SSN Lab Assignment:

UEFI Secure Boot

Introduction

UEFI Secure Boot ensures by means of digital signatures that the code that loads the operating system and the operating system itself have not been tampered with. It is used by Microsoft to guarantee safe startup but is also used by Ubuntu and Fedora.

In this assignment, you will learn how Ubuntu implements secure booting of the Linux kernel. You will need access to a Ubuntu machine that was installed with UEFI Secure Boot enabled (like your desktop PC).

Ubuntu provides the following description of its implementation of UEFI Secure Boot at https://wiki.ubuntu.com/UEFI/SecureBoot:

"In order to boot on the widest range of systems, Ubuntu uses the following chain of trust:

1. *Microsoft signs Canonical's "shim" 1st stage bootloader with their "Microsoft Corporation UEFI CA". When the system boots and Secure Boot is enabled, firmware verifies that this 1st stage bootloader (from the "shim-signed" package) is signed with a key in DB (in this case "Microsoft Corporation UEFI CA")*

2. *The second stage bootloader (grub-efi-amd64-signed) is signed with Canonical's "Canonical Ltd. Secure Boot Signing" key. The shim 1st stage bootloader verifies that the 2nd stage grub2 bootloader is properly signed.*

3. *The 2nd stage grub2 bootloader boots an Ubuntu kernel (as of 2012/11, if the kernel (linux-signed) is signed with the "Canonical Ltd. Secure Boot Signing" key, then grub2 will boot the kernel which will in turn apply quirks and call ExitBootServices. If the kernel is unsigned, grub2 will call ExitBootServices before booting the unsigned kernel)*

4. *If signed kernel modules are supported, the signed kernel will verify them during kernel boot"*

The overall task for this assignment is to verify that Ubuntu actually uses the following chain of trust and that the signatures are correct. The above steps will be referred to as "Step 1", "Step 2", etc. in the text that follows.

Please make your report detailed and include all proofs!

Start by reading the "Introduction" section of the page on Ubuntu Secure Booting (https://wiki.ubuntu.com/UEFI/SecureBoot). Also, please get familiar with the official UEFI specification (https://uefi.org/specifications) for a generic description of UEFI Secure boot.

If you do not know what a Certification Authority (CA) or certificate chain is, please read (https://en.wikipedia.org/wiki/Certificate_authority)

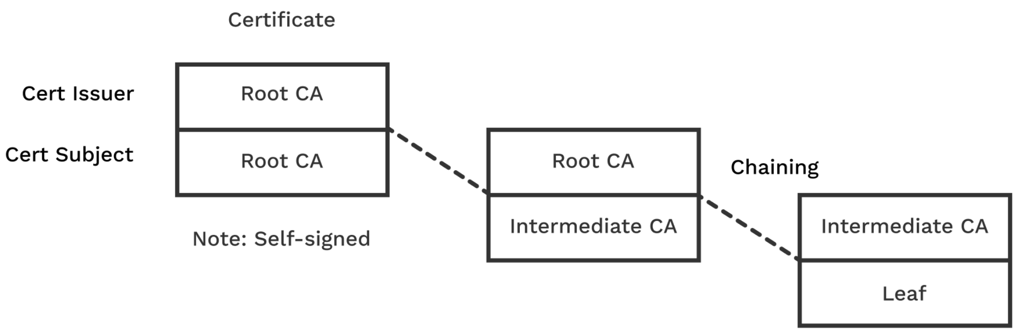

Figure 1 - Certificate chain

Task 1 - Firmware Databases

1. Extract the Microsoft certificate that belongs to the key referred to in Step 1 from the UEFI firmware, and show its text representation.
   Hint: *efitools, openssl x509*

2. Is this certificate the root certificate in the chain of trust? What is the role of the Platform Key (PK) and other variables?

Task 2 - SHIM

3. Verify that the system indeed boots the *shim* boot loader in the first stage. What is the full path name of this bootloader?
   Hint: *efibootmgr*

4. Verify that the *shim* bootloader is indeed signed with the "Microsoft Corporation UEFI CA" key.
   Hint: *sbsigntool, PEM format, pesign, sbverify*

UEFI binaries such as OS/boot loaders, drivers and kernels on Linux are executables in the Windows Authenticode Portable Executable Signature Format. This format is described on https://www.symbolcrash.com/wp-content/uploads/2019/02/Authenticode_PE-1.pdf. On this page you can download version 1.0 of the specification of the format. Read the first 9 pages of the specification (up to “Authenticode-Specific Structures”). Focus on the structure of the binaries.

5. What is the exact name of the part of the binary where the actual signature is stored?
6. In what standard cryptographic format is the signature data stored?

To extract the signature data from the binary, one needs to determine the exact location and size of this data in the binary. This information is stored in the Data Directories section of the Optional Header of the executable, as shown in Figure 1 of the specification. To retrieve location and size you can used the *pyew* package, as follows:

- Start *pyew* from the command line with the full name of the *shim* binary as argument.
- Wait for the *pyew* prompt to appear (it will do an analysis on the binary which you can interrupt with Ctrl-C if it takes too long)
- Type `pyew.pe.OPTIONAL HEADER.DATA DIRECTORY` to get a listing of all data directory entries.
- Locate the one called `IMAGE DIRECTORY ENTRY SECURITY`. The location and size are given as the “VirtualAddress:” and “Size:” fields.

7. Extract the signature data from the *shim* binary using dd. Add 8 bytes to the location as given in the data directory to skip over the Microsoft WIN CERTIFICATE structure header (see page 14 of the specification if you are interested).
   Hint: pyew.py, -/+8.
8. Show the subject and issuer of all X.509 certificates stored in the signature data. Draw a diagram relating these certificates to the “Microsoft Corporation UEFI CA” certificate.
   Hint: binwalk

Now we know that the system indeed boots the *shim*, and that this OS loader is indeed signed by Microsoft and where this signature is stored.

Task 3 - GRUB (BONUS)

In Step 2 of the Ubuntu description, it says that “the second stage bootloader (grub-efiamd64-signed) is signed with Canonical’s “Canonical Ltd. Secure Boot Signing” key.” This boot loader binary is called `grubx64.efi` and is located in the same directory as the *shim* binary.

9. Using your new knowledge about Authenticode binaries, extract the signing certificates from the GRUB boot loader, and show the subject and issuer.

You now know that the GRUB binary is indeed signed, but how does *shim* verify this signature? To do so *shim* needs a certificate that it trusts and which must not be modifiable by an external party without detection. The solution chosen by Ubuntu is to store a certificate in the *shim* binary. We will call this certificate X.

10. Why is storing the certificate X in the *shim* binary secure?
11. What do you think is the subject CommonName (CN) of this X certificate?
12. Obtain the X certificate used by the *shim* to verify the GRUB binary.

There are two ways to obtain it: from the source code or from the binary directly.
Hints for the latter case:

- *The certificate is in X.509 DER/ASN.1 format (see openssl asn1parse)*
- *DER/ASN.1 leaves the CommonName readable*
- *The certificate is 1080 bytes long*
- *binwalk/dd*

13. Verify that this X certificate’s corresponding private key was indeed used to sign the GRUB binary.

Task 4 - The Kernel (Optional)

GRUB allows the user to select from a list of kernels. According to Step 3, GRUB will check if the chosen kernel is signed. If so, GRUB leaves the system in UEFI mode before loading and running the kernel, so that it still has access to the UEFI services. As with *shim*, GRUB needs a trusted certificate against which it can verify the signed kernel. This is the same certificate as the *shim* uses.

14. Verify the kernel you booted against the X certificate.

15. Where does GRUB get its trusted certificate from? Hint: It is not stored in the binary, and it is not stored on the file system.

16. Draw a diagram that shows the chain of trust from the UEFI PK key to the signed kernel. Show all certificates, binaries, and signing relations involved.