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)

Copyright © 2012-2013 Fedora Project Contributors.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at http://creativecommons.org/licenses/by-sa/3.0/. The original authors of this document, and Red Hat, designate the Fedora Project as the "Attribution Party" for purposes of CC-BY-SA. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

For guidelines on the permitted uses of the Fedora trademarks, refer to https://fedoraproject.org/wiki/Legal:Trademark_guidelines.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

All other trademarks are the property of their respective owners.
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 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 independent 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](image)

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.
Chapter 1. What is UEFI Secure Boot?

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:
61:07:76:56:00:00:00:00:08
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
(2048 bit)
Modulus:
[...]
48:03
Exponent: 65537 (0x10001)
X509v3 extensions:
1.3.6.1.4.1.311.21.1:
02:01:00
X509v3 Subject Key Identifier:
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
[...]
-----BEGIN CERTIFICATE-----
MIIF1zCCA7+gAwIBAgIKYQd2VgAAAAAACDANBgkqhkiG9w0BAQsFADCBiDELMAkGA1UEBhMCVVMxEzARBgNVBAgTCldhc2hpbmd0b24xEDAOBgNVBAcTB1JlZG1vbmQx
HjAcBgNVBAoTFU1pY3Jvc29mdCBDb3Jwb3JhdGlvbiBQQ0EgMjAxMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAN0Mu6LkLgnj58X3lmm8ACG9aTMz760Ey1SA7gaDu8UghNn30ovzOLCrpK0tfjG3jBF/j]
-----END CERTIFICATE-----

Figure 1.2. Microsoft Trusted X.509 Certificate for their Secure Boot implementation
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.
Figure 1.3. Microsoft X.509 certificate for third-party UEFI applications
Chapter 1. What is UEFI Secure Boot?

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 acpirspd 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).

![Figure 2.1. Firmware activation instructions](image)

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.
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”.

Figure 2.2. UEFI firmware start screen
Disabling UEFI Secure Boot

**Lenovo BIOS Setup Utility**

<table>
<thead>
<tr>
<th>Main</th>
<th>Devices</th>
<th>Advanced</th>
<th>Power</th>
<th>Security</th>
<th>Startup</th>
<th>Exit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Password Manager</td>
<td>[Enabled]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure Boot Status</td>
<td>[Enabled]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Administrator Password</td>
<td>Not Installed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power-On Password</td>
<td>Not Installed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set Administrator Password</td>
<td>Enter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set Power-On Password</td>
<td>Enter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allow Flashing BIOS to a Previous Version</td>
<td>[Yes]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Require Admin. Pass. when Flashing</td>
<td>[No]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Require POP on Restart</td>
<td>[No]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>► Fingerprint Setup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>► Hard Disk Password</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>► System Event Log</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>► Secure Boot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration Change Detection</td>
<td>[Disabled]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Help Message**

- Hardware Password Manager: [Enabled]
- Secure Boot Status: [Enabled]
- Administrator Password: Not Installed
- Power-On Password: Not Installed
- Set Administrator Password: Enter
- Set Power-On Password: Enter
- Allow Flashing BIOS to a Previous Version: [Yes]
- Require Admin. Pass. when Flashing: [No]
- Require POP on Restart: [No]

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”).
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”. 

---

**Figure 2.4. UEFI firmware Secure Boot settings**

- **Secure Boot Status**: Secure Boot [Enabled]
- **Reset to Setup Mode**: [Disabled]

Select whether to enable or disable Secure Boot. [Enabled] Enable Secure Boot, BIOS will prevent un-authorised OS be loaded. [Disable] Disables Secure Boot.
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”).
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:
Chapter 3. UEFI Secure Boot Implementation

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:
      24:1c:4d:53:16:5a:ba:1c:29:bb:5e:7d:7f:0c:04:e2:
      05:55
  Exponent: 00:ae:f5:f7:52:81:a9:5c:3e:2b:f7:1d:55:f4:5a:
X509v3 extensions:
  Authority Information Access:
    URI:https://fedoraproject.org/wiki/Features/SecureBoot
X509v3 Authority Key Identifier:
X509v3 Extended Key Usage:
  Code Signing
X509v3 Subject Key Identifier:
Signature Algorithm: sha256WithRSAEncryption
  71:3a:a4:4c:3e:39:03:02:15:12:08:1f:4e:cd:97:50:ff:ff:
  50:cc:b6:3e:03:7d:7a:72:7a:cc:2e:be:c9:0e:11:ff:16:
  a6:89:e9:3f

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

16
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
Chapter 3. UEFI Secure Boot Implementation

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

<table>
<thead>
<tr>
<th>Revision 18.4</th>
<th>Tue 11 March 2013</th>
<th>Eric Christensen</th>
<th><a href="mailto:sparks@fedoraproject.org">sparks@fedoraproject.org</a></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Added clarifications to certain sections.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Revision 18.3</th>
<th>Tue 19 February 2013</th>
<th>Eric Christensen</th>
<th><a href="mailto:sparks@fedoraproject.org">sparks@fedoraproject.org</a></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Included information from Florian's repo.</td>
<td>Added figures.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Revision 18.2</th>
<th>Wed 06 February 2013</th>
<th>Eric Christensen</th>
<th><a href="mailto:sparks@fedoraproject.org">sparks@fedoraproject.org</a></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Added text to all chapters to include information on the public keys.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Revision 18.1</th>
<th>Fri 04 January 2013</th>
<th>Eric Christensen</th>
<th><a href="mailto:sparks@fedoraproject.org">sparks@fedoraproject.org</a></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Updated 'What is Secure Boot' chapter. (BZ 891758)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated 'Implementation' chapter. (BZ 891924)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated Josh Boyer's email address. (BZ 891932)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Revision 0</th>
<th>Thu Jul 12 2012</th>
<th>Eric Christensen</th>
<th><a href="mailto:sparks@fedoraproject.org">sparks@fedoraproject.org</a></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Initial creation of book by publican</td>
<td></td>
</tr>
</tbody>
</table>
Index

E
EFIKeyGen
  definition, 21

F
Fedora Secure Boot CA, 15
feedback
  contact information for this manual, vii

G
GRUB
  integrity, 15
  key, 15

K
kernel
  integrity, 15
  key, 15
keys
  expiration
    Fedora 18, 19
Fedora Secure Boot CA
  public key, 15
GRUB, 15
kernel, 15
shim, 15

P
pesign
  definition, 21
  sbsetup, 21

S
Secure Boot
  blacklist, 17
  implementation, 15
  keys, 15
    (see also keys)
    restrictions, 18
shim
  definition, 21
  explanation, 17
  key, 15
sign-file
  definition, 21