Innopolis University

SYSTEM AND NETWORKING ENGINEERING

Security of Systems and Networks

LABORATORY REPORT 3

UEFI Secure Boot

Student | **ID**
Grebennikov Sergey | 47611

Lecturer
Dr. Rasheed Hussain, PhD

October 29, 2017

Contents

1 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 also by Ubuntu and Fedora. In this assignment you will learn how Ubuntu implements secure booting of the Linux kernel. For the first part of this assignment you will need access to a Ubuntu 15.04 machine that was installed with UEFI Secure Boot enabled (like your desktop PC). For the latter parts, a system that uses UEFI boot suffices (like your server in the default case), as the same binaries are used for insecure boot.

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

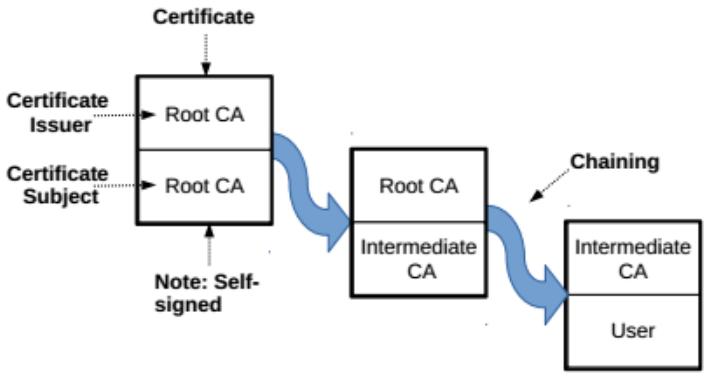

Figure 1: A chain of certificates from the root Certificate Authority to a user's certificate.

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

In this assignment you will verify that Ubuntu actually uses this chain and that the signatures are correct. We will refer to the steps in this description as "Step 1", "Step 2", etc. in the text that follows. Start by reading the Introduction section of the page on Ubuntu Secure Booting, `https://wiki.ubuntu.com/SecurityTeam/SecureBoot`. If you do not know what a Certification Authority (CA) or certificate chain is (see Figure 1), please read `http://en.wikipedia.org/wiki/Certificate_authority`.

2 Firmware Databases

Questions

1. Extract the Microsoft certificate that belongs to the key referred to in Step 1 from the UEFI firmware, and show its text representation on your log. *Hint: efitools, openssl x509*
   Result:
   First, a secure boot check is required:

```
$ sudo apt install mokutil
$ mokutil --sb-state
SecureBoot enabled
```

Second, tools for working with secure boot options are required:

```
$ sudo apt install efitools
```

'Microsoft Corporation UEFI CA' key:

```
$ mokutil --db
[key 1]
SHA1 Fingerprint: 46:de:f6:3b:5c:e6:1c:f8:ba:0d:e2:e6:63:9c:10:19:d0:ed
    :14:f3
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            61:08:d3:c4:00:00:00:00:00:04
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=Washington, L=Redmond, O=Microsoft Corporation,
             CN=Microsoft Corporation Third Party Marketplace Root
        Validity
            Not Before: Jun 27 21:22:45 2011 GMT
            Not After : Jun 27 21:32:45 2026 GMT
        Subject: C=US, ST=Washington, L=Redmond, O=Microsoft Corporation
            , CN=Microsoft Corporation UEFI CA 2011
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a5:08:6c:4c:c7:45:09:6a:4b:0c:a4:c0:87:7f:
                    06:75:0c:43:01:54:64:e0:16:7f:07:ed:92:7d:0b:
                    b2:73:bf:0c:0a:c6:4a:45:61:a0:c5:16:2d:96:d3:
                    f5:2b:a0:fb:4d:49:9b:41:80:90:3c:b9:54:fd:e6:
                    bc:d1:9d:c4:a4:18:8a:7f:41:8a:5c:59:83:68:32:
                    bb:8c:47:c9:ee:71:bc:21:4f:9a:8a:7c:ff:44:3f:
                    8d:8f:32:b2:26:48:ae:75:b5:ee:c9:4c:1e:4a:19:
                    7e:e4:82:9a:1d:78:77:4d:0c:b0:bd:f6:0f:d3:16:
                    d3:bc:fa:2b:a5:51:38:5d:f5:fb:ba:db:78:02:db:
                    ff:ec:0a:1b:96:d5:83:b8:19:13:e9:b6:c0:7b:40:
                    7b:e1:1f:28:27:c9:fa:ef:56:5e:1c:e6:7e:94:7e:
                    c0:f0:44:b2:79:39:e5:da:b2:62:8b:4d:bf:38:70:
                    e2:68:24:14:c9:33:a4:08:37:d5:58:69:5e:d3:7c:
                    ed:c1:04:53:08:e7:4e:b0:2a:87:63:08:61:6f:63:
                    15:59:ea:b2:2b:79:d7:0c:61:67:8a:5b:fd:5e:ad:
                    87:7f:ba:86:67:4f:71:58:12:22:04:22:22:ce:8b:
                    ef:54:71:00:ce:50:35:58:76:95:08:ee:6a:b1:a2:
                    01:d5
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            1.3.6.1.4.1.311.21.1:
                .....
            1.3.6.1.4.1.311.21.2:
                ....k..wSJ.%7.N.&{. p.
            X509v3 Subject Key Identifier:
                13:AD:BF:43:09:BD:82:70:9C:8C:D5:4F:31:6E:D5:22:98:8A:1B
                    :D4
            1.3.6.1.4.1.311.20.2:
                .
.S.u.b.C.A
            X509v3 Key Usage:
                Digital Signature, Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Authority Key Identifier:
```

```
                keyid:45:66:52:43:E1:7E:58:11:BF:D6:4E:9E:23:55:08:3B:3A
                    :22:6A:A8

            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://crl.microsoft.com/pki/crl/products/
                      MicCorThiParMarRoo_2010-10-05.crl

            Authority Information Access:
                CA Issuers - URI:http://www.microsoft.com/pki/certs/
                    MicCorThiParMarRoo_2010-10-05.crt

    Signature Algorithm: sha256WithRSAEncryption
         35:08:42:ff:30:cc:ce:f7:76:0c:ad:10:68:58:35:29:46:32:
         76:27:7c:ef:12:41:27:42:1b:4a:aa:6d:81:38:48:59:13:55:
         f3:e9:58:34:a6:16:0b:82:aa:5d:ad:82:da:80:83:41:06:8f:
         b4:1d:f2:03:b9:f3:1a:5d:1b:f1:50:90:f9:b3:55:84:42:28:
         1c:20:bd:b2:ae:51:14:c5:c0:ac:97:95:21:1c:90:db:0f:fc:
         77:9e:95:73:91:88:ca:bd:bd:52:b9:05:50:0d:df:57:9e:a0:
         61:ed:0d:e5:6d:25:d9:40:0f:17:40:c8:ce:a3:4a:c2:4d:af:
         9a:12:1d:08:54:8f:bd:c7:bc:b9:2b:3d:49:2b:1f:32:fc:6a:
         21:69:4f:9b:c8:7e:42:34:fc:36:06:17:8b:8f:20:40:c0:b3:
         9a:25:75:27:cd:c9:03:a3:f6:5d:d1:e7:36:54:7a:b9:50:b5:
         d3:12:d1:07:bf:bb:74:df:dc:1e:8f:80:d5:ed:18:f4:2f:14:
         16:6b:2f:de:66:8c:b0:23:e5:c7:84:d8:ed:ea:c1:33:82:ad:
         56:4b:18:2d:f1:68:95:07:cd:cf:f0:72:f0:ae:bb:dd:86:85:
         98:2c:21:4c:33:2b:f0:0f:4a:f0:68:87:b5:92:55:32:75:a1:
         6a:82:6a:3c:a3:25:11:a4:ed:ad:d7:04:ae:cb:d8:40:59:a0:
         84:d1:95:4c:62:91:22:1a:74:1d:8c:3d:47:0e:44:a6:e4:b0:
         9b:34:35:b1:fa:b6:53:a8:2c:81:ec:a4:05:71:c8:9d:b8:ba:
         e8:1b:44:66:e4:47:54:0e:8e:56:7f:b3:9f:16:98:b2:86:d0:
         68:3e:90:23:b5:2f:5e:8f:50:85:8d:c6:8d:82:5f:41:a1:f4:
         2e:0d:e0:99:d2:6c:75:e4:b6:69:b5:21:86:fa:07:d1:f6:e2:
         4d:d1:da:ad:2c:77:53:1e:25:32:37:c7:6c:52:72:95:86:b0:
         f1:35:61:6a:19:f5:b2:3b:81:50:56:a6:32:2d:fe:a2:89:f9:
         42:86:27:18:55:a1:82:ca:5a:9b:f8:30:98:54:14:a6:47:96:
         25:2f:c8:26:e4:41:94:1a:5c:02:3f:e5:96:e3:85:5b:3c:3e:
         3f:bb:47:16:72:55:e2:25:22:b1:d9:7b:e7:03:06:2a:a3:f7:
         1e:90:46:c3:00:0d:d6:19:89:e3:0e:35:27:62:03:71:15:a6:
         ef:d0:27:a0:a0:59:37:60:f8:38:94:b8:e0:78:70:f8:ba:4c:
         86:87:94:f6:e0:ae:02:45:ee:65:c2:b6:a3:7e:69:16:75:07:
         92:9b:f5:a6:bc:59:83:58
...
```

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

   Answer:
   No, the 'Microsoft Corporation UEFI CA' key is not the root certificate in the chain of trust. **PK** (**Platform Key**) represents the root of trust and is used to protect the **KEK (Key Exchange Key)** database. The platform vendor puts public portion of the Platform Key (PK) into UEFI Firmware during manufacturing. Its private portion stays with the vendor. When updating the PK, the new PK certificate must be signed with the old one.

   Hashes and signatures used to verify bootloaders and other pre-boot components are stored in 3 different databases: As you can see from the figure 2 the **KEK** signs 3 different Data Bases) where hashes and signatures used to verify bootloaders and other pre-boot components are stored:

   - **DB** (Allowed Signature database) - this database may contain Certification Authority certificates or their hashes that were used to generate code-signing certificates used to sign bootloader and other pre-boot components. If the bootloader is signed by any of certificates chaining to the CA certificate present in this database, it is permitted to execute.
   - **DBX** (Disallowed Signature database) - this database may contain the hash of a specific binary, an X.509 certificate, or the hash of a certificate that were compromised and/or

revoked. If the bootloader is signed by any certificate present in this database, or its hash is present here, it will be denied from execution.

- **DBT** (timestamp signature database) - contains timestamping certificates using when signing bootloader images.

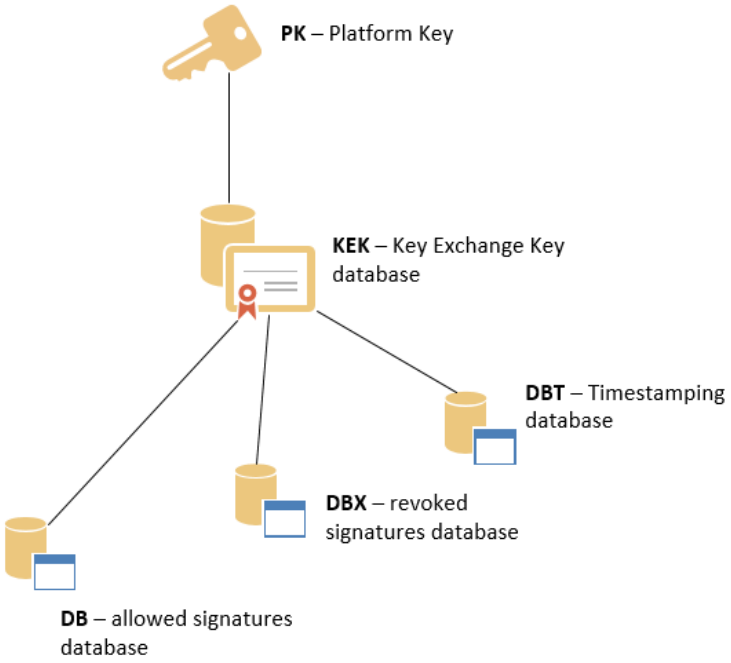

Figure 2: Hierarchy of trust

3 SHIM

Questions

3. Verify that the system indeed boots the 'shim' boot loader in the first stage. What is the full path name of this boot loader?

   Answer:

```
$ efibootmgr -v
BootCurrent: 0000
Timeout: 0 seconds
BootOrder: 0000,0001,0002,0005,0006,0003,0004
Boot0000* ubuntu         HD(1,GPT,c355e600-4097-4791-8389-e11d473ea9a1,0
    x800,0x100000)/File(\EFI\ubuntu\shimx64.efi)
...

$ sudo ls -lh /boot/efi/EFI/ubuntu/shimx64.efi
-rwx------ 1 root root 1,2M aug 14 14:13 /boot/efi/EFI/ubuntu/shimx64.
    efi
```

4. Verify that the 'shim' boot loader is indeed signed with the 'Microsoft Corporation UEFI CA' key. Hint: *sbsigntool, PEM format*

   Answer:

   We'll use the **efi-readvar** tool to store off the current values of db in a machine-readable signature list format:

```
$ efi-readvar -v db -s 0 -o cert.esl
Variable db, length 4600
```

   To convert a machine-readable signature list format to openssl certificate we need to use the **sig-list-to-certs**:

```
$ sig-list-to-certs cert.esl cert
X509 Header sls=1600, header=0, sig=1556
file cert-0.der: Guid 77fa9abd-0359-4d32-bd60-28f4e78f784b
Written 1556 bytes
X509 Header sls=1543, header=0, sig=1499
file cert-1.der: Guid 77fa9abd-0359-4d32-bd60-28f4e78f784b
Written 1499 bytes
X509 Header sls=1457, header=0, sig=1413
file cert-2.der: Guid f5a96b31-dba0-4faa-2aa4-7a0c9832768e
Written 1413 bytes
```

Now we have three DER formatted certificates. We will be interested in **cert-0.der** certificate. Converting a required DER formatted certificate to PEM:

```
$ openssl x509 -inform DER -in cert-0.der -outform PEM -out cert.pem
```

Verifying the Microsoft's signature on the signed shim bootloader:

```
$ sudo sbverify --cert cert.pem /boot/efi/EFI/ubuntu/shimx64.efi
[sudo] password for sergey:
warning: data remaining[1044920 vs 1169992]: gaps between PE/COFF
    sections?
Signature verification OK
```

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 `http://www.microsoft.com/whdc/winlogo/drvsign/Authenticode_PE.mspx`. On this page you can download revision 1.0 of the specification of the format.

5. Read the first 9 pages of the specification (up to "Authenticode-Specific Structures"). Focus on the structure of the binaries. What is the name of the part of the binary where the actual signature data is stored?

   Answer:
   The **signerInfos** field in the **SignedData** structure contains a set of **SignerInfo** structures, which contains information about the signatures. The **encryptedDigest** field in the **SignerInfo** structure contains the signature created by the signing certificate's private key, calculated as defined by the PKCS#7 specification.

6. In what standard cryptographic format is the signature data stored?

   Answer:
   The signature data is stored in a format based on the '**Public-Key Cryptography Standard (PKCS)#7: Cryptographic Message Syntax Standard**'.

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.

We will now study this signature data:

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_CERTIFICAT` structure header (see page 14 of the specification if you are interested). Show the command you used.

   Answer:

```
#dd if=/boot/efi/EFI/ubuntu/shimx64.efi of=shim  bs=1 skip=$((0x11b898))
     count=$((0x21b8))
```

8. Show the subject and issuer of any X.509 certificates stored in the signature data. Draw a diagram relating these certificates to the 'Microsoft Corporation UEFI CA' certificate. Hint: *openssl, strongswan-starter*

Answer:

```
# binwalk -e shim

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------

141           0x8D            Certificate in DER format (x509 v3),
    header length: 4, sequence length: 1317
1462          0x5B6           Certificate in DER format (x509 v3),
    header length: 4, sequence length: 1552
3367          0xD27           Unix path: /www.microsoft.com/whdc/hcl/
    default.mspx0
4074          0xFEA           Certificate in DER format (x509 v3),
    header length: 4, sequence length: 1649
5727          0x165F          Certificate in DER format (x509 v3),
    header length: 4, sequence length: 1242

# ls
_shim.extracted

# cd _shim.extracted/

# ls
165F.crt  5B6.crt  8D.crt  FEA.crt

# openssl x509 -inform DER -in 8D.crt -noout -subject -issuer
subject= /C=US/ST=Washington/L=Redmond/O=Microsoft Corporation/OU=MOPR/
    CN=Microsoft Windows UEFI Driver Publisher
issuer= /C=US/ST=Washington/L=Redmond/O=Microsoft Corporation/CN=
    Microsoft Corporation UEFI CA 2011

# openssl x509 -inform DER -in 5B6.crt -noout -subject -issuer
subject= /C=US/ST=Washington/L=Redmond/O=Microsoft Corporation/CN=
    Microsoft Corporation UEFI CA 2011
issuer= /C=US/ST=Washington/L=Redmond/O=Microsoft Corporation/CN=
    Microsoft Corporation Third Party Marketplace Root

# openssl x509 -inform DER -in FEA.crt -noout -subject -issuer
subject= /C=US/ST=Washington/L=Redmond/O=Microsoft Corporation/CN=
    Microsoft Time-Stamp PCA 2010
issuer= /C=US/ST=Washington/L=Redmond/O=Microsoft Corporation/CN=
    Microsoft Root Certificate Authority 2010

# openssl x509 -inform DER -in 165F.crt -noout -subject -issuer
subject= /C=US/ST=Washington/L=Redmond/O=Microsoft Corporation/OU=MOPR/
    OU=nCipher DSE ESN:BBEC-30CA-2DBE/CN=Microsoft Time-Stamp Service
issuer= /C=US/ST=Washington/L=Redmond/O=Microsoft Corporation/CN=
    Microsoft Time-Stamp PCA 2010
```

Draw: Figure 3

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.

4 GRUB

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.

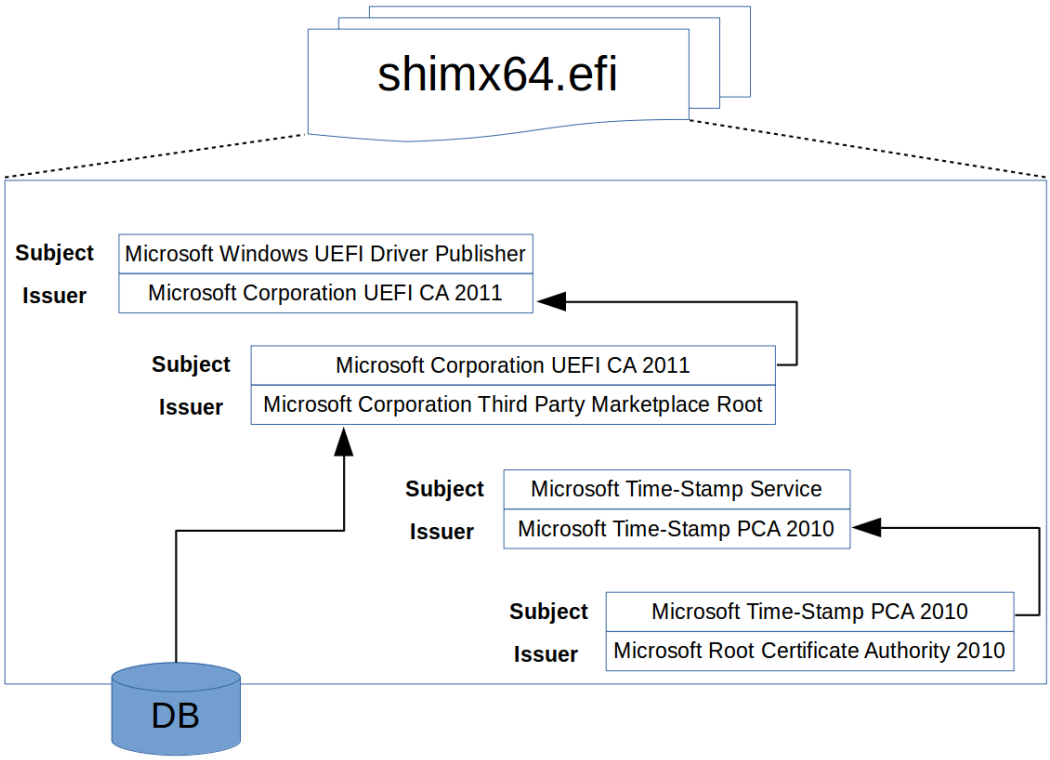

Figure 3: ertificates associated with the certificate "Microsoft Corporation UEFI CA"

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**.

   Answer:

```
# dd if=/boot/efi/EFI/ubuntu/grubx64.efi of=grub bs=1 skip=$((0x114408))
    count=$((0x778))
1904+0 records in
1904+0 records out
1904 bytes (1,9 kB, 1,9 KiB) copied, 0,00217476 s, 875 kB/s
# binwalk -e grub

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------

169           0xA9            Certificate in DER format (x509 v3),
   header length: 4, sequence length: 1056

# cd _grub.extracted/

# ls
A9.crt

# openssl x509 -inform DER -in A9.crt -noout -subject -issuer
subject= /C=GB/ST=Isle of Man/O=Canonical Ltd./OU=Secure Boot/CN=
   Canonical Ltd. Secure Boot Signing
issuer= /C=GB/ST=Isle of Man/L=Douglas/O=Canonical Ltd./CN=Canonical Ltd
   . Master Certificate Authority
```

10. Why is storing the certificate **X** in the shim binary secure?

    Answer:

    Because the certificate **X** as well as five others in the 'shim' is signed by the 'Microsoft Corpora-

tion UEFI CA' key, which is included in the chain of trust that starts from the root Certification Authority.

11. What do you think is the subject CommonName (CN) of this **X** certificate?

    Answer:
    Canonical Ltd. Master Certificate Authority

12. Obtain the **X** certificate used by '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. Show the X certificate on your log in text format.

    Answer:

```
# dd if=/boot/efi/EFI/ubuntu/grubx64.efi of=grub bs=1 skip=$((0x114408))
    count=$((0x778))
1904+0 records in
1904+0 records out
1904 bytes (1,9 kB, 1,9 KiB) copied, 0,00217476 s, 875 kB/s
# binwalk -e grub

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------

169           0xA9            Certificate in DER format (x509 v3),
   header length: 4, sequence length: 1056

# cd _grub.extracted/

# ls
A9.crt

# openssl x509 -inform DER -in A9.crt -noout -subject -issuer
subject= /C=GB/ST=Isle of Man/O=Canonical Ltd./OU=Secure Boot/CN=
   Canonical Ltd. Secure Boot Signing
issuer= /C=GB/ST=Isle of Man/L=Douglas/O=Canonical Ltd./CN=Canonical Ltd
   . Master Certificate Authority

# binwalk /boot/efi/EFI/ubuntu/shimx64.efi | grep "Cert"
776992  0xBDB20   Certificate in DER format (x509 v3), header length: 4,
    sequence length: 924
863248  0xD2C10   Certificate in DER format (x509 v3), header length: 4,
    sequence length: 1076
1161509 0x11B925  Certificate in DER format (x509 v3), header length: 4,
    sequence length: 1317
1162830 0x11BE4E  Certificate in DER format (x509 v3), header length: 4,
    sequence length: 1552
1165442 0x11C882  Certificate in DER format (x509 v3), header length: 4,
    sequence length: 1649
1167095 0x11CEF7  Certificate in DER format (x509 v3), header length: 4,
    sequence length: 1242

# binwalk -e /boot/efi/EFI/ubuntu/shimx64.efi

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0               0x0                     Microsoft executable, portable (
   PE)
37987   0x9463          mcrypt 2.2 encrypted data, algorithm: blowfish
   -448, mode: CBC, keymode: 8bit
76275   0x129F3         mcrypt 2.2 encrypted data, algorithm: blowfish
   -448, mode: CBC, keymode: 8bit
91475   0x16553         mcrypt 2.2 encrypted data, algorithm: blowfish
   -448, mode: CBC, keymode: 8bit
704640  0xAC080         SHA256 hash constants, little endian
```

```
759228  0xB95BC         Unix path: /usr/local/ssl/private
766912  0xBB3C0         Base64 standard index table
776992  0xBDB20         Certificate in DER format (x509 v3), header
    length: 4, sequence length: 924
863248  0xD2C10         Certificate in DER format (x509 v3), header
    length: 4, sequence length: 1076
1033583 0xFC56F         mcrypt 2.2 encrypted data, algorithm: blowfish
    -448, mode: CBC, keymode: 8bit
1101456 0x10CE90        mcrypt 2.2 encrypted data, algorithm: blowfish
    -448, mode: nOFB, keymode: 8bit
1161509 0x11B925        Certificate in DER format (x509 v3), header
    length: 4, sequence length: 1317
1162830 0x11BE4E        Certificate in DER format (x509 v3), header
    length: 4, sequence length: 1552
1164735 0x11C5BF        Unix path: /www.microsoft.com/whdc/hcl/default.
    mspx0
1165442 0x11C882        Certificate in DER format (x509 v3), header
    length: 4, sequence length: 1649
1167095 0x11CEF7        Certificate in DER format (x509 v3), header
    length: 4, sequence length: 1242

# cd _shimx64.efi.extracted/

# ls
11B925.crt  11BE4E.crt  11C882.crt  11CEF7.crt  BDB20.crt  D2C10.crt

# openssl x509 -inform DER -in BDB20.crt -noout -subject -issuer
subject= /C=US/L=SomeCity/O=SomeOrg/CN=shim
issuer= /C=US/L=SomeCity/O=SomeOrg

# openssl x509 -inform DER -in D2C10.crt -noout -subject -issuer
subject= /C=GB/ST=Isle of Man/L=Douglas/O=Canonical Ltd./CN=Canonical
    Ltd. Master Certificate Authority
issuer= /C=GB/ST=Isle of Man/L=Douglas/O=Canonical Ltd./CN=Canonical Ltd
    . Master Certificate Authority
```

13. Verify that this **X** certificates corresponding private key was indeed used to sign the GRUB binary.

    Answer:

```
# openssl x509 -inform DER -in D2C10.crt -outform PEM -out D2C10.pem
# sbverify --cert D2C10.pem /boot/efi/EFI/ubuntu/grubx64.efi
Signature verification OK
```

5 The Kernel

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 has still 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 'shim' uses.

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

    Answer:

```
# uname -r
4.10.0-37-generic
# sbverify --cert D2C10.pem /boot/vmlinuz-4.10.0-37-generic.efi.signed
Signature verification OK
```

15. BONUS: 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. We have now verified Steps 13 of the Ubuntu chain of trust. Verifying Step 4 is left asa bonus.

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.

Answer: Figure 4

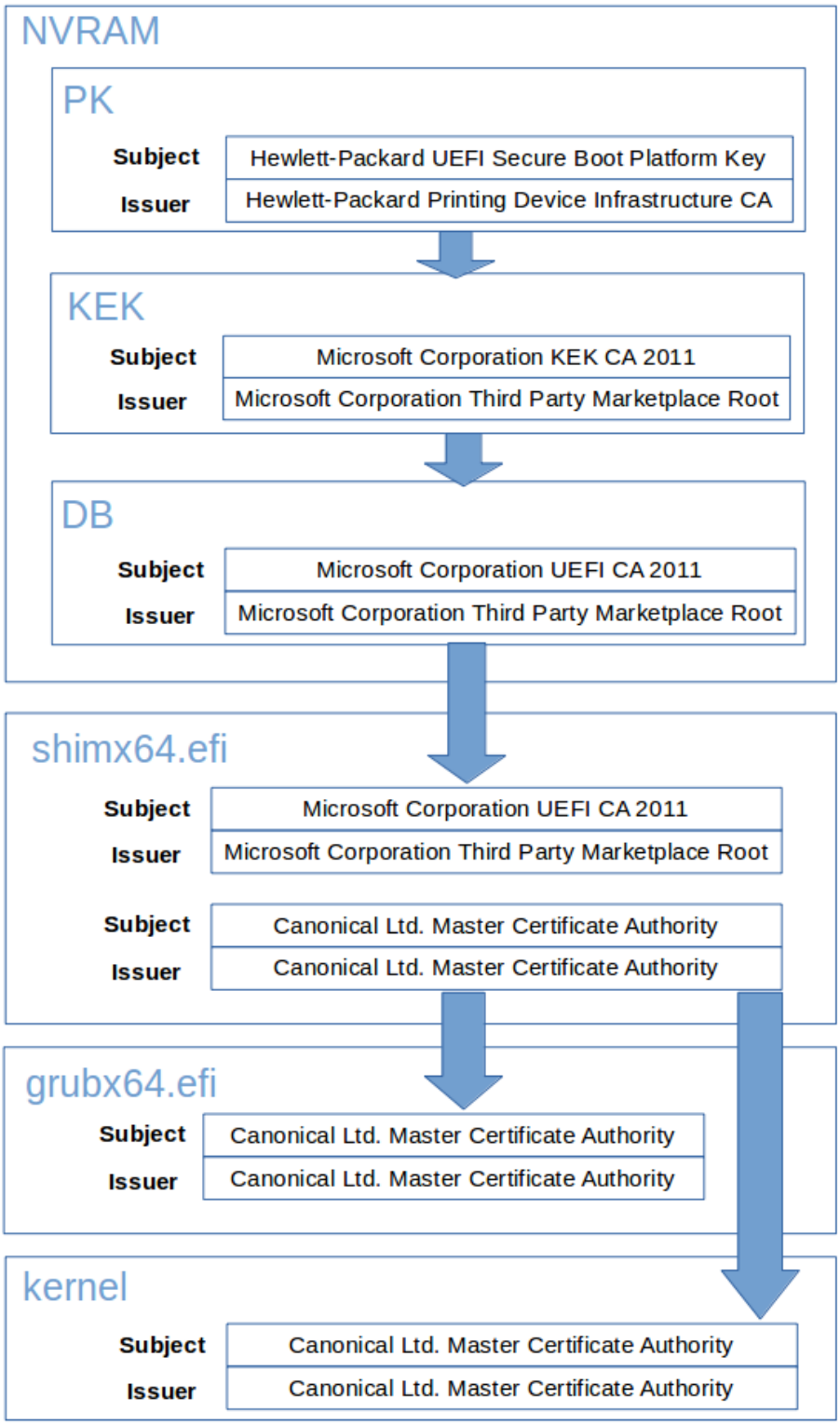

Figure 4: The chain of trust from the UEFI PK key to the signed kernel.

6 Conclusion

When a CPU starts up, it executes only a few very specific instructions at a very specific address. Nothing is initialized yet. There are simply not enough resources to start any code. At this point, all the system can do is to find, validate, install and run a small initial piece of firmware. In the UEFI Secure Boot process this SEC (Security) Phase, as initial turn-on is called, forms the basis for the Root of Trust. Changing what happens during the SEC Phase, while theoretically possible, would be very difficult and would involve gaining physical access to the system and modifying or replacing hardware.

From this very secure basis flows the Chain of Trust used in UEFI Secure Boot. The trust is maintained via public key cryptography. Hardware manufacturers put whats known as a Platform Key (PK) into the firmware, representing the Root of Trust. The trust relationship with operating system vendors and others is documented by signing their keys with the Platform Key.

Security is established by requiring that no code will be executed by firmware unless it has been signed by a trusted key whether its an operating system boot loader, a driver located in the flash memory of a PCI Express card or on disk, or an update of the firmware itself. Key Exchange Keys (KEK) can be added to the UEFI key database so trusted third-party applications can use other certificates as long as they are signed with the private part of the PK.

To manage the signing process, a centralized Certificate Authority (CA) is used, currently operated by Microsoft and open to everyone in the industry. The opportunity exists for other organizations to set up a certificate authority on a broader basis or for their exclusive use as well. The Microsoft-operated UEFI CA, which is available for a nominal fee, has been accepted broadly across the industry, including by most major Linux distributions and third-party developers.

References

[1] M. Stamp, Information Security: Principles and Practice, Second Edition, 2011, 606 pages.

[2] SecureBoot https://wiki.ubuntu.com/SecurityTeam/SecureBoot.

[3] Unified Extensible Firmware Interface Specification, 1785 p http://www.uefi.org/sites/default/files/resources/UEFI\%202_5.pdf.

[4] THE CHAIN OF TRUST. Keeping Computing Systems More Secure. Authors: Richard Wilkins, Ph.D. Phoenix Technologies, Ltd. http://www.uefi.org/sites/default/files/resources/UEFI%20Forum%20White%20Paper%20-%20Chain%20of%20Trust%20Introduction_Final.pdf.