Improving security of the FreeBSD boot process

Kornel Duleba, Michal Stanek
Semihalf
mindal@semihalf.com, mst@semihalf.com

Abstract

This paper describes recent security additions in the FreeBSD boot process.

TPM 2.0 devices are now supported in FreeBSD. They are most often referred to in the context of Measured Boot, i.e. secure measurements and attestation of all images in the boot chain. The TPM 2.0 specification defines versatile HSM devices which can also strengthen security of various other parts of your system. We describe the basic features of TPM 2.0 which we have made available in FreeBSD. We also mention some caveats and shortcomings of the technology which may have contributed to the limited adoption of TPMs.

The article includes practical TPM use cases such as hardening Strongswan IPSec tunnels by performing IKE-related cryptographic operations within the TPM, using private keys which never leave the chip. Another example will be sealing secrets in TPM NVRAM with specific boot measurements (hashes) stored in PCR registers so that the secrets are locked in to a specific boot chain.

Furthermore, we describe our recent work on UEFI Secure Boot support in the FreeBSD loader and kernel. The loader is now able to parse UEFI databases of keys and certificates which are used to verify a signed FreeBSD kernel binary, using BearSSL as the cryptographic backend. In addition, we employ FreeBSD veriexec capability to verify various userland binaries and configuration files. We have extended veriexec with the ability to use UEFI trust anchors as base for veriexec manifest verification.

1 Introduction

There are several concepts which can improve security of the FreeBSD boot process. Measured Boot may be employed by using a hardware device called TPM (Trusted Platform Module) to make measurements of the code executed at startup and verify its integrity. Another, similar technology is UEFI Secure Boot, which makes use of cryptographic certificates stored in UEFI bootloader to authenticate the image being loaded at each step of the boot process.

After the system has booted, there may be a need for verified execution i.e. authenticating binaries and configuration files being loaded at runtime. Veriexec is a kernel component which provides such guarantees by performing file integrity checks during certain system calls such as open and exec. This ensures that an active adversary has not modified critical system files.

In this paper we present our work on TPM 2.0 support in FreeBSD, as well as integration of UEFI Secure Boot with Veriexec to allow authentication of the loader, kernel, modules and sensitive system files. We also describe several TPM features which may be used by FreeBSD userspace software for secure computation and storage. Finally, we discuss open issues concerning the current implementation of the above technologies in FreeBSD, while listing possible ways of improving it in the future.

2 TPM overview

TPM (Trusted Platform Module) is a technology specified by Trusted Computing

Group which allows systems to validate basic boot properties, such as the integrity of the boot images executed at startup. TPMs are usually discrete hardware components which provide secure storage and secure cryptographic computation. They may be classified as HSMs (Hardware Security Module) or as a limited TEE (Trusted Execution Environment). There have been two major versions of TPM specification - 1.2 and 2.0. Version 1.2 has been supported in FreeBSD since 2010. Version 2.0 is not backwards-compatible with 1.2 and introduces numerous improvements over the initial specification. We created and merged a driver for TPM 2.0 at the end of December[1]. Combined with IBM TSS library[2], which provides a software API to user applications, nearly all TPM features can now be used in FreeBSD.

TPM is most commonly used for Measured Boot. During system startup, each image which takes part in the boot chain measures (hashes) the next image before passing execution to it. The resulting hash is saved to a PCR register in the TPM. Apart from binary images, various other data can be included in the measurements - for example EFI environmental variables. The PCR registers can only be reset by firmware or during reboot. Write operation to PCR does not replace the hash value but only extends it. The TPM takes the new hash measurement, concatenates it with the current PCR value and stores the hash of the concatenation in the PCR. This way a hash chain is formed which serves as proof of a particular chain of boot images, loaded in a particular sequence. As a result, the OS may query the PCR value from the TPM and compare it to a known good hash which represents the desired system state. On mismatch, the OS finds out that some of the boot code was modified and may take appropriate action. In addition, an event log is created. It contains a list of names of hashed objects and their fingerprints. One can compare the digests against expected values and replay the process of extending PCRs to verify the log integrity.

Another TPM feature strictly linked to Measured Boot is attestation. A remote server may request measurements of the local platform as proof that the system can be trusted, before providing up keys, secrets or other information. This is achieved with the Quote operation, which consists of the TPM signing its PCR values along with a nonce provided by the requester to prevent replay attacks. The TPM signs the PCRs using a private key embedded in the chip which is unavailable to software. Several key types are supported such as RSA and ECC.

The measurements are normally made by both the firmware and OS. Depending on UEFI implementation usually all loaded binaries (*.efi files) and some variables are measured. Currently the FreeBSD loader and kernel do not support the extend operation, however depending on UEFI implementation, boot1.efi and loader.efi may be measured. There is currently no possibility to read the event log associated with the measurements in FreeBSD. Userspace software may, however, request signed PCR measurements from the TPM, which were made by UEFI.

3 TPM usage in FreeBSD

Apart of Measured Boot image measurements, there are several interesting features of the TPM which can be used in FreeBSD:

- Strongswan IPSEC

  Strongswan is a multiplatform IPsec VPN implementation available in FreeBSD. It offers optional secure storage of private keys and certificates on smartcards and TPMs. We have created patches for Strongswan which enable FreeBSD users to take advantage of the TPM plugin to secure their VPN networks. We describe an example Strongswan configuration using TPM in later sections.

- Secure NVRAM storage

  TPMs typically contain limited NVRAM memory in the order of several kilobytes for storing secrets and other data. NVRAM

data in the TPM may be locked to a specific password, pin code or a specific set of PCR values. Some operating systems support storing disk encryption keys in TPM NVRAM. Common examples are Bitlocker and LUKS. FreeBSD GELI does not support storing encryption keys in the TPM.

- Data sealing

  The TPM offers a Seal operation which allows encryption of arbitrary user data by the TPM using embedded symmetric keys. The data may also be sealed to a particular set of PCR values. Note that TPM cryptographic operations are usually much slower than software. Therefore, for large data it is best to use the TPM to seal a symmetric key which may then be kept on vulnerable storage in encrypted form and later used to decrypt the actual user data in software.

- SSH key storage

  It is possible to store SSH private keys in the TPM NVRAM. A third-party library is required which works as a PKCS11 provider. We did not evaluate this option.

4 TPM limitations and caveats

One of the biggest shortcomings of discrete TPM chips is their performance, especially regarding asymmetric cryptography. In case of Infineon SLB9665 the operation of signing SHA256 digest with a RSA-2048 key takes approximately 0.15s. TPMs are not supposed to be cryptographic accelerators. Better performance may be achieved with a non-discrete TPM implementation. As an example, fTPM 2.0 is available as part of Intel Management Engine in 5+ gen processors. Another firmware implementation of TPM was created by Microsoft[3]. It leverages ARM TrustZone and is used in all ARM mobile devices running Windows.

Another limitation is the size of NVRAM storage. Usually it is in the order of several kilobytes, which is hardly enough to store a few RSA keys. Infineon SLB9665 contains exactly 7206 bytes of NVRAM.

Since the TPM 2.0 specification was introduced fairly recently, some of the software interacting with it may still lack important features. For example IBM TSS library lacks support for encrypted communication with the TPM, a feature that is defined in the specification. Also, there is no in-tree support for TPM in OpenSSL.

Most TPM implementations are closed-source with limited public information available on the specifics of the hardware. This is often the reason for common mistrust in TPMs and their vendors. Some major companies, however, have released their open-source TPM implementations including Microsoft[4] and Google.

Bus communication with the TPM is not encrypted by default. The most common bus used with discrete TPMs is LPC on Intel systems (I2C/SPI on others) which is a low speed bus integrated in the Southbridge. The TPM specification allows encryption of sensitive parameters within a command (not the entire bus traffic) with AES-CFB together with HMAC for authentication and protection against tampering, as well as rolling nonces for protection against replay attacks. The keys for encryption and authentication must initially be uploaded to the TPM on a trusted system.

TPM boot integrity measurement functionality contains several caveats. It is less flexible than Secure Boot as every TPM object protected by PCR-based authorization policy must be replaced each time a boot image or configuration included in PCR measurements has been updated. Furthermore, security of Measured Boot relies on the assumption that PCR registers are only reset when the entire system is reset. If there is any vulnerability in the software, OS or TPM which allows the attacker to issue a reset to the TPM without resetting the rest of the system, then the PCRs get zeroed out and the attacker may then replay the proper hash sequence into the PCRs, spoofing the integrity

measurements and possibly unlocking secrets. This has been the major method used in attacks on Measured Boot[5].

5 UEFI Secure Boot and the FreeBSD loader

UEFI Secure Boot is a method of ensuring that only authenticated boot images are allowed to run on the system. The security goals of Secure Boot are similar to TPM Measured Boot, however there are significant differences between the two technologies. TPM Measured Boot performs hash measurements of subsequent boot images and configuration, without disturbing the boot process. It is up to the OS and user software to verify PCR measurements and take appropriate action. For example, one could include remote attestation, in which a remote server verifies signed PCR values against its own database using Quote operation[6]. The TPM may also release secrets based on specific PCR values.

UEFI Secure Boot, on the other hand, never passes execution to an unauthenticated image. Furthermore, instead of only hashing, it verifies certificates and signatures of the images it loads, with trust anchors and revoked certificates stored in flash - DB and DBX variables respectively.

UEFI Secure Boot provides an important advantage over TPM Measured Boot from the system administration point of view. In Measured Boot, PCR values will change on firmware update. This requires that any software and/or OS which relies on specific PCR values must be updated with new configuration. Thanks to the usage of certificates in Secure Boot, one must simply sign the new firmware image and it will be successfully authenticated. Certificate chains can be employed to allow for handling of complex certificate trust schemes.

Another significant difference is that UEFI Secure Boot does not require the use of TEE (Trusted Execution Environment) for secure operations, although some ARM systems make use of TrustZone technology along with Arm Trusted Firmware to separate sensitive operations from main execution mode. TPM, however, may be regarded as a TEE as it offers both secure cryptographic operations and isolated secure storage.

The FreeBSD EFI loader is a regular EFI application which can be verified by UEFI as part of Secure Boot. The simplest approach of including the FreeBSD kernel in Secure Boot is to bundle the kernel into the loader binary and make UEFI verify the whole package. The loader would find the kernel image embedded in its binary and pass execution to the kernel. The ideal solution, however, would be to modify the EFI loader to be able to read UEFI certificate lists and verify the kernel on its own. This would ensure the proper chain of trust where each boot element is clearly separated and verifies the next image to be loaded. It would also eliminate the necessity to rebundle the loader with the kernel on each kernel update.

The latter approach is implemented with the help of Juniper veriexec[7].

6 FreeBSD Veriexec and UEFI

Veriexec is a system developed by Juniper Networks which allows restricting execution only to verified code. It uses a signed manifest which is a list containing a path and hash for each verified file. The manifest is passed over to /dev/veriexec char device by /sbin/veriexec[8]. Veriexec consists of two parts, mac_veriexec[7] - a kernel module that manages verification during runtime and libsecureboot[9] which is used by the loader.

Up to now the authenticity of the manifest could only be verified using an embedded trust anchor.

In our contribution we extend it to load trust anchors from UEFI and implement a revocation system which is also based on data stored in firmware[10].

Manifest verification is done in userspace, which opens a time window for someone to inject their own malicious data. To fix this issue, we introduce a kernel module which loads a manifest based on data passed by the loader through environmental variables[11]. This allows verifying the signature of the manifest in kernel space (by the loader) which eliminates previous security concerns.

7 Acknowledgements

The work on TPM 2.0 driver support, libsecureboot and UEFI Veriexec support in FreeBSD was initiated and sponsored by Stormshield who also provided reference hardware.

Work on this paper was sponsored by Semihalf.

8 References

[1] FreeBSD TPM 2.0 driver `https://svnweb.freebsd.org/base/head/sys/dev/tpm/`

[2] IBM TSS `https://sourceforge.net/projects/ibmtpm20tss/`

[3] fTPM: A Software-only Implementation of a TPM Chip `http://ssaroiu.azurewebsites.net/publications/usenixsecurity/2016/ftpm.pdf`

[4] Microsoft TPM 2.0 reference implementation `https://github.com/Microsoft/ms-tpm-20-ref`

[5] Attacks on Measured Boot `https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-han.pdf`

[6] IBM TPM attestation `https://sourceforge.net/projects/ibmtpm20acs/`

[7] mac_veriexec `https://svnweb.freebsd.org/base/head/sys/security/mac_veriexec/`

[8] /sbin/veriexec `https://reviews.freebsd.org/rS344567`

[9] Juniper libsecureboot `https://reviews.freebsd.org/rS344565`

[10] Veriexec UEFI support `https://reviews.freebsd.org/D19093`

[11] Veriexec in-kernel manifest parsing `https://reviews.freebsd.org/D19281`