For example, Fig. 8 shows the tamper-evidence of using a boot live attack to change the bootloader on a client machine, while the original PCR values of the same machine are shown in Fig. 2.
- *Tampered OS Kernel* – The OS kernel can be changed by an attacker or updated by a new compromised kernel.

Any changes incurred to the OS kernel can be detected by PCR-20.

B. Data Provenance with Integrity Guarantees

In this framework, all chunks of provenance logs are hashed using HMAC on client machines. Any changes can occur to a chunk during the data at transmission and data at rest can be detected by comparing the hash value of the tampered chunk with the original hash value using the HMAC. With this technique, the integrity of the data provenance can be guaranteed.

C. Data Provenance with Confidentiality Guarantees

The provenance and backup servers apply TCG Opal technology to provide full disk encryption such that the confidentiality of the data provenance can be obtained. For the backup server, the TPM-sealed feature using the TPM SRK key is applied to encrypt the provenance logs. Moreover, the confidentiality of communications among the provenance server, backup server, and client machines are guaranteed by TLS connections. This confidentiality can preserve the privacy of the data provenance.

D. Data Provenance with Availability Guarantee

Originally, the provenance logs are generated and stored in the client machines. However, these logs can be damaged and unavailable. For example, a virtual machine stores provenance logs, and then the virtual machine is terminated such that the logs are unavailable. With the proposed framework, the availability of the provenance logs can be guaranteed by storing the logs in the provenance and backup servers for short-term and long-term usage, respectively.

VII. Conclusions and Future Work

We proposed a novel data provenance tamper-evidence and remote attestation framework based on the TPM. Our technique can be applied to cloud computing environments, provides a much-needed tamper-evidence for data provenance and complies with five major rules of evidence including admissibility, authenticity, completeness, reliability, and believability.

This framework also assures the integrity, confidentiality, and availability of provenance logs. Confidentiality and integrity of the logs can be provided by using the features of TPM. The availability of the logs can be achieved by storing the provenance logs in our provenance and backup servers.

Some research directions can be further addressed in future work as follows:

- *Virtual TPM* – A TPM for virtualization technologies is challenging since the TPM chip is originally designed for a physical machine. In the physical machine, the TPM chip can be fully used to measure sensitive components inside the machine. However, virtual machine environments have no actual TPM chips and virtual or software-based TPM chips can be easily tampered.
- *Cloud API* – A cloud Application Programming Interface (API) should be provided to help cloud users access capabilities of the TPM chip. However, this API needs to ensure that cloud users can access the TPM chip even though they usually do not have root access to their subscribed virtual hosts or underlying physical machines.
- *TPM-enabled Private Cloud* – A private cloud environment can be provided a trusted cloud computing environment by using TPM. Open-source cloud software (e.g., OpenStack) can be integrated fully with TPM to provide a trusted environment.
- *Optimisation for Large Scale Environments* – When we implement trusted computing for virtual machines in data centres, there could be a large number of physical and virtual machines – generating a large amount of provenance logs. This large amount of logs transferred among client machines and the provenance server will be a major bottleneck of the whole system. We would need an optimised system for this scenario.

Acknowledgments

This research is supported by STRATUS (Security Technologies Returning Accountability, Trust and User-Centric Services in the Cloud) (https://stratus.org.nz), a science investment project funded by the New Zealand Ministry of Business, Innovation and Employment (MBIE). The authors would like to acknowledge the students and staff of CROW (Cyber Security Researchers of Waikato) - in particular, Jeff Garae, Baden Delamore and Brandon Nicholson - for their support on this work.

References

[1] Trousers. Online [Accessed 29/03/15]http://sourceforge.net/projects/trousers/files/trousers/, March 2015.

[2] Alam Ansari, Arijit Chattopadhayay, and Suvrojit Das. A kernel level vfs logger for building efficient file system intrusion detection system. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 273–279. IEEE, 2010.

[3] Benjamin Bock, David Huemer, and A Min Tjoa. Towards more trustable log files for digital forensics by means of "trusted computing". In *24th IEEE International Conference on Advanced Information Networking and Applications (AINA), 2010*, pages 1020–1027. IEEE, 2010.

[4] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Data provenance: Some basic issues. In *FST TCS 2000: Foundations of software technology and theoretical computer science*, pages 87–93. Springer, 2000.

[5] William Futral and James Greene. *Intel® Trusted Execution Technology for Server Platforms: A Guide to More Secure Datacenters*. Apress, 1st edition, September 2013.

[6] Ashvin Goel, W-C Feng, David Maier, and Jonathan Walpole. Forensix: A robust, high-performance reconstruction system. In *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, pages 155–162. IEEE, 2005.

[7] Ragib Hasan, Radu Sion, and Marianne Winslett. The case of the fake picasso: Preventing history forgery with secure provenance. In *FAST '09 Proccedings of the 7th conference on File and storage technologies*, pages 1–14. Berkeley, CA, USA USENIX Association, 2009.

[8] Ryan K L Ko. Cloud computing in plain english. In *ACM Crossroads*, volume 16, pages 5–6. ACM, 2010.

[9] Ryan K L Ko. Data accountability in cloud systems. In *Security, Privacy and Trust in Cloud Systems*, pages 211–238. Springer Berlin Heidelberg, 2014.

[10] Ryan K L Ko, Peter Jagadpramana, and Bu Sung Lee. Flogger: A file-centric logger for monitoring file access and transfers within cloud computing environments. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 765–771. IEEE, 2011.

[11] Ryan K L Ko, Peter Jagadpramana, Miranda Mowbray, Siani Pearson, Markus Kirchberg, Qianhui Liang, and Bu Sung Lee. Trustcloud: A framework for accountability and trust in cloud computing. In *Services (SERVICES), 2011 IEEE World Congress on*, pages 584–588. IEEE, 2011.

[12] Ryan K L Ko, Bu Sung Lee, and Siani Pearson. Towards achieving accountability, auditability and trust in cloud computing. In *International Workshop on Cloud Computing- Architecture, Algorithms and Applications (Cloudcom 2011)*, pages 432–444. IEEE, 2011.

[13] Ryan K L Ko and Mark A Will. Progger: An efficient, tamper-evident kernel-space logger for cloud data provenance tracking. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 881–889. IEEE, 2014.

[14] Tang Ling. The study of computer forensics on linux. In *Computational and Information Sciences (ICCIS), 2013 Fifth International Conference on*, pages 294–297. IEEE, June 2013.

[15] Hiroshi Maruyama, Taiga Nakamura, Seiji Munetoh, Yoshiaki Funaki, and Yuhji Yamashita. Linux with tcpa integrity measurement. *IBM Japan, Ltd.(January 28, 2003)*, 2003.

[16] Kiran-Kumar Muniswamy-Reddy, David A Holland, Uri Braun, and Margo I Seltzer. Provenance-aware storage systems. In *ATEC '06 Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 4–4. Berkeley, CA, USA USENIX Association, 2006.

[17] Joanna Rutkowska. Evil maid goes after truecrypt! Online [Accessed 29/03/15] http://theinvisiblethings.blogspot.co.nz/2009/10/evil-maid-goes-after-truecrypt.html, 2015.

[18] Joanna Rutkowska, Alexander Tereshkin, and Rafal Wojtczuk. Thoughts about trusted computing, 2009.

[19] Margo I. Seltzer, Peter Macko, and Marc A. Chiarini. Collecting Provenance via the Xen Hypervisor. In *In Proceedings of 3rd USENIX Workshop on the Theory and Practice of Provenance (TaPP '11), Heraklion, Crete, Greece.* Berkeley, CA: USENIX Association, 20-21 June 2011.

[20] SourceForge. Linux integrity subsystem. Online [Accessed 29/03/15] http://linux-ima.sourceforge.net, Feburary 2015.

[21] Chun Hui Suen, Ryan K L Ko, Yu Shyang Tan, Peter Jagadpramana, and Bu Sung Lee. S2logger: End-to-end data tracking mechanism for cloud data provenance. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 12th IEEE International Conference on*, pages 594–602. IEEE, 2013.

[22] Yu Shyang Tan, Ryan K L Ko, and Geoff Holmes. Security and data accountability in distributed systems: A provenance survey. In *Proceedings of the 15th IEEE International Conference on High Performance Computing and Communications(IEEE HPCC13)*, Zhang JiaJie, China, 2013. IEEE Computer Society.

[23] Trusted Computing Group. Enterprise security: Putting the tpm to work. Online [Accessed 29/03/15] http://www.trustedcomputinggroup.org/resources/enterprise_security_putting_the_tpm_to_work, 2015.

[24] Trusted Computing Group. Opal ssc. Online [Accessed 29/03/15] http://www.trustedcomputinggroup.org/resources/storage_work_group_storage_security_subsystem_class_opal, March 2015.

[25] Jing Zhang, Adriane Chapman, and Kristen LeFevre. Do You Know Where Your Data's Been? – Tamper-Evident Database Provenance. In Willem Jonker and Milan Petković, editors, *Secure Data Management*, number 5776 in Lecture Notes in Computer Science, pages 17–32. Springer Berlin Heidelberg, 2009.

[26] Olive Q. Zhang, Markus Kirchberg, Ryan K. L. Ko, and Bu Sung Lee. How to track your data: The case for cloud computing provenance. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 446–453. IEEE, Nov 2011.