

Concerning about Trust of Platform Hardware

Fan Zhang¹, Guoqing Wu¹, Min Jiang², Xiaoli Liu³

¹Computer School, Wuhan University, Wuhan, 430072, China

²Department of Cognitive Science, Xiamen University, Xiamen, 361005, China

³College of Information Science and Technology, Jinan University, Guangzhou, 510630, China

Zhangfan-cixjy@163.com

Abstract

This paper investigates the hardware trust of a trusted computing platform. Initially, some prior researches are discussed. Based on these researches, we point out that hardware trust is the same important as the software trust in a trusted computing platform. Then, we introduce the notion of Hardware Integrity Measurement (HIM), which is similar to the software integrity measurement required by the Trusted Computing Group (TCG). Its purpose is to collect and measure integrity of the whole-platform hardware devices, and the HIM result can be used for the further reporting and attestation. Next, a protocol for hardware integrity measurement is discussed in detail. Finally, we conclude that trust of platform hardware is pretty important, and shares of research attentions are deserved.

1. Introduction

In a trusted computing platform, trust is typically established solely based on software measurements. Few works consider the trust of platform hardware. However, trust of platform hardware is, at least to some extent, the same important as the trust of platform software.

Robert, who is a member of the Technical Committee and Board of Directors of the Trusted Computing Group, proposes in [1] that peripherals are critical to the security, privacy and trust of a computing infrastructure. In that paper, he presents how to secure common devices so that they can be used in a highly trusted computing environment. Reference [2] investigates the “trusted hardware”, the author concludes that with the rapidly increasing reliance on sophisticated hardware components, the attacks to hardware devices, which will in turn result in security vulnerabilities of the whole platform, are becoming more and more serious and troublesome. So, some initiative and ongoing researches are needed to solve this problem. In [3], the authors successfully read data from memory in an unusual way which can bypass the software-based access control. Based on the experiment, they draw the conclusion that hardware devices, especially those contain embedded memory, need necessary security reinforcements. With these reinforcements, both data stored in these devices

and communications among these devices can be initiatively encrypted so as to avoid illegal information theft. A more dangerous example is the attack to the Trusted Platform Module (TPM). As we know, in a trusted computing platform, TPM is the root trust of the platform and it must be absolutely trusted. However, TPM is also at risk in the hardware attack scenario. For example, reference [4] analyzes the trusted platform communication. The authors present us how they sniff data from the unprotected channel (or bus/ or interface) between a TPM and the host in which the TPM is embedded. After the successful experiment, the authors further propose that next they will try to initiatively inject fake data/commands into the channel so as to cheat TPM, thus breaking the platform integrity completely. In [5], the author discusses some widely-used peripherals that can lead to various security dangers. Finally, worst of all, a new kind of attack has emerged: the hardware-based attack to software, which is directly launched by a malicious hardware device. In [6], the authors show us how they design and implement a malicious processor to provide hidden services. Based on the malicious processor, attackers can escalate privilege, login through backdoor and steal password without being detected by traditional software-based access controls.

From what discussed above, we can conclude that trust of platform hardware is also fundamental to a trusted computing platform. Thus, to measure and collect integrity information of platform hardware devices, we propose the idea of Hardware Integrity Measurement (HIM), and also introduce a related protocol.

This paper is organized as follows. Section 2 describes the HIM. Section 3 discusses the related protocol for HIM. Section 4 shows part of the experiment result based on the trusted computer designed by us [7] [8]. Section 5 draws the conclusion.

2. Hardware Integrity Measurement

2.1. What is the hardware integrity measurement

In the former section, we point out that hardware trust of a trusted computing platform is the same important as the software trust. However, how can we say the software stack of a platform is trusted? This is done through the

Software Integrity Measurement (*SIM*), which is based on the “trust chain” [9] defined by TCG: $CRTM \rightarrow BIOS \rightarrow OSLoader \rightarrow OS \rightarrow Applications$.

Similarly, to establish the hardware trust of a platform, we propose the conception of Hardware Integrity Measurement (*HIM*). The *HIM* is also a chaining-style: $TPM \rightarrow Motherboard \rightarrow CPU \rightarrow Memory \rightarrow \dots$, its purpose is to firstly collection information from the vulnerability points of platform hardware devices, which can be located in the generic risk model of peripherals shown in [1], and then to determine whether these measured devices are trusted or not. More details about *HIM* definitions and how to build an *HIM* framework can be found in [11]. We do not repetitively discuss here.

2.2. Classifications of untrusted hardware

Next, in order to measure hardware integrity, what is the untrusted hardware should be defined first. Generally, untrusted hardware can be classified into three types.

(1) **Type I**: a hardware device is made by a malicious vendor, or there is at least one malicious integrated circuit (IC) provider in its IC supply chain. This kind of device necessarily contains malign hardware functions (intentionally-inserted malign circuits) or firmware functions (malign firmware) [6] [2]. Different from type II, even if nothing of the device is modified, it is not trusted at all.

(2) **Type II**: a hardware device is maliciously attacked by an attacker. For example, the firmware attack discussed in [12]. Different from type I, the device itself is certainly made by a trusted vendor, so if its integrity is guaranteed, it is trusted. However, due to malicious modifications to its integrated circuits (or firmware and so on) by attackers, it becomes from trusted to untrusted.

(3) **Type III**: the hardware configuration of platform is changed. Unauthorized variations of platform hardware devices may also lead to security problems [2] [5]. For example, an inside attacker can physically disconnect a harddisk from a trusted computing platform in which the harddisk is rigorously protected, and then connect it to a common platform. If data stored in the harddisk are not encrypted, the attacker can steal confidential information.

In section 3, we will introduce how to detect these untrusted hardware devices based on an *HIM* protocol.

2.3. Security enhancements to traditional hardware devices

Why hardware attacks are so dangerous and troublesome? Existing works conclude that traditional hardware devices do not pay enough attentions to the security when they are designed. They also point out traditional hardware devices need security enhancements so that they can be used in a highly trusted computing

environment [1]. The following are two examples of hardware security enhancements:

(1) A digital certificate is needed to indicate that the hardware itself is made by a trusted vendor. Thus a malicious vendor can no longer provide hardware devices containing malicious circuits to users.

(2) Cryptography related enhancements are needed for trusted hardware. According to references [3] [4], both data stored in a hardware device, and communications among hardware devices should be encrypted, so cryptography enhancements are necessary.

To sum up, security enhancements are necessary for traditional hardware devices. Without these enhancements, directly using them in a platform may cause various security holes, especially in the situation that hardware attacks becomes more and more widespread and harmful.

3. Protocol for hardware integrity measurement

In this section, a protocol for *HIM* is presented in detail by referencing [10].

3.1. Background of the protocol

Suppose hardware devices are secured as the independent root of trust as [1]. If this cannot be achieved currently, at least, they should have the security enhancement as assumption 3.1.

Assumption 3.1: Suppose each platform hardware device has three security enhancements: (1) a self-measurement chip, (2) a register named *Device Status Register* (*DSR*) and (3) an encryption engine, where:

(1) The self-measurement chip is responsible for collecting information

$Self_MeaInfo = (I_1, I_2, I_3, \dots, I_n)$ from vulnerability points of hardware devices based on the generic risk model of peripherals shown in [1]. By comparing the collected information against the expected one, TPM can determine whether the measured device is trusted or not.

(2) The *DSR* is cleared to zero automatically when the device powers up. After the self-measurement chip finishes collecting the measurement information $Self_MeaInfo = (I_1, I_2, I_3, \dots, I_n)$, it stores the information into *DSR* like this: $DSR = SHA1(I_n \parallel \dots \parallel SHA1(I_2 \parallel (SHA1(I_1 \parallel 0))))$.

(3) The encryption engine is responsible for encryption/decryption operations.

We say that TPM can determine whether a device is attacked or not by comparing information collected against the expected value, but where is the expected one? In our implementation, the expected information is stored in a *Standard Platform Device Table* (*SPDT*). The

SPDT consists of all of the configuration information of hardware devices of the platform, and it is stored into the TPM by security administrators in advance. Table 3.1 shows parts of the *SPDT*.

Table 3.1 Illustration of *SPDT*

<i>device_info</i>		<i>expected_info</i>	<i>extended</i>
EliteGroup Mainboard	/	ID=02/19/2002-SiS-745-6A6IUE19C-00	Not used
Jet BIOS (Test)	ECS	Revision 0000.0015	Not used
AMD CPU	ECS	Athlon(tm) processor, CPUID=000006A0h	Not used
Seagate Harddisk	ATA	Capacity=40G, C=77545, H=16, S=63	Not used

In an *SPDT*, there are many entries. Each entry is related to the corresponding expected measurement information of a hardware device. We call each entry a Configuration (shorted for *Conf*). Formally, each *Conf* are expressed as the form of $Conf = (device_info, expected_info, extended)$, where:

device_info = (*device_name*, *interface*) indicates the name of the device, and the interface to which it connects. For example, in table 3.1, there are four entries, which are related to “Motherboard, BIOS, CPU and Harddisk” respectively. The last entry is related to a harddisk named “Seagate Harddisk”, and it is connected to an “ATA” interface.

expected_info indicates the expected measurement result of the device, including ID, Version, SHA1 of firmware (not implemented yet) and so on. Not all of the information in *expected_info* is shown in table 3.1 because of space limitations.

extended is reserved for further use. Whether to use and how to use it is decided by administrators.

More details about *SPDT* can be found in [11].

3.2. Hardware integrity measurement protocol

The protocol is as follows and contains six main steps:

- (1) TPM: creates a random number n , and then sends the self measurement command *Self_MeaCmd* and n to the hardware to be measured: $Challreq(Self_MeaCmd + n)$.
- (2) Hardware: does the self measurement and then collects self-measurement information $Self_MeaInfo = (I_1, I_2, I_3, \dots, I_n)$. After this, it uses the private key K_{priv} to sign *DSR* and n : $SIG = sig(DSR, n)_{K_{priv}}$.
- (3) Hardware: sends *SIG* together with the self-measurement information *Self_MeaInfo* to TPM $Attesresp(SIG, Self_MeaInfo)$.
- (4) TPM: validates the signature

$SIG = sig(DSR, n)_{K_{priv}}$.

(5) TPM: validates the self-measurement information $Self_MeaInfo = (I_1, I_2, I_3, \dots, I_n)$.

(6) TPM: validates the trust of platform hardware by comparing *Self_MeaInfo* against *SPDT*.

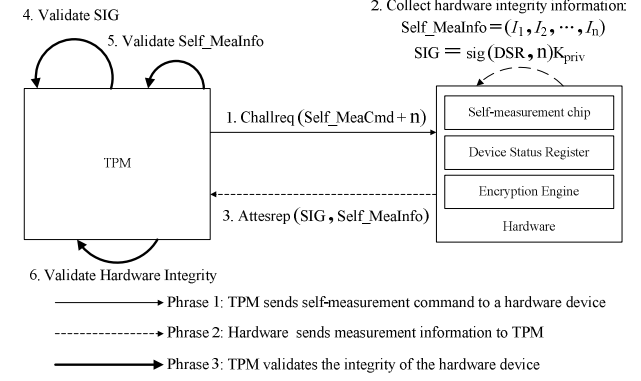


Figure 3.1 Hardware Integrity Measurement Protocol

3.3. Analysis of the protocol

Figure 3.1 shows the protocol. Generally, the security protocol can be divided into three major phases:

Phase (i): TPM initiates the hardware measurement by sending *Self_MeaCmd* to the device to be measured (step (1) in section 3.2).

Phase (ii): The device starts the self-measurement, and then returns the collected information *Self_MeaInfo*, together with signature *SIG*, to TPM (step (2) ~ step (3)). In this phase, the main purpose is to ensure that TPM receives the exact self-measurement information without being tampered with.

Phase (iii): TPM verifies the authenticity of *Self_MeaInfo* and *SIG* respectively (step (4) ~ step (5)), and then checks the integrity of the device by comparing measurement information received against *SPDT* in step (6).

Phase (i) is easy to understand, so we give a detailed discussion of Phase (ii) and Phase (iii) here.

Analysis of Phase (ii): In this phase, the main purpose is to ensure that what TPM receives is clear, fresh and not tampered with, so three attacks should to be prevented:

- (a) Replay: an attacker may intercept the measurement information, and then resends the measurement information before the hardware is compromised to TPM.
- (b) Tampering: an attacker may maliciously tamper the measurement information, and then resends the tampered information to TPM.
- (c) Masquerading: an attacker may send the measurement information of another device which is not compromised to TPM.

Now, let's take a look at how the security protocol can defend these three attacks in the following.

(a) Suppose an attacker adopts the Replay Attack. Because n is generated by a true RNG, it can not be predicated. So, in step (4), TPM will find the n is illegal.

(b) Suppose an attacker adopts the Tampering Attack, then *Self_MeaInfo* or *DSR* must have been tampered with. If *DSR* is tampered with, the attacker cannot rebuild legal $SIG = sig(DSR, n)_{K_{priv}}$ due to the randomness of n and the security of K_{priv} . Thus in step (4), TPM will find *DSR* is attacked. On the other hand, if *Self_MeaInfo* is tampered with, by reconstruct *DSR'* according to *Self_MeaInfo*, and then compare *DSR'* against *DSR*, TPM will find the tampering to *Self_MeaInfo* in step (5).

(c) Suppose an attacker adopts the Masquerading Attack. If step (5) also passes, TPM can trust what it receives is the exact configuration of the hardware without being tampered with. Subsequently, TPM will compare the hardware measurement information against the expected one stored in the *SPDT*. Thus the masquerading attack can be found in step (6), because the hardware configuration information of other devices will never occur in the *SPDT* of this platform. How TPM can find this will be introduced in Phase (iii).

Analysis of Phase (iii): In section 2.2, untrusted hardware devices have been classified into three types. Here, let us see how these three types of untrusted hardware can be detected.

(a) Detecting untrusted hardware of Type I. By verifying the digital certificate of the hardware device, whether the vendor is trusted or not can be determined.

(b) Detecting untrusted hardware of Type II. Type II is due to the malign modification to a hardware device. Intuitively, this is because the measurement result of a device is different from the expected one. Formally, let *DetectedDEVICE* be the set of the detected hardware devices of the platform, we have: $\exists device.device \in DetectedDEVICE \wedge Conf(device) \notin SPDT$. In which $Conf(device) \notin SPDT$ means that the *Conf* of the being measured device does not match any entry of *SPDT* (see table 3.1). The more fine-grained reason is that: for the collected measurement information (*device_meainfo* and *expected_meainfo*), we have: $\exists Conf \in SPDT.device_meainfo \in Conf \wedge expected_meainfo \notin Conf$.

(c) Detecting untrusted hardware of Type III. Type III results from the unauthorized changes of hardware in a platform. Commonly, hardware configuration changes include two possibilities.

■ A hardware device that does not belong to the platform is connected. Intuitively, this means that the device being measured cannot match any entry of the *SPDT*. Formally, let $device \in DetectedDEVICE$, we have: $\exists device \forall Conf.get_device_info(device) \notin Conf$, where *get_device_info* returns the measured *device_info* of a *device* (see Table 3.1).

■ A critical hardware device of the platform is intentionally disconnected. It is the contrary of the former one. Intuitively, there is an entry existed in the *SPDT*, representing that a certain hardware device should be connected to the platform, however, this entry can not be matched by any detected device. Formally, let $device \in DetectedDEVICE$, we have: $\forall device \exists Conf.get_device_info(device) \notin Conf$.

4. Experiment

Though related hardware security enhancements cannot be completely achieved, the protocol introduced in section 3 is partly a paper-design one. However, we still partly implement hardware integrity measurement as an illustration. Our experiment is based on the trusted computer [7] [8] designed by us. Figure 4.1 shows a part of the measurement result. More details about figure 4.1 can be found in [11]. Figure 4.2 shows the access control by considering hardware integrity.

```

BIOS: Trusted
Vendor: Jetway Information Security Corp.
Version: Revision 0000.0015
SHA1: OK

.....
Harddisk1: Warning
ID: ST340015A
Serial Number: 5LAAD1J8
CHS: 77545, 18, 63
!!!Warning: MBR is corrupted!!!
ErrorCode: 03
Description: this disk may connect to other platforms.

.....
USB: Warning
Vendor: USB 2.0 (HS) Flash Disk
Serial Number: 000000000003D5
!!!Warning: a USB disk was connected at 19:23!!!

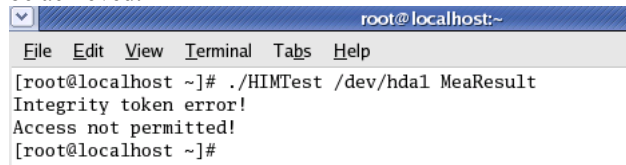
```

Figure 4.1 Hardware integrity measurements

Normally, hardware integrity ought to be measured in two stages. One is when a platform powers up. In this stage, some “fix devices” are measured. The other is after the OS is booted up. In this stage, some “non-fix devices” are measured. In figure 4.1, the BIOS and Harddisk1 are measured in the former stage, while the USB is measured in the latter. Measurements in both stages have their specific purposes, and they cannot take the place of each other.

In the experiment, because currently hardware devices have no bounded digital certificate, we cannot determine whether a device is made by a trusted vendor or not. Meanwhile, because we can neither read the content of firmware (it is under the protection of copyrights and laws) nor get its hash, the firmware’s integrity cannot be

verified, either. However, both of the validations are fundamental to the hardware trust. In the future, necessary security enhancements should be considered when designing a hardware device, so that the validations can be achieved.



```

root@localhost:~
File Edit View Terminal Tabs Help
[root@localhost ~]# ./HIMTest /dev/hda1 MeaResult
Integrity token error!
Access not permitted!
[root@localhost ~]#

```

Figure 4.2 Access control considering hardware integrity

Figure 4.2 shows the access control by considering hardware integrity. In figure 4.1, we measured that the Harddisk1 may be connected to other platform, thus some data stored in it may be tampered with. Here, we just suppose that the whole harddisk is not trusted for simplicity. With the measurement result, when program HIMTest wants to access data from /dev/hda1, the security-enhancement module finds that /dev/hda1 is not trusted from the *HIM* result (MeaResult), so denies the access request. This is only the coarse-grained access control by considering hardware integrity. Fine-grained access control involving *HIM* will be discussed in the other paper.

5. Conclusions and future work

In this paper, we firstly point out that trust of platform hardware is fundamental to a trusted computing platform. Then, we propose the notion of Hardware Integrity Measurement (*HIM*), whose purpose is to measure and collect the trust information of platform hardware devices. Finally, a protocol for *HIM* is presented.

In [6], the authors show how to design and implement a malicious processor, and then use the malign processor to attack a system. This kind of hardware-based attack can easily bypass traditional access control and, therefore, is hard to detect. The authors in [6] further discuss the possibilities of using testing, reverse engineering and fault-tolerance to defend against the hardware-based attacks. However, they draw the discouraging conclusion that even the seeming effective fault-tolerance cannot be directly applied into practices, because the cost may be too high. So, our notion of measuring hardware integrity can be regarded as its supplement.

Of more importance, as existing work shows, hardware-based attacks like [6] can easily defeat the traditional security models, because these models are software-based without taking hardware into consideration. In the future, we will try to improve the traditional security models by involving trust of platform hardware.

Acknowledgement

This work is supported by Chinese National 863 High-tech Research and Development Projects, Grant No. 2007AA01Z85.

References

- [1] R. Thibadeau, Trusted Computing for Disk Drives and Other Peripherals, IEEE Security & Privacy, vol. 4, no. 5, Sep, 2006, pp: 26-33.
- [2] C. E. Irvine, K. Levitt, Trusted hardware: can it be trustworthy?, Proceedings of the 44th annual Conference on Design Automation, Annual ACM IEEE Design Automation Conference, 2007, pp:1-4.
- [3] D. Samyde, S. Skorobogatov, R. Anderson et al., On a New Way to Read Data from Memory, Proceedings of the First International IEEE Security in Storage Workshop, 2002, pp:65-69.
- [4] K. Kursawe, D. Schellekens, B. Preneel, Analyzing trusted platform communication, In ECRYPT Workshop, CRASH – Cryptographic Advances in Secure Hardware, September 2005.
- [5] I. Arce, Bad Peripherals, IEEE Security & Privacy, vol. 3, no. 1, 2005, pp: 70–73.
- [6] S. T. King, J. Tucek, A. Cozzie et al., Designing and Implementing Malicious Hardware, USENIX'08, http://www.usenix.org/event/leet08/tech/full_papers/king/king.pdf.
- [7] H.G. Zhang, G.Q. Wu, Z.P. Qin et al., A New Type of Secure Microcomputer (Chinese), J. Wuhan Univ. (Nat. Sci. Ed.), vol.50, no. S1, Oct., 2004, pp: 1-6.
- [8] H.G. Zhang, Y.Z. Liu, F.J. Yu et al., A New Type of Embedded Security Module (Chinese), J. Wuhan Univ. (Nat. Sci. Ed.), vol 50, no. S1, Oct, 2004, pp: 7-11.
- [9] TCG Specification Architecture Overview, Specification Revision 1.2, 28, Apr. 2004, http://www.trustedcomputinggroup.org/downloads/TCG_1_0_Architecture_Overview.pdf.
- [10] R. Sailer, X. L. Zhang, T. Jaeger et al., Design and Implementation of a TCG-based Integrity Measurement Architecture, In Proceedings of the 13th USENIX Security Symposium, Aug, 2004.
- [11] F. Zhang, G. Q. Wu, J. Tao, M. T. Yuan, Trust of Hardware, Symposia of the IEEE 2008 International Conference on Embedded Software and System, July 29-31, 2008.
- [12] J. Hendricks, L. V. Doorn, Secure Bootstrap Is Not Enough: Shoring up the Trusted Computing Base, Proceeding of the 11th workshop on ACM SIGOPS European workshop: beyond the PC, 2004.