Fedora 18

UEFI Secure Boot Guide

Josh Boyer

Kevin Fenzi

Peter Jones

Josh Bressers

Florian Weimer

Fedora 18 UEFI Secure Boot Guide
Edition 18.4

Author	Josh Boyer	`jwboyer@redhat.com`
Author	Kevin Fenzi	`kevin@fedoraproject.org`
Author	Peter Jones	`pjones@redhat.com`
Author	Josh Bressers	`bressers@redhat.com`
Author	Florian Weimer	`fweimer@redhat.com`
Editor	Eric Christensen	`sparks@redhat.com`

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*[1] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keycaps and key combinations. For example:

> To see the contents of the file **`my_next_bestselling_novel`** in your current working directory, enter the **`cat my_next_bestselling_novel`** command at the shell prompt and press **`Enter`** to execute the command.

The above includes a file name, a shell command and a keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F2`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F1`** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set pressed simultaneously).

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **`mono-spaced bold`**. For example:

> File-related classes include **`filesystem`** for file systems, **`file`** for files, and **`dir`** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System**□□□ **Preferences**□□□ **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box

[1] https://fedorahosted.org/liberation-fonts/

> and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications**□□□ **Accessories** □□ **Character Map** from the main menu bar. Next, choose **Search**□□□ **Find…** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit**□□ **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

`Mono-spaced Bold Italic` or ***`Proportional Bold Italic`***

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type **`ssh`** ***`username@domain.name`*** at a shell prompt. If the remote machine is **`example.com`** and your username on that machine is john, type **`ssh john@example.com`**.
>
> The **`mount -o remount`** ***`file-system`*** command remounts the named file system. For example, to remount the **`/home`** file system, the command is **`mount -o remount /home`**.
>
> To see the version of a currently installed package, use the **`rpm -q`** ***`package`*** command. It will return a result as follows: ***`package-version-release`***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **`mono-spaced roman`** and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in **`mono-spaced roman`** but add syntax highlighting as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
```

```
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.

Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: *http://bugzilla.redhat.com/bugzilla/* against the product **Fedora.**

When submitting a bug report, be sure to mention the manual's identifier: *UEFI_Secure_Boot_Guide*

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

What is UEFI Secure Boot?

Secure Boot is a technology where the system firmware checks that the system boot loader is signed with a cryptographic key authorized by a database contained in the firmware. With adequate signature verification in the next-stage boot loader(s), kernel, and, potentially, user space, it is possible to prevent the execution of unsigned code.

Secure Boot is a form of *Verified Booting*. Boot path validation is also part of other technologies such as *Trusted Boot*. Boot path validation is indepedent of secure storage of cryptographic keys and remote attestation.

1.1. UEFI Secure Boot

UEFI Secure Boot is the boot path validation component of the *UEFI* specification (*Unified Extensible Firmware Interface*)as of version 2.3. Roughly speaking, it specifies the following:

- a programming interface for cryptographically protected UEFI variables in non-volatile storage,
- how the trusted X.509 root certificates are stored in UEFI variables,
- validation of UEFI applications (boot loaders and drivers) using AuthentiCode signatures embedded in these applications, and
- procedures to revoke known-bad certificates and application hashes.

UEFI Secure Boot does not require specialized hardware, apart from non-volatile (flash) storage which can be switched from read-write mode to read-only mode during system boot. This storage has to be used to store the UEFI implementation itself and some of the protected UEFI variables (including the trusted root certificate store).

From a user point of view, a system which has enabled UEFI Secure Boot and which is confronted with a tampered boot path simply stops working until UEFI Secure Boot is disabled or a signed next-stage boot loader is available on boot media. (*Figure 1.1, “Typical error message from UEFI Secure Boot”* shows a typical error message.) Similarly, operating system installers without a cryptographically valid signature do not run and result in an error message. Users are not offered a way to override the boot loader decision to reject the signature, unlike the similar scenario with web server certificates. No certificate issuer information is provided to the user.

```
┌────────── Secure Boot Violation ──────────┐
│                                           │
├───────────────────────────────────────────┤
│ Invalid signature detected. Check Secure  │
│          Boot Policy in Setup             │
│                                           │
│                                           │
│                   [OK]                    │
└───────────────────────────────────────────┘
```

Figure 1.1. Typical error message from UEFI Secure Boot

UEFI Secure Boot does not prevent the installation or removal of second-stage boot loaders or require explicit user confirmation of such changes. Signatures are verified during booting, and not when the boot loader is installed or updated. Therefore, UEFI Secure Boot does not stop boot path manipulations. It only prevents the system from executing a modified boot path once such a modification has occurred, and simplifies their detection.

Client Technology

UEFI Secure Boot is currently only generally enabled on client devices, and is currently not recommended for deployment on server machines. It is expected that server technology will enable Secure Boot at a future date.

1.2. Microsoft Requirements for Secure Boot

Microsoft has not published many details about their implementation of Secure Boot, which is based on UEFI Secure Boot.

Microsoft supports UEFI Secure Boot only with Windows 8, but it is not required for running Windows 8. Systems in a UEFI Secure Boot environment still boot if Secure Boot support is removed from the UEFI environment. Customers who want to use previous versions of Windows need to disable Secure Boot because Microsoft does not provide signed boot loaders for them.

Based on the public record, the following security objectives for Microsoft Secure Boot appear likely:

- integrity protection of installation media which is stored on writable media (such as hard disk recovery partitions),
- protection of the boot path until heuristic countermeasures (such as kernel mode anti-virus software) can be loaded during early boot, and
- automatic restoration of the original boot path, perhaps after the system has been compromised by malware, without complete reinstallation of the entire operating system.

It is unclear whether there are plans to restrict access to certain (online) content to systems which have enabled UEFI Secure Boot and whose boot path is cryptographically valid. This would imply a remote attestation aspect which is not part of the UEFI Secure Boot specification, and which cannot be implemented securely without additional hardware support. It also would imply to that UEFI Secure Boot is no longer truly optional.

There is no evidence that Microsoft intended to lock out anyone with their implementation of Secure Boot. However, automatic restoration of the original boot path is much more complicated once other boot loaders can co-exist on the same system.

1.2.1. Implementation details

Microsoft requires that client PCs which carry the Windows 8 logo must enable UEFI Secure Boot and install Microsoft-provided key material. The required X.509 certificate is shown in *Figure 1.2, “Microsoft Trusted X.509 Certificate for their Secure Boot implementation”*. System vendors are encouraged to include other root certificates as needed, but those are not required to be present.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            61:07:76:56:00:00:00:00:00:08
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=Washington, L=Redmond, O=Microsoft Corporation,
         CN=Microsoft Root Certificate Authority 2010
        Validity
            Not Before: Oct 19 18:41:42 2011 GMT
            Not After : Oct 19 18:51:42 2026 GMT
        Subject: C=US, ST=Washington, L=Redmond, O=Microsoft Corporation,
                 CN=Microsoft Windows Production PCA 2011
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:dd:0c:bb:a2:e4:2e:09:e3:e7:c5:f7:96:69:bc:
      […]
                    87:65:b4:43:18:a8:b2:e0:6d:19:77:ec:5a:24:fa:
                    48:03
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            1.3.6.1.4.1.311.21.1:
                02:01:00
            X509v3 Subject Key Identifier:
                A9:29:02:39:8E:16:C4:97:78:CD:90:F9:9E:4F:9A:E1:7C:55:AF:53
            1.3.6.1.4.1.311.20.2:
                1E:0A:00:53:00:75:00:62:00:43:00:41
            X509v3 Key Usage:
                Digital Signature, Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Authority Key Identifier:
                keyid:D5:F6:56:CB:8F:E8:A2:5C:62:68:D1:3D:94:90:5B:D7:CE:9A:18:C4

            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://crl.microsoft.com/pki/crl/products/MicRooCerAut_2010-06-23.crl

            Authority Information Access:
                CA Issuers - URI:http://www.microsoft.com/pki/certs/
MicRooCerAut_2010-06-23.crt

    Signature Algorithm: sha256WithRSAEncryption
         14:fc:7c:71:51:a5:79:c2:6e:b2:ef:39:3e:bc:3c:52:0f:6e:
  […]
         04:cf:77:a4:62:1c:59:7e

-----BEGIN CERTIFICATE-----
MIIF1zCCA7+gAwIBAgIKYQd2VgAAAAAACDANBgkqhkiG9w0BAQsFADCBiDELMAkG
A1UEBhMCVVMxEzARBgNVBAgTCldhc2hpbmd0b24xEDAOBgNVBAcTB1JlZG1vbmQx
HjAcBgNVBAoTFU1pY3Jvc29mdCBDb3Jwb3JhdGlvbjEyMDAGA1UEAxMpTWljcm9z
b2Z0IFJvb3QgQ2VydGlmaWNhdGUgQXV0aG9yaXR5IDIwMTAwHhcNMTExMDE5MTg0
MTQyWhcNMjYxMDE5MTg1MTQyWjCBhDELMAkGA1UEBhMCVVMxEzARBgNVBAgTCldh
c2hpbmd0b24xEDAOBgNVBAcTB1JlZG1vbmQxHjAcBgNVBAoTFU1pY3Jvc29mdCBD
b3Jwb3JhdGlvbjEuMCwGA1UEAxMlTWljcm9zb2Z0IFdpbmRvd3MgUHJvZHVjdGlv
biBQQ0EgMjAxMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAN0Mu6Lk
Lgnj58X3lmm8ACG9aTMz760Ey1SA7gaDu8UghNn30ovzOLCrpK0tfGJ5Bf/jSj8E
NSBw48Tna+CcwDZ16Yox3Y1w5dw3tXRGlihbh2AjLL/cR6Vn91EnnnLrB6bJuR47
UzV85dPsJ7mHHP65ySMJb6hGkcFuljxB08ujP10Cak3saR8lKFw2//1DFQqU4Bm0
z9/CEuLCWyfuJ3gwi1sqCWsiiVNgFizAaB1TuuxJ851hjIVoCXNEXX2iVCvdefcV
zzVdbBwrXM68nCOLb261Jtk2E8NP1ieuuTI7QZIs4cfNd+iqVE73XAsEh2W0Qxio
suBtGXfsWiT6SAMCAwEAAaOCAUMwggE/MBAGCSsGAQQBgjcVAQQDAgEAMB0GA1Ud
```

Figure 1.2. Microsoft Trusted X.509 Certificate for their Secure Boot implementation

```
AEMAQTALBgNVHQ8EBAMCAYYwDwYDVR0TAQH/BAUwAwEB/zAfBgNVHSMEGDAWgBTV
9lbLj+iiXGJo0T2UkFvXzpoYxDBWBgNVHR8ETzBNMEugSaBHhkVodHRwOi8vY3Js
Lm1pY3Jvc29mdC5jb20vcGtpL2NybC9wcm9kdWN0cy9NaWNSb29DZXJBdXRfMjAx
MC0wNi0yMy5jcmwwWgYIKwYBBQUHAQEETjBMMEoGCCsGAQUFBzAChj5odHRwOi8v
d3d3Lm1pY3Jvc29mdC5jb20vcGtpL2NlcnRzL01pY1Jvb0NlckF1dF8yMDEwLTA2
LTIzLmNydDANBgkqhkiG9w0BAQsFAAOCAgEAFPx8cVGlecJusu85Prw8Ug9uKz8Q
```

Microsoft is expected to eventually ship signed revocation requests in Windows Update. These requests are installed into the UEFI Secure Boot configuration variables during the next system boot, after validating them against the Microsoft key. At the time of this writing, such a blacklist does not yet exist.

Third-party boot loaders currently do not have access to the *Microsoft Windows Production PCA 2011* certificate Microsoft uses for their own products. Instead, Microsoft provides a signing service originally intended for UEFI drivers. This service has been extended to include third-party boot loaders, too. Microsoft reviews the submitted UEFI applications, provides an AuthentiCode signature using its own key, and sends the result back to the author. This signature does not identify the application author (it is pseudonymous), and, more importantly, it is chained via the intermediate certificate *Microsoft Corporation UEFI CA 2011* (see *Figure 1.3, "Microsoft X.509 certificate for third-party UEFI applications"*) to the *Microsoft Corporation Third Party Marketplace Root* root certificate.

Warning

Third-party UEFI boot loaders are not guaranteed to work on Microsoft Secure Boot systems because the necessary certificates are not part of the Microsoft Secure Boot specification.

```
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:
                02:03:01:00:01
            1.3.6.1.4.1.311.21.2:
                04:14:F8:C1:6B:B7:7F:77:53:4A:F3:25:37:1D:4E:A1:26:7B:0F:20:70:80
            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:
         1E:0A:00:53:00:75:00:62:00:43:00:41
            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:
         […]
```

Figure 1.3. Microsoft X.509 certificate for third-party UEFI applications

```
-----BEGIN CERTIFICATE-----
MIIGEDCCA/igAwIBAgIKYQjTxAAAAAAABDANBgkqhkiG9w0BAQsFADCBkTELMAkG
A1UEBhMCVVMxEzARBgNVBAgTCldhc2hpbmd0b24xEDAOBgNVBAcTB1JlZG1vbmQx
HjAcBgNVBAoTFU1pY3Jvc29mdCBDb3Jwb3JhdGlvbjE7MDkGA1UEAxMyTWljcm9z
b2Z0IENvcnBvcmF0aW9uIFRoaXJkIFBhcnR5IE1hcmtldHBsYWNlIFJvb3QwHhcN
```

A regular code signing certificate is *not* sufficient to guarantee booting on a Microsoft Secure Boot system. Microsoft requires a code signing certificate when communicating with UEFI application authors, but this is an internal detail not visible to the end user in any way.

In the booted operating system, Microsoft Windows 8 supports AuthentiCode validation and loading of signed third-party kernel modules. Windows has infrastructure to extend cryptographic validation to user space programs, again based on AuthentiCode.

1.3. Fedora Secure Boot

The Fedora Secure Boot implementation has a single security objective: it prevents the execution of unsigned code in kernel mode.

Fedora can boot on systems with Microsoft Secure Boot enabled, provided the Microsoft certificate for third-party UEFI applications is installed. This mode of operation is most important for installing Fedora on machines which have been prepared for Windows 8. Other hardware is not likely to provide a Microsoft Secure Boot environment.

Warning

Third-party UEFI boot loaders (such as the Fedora boot loader) are not guaranteed to work on Microsoft Secure Boot systems because the necessary certificates are not part of the Windows 8 Hardware Certification Requirements. If your hardware is in this category, you need to switch off UEFI Secure Boot, enroll the missing Microsoft certificate, or enroll the Fedora certificate.

Fedora boots on UEFI systems which do not support or have disabled Secure Boot, too. This works with all UEFI boot loaders. These boot loaders also support running in an environment which performs boot path validation by other (non-UEFI) means. In this mode, there are no restrictions on executing code in kernel mode.

Details of the Fedora Secure Boot implementation are covered in *Chapter 3, UEFI Secure Boot Implementation*. Restrictions on kernel mode code execution disables certain functionality, see *Section 3.4.1, “Restrictions”*.

1.4. What does Secure Boot protect you from?

On the most basic level, UEFI Secure Boot prevents running unsigned boot loaders. The effect of running the boot loader obviously depends on that boot loader. The following refers to the Fedora implementation of Secure Boot. The Microsoft implementation is different, see *Section 1.2, “Microsoft Requirements for Secure Boot”*.

Fedora has extended the chain of trust from the UEFI environment into the kernel. Verification happens before loading kernel modules, but it does not extend to user space applications. We can be certain that no unsigned executable code is present until the initial ramdisk (initrd) is loaded. Since initrd contents are not cryptographically signed and contain executable code, booting a signed Fedora boot loader can eventually lead to arbitrary effects.

The Fedora Secure Boot implementation simplifies verification of the boot path in digital forensics. Legacy BIOS booting executes potentially malicious code loaded from the disk in a very early stage, which makes it difficult to rule out that the operating system has been tampered with at a very low level. (Parts of this benefit comes from the more declarative nature of the UEFI boot configuration on disk.)

1.5. Potential Secure Boot Risks

Secure Boot will not protect your PC from most malware or attackers. Secure Boot itself protects the boot phase of a system, but does not protect against attacks against your running system or data. In Fedora if you use Secure Boot, what modules the kernel loads can be restricted, but no additional protection is provide against user space malware. The initial ramdisk (initrd) disk image used during boot is not protected by this feature, and could contain malicious code.

1.5.1. Forced removal of features in Secure Boot mode

We currently use a blacklist approach to disable known-unsafe kernel functionality. In some cases, specific functionality has been disabled. In most cases such functionality is obscure and seldom used. Currently, the list of restricted functionality includes:

- modification of device memory address maps through "setpci", and writes to that memory from user space via sysfs
- hibernate (also known as *suspend to disk*)
- kexec and kdump (addressing this is a work in progress)
- writes to *Machine Specific Registers* writes via the "msr" kernel module.
- the acpi_rspd command line option, which is used to specify custom ACPI data
- io operations on /dev/kmem

In addition, use of systemtap modules is restricted to those modules which have been signed with an appropriate certificate. It is recommended that such a certificate be generated on-site, taking care to follow appropriate security practices for storage and use of a signing certificate. When using a site-local certificate, it can be enrolled in the machine's local database using the *mokutil* utility, which will then require a reboot before taking effect. Upon rebooting, *shim* will start *MokManager* to present you with an interface to enroll the certificate.

Warning

It is critically important that proper precautions be employed when using site-local certificates, so that they cannot be found and used by an attacker to subvert module security. Treat such certificates as you would any critical secrets, and follow industry best practices for cryptographic keys and certificates.

1.5.2. System Transitions out of Secure Boot

A BIOS upgrade or mainboard replacement can disable Secure Boot on systems where it was previously enabled. Additional hardware installed into a machine may have requirements incompatible with Secure Boot (user space DMA, SystemTap support, non-Red Hat kernel modules). This means Secure Boot cannot be a requirement for system functionality, and it is extremely unwise to make it a policy requirement.

1.5.3. No provisioning infrastructure beyond Microsoft Windows

Some system vendors reportedly require a Windows 8 installation to activate Secure Boot. Customers who want to enable Secure Boot might have to obtain a Windows 8 client license, personalized by their system vendor. This scenario would not be relevant if Red Hat relied on a separate trust root under our control, provisioned in cooperation with hardware vendors.

1.5.4. Unproven Revocation Procedures

We do not know if the business processes surrounding revocation actually work. Revocations are complex because they have to be synchronized among operating system vendors to support dual-boot configurations. Without such coordination, a signature on a boot path could be revoked before the underlying operating system had a chance to update it. This would leave systems unbootable.

It is not clear under what circumstances Microsoft will issue an unsolicited revocation. Potential revocation reasons are a failure to reach the security objective (that is, execution of unsigned code in kernel mode is possible under lab conditions), or actual exploitation of such a failure to compromise the boot path of Windows 8 systems outside labs. The latter could also apply to Secure Boot workarounds which load unsigned code after user interaction.

System Configuration

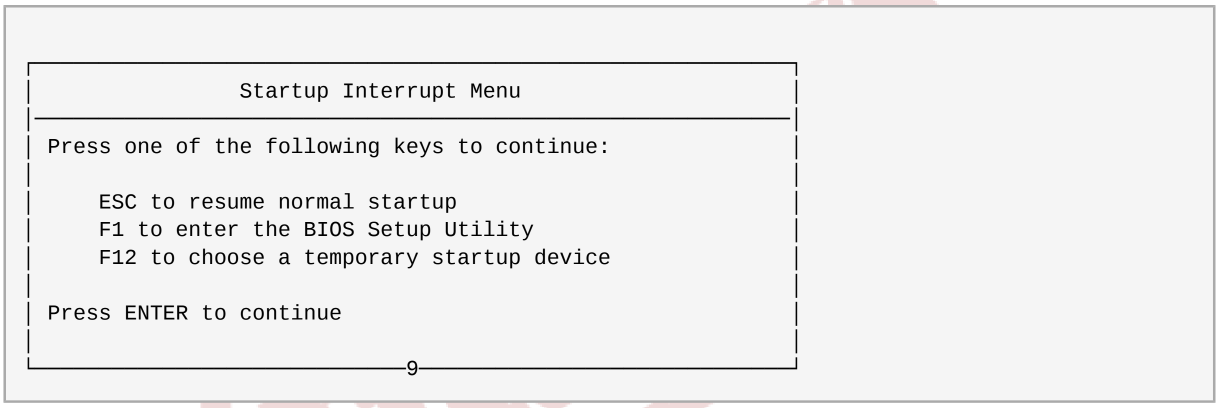
This chapter describes how to configure systems for use with UEFI Secure Boot. The required steps vary from system to system because they depend on how the firmware implements the UEFI specification, but the descriptions should give you an idea where to look.

System can become unbootable

If the operating systems installed on your machine do not provide UEFI boot loaders, or if those boot loaders are not accepted by the new Secure Boot configuration because the signatures are not recognized, your system will no longer boot after switching on Secure Boot.

2.1. Entering the UEFI firmware

When the system is booting, try pressing the **`Enter`**, **`Del`**, **`F1`** or **`Esc`** keys. Usually, instructions will appear telling you how to enter the firmware, similar to *Figure 2.1, “Firmware activation instructions”* (which still refers to the UEFI firmware as *BIOS Setup Utility*, so you are expected to press **`F1`**).



```
┌─────────────────────────────────────────────────────────────┐
│                Startup Interrupt Menu                       │
├─────────────────────────────────────────────────────────────┤
│ Press one of the following keys to continue:                │
│                                                             │
│     ESC to resume normal startup                            │
│     F1 to enter the BIOS Setup Utility                      │
│     F12 to choose a temporary startup device                │
│                                                             │
│ Press ENTER to continue                                     │
│                                                             │
└──────────────────────────────9──────────────────────────────┘
```

Figure 2.1. Firmware activation instructions

You may want to insert a USB stick to prevent a full operating system from booting, so that you can reboot more quickly and try out different keys faster.

If the firmware is protected by a password and you do not know this password, you need to check your system or mainboard manual for password reset instructions.

Once you have entered the firmware, a screen similar to *Figure 2.2, “UEFI firmware start screen”* will be shown.

```
                                 Lenovo BIOS Setup Utility
     Main  Devices  Advanced  Power  Security  Startup  Exit
 ┌──────────────────────────────────────────────────────────────────────────────────────┐
 │                                                                                      │
 │► System Summary                                                                      │
 │► System Time & Date                                                                  │
 │                                                                                      │
 │  Machine Type and Model               0896A9G                                        │
 │  System Brand ID                      Lenovo Product                                 │
 │  System Serial Number                 RUYWEQZ                                        │
 │  Asset Tag                            INVALID                                        │
 │  System UUID                          1846F489-64F1-4714-83D8-A02FD2C79AD1           │
 │  Ethernet MAC address                 D5-3D-7E-60-29-2C                              │
 │  BIOS Revision Level                  F1KT44AUS                                      │
 │  Boot Block Revision Level            F144A                                          │
 │  BIOS Date (MM/DD/YY)                 12/21/2012                                     │
 │  License Status                                                                      │
 │  Language                             [English]                                      │
 │                                                                                      │
 │                                                                                      │
 │                                                                                      │
 │                                                                                      │
 │                                                                                      │
 │                                                                                      │
 │                                                                                      │
 │                                                                                      │
 └──────────────────────────────────────────────────────────────────────────────────────┘
  F1     Help     ↑↓      Select Item     +/-     Change Values       F9     Setup Defaults
  ESC    Exit     ←→      Select Menu     Enter   Select►Sub-Menu     F10    Save and Exit
```

Figure 2.2. UEFI firmware start screen

2.2. Disabling UEFI Secure Boot

Systems which come with Microsoft Windows 8 pre-installed typically have enabled UEFI Secure Boot, and ship the Microsoft keys in the firmware.

The Lenovo desktop system we use as an example makes disabling Secure Boot fairly straightforward. First, enter the firmware as described in *Section 2.1, “Entering the UEFI firmware”*. Press the → key until you reach the *Security* tab, as shown in *Figure 2.3, “UEFI firmware Security tab”*.

```
                              Lenovo BIOS Setup Utility
   Main  Devices  Advanced  Power  Security  Startup  Exit
┌────────────────────────────────────────────────────────┬────────────────────────────────┐
│                                                        │          Help Message          │
│  Hardware Password Manager            [Enabled]        │────────────────────────────────│
│  Secure Boot Status                   [Enabled]        │Select whether to enable or     │
│                                                        │disable Secure Boot             │
│  Adminstrator Password                Not Installed    │[Enabled] Enable Secure         │
│  Power-On Password                    Not Installed    │Boot,BIOS will prevent          │
│                                                        │un-authorised OS be loaded.     │
│  Set Administrator Password           Enter            │[Disable] Disables Secure       │
│  Set Power-On Password                Enter            │Boot.                           │
│                                                        │                                │
│  Allow Flashing BIOS to a Previous    [Yes]            │                                │
│  Version                                               │                                │
│                                                        │                                │
│  Require Admin. Pass. when Flashing   [No]             │                                │
│  Require POP on Restart               [No]             │                                │
│                                                        │                                │
│► Fingerprint Setup                                     │                                │
│► Hard Disk Password                                    │                                │
│► System Event Log                                      │                                │
│► Secure Boot                                           │                                │
│                                                        │                                │
│  Configuration Change Detection       [Disabled]       │                                │
│                                                        │                                │
└────────────────────────────────────────────────────────┴────────────────────────────────┘
 F1      Help    ↑↓      Select Item     +/-     Change Values       F9      Setup Defaults
 ESC     Exit    ←→      Select Menu     Enter   Select►Sub-Menu     F10     Save and Exit
```

Figure 2.3. UEFI firmware Security tab

Press ↓ until you reach the *Secure Boot* item and hit **`Enter`**. The *Image Execution Policy* screen appears (*Figure 2.4, “UEFI firmware Secure Boot settings”*).

```
                              Lenovo BIOS Setup Utility
  Main  Devices  Advanced  Power  Security  Startup  Exit
┌─────────────────────────────────────────────────────────┬────────────────────────────────┐
│                  Image Execution Policy                 │          Help Message          │
├─────────────────────────────────────────────────────────┼────────────────────────────────┤
│  Secure Boot Status                    User Mode        │Select whether to enable or     │
│  Secure Boot                           [Enabled]        │disable Secure Boot             │
│                                                         │[Enabled] Enable Secure         │
│  Reset to Setup Mode                                    │Boot,BIOS will prevent          │
│                                                         │un-authorised OS be loaded.     │
│                                                         │[Disable] Disables Secure       │
│                                                         │Boot.                           │
│                                                         │                                │
│                                                         │                   .            │
│                                                         │                                │
│                                                         │                                │
│                                                         │                                │
│                                                         │                                │
│                                                         │                                │
│                                                         │                                │
│                                                         │                                │
│                                                         │                                │
│                                                         │                                │
│                                                         │                                │
│                                                         │                                │
└─────────────────────────────────────────────────────────┴────────────────────────────────┘
F1     Help     ↑↓      Select Item     +/-     Change Values       F9     Setup Defaults
ESC    Exit     ←→      Select Menu     Enter   Select►Sub-Menu     F10    Save and Exit
```

Figure 2.4. UEFI firmware Secure Boot settings

Make sure that *Secure Boot* is selected, and press **`Enter`**, hit ↑ to choose *Disabled*, and press **`Enter`** again.

The previous step only disables verification of cryptographic signatures, it does not remove some restrictions Microsoft imposes on firmware settings. If you want to boot non-UEFI operating systems, it is necessary to disable the *OS Optimized Defaults*.

2.3. Enabling Microsoft Secure Boot

Systems which do not ship with Microsoft Windows 8 typically do not enable UEFI Secure Boot (or its Microsoft variant). However, many of these systems still contain the Microsoft keys in the firmware, and enabling Microsoft Secure Boot is relatively straightforward.

For example, on a Lenovo desktop system, you need to enter the firmware as described in *Section 2.1, “Entering the UEFI firmware”*. Then press the → key until you reach the *Exit* tab, as shown in *Figure 2.5, “UEFI firmware Exit tab”*.

```
                                Lenovo BIOS Setup Utility
  Main  Devices  Advanced  Power  Security  Startup  Exit
┌────────────────────────────────────────────────────────┬────────────────────────────────┐
│                                                        │          Help Message          │
│  Save Changes and Exit                                 │────────────────────────────────│
│  Discard Changes and Exit                              │Some settings below are         │
│                                                        │changed accordingly. Select     │
│  Load Optimal Defaults                                 │"Enabled" to meet Microsoft(R)  │
│  OS Optimized Defaults                 [Disabled]      │Windows 8 (R) Certification     │
│                                                        │Requirement.                    │
│                                                        │Affected settings are CSM       │
│                                                        │Support, Boot mode, Boot        │
│                                                        │Priority, Secure Boot, Secure   │
│                                                        │RollBack Prevention.            │
│                                                        │                                │
└────────────────────────────────────────────────────────┴────────────────────────────────┘
 F1     Help    ↑↓     Select Item    +/-     Change Values       F9     Setup Defaults
 ESC    Exit    ←→     Select Menu    Enter   Select►Sub-Menu     F10    Save and Exit
```

Figure 2.5. UEFI firmware Exit tab

Press ↓ to select the *OS Optimized Defaults* entry. Press **`Enter`** to change the settings. A confirmation dialog will appear, and need to choose *Yes*. (See *Figure 2.6, “UEFI firmware confirmation for OS Optimized Defaults”*).

```
                                Lenovo BIOS Setup Utility
     Main  Devices  Advanced  Power  Security  Startup  Exit
┌─────────────────────────────────────────────────────────┬────────────────────────────────┐
│                                                         │         Help Message           │
│  Save Changes and Exit                                  │────────────────────────────────│
│  Discard Changes and Exit                               │Some settings below are         │
│                                                         │changed accordingly. Select     │
│  Load Optimal Defaults                                  │"Enabled" to meet Microsoft(R)  │
│  OS Optimized Defaults      ┌───────────────────────────────────────────┐Certification   │
│                             │               Attention!                  │                │
│                             ├───────────────────────────────────────────┤ngs are CSM      │
│                             │  If OS Optimized Defaults is changed to   │mode, Boot      │
│                             │  Enable, some settings including Secure   │re Boot, Secure │
│                             │  Boot,CSM,IPV4 and IPV6 will be changed.  │ntion.          │
│                             │      Do you really want to continue?      │                │
│                             │  Select Yes to continue to Enable the OS  │                │
│                             │            Optimized Defaults.            │                │
│                             │ Select No  to discontinue the operation.  │                │
│                             │                                           │                │
│                             │                                           │                │
│                             │            [Yes]          [No]            │                │
│                             └───────────────────────────────────────────┘                │
│                                                         │                                │
│                                                         │                                │
│                                                         │                                │
│                                                         │                                │
└─────────────────────────────────────────────────────────┴────────────────────────────────┘
  F1     Help     ↑↓     Select Item    +/-     Change Values      F9     Setup Defaults
  ESC    Exit     ←→     Select Menu    Enter   Select►Sub-Menu    F10    Save and Exit
```

Figure 2.6. UEFI firmware confirmation for OS Optimized Defaults

Afterwards, check that *OS Optimized Defaults* has changed to *Enabled*. Press ← several times until you reach the *Security* tab (*Figure 2.3, “UEFI firmware Security tab”*), press ↓ to select *Secure Boot*, hit **`Enter`**, and check that *Secure Boot* is enabled, as in *Figure 2.4, “UEFI firmware Secure Boot settings”*.

Return to the *Exit* tab, choose *Save Changes and Exit*, and press **`Enter`**. Confirm saving the settings, and reboot. Microsoft Secure Boot is now enabled.

2.4. Known issues

When Fedora is installed on an UEFI system, existing boot loaders (for example, the code found in the Master Boot Record) are not overwritten. Therefore, Fedora has considerably less control over the boot process. In some cases, systems cannot dual-boot between Fedora and other operating systems. Even if Fedora is selected manually in the firmware boot loader selection dialog (*choose a temporary startup device* in *Figure 2.1, “Firmware activation instructions”*), the other operating system is started. This is not a problem with UEFI Secure Boot; on the affected systems, it also happens with Secure Boot disabled.

UEFI Secure Boot (and its Microsoft variant) require secure firmware updates. Typically, this is implemented by writing a signed update to a staging area, where the firmware picks it up during the next boot, verifies it, and then proceeds to overwrite the actual firmware. However, this process is still far from foolproof and firmware updates still can make devices unusable, requiring a firmware replacement.

UEFI Secure Boot Implementation

The Fedora Secure Boot implementation includes support for two methods of booting under the Secure Boot mechanism. The first method utilizes the signing service hosted by Microsoft to provide a copy of the shim bootloader signed with the Microsoft keys. The second method is a more general form of the first, wherein a site or user can create their own keys, deploy them in system firmware, and sign their own binaries.

3.1. Keys

The solution to use the Microsoft signing service was one of simplicity. The key Microsoft uses is shipped on all known hardware, which should result in Fedora being able to boot on this hardware without issue. There are of course risks having to rely on a third party for this service. Fedora Project is committed to closely watching activity in this space and will respond to any new information appropriately.

The key usage in the Fedora implementation can be confusing due to its complexity. Here is how the various components are signed.

Shim: This is signed by the UEFI signing service. We do not have control over this key. The shim contains the Fedora Boot CA public key.

GRUB: This is signed by the "Fedora Boot Signer" key, which chains off the Fedora Boot CA key. GRUB doesn't contain any keys, it calls into shim for its verification.

Kernel: This is also signed by the Fedora Boot Signer. The kernel contains the public key used to sign kernel modules.

Kernel Modules: These are signed with a private key generated during build. This key is not saved, a new key is used with each kernel build.

The Fedora Secure Boot CA is used to verify the integrity of GRUB and the kernel. The public key can currently be found in the shim source package. The details of the key are:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2574709492 (0x9976f2f4)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=Fedora Secure Boot CA
        Validity
            Not Before: Dec  7 16:25:54 2012 GMT
            Not After : Dec  5 16:25:54 2022 GMT
        Subject: CN=Fedora Secure Boot CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ae:f5:f7:52:81:a9:5c:3e:2b:f7:1d:55:f4:5a:
                    68:84:2d:bc:8b:76:96:85:0d:27:b8:18:a5:cd:c1:
                    83:b2:8c:27:5d:23:0a:d1:12:0a:75:98:a2:e6:5d:
                    01:8a:f4:d9:9f:fc:70:bc:c3:c4:17:7b:02:b5:13:
                    c4:51:92:e0:c0:05:74:b9:2e:3d:24:78:a0:79:73:
                    94:c0:c2:2b:b2:82:a7:f4:ab:67:4a:22:f3:64:cd:
                    c3:f9:0c:26:01:bf:1b:d5:3d:39:bf:c9:fa:fb:5e:
                    52:b9:a4:48:fb:13:bf:87:29:0a:64:ef:21:7b:bc:
                    1e:16:7b:88:4f:f1:40:2b:d9:22:15:47:4e:84:f6:
                    24:1c:4d:53:16:5a:b1:29:bb:5e:7d:7f:c0:d4:e2:
                    d5:79:af:59:73:02:dc:b7:48:bf:ae:2b:70:c1:fa:
                    74:7f:79:f5:ee:23:d0:03:05:b1:79:18:4f:fd:4f:
                    2f:e2:63:19:4d:77:ba:c1:2c:8b:b3:d9:05:2e:d9:
                    d8:b6:51:13:bf:ce:36:67:97:e4:ad:58:56:07:ab:
                    d0:8c:66:12:49:dc:91:68:b4:c8:ea:dd:9c:c0:81:
                    c6:91:5b:db:12:78:db:ff:c1:af:08:16:fc:70:13:
                    97:5b:57:ad:6b:44:98:7e:1f:ec:ed:46:66:95:0f:
                    05:55
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            Authority Information Access:
                CA Issuers -
URI:https://fedoraproject.org/wiki/Features/SecureBoot

            X509v3 Authority Key Identifier:
                keyid:FD:E3:25:99:C2:D6:1D:B1:BF:58:07:33:5D:7B:20:E4:CD:96:3B:42

            X509v3 Extended Key Usage:
                Code Signing
            X509v3 Subject Key Identifier:
                FD:E3:25:99:C2:D6:1D:B1:BF:58:07:33:5D:7B:20:E4:CD:96:3B:42
    Signature Algorithm: sha256WithRSAEncryption
         37:77:f0:3a:41:a2:1c:9f:71:3b:d6:9b:95:b5:15:df:4a:b6:
         f4:d1:51:ba:0d:04:da:9c:b2:23:f0:f3:34:59:8d:b8:d4:9a:
         75:74:65:80:17:61:3a:c1:96:7f:a7:c1:2b:d3:1a:d6:60:3c:
         71:3a:a4:c4:e3:39:03:02:15:12:08:1f:4e:cd:97:50:f8:ff:
         50:cc:b6:3e:03:7d:7a:e7:82:7a:c2:67:be:c9:0e:11:0f:16:
         2e:1e:a9:f2:6e:fe:04:bd:ea:9e:f4:a9:b3:d9:d4:61:57:08:
         87:c4:98:d8:a2:99:64:de:15:54:8d:57:79:14:1f:fa:0d:4d:
         6b:cd:98:35:f5:0c:06:bd:f3:31:d6:fe:05:1f:60:90:b6:1e:
         10:f7:24:e0:3c:f6:33:50:cd:44:c2:71:18:51:bd:18:31:81:
         1e:32:e1:e6:9f:f9:9c:02:53:b4:e5:6a:41:d6:65:b4:2e:f1:
         cf:b3:b8:82:b0:a3:96:e2:24:d8:83:ae:06:5b:b3:24:74:4d:
         d1:a4:0a:1d:0a:32:1b:75:a2:96:d1:0e:3e:e1:30:c3:18:e8:
         cb:53:c4:0b:00:ad:7e:ad:c8:49:41:ef:97:69:bd:13:5f:ef:
         ef:3c:da:60:05:d8:92:fc:da:6a:ea:48:3f:0e:3e:73:77:fd:
         a6:89:e9:3f
```

Figure 3.1. Fedora X.509 certificate for signing Kernel and GRUB

3.2. Shim

Fedora uses a first-stage boot loader called shim which embeds a self-signed CA certificate. This CA is then used to verify the GRUB 2 boot loader (UEFI version, a PE/COFF program signed with AuthentiCode). Before booting a kernel, GRUB 2 calls back into shim to verify the AuthentiCode signature of the kernel.

In addition to the Microsoft key, shim also supports additional trusted certificates provided by the owner and a mechanism to disable signature verification. The information is kept in UEFI variables which cannot be written after an operating system has booted. Shim contains a physical presence check and asks for confirmation before it updates the settings stored in these UEFI variables.

In Fedora there are two packages that make up shim. The package named "shim" is the result of compiling the source code that makes up shim. This package will not boot the system as it is not signed. The results of building the shim package are signed, then incorporated into the shim-signed package. The shim-signed package contains the signed binary that is capable of booting the system.

The shim package also contains a blacklist of known bad keys or binaries that should not be allowed to boot. The blacklist is a file called dbx.esl in the shim package. This blacklist is currently embedded into the shim binary at build time. It exists to prevent a known exploitable version of grub from being booted. Future developments may see this blacklist moved into UEFI memory. In its current form, updating the blacklist will not provide additional security as you could downgrade the shim package to avoid updating the blacklist. If the blacklist is stored in the BIOS, a blacklist update would survive a shim downgrade.

Additionally there is a blacklist which Microsoft maintains, signs, and is stored in the BIOS for checking. Microsoft will provide this list to Fedora Project for inclusion. This may create periodic updates to the shim-signed package that do not change the actual shim binary. This blacklist file does not currently exist as nothing has been blacklisted. The blacklist will likely be packaged on its own to avoid having to update the shim-signed package.

The details about this blacklist come from Microsoft and Fedora Project is not able to update this blacklist. The data is signed with a Microsoft key which will prevent unauthorized updates to this list. Microsoft has suggested that the blacklist is to be used to prevent the computer from booting compromised keys and known vulnerabilities.

In both boot methods, shim, GRUB, and the kernel will detect that they are started in what UEFI describes as "User mode" with Secure Boot enabled, and upon detecting this they will validate the next stage with a Fedora-specific cryptographic public key before starting. The validation is done via shim for GRUB, and GRUB calls back to shim to validate the kernel as well. Once the kernel is booted, it will also detect that it is in Secure Boot mode, which will cause several things to be true:
it will validate the boot command line to only allow certain kernel settings
it will check modules at load time for signatures and refuse to load them if they are unsigned or signed with a signature not found in the UEFI key store variables (see note)
it will refuse any operations from userland which cause userland-defined DMA.

Additional information on shim can be found on the *shim development repository*[1].

[1] https://github.com/mjg59/shim

Note

Other distributions have chosen to not require signed kernel modules in their Secure Boot implementation. Fedora believes that to fully support Secure Boot this is required. We are working to limit the impacts of this while ensuring that untrusted module code is not allowed to execute.

3.3. GRUB

GRUB is contained in the grub2 package and is signed with the Fedora CA key. Once the binary is cryptographically verified it is executed by shim. The GRUB package does not contain any key material. When GRUB needs to verify the integrity of the Kernel it will call back into shim to execute the actual check.

3.4. Kernel

The Kernel is signed with the Fedora CA key and is verified by shim before GRUB will allow execution. Modules the kernel loads are signed with a key that is generated at kernel build time, then destroyed.

The Kernel will import the UEFI key database and use that to verify Kernel modules. This means that 3rd party kernel modules signed with a key trusted by UEFI (Microsoft), will load in the Kernel while Secure Boot is enabled.

Important

It is important to note that once the kernel loads the initial ramdisk (initrd), execution is no longer trusted. While the initrd may contain signed kernel modules, it also contains unsigned userspace code. The integrity of this code cannot be guaranteed.

3.4.1. Restrictions

Restrictions are in place to be fully compliant with Secure Boot. This requires us to prevent any execution of unverified code at the supervisor level. Most users won't notice these restrictions as most of the userspace packages that required such access have been fixed to work without it. There are, however, a few services or features that will not work in a Secure Boot enabled machine at this time. They include:

kexec/kdump
hibernate (suspend to disk)
third party modules that are unsigned, or signed with an unknown key
systemtap kernel probing (and kprobes)

Note

In future iterations of Secure Boot support the above may also be possible, however secure implementations were not feasible in the Fedora 18 timeframe. If you require support of any of these features, Secure Boot must be disabled.

Important

Currently, the Fedora shim was signed in a way that gives it an expiration date of October 2013, prior to the Fedora 18 end-of-life. We are not aware of any hardware that honors this expiration date, but it's not out of the question. This is the date Microsoft gave the signature, Fedora has no control over it. We are investigating this issue and expect to resolve it in the future.

Tools

Several tools have been developed to allow Fedora to work with the UEFI Secure Boot firmware.

4.1. Shim

Shim is the cryptographically signed software that creates the trust between the UEFI firmware and GRUB and the kernel software. Shim is cryptographically signed by Verisign (via Microsoft) so that the UEFI firmware will cryptographically recognize the Fedora system and allow the software to continue through the boot process. The shim validates GRUB and kernel through a cryptographic verification based on a Fedora key used to sign all three.

4.2. Pesign

Pesign allows users to create their own shim and use their own cryptographic keys. Using this tool, one can create their own trust model and not be required to trust the Microsoft trust model. Once the user has created their keys and signed their shim, and optionally signed and built GRUB and kernel, they can use the setup mode in the firmware to install Fedora and use the *sbsetup* tool as provided by pesign to enroll their keys in the firmware.

4.3. EFIKeyGen

4.4. sign-file

Using your own keys

5.1. Creating keys

5.2. Making keys for shim's build

5.3. Packages that need rebuilding

5.4. Enrolling your keys in firmware

Appendix A. Revision History

Revision 18.4	**Tue 11 March 2013**	**Eric Christensen** *sparks@fedoraproject.org*

Added clarifications to certain sections.

Revision 18.3	**Tue 19 February 2013**	**Eric Christensen** *sparks@fedoraproject.org*

Included information from Florian's repo.
Added figures.

Revision 18.2	**Wed 06 February 2013**	**Eric Christensen** *sparks@fedoraproject.org*

Added text to all chapters to include information on the public keys.

Revision 18.1	**Fri 04 January 2013**	**Eric Christensen** *sparks@fedoraproject.org*

Updated 'What is Secure Boot' chapter. (BZ 891758)
Updated 'Implementation' chapter. (BZ 891924)
Updated Josh Boyer's email address. (BZ 891932)

Revision 0	**Thu Jul 12 2012**	**Eric Christensen** *sparks@fedoraproject.org*

Initial creation of book by publican

Index