Tux: Trust Update on Linux booting

No Author Given

No Institute Given

Abstract. Preserving integrity is one of the essential requirements in trusted computing. However, When it comes to system update, even with the state-of-the-art integrity management system such as OpenCIT cannot properly manage integrity. This is because the updates are not transparent to the remote attestation server and the integrity value is not updated according to the updates.
This paper presents Trust update on Linux booting, TUX. TUX collaboratively manages the integrity along with the kernel update, so that the update is transparent to the attestation server. With TUX, we can successfully maintain trust for the managed machines, even with frequent OS kernel updates. Also, TUX guarantees robust verified and measured boot to safeguard the integrity of a system's booting process.

Keywords: Open CIT · Intel TXT · Integrity · Security Update · TPM· Trusted Computing Group· Grub · Shim · UEFI Secure boot · Linux Kernel.

1 Introduction

As lethal security attacks, such as Spectre[5] and Meltdown[6], arise, the system administrators (admins) have been deploying security updates and patches continuously[10][8]. Considering the amount of efforts that have been putting into the security updates, the changing integrity of a system due to the updates has not drawn much attention. Maintaining integrity and applying security update share the same goal, securing the system; however, they take contradicting approaches. The former tries to prevent the system from unintended modifications, but the latter seeks to mitigate the vulnerabilities by modifying the system. This paradox makes guaranteeing the integrity challenging because the integrity is inevitably changed when the system update is conducted. Also, managing changed integrity can be frustrating when updates are frequent as is today.

Although several trust/integrity management schemes are presented[11][2][15][1], these schemes cannot manage changing integrity according to the frequent updates. For example, Open CIT[11] implements remote attestation defined by TCG[13]. In Open CIT, the remote attestation server stores the whitelist value of its managed systems (clients) and validates the platform integrity based upon the whitelist value. However, Open CIT fails to validate integrity whenever a client is updated. This is because the Open CIT server is not aware of the client's update state. Consequently, the server tries to validate the client with outdated integrity information and inevitably fails to attest the managed machine.

This paper presents Trust update on Linux booting (TUX), a novel integrity management scheme which can adequately manage the integrity of Linux booting along with frequent updates. First of all, TUX makes kernel update transparent to the attestation server by integrating the update server to the Open CIT. Thus, Open CIT can manage kernel updates for each client, and Open CIT is fully aware of the updates of its clients. TUX allows Open CIT to maintain up-to-date whitelist values; thereby TUX eliminates attestation failures caused by the unnoticed updates.

Secondly, TUX provides a robust verified boot which can verify the integrity of entire booting process using the TUX-secure boot (TS-Boot). TS-Boot is a combination of the UEFI secure boot, Shim bootloader, and Grub bootloader. Using the TS-Boot, TUX supports integrity verification using firmware key and TPM. The core idea is to measure entire booting process using TPM and compare it with known-good integrity value using the secure boot to check whether if the booting process has maintained its integrity. TUX will immediately halt the booting when the two values (TPM measurements and the known-good integrity) do not match. Additionally, this allows the distributors to manage client's booting policy, and restrict the attacks from compromised booting process.

The key contributions of this work are:

- TUX extends Intel Open CIT enabling integrity management under frequent updates.
- TUX hardens Linux booting by consolidating secure boot and measured boot.

This paper consists of the following sections. Section 2 provides background knowledge for contemporary integrity management schemes. Section 3 presents some threat models. Section 4 illustrates the design considerations of TUX. Section 5 describes some implementation issues in TUX. Section 6 evaluates TUX. Section 7 discusses further considerations not covered in this paper. Finally, we conclude the paper in section 8.

2 Background

2.1 TPM

Trusted Platform Module (TPM)[14] is an independent hardware which is designed to measure integrity safely. TPM was proposed by the Trusted Computing Group (TCG)[13] to provide a trusted execution environment that can measure and store the integrity. TPM measures platform integrity with its cryptography processor and stores it to the Platform Configuration Registers (PCRs). PCRs are tamper proof registers that stores integrity hash values. Also, TPM creates chain-of-trust with *extend* operation, which allows TPM to calculate new integrity value from the old value.

2.2 tboot and LCP

Trusted Boot (tboot)[16] is a bootloader module which adopts Intel Trust Execution Technology (TXT)[2] to support measured and verified boot. Using the TXT, tboot measures platform's pre-boot environment from the hardware initialization to the loaded kernel binary. Also, tboot verifies platform integrity with the Launch Control Policy(LCP), a pre-defined list of known-good values of allowed properties. LCP defines integrity of the BIOS configurations, bootloader, OS kernel, and LCP itself with its three policies: SINIT, Platform configuration (PCONF), and Measured Launch Environment (MLE). tboot verifies the integrity by comparing TPM measurements against the LCP and state the booting is trusted when all three policies are met.

Although the tboot guarantees the robust integrity of the pre-boot environment, it can be restricted when frequent updates are conducted. The LCP is bound to a specific state of the platform, and thus, when the platform is updated, the LCP also needs to be updated according to up-to-date status. LCP update regarding a small number of devices may be manageable, but the difficulty of management will increase exponentially as the number of devices to manage and issued updates increase. Therefore, tboot is not suitable for an environment with frequent updates. Additionally, tboot works as a bootloader module, making it vulnerable to rootkit attacks[3].

2.3 UEFI secure boot

UEFI secure boot[15] is a security standard defined in the UEFI BIOS. The secure boot enables robust firmware level verified booting by employing asymmetric-key cryptography. Secure boot mandates every booting components, such as bootloaders and OS kernels, are digitally signed with the distributor's private key. At the boot time, the UEFI secure boot decrypts the signatures of the binaries using the distributor's public key stored in the firmware's db. Then, the UEFI secure boot verifies integrity by comparing the hash of the loaded binary against the decrypted signature. When they match, secure boot confirms the integrity of the file and executes the binary.

For Linux to support UEFI secure boot, it uses a first-stage bootloader, Shim[9]. This is because Grub, a bootloader to load Linux kernel, is not supported by the UEFI secure boot. Therefore, at booting, secure boot loads the Shim first, then Shim loads Grub bootloader. Also, Shim is capable of binary verification with the firmware key using the shim_lock verification.

Adopting UEFI secure boot can result in several advantages. First, it is supported by most of the platforms. Second, secure boot provides simple update procedure for the clients, compared to the Intel TXT. When a component is updated, the distributor only needs to sign the updated file with the private key and deploy it. Also, the client only needs to update the binary, no LCP update required.

However, UEFI secure boot is not satisfactory to guarantee thorough trusted booting. This is because the current secure boot does not provide integrity verification for some vital booting components, such as the kernel booting parameters,

grub commands, and grub configuration files. Also, it does not support measured environment for the remote attestation.

2.4 Trusted Grub

Trusted Grub[1] is a bootloader for Linux that supports measured launch. Trusted GRUB adopts TPM to establish the chain-of-trust of the booting stages. Trusted GRUB measures thorough booting process by measuring not only the kernel binaries, but also the GRUB configuration files, and loadable grub modules.

Unfortunately, the official Trusted Grub only supports TPM v1.2. Note that the TPM v1.2 only support SHA-1 hash algorithm, which is vulnerable to hash collision attacks[12] and deprecated. Furthermore, Trusted Grub only performs measurements and leaves verification up to the remote attestation. Thus, it cannot guarantee the integrity of the platform on run-time as Intel TXT or UEFI secure boot.

2.5 Open CIT and remote attestation

Intel Open Cloud Integrity Technology (CIT)[11] is an Intel's implementation of TCG's remote attestation. Open CIT claims the integrity of the platform by verifying the TPM measurements measured using the Intel TXT. Also, Open CIT uses a unique application called *Trust Agent* to enable remote attestation. Trust Agent resides on the managed machines, and it implements *quote operation* to pass on the TPM measurements to the attestation server for verification. Also, it provides web-based monitoring tool, which allows users to initiate and manage remote attestation efficiently.

The remote attestation begins with the machine registration to the remote attestation server. At the registration, the attestation server imports measured values as the initial known-good values (whitelist) from each machine. Then, the imported values become the comparison basis to check the machines' integrity. After the registration, the server can perform remote attestation by requesting TPM measurements from the local hosts. When the Trust Agent receives the request, it uses quote operation to pass on the measurements. Then, received measurements are compared against the whitelist to verify the integrity. If the values do not match, Open CIT states broken integrity.

However, Open CIT has some limitation when it comes to an environment with frequent updates. This is because the update status of the clients are not transparent to the attestation server and the Open CIT can establish whitelist only by importing. Namely, when the update is conducted on the client side, the Open CIT encounters broken integrity since it does not have information about the update. Also, for every update, the attestation server must re-import updated values to re-establish comparison basis. This can be a considerable overhead for the Open CIT when updates are frequently conducted.

3 Threat Model

3.1 Subverting Open CIT

We assume that the adversary can update OS kernel and perform measured boot. In this case, the adversary can thwart remote attestation with the following scenario. First, the adversary updates the victim with a compromised kernel such as old kernel that includes known vulnerabilities. After the reboot, the TPM measurements are also compromised since it measures compromised kernel. Then, since the update was conducted, victim admin tries to import new measurements to the Open CIT server to re-establish the comparison basis. Note that Open CIT has no means to distinguish between legitimate updates from malicious updates. Therefore, Open CIT administrator, unfortunately, has no way but to import new measurement values to the server. Now Open CIT has compromised whitelist, and the remote attestation is performed using the compromised whitelist values. Therefore, the adversary can run malicious kernel even though the Open CIT confirms that the system is trusted.

3.2 Circumventing verified boot

The pre-boot environment can be a security hole[4], and this can be mitigated by verified boots, such as tboot and UEFI secure boot. However, [3] describes an attack that can undermine tboot using a thin layer of virtual TPM. Also, it is possible to modify the booting process using the Grub command line even though the UEFI secure boot is on. Both limitations are because they do not validate the executed Grub commands and modules. Thus, an adversary, who has physical access to Grub, can compromise Grub commands to run malicious modules, breaking the integrity in Linux booting process.

4 Design of TUX

In this section, we present the design of TUX, Trust update on Linux booting. TUX suggests novel integrity management scheme, which can maintain thorough integrity verification along with frequent updates. TUX extends the Intel Open CIT by taking advantage of its robust attestation scheme and consolidating it with kernel update and whitelist management.

Assumptions. Before diving into TUX design, there are several assumptions to be considered. First, the attestation server owner, also known as the *TUX owner*, is considered as the administrator for update and management. Second, TUX verifies the integrity of Linux booting process only. This is because we believe that booting process is essential to establish the trusted environment and after boot environment can be handled with operating system's powerful security solutions. Third, TUX assumes that the attestation server and its components are trustworthy and secure. Fourth, TUX assumes that the TUX owner holds manifest of the entire booting process of each managed machine. Booting

process includes all the hardware configurations, loaded software, executed grub commands, etc. IT department usually manages hardware information of every computer because they are assets of an organization, and thus, it is reasonable to assume that the owner knows the hardware configurations of the managed machines. Finally, we assume that all of the managed machines have TUX owner's private key pre-installed.

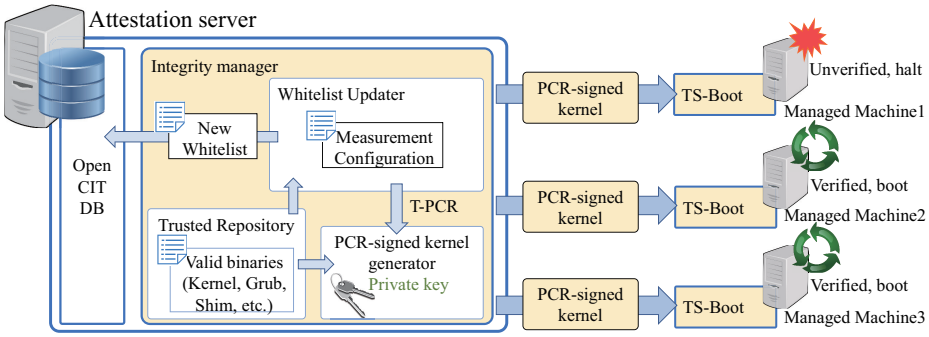


Fig. 1: The design of TUX

Figure 1 illustrates the design of TUX. As shown in the figure, TUX consists of three components: Integrity manager, PCR-signed kernel, and TUX-Secure boot. First, The TUX Integrity manager resides in Open CIT's remote attestation server to manage integrity along with updates. The TUX Integrity manager is composed of the Trusted repository, Whitelist updater, and PCR-signed kernel generator. Second, the PCR-signed kernel is a unique kernel that is deployed from the attestation server, which holds known-good integrity value inside its signature. Lastly, The TUX-Secure boot (TS-boot) is located at the managed machine. TS-boot provides solid chain-of-trust and integrity verification by consolidating UEFI Secure boot, Shim, and Grub. By using the PCR-signed kernel, TS-boot can verify entire booting process without LCP.

4.1 TUX Components

Here, we describe the three major components of TUX: Integrity manager, PCR-signed kernel, and TS-Boot.

Integrity manager TUX presents Integrity manager, robust integrity management and kernel update component integrated to Intel Open CIT's attestation server. With the integrity manager, Open CIT server transparently updates the managed machines and prevent un-intended attestation failure due to the updates. The Integrity manager focuses on whitelist value management and the PCR-signed kernel generation for integrity value deployment. As shown in Figure 1, Tux Integrity manager is consisted of three units to manage integrity: Trusted repository, Whitelist updater, and PCR-signed kernel generator.

Trusted repository is an update repository, which stores valid kernel and bootloader binaries that are not stated as End-Of-Life (EOL). Binaries stored in the Trusted repository are under the control of TUX owner, allowing the TUX owner to take full control of the software used on the managed machines. Also, TUX Trusted repository provides the binaries to Whitelist updater and PCR-signed kernel generator for integrity management and deployment. Note that the attestation server is secure; hence, the files in the trusted repository are also safe and trusted.

Whitelist updater calculates and updates per-machine integrity values within the attestation server. Firstly, the Whitelist updater dynamically calculates whitelist values and the Trusted PCR value (*t-PCR*) according to the Measurement configuration and the binaries stored in the trusted repository. Measurement configuration is a manifest of instruction sequence resembling the entire booting process, and the *t-PCR* is a known-good integrity value calculated by extending entire booting process declared in the Measurement configuration. Secondly, the Whitelist updater updates whitelist values to the Open CIT database and pass on *t-PCR* value to the PCR-signed kernel generator.

Using the TUX Whitelist updater, TUX maintains up-to-date whitelist with the values produced by the Whitelist updater. Because the whitelist values are self-produced and updated within the attestation server, TUX prevents attestation failure after system updates. Figure 2A illustrates a comparison between regular open CIT remote attestation and TUX remote attestation.

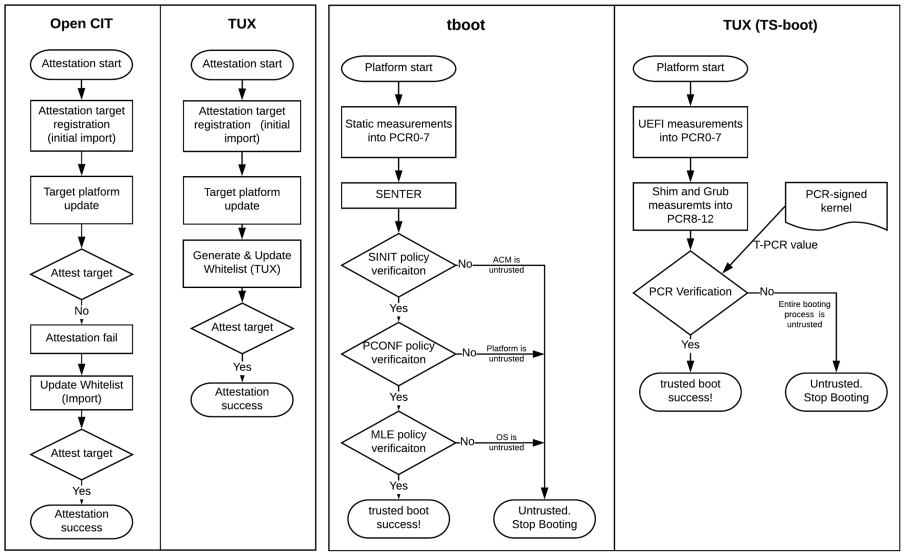


Fig. 2: Comparison between TUX, Open CIT, and tboot

PCR-signed kernel generator generates PCR-signed kernel by leveraging Secure boot sign tool[7]. PCR-signed kernel generator takes *t-PCR* from the Whitelist updater and encrypts it with the TUX owner's private key. The encrypted value gets stored into the digital signature and attached to the kernel binary from the Trusted repository. Then, the finalized the PCR-signed kernel is deployed to the local machine. The following section will further explain the PCR-signed kernel in detail.

PCR-signed kernel The PCR-signed kernel is a unique kernel that contains integrity information of the managed system. As shown in the Figure 3, a PCR-signed kernel is comprised of a kernel binary and a digital signature, which holds encrypted *t-PCR*. Using the PCR-signed kernel, TUX can easily deploy known-good integrity value to the managed machines. TUX mandates each managed machine to use the PCR-signed kernel. The PCR-signed kernel is used in PCR-verification to verify the integrity of the system's booting process, instead of LCP. Details about the PCR-verification is mentioned at the following section.

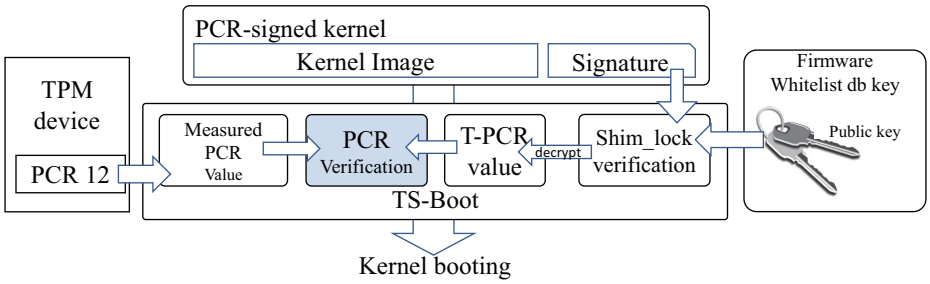


Fig. 3: The PCR-verification components and its process

TUX-secure boot (TS-boot) TS-boot is a batch of booting schemes collaborate to enable verified and measured booting scheme. It is composed of UEFI secure boot, Shim, and Trusted Grub. Note that the Grub is the most popular Linux bootloader, and thus, TUX maximizes Linux compatibility. By utilizing TS-boot, TUX can guarantee the robust and strict integrity of Linux booting process. TS-boot takes place in the managed machine as shown in Figure 1.

TS-boot provides three services to guarantee the integrity of managed machine's booting process. First of all, TS-boot uses firmware keys to verify booting components. TS-boot adopts *Shim*, the first-stage bootloader, to perform firmware level integrity check. Shim bootloader defines a function to use firmware key verification, called shim_lock verification. Using the shim_lock verification, TS-boot verifies every component used in the Linux booting (e.g., Grub, OS kernel, and Initrd) with keys stored in the firmware.

Second, TS-boot enforces integrity measurement using TPM from the beginning of booting to OS kernel. As soon as the booting starts, the UEFI secure boot measures hardware and BIOS configurations to first eight PCR registers respectively. Then, the UEFI BIOS verifies and executes Shim, Shim measures and verifies Grub, OS kernel, and initrd binary before execution. Finally, the Trusted

Grub measures grub configuration file and executed grub commands. Note that all values are extended from the measurements from the previous stage.

Lastly, TS-boot introduces *PCR-verification* to guarantee the strict and robust integrity of the entire booting process. PCR-verification verifies the integrity of the booting process by comparing TPM measurements of the entire booting process against the *t-PCR* in the kernel's signature. Figure 3 shows the overall PCR-verification process. TS-Boot measures all the hardware configuration, executed binaries, and GRUB commands are extended to the PCR12 so that it can capture the integrity of the entire booting process. Therefore, if anything of the above components or sequences change, the value of the PCR12 also changes. To achieve *t-PCR* value, TS-boot decrypts the signature using the TUX owner's public key stored in the firmware. Then, by comparing the value of the PCR12 against the *t-PCR* TS-boot verifies the thorough integrity of the booting process. TS-Boot allows kernel execution only when the two values are matched, meaning that the hardware configurations and software executions are intact and not compromised. Figure 2B presents a comparison of boot-time verification with tboot and TUX.

5 Implementation

5.1 Integrity manager

PCR calculation TPM 2.0 provides multiple hash algorithms for measuring the integrity of the platform. For TUX, the SHA-256 algorithm was used for calculating the *t-PCR* value and the up-to-date whitelist. This is because, recently, SHA-1 collision[12] was announced, and most security vendors suggest not to use SHA-1 but use SHA-2 algorithms for robust security.

For PCR calculation, TUX owner first configures the Measurement configuration. Then, the whitelist values and *t-PCR* is calculated according to the Measurement configuration by using the PCR extend operation. The PCR extend operation is defined as follows:

$$PCR_{new} = H(PCR_{previous} \oplus H(Data))$$

where H() is a TPM's hash function, which calculates SHA-256 hash, and $\oplus$ is a concatenation operator. The PCR extend operation takes the value from the previous stage and data of the current stage to extend integrity values. By following the operations defined in the Measurement configuration.

PCR-signing PCR-signed kernel generator utilizes Secure Boot signing tool, SB-signing tool[7], to make a digital signature. The generator takes three inputs: OS kernel binary, *t-PCR* value, and TUX owner's private key. To generate the signed kernel, PCR-signed kernel generator first encrypts the *t-PCR* with the owner's private key, instead of the digest of a file, and creates a digital signature. Then the signature is attached to the kernel binary creating the PCR-signed kernel. The produced PCR-signed kernel is deployed to the managed machine

and update the system. Note that the PCR-signed kernel generator runs inside the attestation server, which should be safe and trustworthy. Therefore, vital assets such as TUX owner's private key are secure.

5.2 TS-Boot

PCR Read Reading the PCR on the run is essential for PCR-verification. However, reading the PCR value of TPM 2.0 was not implemented in the Shim, where PCR-verification is conducted. Thus, PCR read operation was added to the Shim. Note that TPM 2.0's PCR implementation is more sophisticated than TPM 1.2.

PCR Usages There are multiple PCRs inside TPM. In our implementation, we measure the platform integrity to PCRs from PCR0∼PCR12. Table 1 explains each PCR usages.

PCR	Contents	Measurment host
PCR0-7	BIOS and hardware configurations.	UEFI Secure boot
PCR8	Executed Grub commands.	Trusted Grub
PCR9	Executed Modules from Trusted Grub.	Trusted Grub
PCR10	Trusted Grub binary.	Shim
PCR11	Kernel and initrd	Shim
PCR12	Entire booting process.	UEFI Secure boot, Trusted Grub, and Shim.

Table 1: Platform Configuration Register (PCR) Usages

TPM measurements For integrity verification, additional TPM measurements are added to Shim and Trusted Grub. First of all, we added measurements for the Grub, kernel, and initrd in the Shim to accurately measure each binary to the PCRs described in Table 1. Also, more measurements are added to extend entire booting process to the PCR12. To begin with, Shim merges the hardware PCR values to the PCR12. PCR0∼PCR7 are consecutively read and extended to PCR12. Then, along with the booting process, all the TPM measurements are additionally extended to PCR12, generating the integrity hash of the entire booting process. All the measurements are measured with the SHA-256 algorithm for the brevity.

PCR-verification To implement PCR-verification, we adopt shim_lock verification, which is part of the shim_lock protocol. The shim_lock protocol allows Grub to communicate with the Shim, which has access to firmware database.

The PCR-verification is located at the *linuxefi* command, which is a grub command to load kernel and initrd, to verify the kernel and the boot process before kernel execution. After the kernel is loaded to the memory, it is sent to Shim using the shim_lock protocol. Then Shim measures the binary to the

PCR12. After the measurement, as shown in the following pseudo code, the value of PCR12 is read and compared against the decrypted signature, the *t-PCR*. If the verification fails, the kernel execution halts and TUX discards booting.

```
pcrval;
TPM_readPCR(12,pcrval); //read pcr12
...
// decrypt signature using the firmware db key.
// compare read pcr12 and the decrypted value.
if (check_db_cert(cert, pcrval)) {
    console_notify(L"PCR Verification Success :)\n");
    return EFI_SUCCESS;
} else {
    console_notify(L"PCR Verification Fail\n");
    return EFI_FAIL;
}
```

PCR-verification provides robust integrity, forbidding all changes during the booting. Also, it uses one value, *t-PCR*, to verify whole booting process, taking advantage of the chain-of-trust.

5.3 Kernel Update Procedure

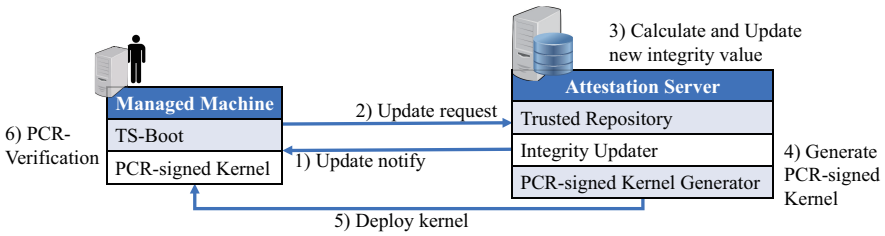


Fig. 4: The TUX update procedure

TUX defines specific kernel update procedure for the managed machines to properly properly manage integrity according to the updates. The overall update process of TUX is shown in Figure 4.

First of all, kernel update repository, defined in each managed machine's sign.list, is set to TUX's Trusted repository. Thus, when a new kernel is released, the user is notified of the new kernel from the Trusted repository.

Second, the update process is triggered by the user. When the update is requested, the attestation server checks the registered information of the managed machine. Hence, the server is aware of managed machine's update state transparently.

Third, the attestation server's Integrity manager calculates Trusted PCR value (*t-PCR*) the Measurement configuration and the requested binary. Note that the new whitelist value is derived and updated at this stage, and thus, the attestation server has up-to-date known-good value.

Fourth, the *t-PCR* is passed on to PCR-signed kernel generator. With the *t-PCR* and the valid kernel binary from the Trusted repository, the PCR-signed kernel is generated.

Finally, the PCR-signed kernel is distributed over the network. The local admin receives and installs the new kernel, finishing the update procedure of TUX. Also, the Grub configuration is updated to adopt new kernel.

5.4 Installation of TS-Boot

The initial installation of TS-Boot is vital because it set the initial root-of-trust. At the initial installation, the IT department (TUX owner) gathers essential information about the managed machine. The initial installation only needs to be performed once.

Initial installation of the TS-boot is conducted in following steps:

First, the managed machine admin checks hardware requirements of its platform. A managed machine should have TPM 2.0, and UEFI BIOS with secure boot.

Second, the managed machine admin enrolls the TUX owner's public key to the firmware of the platform and enables the secure boot. This step is essential to provide root-of-trust and firmware level integrity verification.

Third, to achieve UUID of the boot partition, the managed machine admin installs normal Linux distribution without TS-Boot.

Fourth, the managed machine admin performs initial registration to the attestation server. Here the attestation server imports PCR values and UUID.

Fifth, the TUX owner generates the whitelist values, the PCR-signed kernel, and corresponding Grub configuration file using the TUX integrity manager. The Grub configuration file is driven from the TUX owner's Measurement configuration.

Sixth, the TUX owner uses integrity manager to update the whitelist values and deploy TS-boot with PCR-signed kernel and Grub configuration.

Finally, the managed machine admin installs TS-boot by updating Shim, Grub, Grub configuration file, and kernel image.

6 Experiments

6.1 TPM Measurements

TUX TS-boot measures PCR values along with the booting process. The measurement values of the managed machine are stored in the PCRs. The measured values are then compared with the whitelist values in the attestation server. If the measurements do not match, the attestation server states the machine is not trustworthy.

In this experiment, we show changes in PCR values according to hardware, bios configuration, and executed kernel. Figure 5 shows the result of four different settings. The settings are configured as follow: **A**) Hardware 1 + Secure boot on

+ kernel version vmlinuz-4.4.0-104, **B**) Hardware 1 + Secure boot on + kernel version vmlinuz-4.4.0-109, **C**) Hardware 1 + Secure boot off + kernel version vmlinuz-4.4.0-109, **D**) Hardware 2 + Secure boot on + kernel version vmlinuz-4.4.0-109.

The comparison between configuration **A** and **B** shows PCR change due to the kernel update. Using the PC 1, we updated the kernel from vmlinuz-4.4.0-104 to vmlinuz-4.4.0-109. This caused changes in PCR8, 11, and 12 since the binary content and the kernel loading commands line[1] is modified. Also, PCR11 for configuration **B** and **D** remains the same because they use same Kernel and initrd binary, version 4.4.0-109.

The comparison between configuration **B** and **C** shows PCR changes due to BIOS modification, turning Secure boot on and off. This experiment causes changes in PCR7 and 12. Also, turning Secure boot off disables shim_lock verification. Thus PCR11 was not measured.

The comparison between **C** and **D** illustrates different hardware(PC) having different PCR values for PCR1 and PCR5. Finally, the PCR1, 5, and 11 are different between **A** and **D** since the kernel is changed as well as the hardware.

Additionally, we can see that all configurations have the same values for PCR 10 and 9 because we have used same Grub binary and modules. Furthermore, PCR12, which is used for PCR-verification, is unique for all configurations.

A. PC1+SB on+Kernel 104	B. PC1+SB on+Kernel 109	C. PC1+SB off+Kernel 109	D. PC2+SB on+Kernel 109
Bank/Algorithm: TPM ALG SHA256(0x000b)			
PCR_00: 05 48 02 7e c	PCR_00: 05 48 02 7e c	PCR_00: 05 48 02 7e cf	PCR_00: 05 48 02 7e cf
PCR_01: f1 67 99 3b a	PCR_01: f1 67 99 3b a	PCR_01: f1 67 99 3b a5	PCR_01: c7 84 e6 09 94
PCR_02: 3d 45 8c fe 5	PCR_02: 3d 45 8c fe 5	PCR_02: 3d 45 8c fe 55	PCR_02: 3d 45 8c fe 55
PCR_03: 3d 45 8c fe 5	PCR_03: 3d 45 8c fe 5	PCR_03: 3d 45 8c fe 55	PCR_03: 3d 45 8c fe 55
PCR_04: f5 f8 1f 6b 5	PCR_04: f5 f8 1f 6b 5	PCR_04: f5 f8 1f 6b 5b	PCR_04: f5 f8 1f 6b 5b
PCR_05: de 89 35 69 c	PCR_05: de 89 35 69 c	PCR_05: de 89 35 69 c2	PCR_05: 39 55 01 58 89
PCR_06: 3d 45 8c fe 5	PCR_06: 3d 45 8c fe 5	PCR_06: 3d 45 8c fe 55	PCR_06: 3d 45 8c fe 55
PCR_07: 25 c0 b3 ce 4	PCR_07: 25 c0 b3 ce 4	PCR_07: 47 d9 c1 f4 d9	PCR_07: 25 c0 b3 ce 45
PCR_08: 63 81 11 5c d	PCR_08: f4 1e 86 df 9	PCR_08: b1 5d 09 67 39	PCR_08: a6 fe 12 0a 0f
PCR_09: e2 fa 1b a3 f	PCR_09: e2 fa 1b a3 f	PCR_09: e2 fa 1b a3 f9	PCR_09: e2 fa 1b a3 f9
PCR_10: 0b 74 50 53 8	PCR_10: 0b 74 50 53 8	PCR_10: 0b 74 50 53 8e	PCR_10: 0b 74 50 53 8e
PCR_11: 53 45 a7 13 8	PCR_11: 79 bd 24 78 8	PCR_11: 00 00 00 00 00	PCR_11: 79 bd 24 78 88
PCR_12: 92 5a 80 6e c	PCR_12: 31 68 59 c3 e	PCR_12: 10 5e fc 8c b1	PCR_12: 76 fc 4d 87 a9

Fig. 5: PCR measurements for different configurations.

6.2 PCR-verification

In this experiment, we show robust security enforcement using the PCR-verification. Adjustment in hardware, BIOS configuration, booting commands, and binaries cause modification of PCR12. When booting components are modified, the PCR12 value changes, as shown in Figure 5.

Figure 6a shows modification in Grub configuration file (grub.cfg), which also caused Measured value to change. Thus, when *t-PCR* and modified PCR12 value is compared, TS-boot halts booting, concluding that the integrity of the booting is broken. Figure 6b shows verification fail message produced by the Shim.

[1] the grub's kernel loading command includes the version of the kernel.

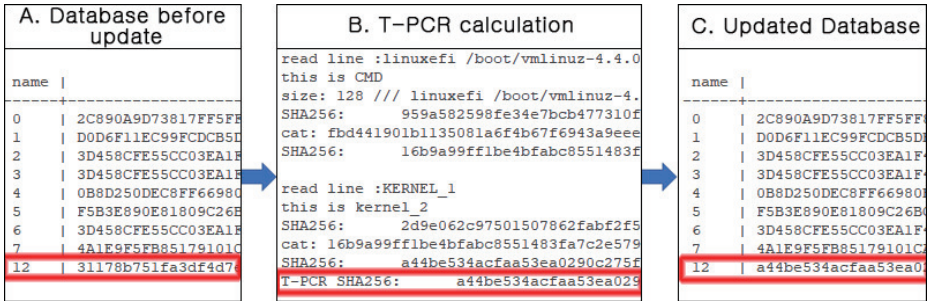
(a) Modified Grub.cfg (b) Fail message from Shim

Fig. 6: PCR-Verficiation

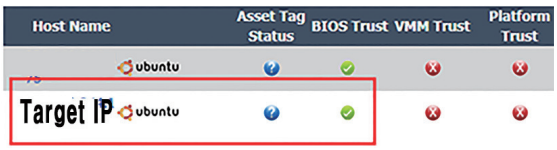
6.3 whitelist update

In this experiment, we show whitelist update, integrity value generation, and attestation result using up-to-date whitelist value. Note that, we experimented only with the PCR12 (*t-PCR*) to simplify the experiment. To update the whitelist, we defined a Measurement configuration according to Ubuntu 16.04's booting process. Using the Measurement configuration, we generated *t-PCR*. Then, we updated whitelist in the Open CIT database with SQL update statement.

Figure 7a-A shows whitelist database before the update, Figure 7a-B illustrates *t-PCR* calculation using different kernel version, and Figure 7a-C is the whitelist database which is updated with the *t-PCR*. Finally, Figure 7b illustrates attestation success for the BIOS, which refers to the booting process, after the update.



(a) whitelist database updated.



(b) Attestation Success for BIOS.

Fig. 7: Attestation with updated whitelist

7 Discussion

7.1 Recovery from broken integrity.

Integrity can be broken when the user unintentionally modifies the booting component. In this case, the system should re-install the TS-boot and the kernel.

Even when the integrity is broken, TUX provides simple recovery procedure compared to Intel TXT. Table 2 compares recovery procedure of TUX and tboot.

	TUX	**tboot**
1	Restore original TS-boot binaries (Grub, kernel, etc.).	TPM clear to take ownership.
2	Restore Grub configuration	Re-install SINIT.
3	-	Restore original binaries (Grub, kernel, etc.).
4	-	Restore Grub configuration.
5	-	Reset Launch Control Policy.
6	-	Update TPM NVRAM.

Table 2: Recovery procedure comparison of TUX and tboot.

7.2 Roll-back and multiple kernel support.

TUX can support version roll-back and multiple kernels. As long as the requested version is stored in the Trusted repository, a user can request specific kernel version to roll-back. Moreover, TUX supports using multiple versions of kernels at the same time by defining multiple kernel information in the Grub configuration file. At boot-time, a user can select kernel version to boot from the Grub menu.

Also, TUX can support attestation for multiple kernels. When the client is attested, it can provide kernel version information in the quote to the attestation server. Then the server updates the whitelist according to the provided kernel version with the Integrity manager. After the whitelist update, the attestation server can successfully attest the client's integrity.

Additionally, TUX is robust to roll-back attacks, in which the adversary replacing a trusted version with an old vulnerable version. TUX trusted repository holds trusted versions of Linux kernel and TUX is capable of version-based remote attestation as mentioned above. Thus, the roll-back attack can be detected by remote attestation.

7.3 TUX owner's key

TUX requires every managed system to have TUX owner's key installed to the firmware and use it as a root-of-trust. We assume that the firmware is protected and difficult to be modified, and thus, the key is safe. Even though the key is modified and the booting is compromised, remote attestation can detect and state the system untrusted.

8 Conclusion

This paper presents the design and implementation of TUX, the Trust update on Linux booting. With TUX, we handle integrity seriously as much as updates to secure the system. TUX integrates update server to the Open CIT to transparently manage system updates. Also, TUX performs remote attestation

with the up-to-date whitelist, eliminating misguided attestation failure caused by updates.

Furthermore, TUX provides robust verified and measured boot with TS-Boot. TS-Boot verifies thorough integrity regarding the Linux booting process using the PCR-verification. With TS-Boot, a system can validate hardware environment, BIOS, bootloaders, executed Grub commands, and OS kernel binary. Thus, by utilizing TUX, we can successfully manage the integrity of the platform along with updates.

References

1. Daniel Neus: Rohde-schwarz-cybersecurity/trustedgrub2: Tpm enabled grub2 bootloader (Jun 2017), https://github.com/Rohde-Schwarz-Cybersecurity/TrustedGRUB2
2. Futral, W., Greene, J.: Intel Trusted Execution Technology for Server Platforms: A Guide to More Secure Data centers. Apress, Berkely, CA, USA, 1st edn. (2013)
3. Sharkey, J.: Breaking Hardware-Enforced Security with Hypervisors (2016), black hat USA, https://www.blackhat.com/docs/us-16/materials/us-16-Sharkey\-Breaking-Hardware-Enforced-Security-With-Hypervisors.pdf
4. Kleissner, P.: Stoned bootkit (2009), black Hat USA, http://www.blackhat.com/presentations/bh-usa-09/KLEISSNER/BHUSA09-Kleissner-StonedBootkit-SLIDES.pdf
5. Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., Yarom, Y.: Spectre attacks: Exploiting speculative execution. ArXiv e-prints (January 2018)
6. Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., Hamburg, M.: Meltdown. ArXiv e-prints (January 2018)
7. Michal Sekletar: Sbsigntool github, https://github.com/msekletar/sbsigntool
8. Microsoft: Protect your windows devices against spectre meltdown (Apr 2018), https://support.microsoft.com/ko-kr/help/4073757/protect-your-windows-devices\\-against-spectre-meltdown
9. Red Hat Bootloader Team: UEFI Shim loader, https://github.com/rhboot/Shim
10. Redhat: Rhsa-2018:0093 - security advisory. https://access.redhat.com/errata/RHSA-2018:0093 (Jan 2018)
11. Savino, R.: Open cit 3.2.1 product guide · opencit/opencit wiki. https://github.com/opencit/opencit/wiki/Open-CIT-3.2.1-Product-Guide (February 2018)
12. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full sha-1. Advances in Cryptology- CRYPTO 2017 Lecture Notes in Computer Science (2017), https://eprint.iacr.org/2017/190
13. Trusted Computing Group: Tcg architecture overview, version 1.4. https://trustedcomputinggroup.org/tcg-architecture-overview\\-version-1-4/ (August 2007), (Accessed on 02/07/2018)
14. Trusted Computing Group: Tpm main specification (October 2014), available https://trustedcomputinggroup.org/tpm-main-specification/
15. UEFI: Unified extensible firmware interface specification. http://www.uefi.org/sites/default/files/resources/UEFI\%20Spec\%202_6.pdf (January 2016)
16. Wei, J., Wang, S., Sun, N., Qiaowei, R.: Trusted boot — sourceforge.net. https://sourceforge.net/projects/tboot/