UEFI Secure Boot in Red Hat Enterprise Linux 7

Author Name: Yogesh Babar
Editor: Chris Negus
08/28/2014

The UEFI Secure Boot specification was created to secure the boot process by protecting attack points from intrusion and preventing malicious component replacement. Red Hat Enterprise Linux 7 offers UEFI Secure Boot support by including a kernel and all associated drivers that are signed by a UEFI CA certificate.

The Unified Extensible Firmware Interface Forum (`www.uefi.org`) developed the UEFI Secure Boot specification with the support of many leading technology companies. Consumers who boot UEFI-enabled systems can feel secure that the firmware, drivers, and operating system components that are used to boot their computers have been certified to belong to reputable players in the computing industry and have not been tampered with.

It is important to note that secure boot was not created as a Microsoft Windows 8 feature. However, Microsoft has chosen to implement UEFI secure boot for their Windows 8 OS and requires all PCs that want to have the Windows 8-certified logo to ship with the secure boot feature enabled. This has, of course, become a bit of a pain for people who want to dual-boot, either with Linux or any other OS.

Although some have complained that Secure Boot can “lock out” other operating systems, the stated purpose of UEFI is to prevent low-level malware from getting between the hardware and operating system.

WHAT IS SECURE BOOT?

Secure Boot is defined in Chapter 28 of the UEFI specification (http://www.uefi.org/specifications). It’s a pretty clever mechanism. But what it does can be described very simply. It says the firmware can contain a set of signatures and refuse to run EFI executables that are not signed with one of those signatures.

We at Red Hat do the same with our RPM packages: we sign them with our private key and put our public key on website where everyone can access it or in our RPM database on the local system. That way, whenever some one tries to install any RPM package its signature can be checked with the public key. Secure boot works the same way.

Microsoft's role in secure boot

Just as Red Hat has its hardware certification program, Microsoft has its Windows Hardware Certification Requirements. Microsoft's program says that it's compulsory to have Windows 8 with secure boot enabled. That means if a hardware vendor wants to certify their PC hardware with Windows 8, then its UEFI must have secure boot enabled. If, on the other hand, a system is not Microsoft certified, then it can boot Windows 8 by disabling the secure boot feature. Computers complying with the Windows Hardware Certification requirements must:

- Ship with Secure Boot turned on (except for servers).
- Have Microsoft’s key in the list of keys they trust.
- Disable BIOS compatibility mode when Secure Boot is enabled.
- Support signature blacklisting.

Any x86 computers complying with the requirements must additionally:

- Allow a physically present person to disable Secure Boot
- Allow a physically present person to enable setup mode and be able to modify the list of keys the firmware trusts.

The Microsoft certification requirements, for x86 machines, explicitly require implementers to give a physically present user complete control over Secure Boot – turn it off, or completely control the list of keys it trusts. Another important note here is that while the certification requirements state that the out-of-the-box list of trusted keys must include Microsoft's key, they don't say, for example, that it must not include any other keys. The requirements explicitly and intentionally allow for the system to ship with any number of other trusted keys, too.

Using signed components

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 UEFI 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.

UEFI specification specifies the following for secure boot:

- 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
- Procedures to revoke known-bad certificates and application hashes.

RHEL 7 AND SECURE BOOT

Red Hat Enterprise Linux 7 includes support for the UEFI Secure Boot feature. This means that RHEL 7 can be installed and run on systems where UEFI secure boot is enabled. (RHEL 7.0, however, does not require the use of Secure Boot on UEFI systems.) When Secure Boot is enabled, the EFI OS boot loaders, the RHEL kernel, and kernel modules must be signed with a private key and authenticated with the corresponding public key.

Most of the PC hardware you can buy these days is Windows 8 certified and if the hardware comes with pre-installed Windows 8, then secure boot will be enabled by default. Because this is the case, we as Red Hat had the following options:

- **Producing a Red Hat key and encourage hardware vendors to incorporate it.** But:
  - ➢ Its not possible to engage each and every hardware vendor.
  - ➢ Creating a Red Hat specific key will create a problem for other Linux distributions.
- **Producing a overall Linux key which will be applicable to all the distributions.** This is very expensive as we need to maintain the key along with its security (as we need to share it securely with every distro).

- **Use the Microsoft's signing service and sign our first stage boot loader with Microsoft's key.** This looks like a more practical way and its less expensive as well, as the key Microsoft uses is shipped on all known hardware. This should result in RHEL 7 being able to boot on this hardware without issue. There are of course risks having to rely on a third party for this service. Red Hat is committed to closely watching activity in this space and will respond to new information appropriately.

So RHEL 7 uses the Microsoft's signing service and signs the first stage boot loader with Microsoft's key. This raises a few questions:

- Can I use RHEL 7 by disabling the secure boot?
- How can I enable secure boot?
- Can I install RHEL 7 with secure boot and then can I disable it?
- What is a first stage boot loader? Is it GRUB 2 or something else?
- How do I validate that kernel, initramfs or kernel modules are not compromised?

Let's review these one by one.

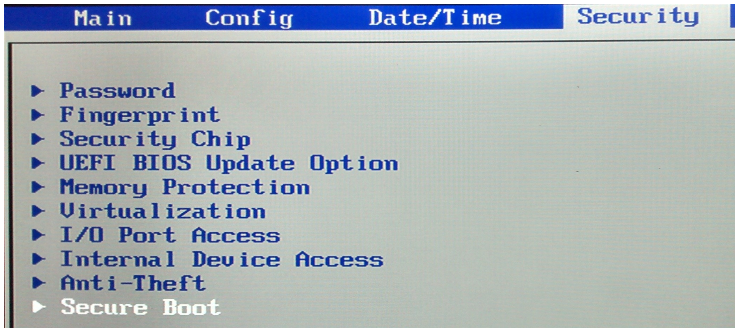
Can I use RHEL 7 by disabling the secure boot?

Yes RHEL 7 does not require secure boot to be enabled.

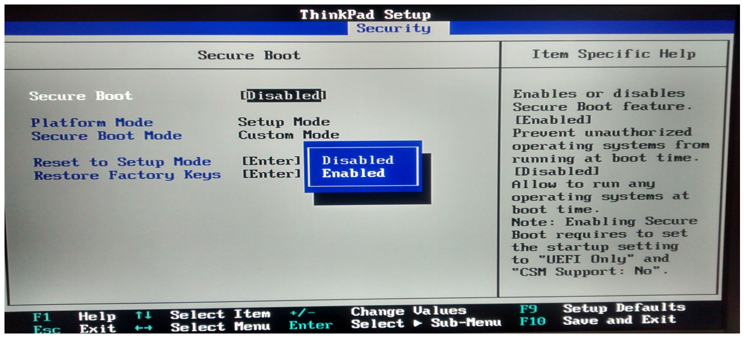
How can I enable secure boot?

How you enable secure boot varies from vendor to vendor. The following example illustrates how to enable secure boot on a Lenovo T440p:

1. From the BIOS screen select Security tab -> Secure Boot:



2. From Secure Boot, select the Enabled option.



```
Secure Boot              [Disabled]
Platform Mode            Setup Mode
Secure Boot Mode         Custom Mode
Reset to Setup Mode      [Enter]   Disabled
Restore Factory Keys     [Enter]   Enabled
```

Enables or disables Secure Boot feature.
[Enabled]
Prevent unauthorized operating systems from running at boot time.
[Disabled]
Allow to run any operating systems at boot time.
Note: Enabling Secure Boot requires to set the startup setting to "UEFI Only" and "CSM Support: No".

```
F1   Help   ↑↓ Select Item   +/-   Change Values        F9  Setup Defaults
Esc  Exit   ↔  Select Menu   Enter Select ▶ Sub-Menu    F10 Save and Exit
```

3. Save and Exit (press F10).

Can I install RHEL 7 with secure boot and then can I disable it?

Yes, you can disable it. The system will boot in insecure mode.

What is first stage boot loader? Is it GRUB 2 or something else?

First stage loader is not GRUB 2. It is called as Shim. Shim will be signed with Microsoft's signing service (Microsoft's key). The hardware will have Microsoft's key, so the shim will be allowed to run by UEFI-Secure-Boot.

Why shim? Why not sign a GRUB itself with Microsoft's key?

Signing through the Microsoft signing service is a manual process, and that means it is time consuming. If we use GRUB 2 instead of shim then we need to sign the GRUB all the time when we do small changes in it. This will unnecessarily delay the boot loader updates. So Red Hat has chosen a multilayer approach. Shim is used as a very simple bootloader, doing nothing other than loading a real boot loader (GRUB 2).

How do I validate that GRUB, kernel, initramfs and kernel modules aren't compromised?

The secure boot mechanism's main purpose is to stop running untrusted programs during boot time. So far, we understand that Shim will be signed with Microsoft's signing service, the key which is in firmware (UEFI) will validate the Shim bootloader and Shim will call the GRUB bootloader. But how can you make sure that GRUB is not compromised? Here is the trick:

The Red Hat public key is embedded inside the shim module. GRUB 2 is signed with a Red Hat Private key. That embedded key from Shim is used to check the signature on GRUB 2. The kernel is again signed with the Red Hat Private key. GRUB 2 then uses the same key from Shim to check the signature on the kernel. Kernel modules are signed with a kernel-specific private key. The kernel's public key is embedded inside the kernel itself. When the kernel attempts to load modules, it checks their signature against the public key embedded in the kernel image. That's confusing, so lets make it easier:

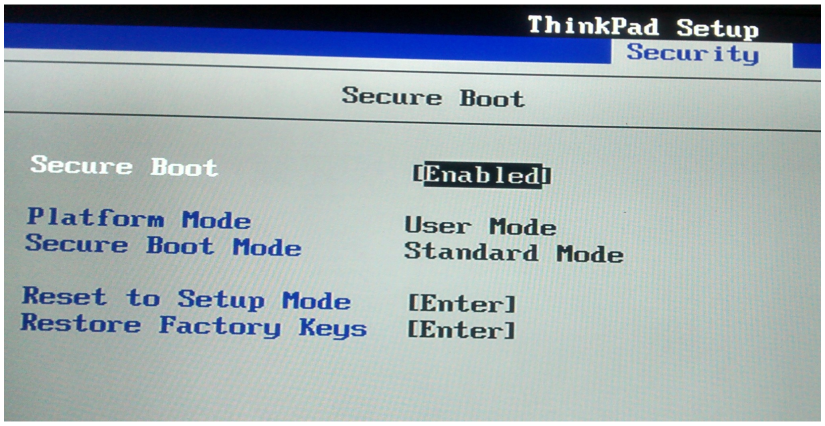
1. Microsoft's public key is available in UEFI firmware.
2. Shim is signed with Microsoft's private key with the help of Microsoft signing services. Red Hat does not have control over this key.
3. Shim is shipped with Red Hat's public key.
4. GRUB 2 is signed with Red Hat's private key.
5. The kernel is signed with Red Hat's private key.
6. GRUB 2 does not contain any key. It uses the public key shipped with shim to validate the kernel.
7. The kernel image is shipped with its own public key, which will be used to validate the modules. A public/private key pair is generated during the kernel build process. The public key is embedded in the kernel image. The modules are signed with the private key and the private key is destroyed at the end of the kernel build process. The private key is not saved. A new key is used with each kernel build.

Secure Boot protects code in kernel space, not user space. So initramfs validation does not fall under the Secure Boot mechanism.

INSTALLING A RHEL 7 BETA WITH SECURE BOOT

Remember that not every UEFI enabled system supports secure boot. While RHEL 7 supports secure boot, RHEL 7 Beta only supported secure boot in standard mode. The following steps show how to check for secure boot support, then install Red Hat Enterprise Linux 7 Beta with UEFI Secure Boot enabled in standard mode:

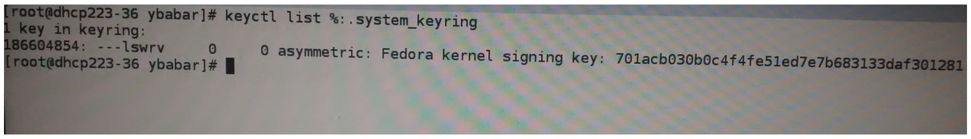
1. **Select the appropriate mode from UEFI firmware**: There are two types of modes in UEFI secure boot. Which is available depends upon the hardware vendor.
   - **Standard mode**: In this mode UEFI firmware does not allow you to enroll your own Key to validate the boot loader.



Secure Boot

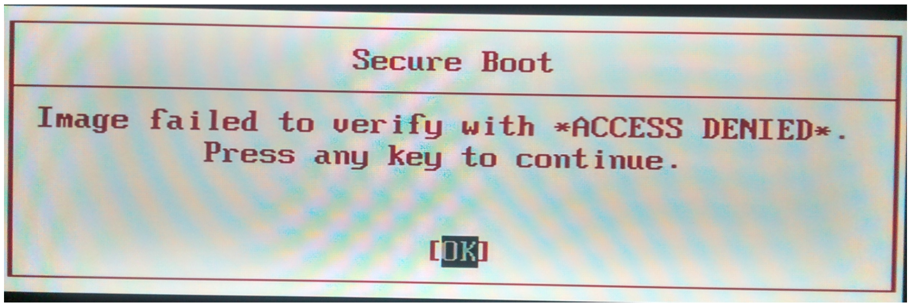
```
Secure Boot              [Enabled]

Platform Mode            User Mode
Secure Boot Mode         Standard Mode

Reset to Setup Mode      [Enter]
Restore Factory Keys     [Enter]
```

In this example, we are installing RHEL 7 Beta. Its shim is not signed with Microsoft's signing service so for secure boot it's an unauthorized program. So secure boot prevents it from running. Earlier on this computer, Fedora was installed by disabling the secure boot. You can display information about the keys on the system key rings using the 'keyctl' utility. The following is an abbreviated example output from a Fedora system where UEFI Secure Boot is not enabled:



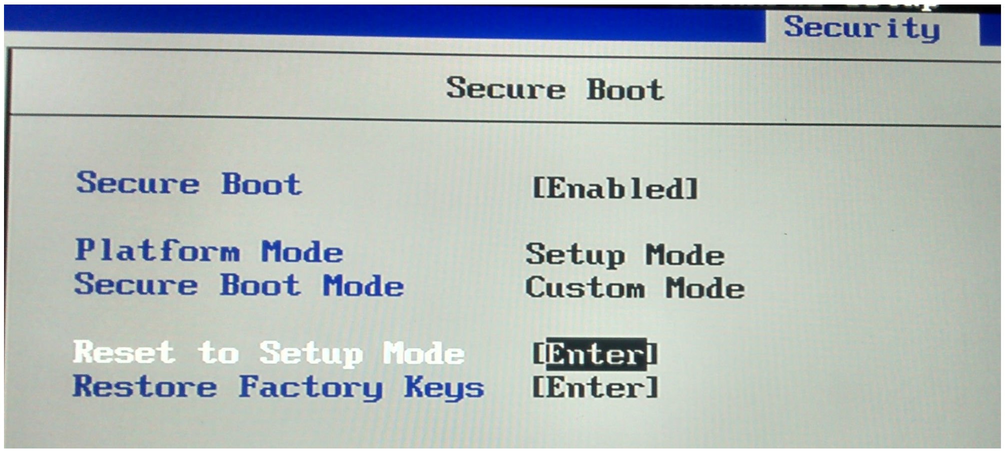
```
[root@dhcp223-36 ybabar]# keyctl list %:.system_keyring
1 key in keyring:
186604854: ---lswrv     0     0 asymmetric: Fedora kernel signing key: 701acb030b0c4f4fe51ed7e7b683133daf301281
[root@dhcp223-36 ybabar]#
```

To install a RHEL 7 Beta system, you would have to enable a secure boot in standard mode. Installation fails with the typical error message from UEFI secure boot:



```
Secure Boot

Image failed to verify with *ACCESS DENIED*.
Press any key to continue.

[OK]
```

- **Setup mode**: With the secure boot you can clear the current Platform key and can install your own platform key. This allows you to customize the secure boot signature databases.



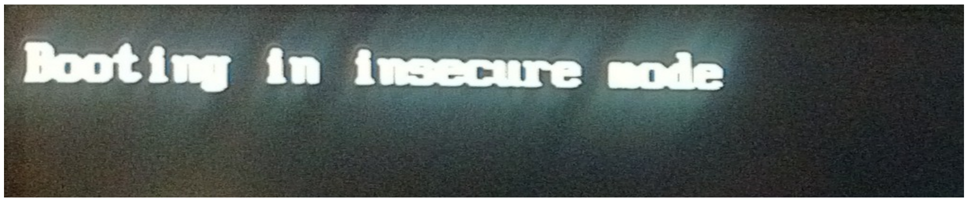
Security

Secure Boot

```
Secure Boot              [Enabled]

Platform Mode            Setup Mode
Secure Boot Mode         Custom Mode

Reset to Setup Mode      [Enter]
Restore Factory Keys     [Enter]
```

2. **Put the system in setup mode**. Consult system documentation provided by the hardware vendor for assistance with this step.

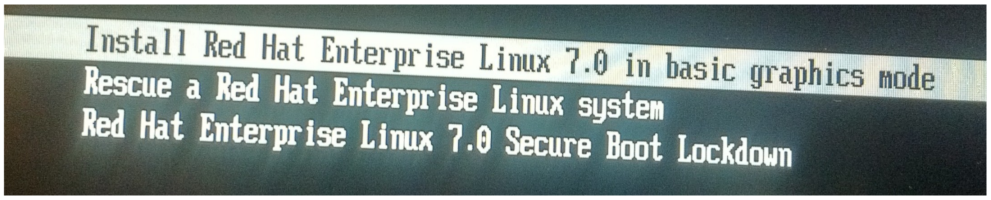
3. **Reboot the machine from Red Hat Enterprise Linux 7 Beta installation media.** This time UEFI firmware allows us to boot with the installation disk:



```
Booting in insecure mode
```

The fact that the system is booting in insecure mode means it has failed to verify the boot loader because we have not enrolled the key yet. Once we include the key in UEFI, the firmware system will not show this Message.

4. **Select Secure Boot Lockdown.** Select the entry on the installation media labeled *Red Hat Enterprise Linux 7.0 Secure Boot Lockdown*:



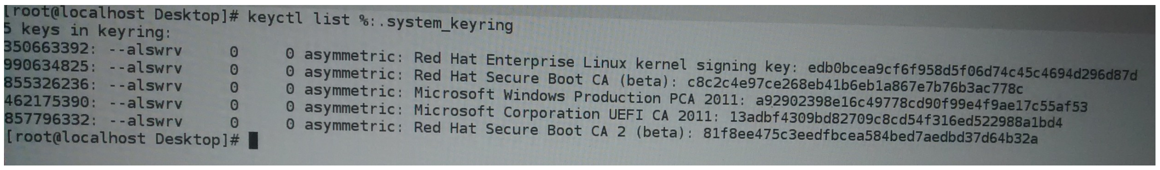
```
Install Red Hat Enterprise Linux 7.0 in basic graphics mode
Rescue a Red Hat Enterprise Linux system
Red Hat Enterprise Linux 7.0 Secure Boot Lockdown
```

The lockdown process will add a RHEL 7 specific platform key in UEFI firmware. Now this key will be used to validate the shim boot loader of RHEL 7.

NOTE: The final release of Red Hat Enterprise Linux 7.0 does not require these steps. This process is only required for Red Hat Enterprise Linux 7.0 Beta because the Beta release has its own specific key. The final release uses a key signed by the standard UEFI vendor certificate authority chain.

Once the lockdown process is complete, reboot the system. Completion of the lockdown process may vary by hardware vendor. Consult system documentation provided by the hardware vendor for assistance with this step.

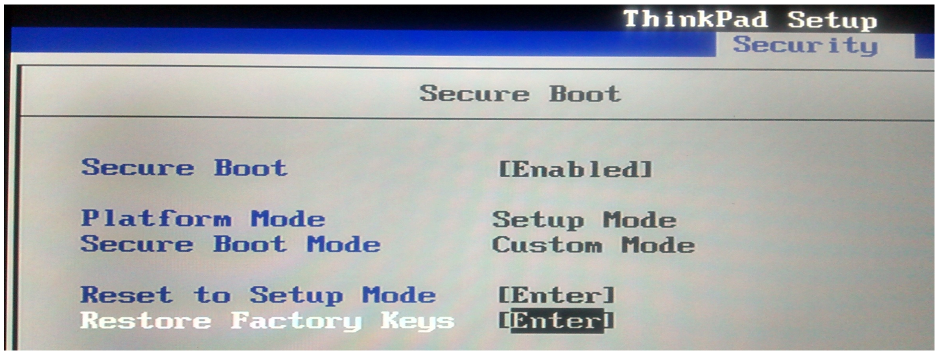
5. **Verify keys.** Verify in your firmware that keys are enrolled and that thee computer is in enforcing mode. Consult system documentation provided by the hardware vendor for assistance with this step. The system which was used for this testing does not have facility to see the enrolled keys.
6. **Reboot.** Reboot from Red Hat Enterprise Linux 7 Beta installation media again.
7. **Install.** Perform the installation. When finished, remove the installation media and reboot the system.
8. **Check keys.** The following is a abbreviated example output from a RHEL 7 beta system where UEFI secure boot is enabled:



```
[root@localhost Desktop]# keyctl list %:.system_keyring
5 keys in keyring:
350663392: --alswrv     0     0 asymmetric: Red Hat Enterprise Linux kernel signing key: edb0bcea9cf6f958d5f06d74c45c4694d296d87d
990634825: --alswrv     0     0 asymmetric: Red Hat Secure Boot CA (beta): c8c2c4e97ce268eb41b6eb1a867e7b76b3ac778c
855326236: --alswrv     0     0 asymmetric: Microsoft Windows Production PCA 2011: a92902398e16c49778cd90f99e4f9ae17c55af53
462175390: --alswrv     0     0 asymmetric: Microsoft Corporation UEFI CA 2011: 13adbf4309bd82709c8cd54f316ed522988a1bd4
857796332: --alswrv     0     0 asymmetric: Red Hat Secure Boot CA 2 (beta): 81f8ee475c3eedfbcea584bed7aedbd37d64b32a
[root@localhost Desktop]# █
```

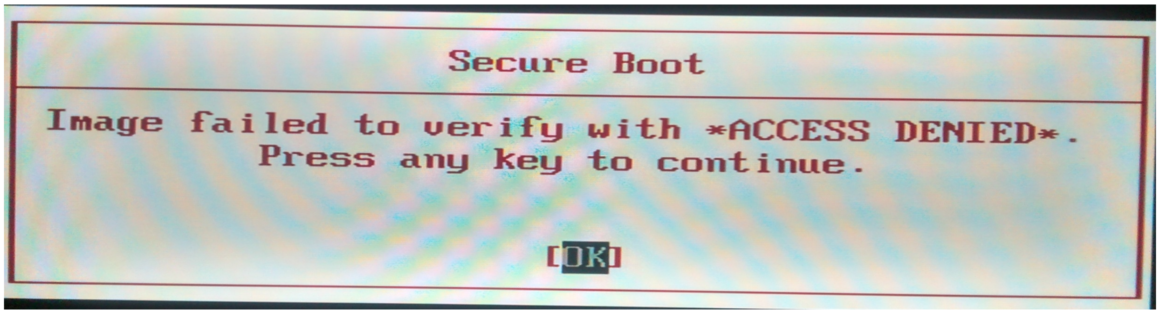
HOW DO YOU FLUSH EXISTING KEYS FROM UEFI FIRMWARE?

The steps depends on hardware vendor. For this demonstration, using a 'Lenovo T440p' system, select the Security tab -> Restore Factory Keys, then choose Yes. This option is used to restore all keys and certificates in Secure Boot databases to factory defaults. Any customized Secure Boot settings will be erased, and the default Platform Key will be re-established along with the original signature databases including certificate for Microsoft Windows 8.



```
ThinkPad Setup
Security

Secure Boot

Secure Boot              [Enabled]

Platform Mode            Setup Mode
Secure Boot Mode         Custom Mode

Reset to Setup Mode      [Enter]
Restore Factory Keys     [Enter]
```

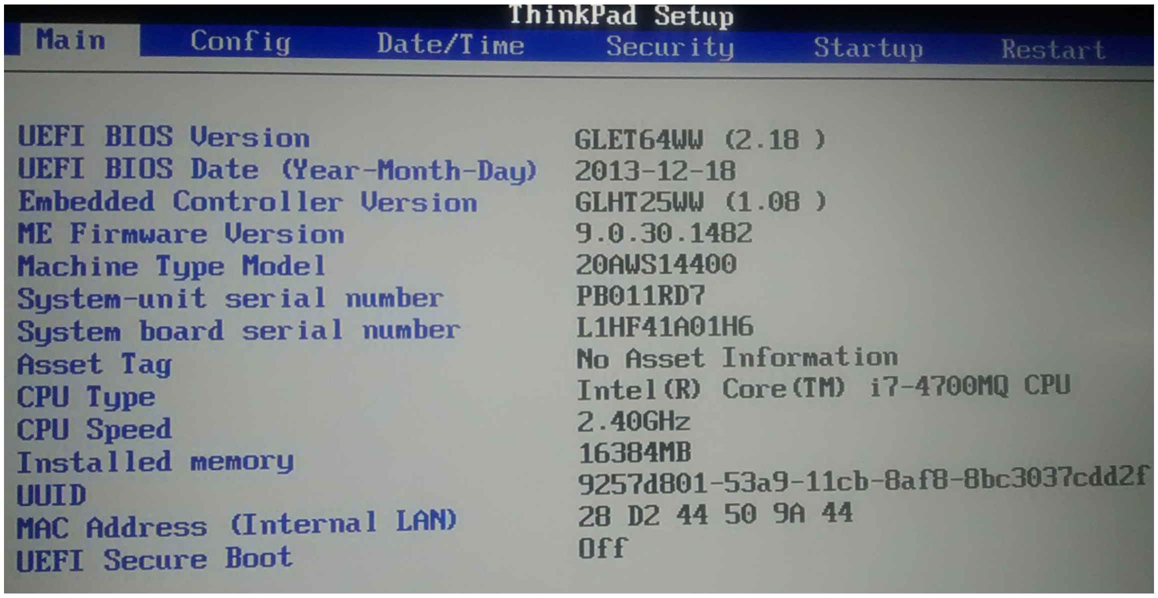
Please note that secure boot is still enabled but there is only Microsoft's key available in firmware and the RHEL 7 Beta key is erased. In this case, Secure boot should not allow RHEL 7 to boot and we should get:



```
Secure Boot

Image failed to verify with *ACCESS DENIED*.
Press any key to continue.

[OK]
```

In this case, however, RHEL 7 beta still managed to boot. This is because you can see in the screen shot below the selected mode is ‘Setup mode’ and secure boot mode is ‘custom mode’ (not enforcing mode). So secure boot allows the system to boot, but will be booted in ‘insecure mode’.

That means that secure boot is enabled, but not enforced:

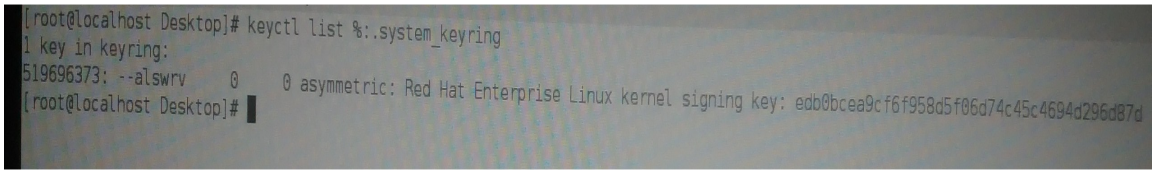


```
UEFI BIOS Version                      GLET64WW (2.18 )
UEFI BIOS Date (Year-Month-Day)        2013-12-18
Embedded Controller Version            GLHT25WW (1.08 )
ME Firmware Version                    9.0.30.1482
Machine Type Model                     20AWS14400
System-unit serial number              PB011RD7
System board serial number             L1HF41A01H6
Asset Tag                              No Asset Information
CPU Type                               Intel(R) Core(TM) i7-4700MQ CPU
CPU Speed                              2.40GHz
Installed memory                       16384MB
UUID                                   9257d801-53a9-11cb-8af8-8bc3037cdd2f
MAC Address (Internal LAN)             28 D2 44 50 9A 44
UEFI Secure Boot                       Off
```

NOTE: Notice the ‘UEFI secure Boot OFF’ though Secure Boot is enabled in Startup Tab.

Remember these all tabs and their settings depend on the hardware vendor so please refer to the firmware documentation for these steps.

After booting with RHEL-7 beta you will find the keys are flushed:



```
[root@localhost Desktop]# keyctl list %:.system_keyring
1 key in keyring:
519696373: --alswrv     0     0 asymmetric: Red Hat Enterprise Linux kernel signing key: edb0bcea9cf6f958d5f06d74c45c4694d296d87d
[root@localhost Desktop]#
```

SIGNING KERNEL MODULES FOR RHEL 7 UEFI SECURE BOOT

If you wish to load your own kernel module on a system where UEFI Secure Boot is enabled, then you will need to sign your kernel module and make sure that the corresponding public key is enrolled on that system.

There are three steps to this process:

1. Generate a private/public key pair.

2. Enroll your public key on each target system where you want to load your kernel module.

3. Build your kernel module the usual way, then sign your kernel module using the private key that you generated.

Tools Required:

Tool	**Provided by Package**	**Where Used**	**Purpose**
Openssl	openssl	Build system	Generates a public/private key pair.
Sign-file	kernel-devel	Build system	Perl script used to sign kernel modules.
Perl	perl	Build system	Perl interpreter used to run the signing script.
Mokutil	mokutil	Target system	Tool used to manually enroll a public key.
Keyctl	keyutils	Target system	Tools used to display public keys in system key ring.

The build system, where you build and sign your kernel module, does not need to have UEFI Secure Boot enabled and does not even need to be a UEFI-based system.

> **NOTE**: At the time of boot kernel loads the keys from UEFI firmware to system key ring which we can see by using the keyctl tool. For example:
>
> # **keyctl list %:.system_keyring**

In RHEL 7, when a kernel module is loaded the module's signature is checked using the public x.509 keys on the kernel's system key ring, excluding those keys that are on the kernel's system black list key ring.

Sources of System key Ring:

Source of x.509 keys	User Ability to add Keys	UEFI secure Boot state	Keys loaded during Boot
Embedded in kernel	No	-NA-	.system_keyring
UEFI Secure Boot “db”	Limited	Enabled	.system_keyring
UEFI Secure Boot “dbx”	Limited	Enabled	.system_blacklist_keyring
Embedded in shim.efi boot loader	No	Enabled	.system_keyring
Machine Owner Key (MOK) List	Yes	Enabled	.system_keyring

Note that if the system is not UEFI-based or if UEFI Secure Boot is not enabled, then only the keys that are embedded in the kernel are loaded onto the system key ring and you have no ability to augment that set of keys without rebuilding the kernel. The system black list key ring is a list of X.509 keys which have been revoked. If your module is signed by a key on the black list then it will fail authentication even if your public key is in the system key ring.

You can display information about the keys on the system key rings using the keyctl utility. The following is abbreviated example output from a RHEL 7 system where UEFI Secure Boot is not enabled.

```
# keyctl list %:.system_keyring
3 keys in keyring:
...asymmetric: Red Hat Enterprise Linux Driver Update Program (key 3): bf57f3e87...
...asymmetric: Red Hat Enterprise Linux kernel signing key: 4249689eefc77e95880b...
...asymmetric: Red Hat Enterprise Linux kpatch signing key: 4d38fd864ebe18c5f0b7...
```

The following is abbreviated example output from a RHEL 7 system where UEFI Secure Boot is enabled:

```
# keyctl list %:.system_keyring
6 keys in keyring:
...asymmetric: Red Hat Enterprise Linux Driver Update Program (key 3): bf57f3e87...
...asymmetric: Red Hat Secure Boot (CA key 1): 4016841644ce3a810408050766e8f8a29...
...asymmetric: Microsoft Corporation UEFI CA 2011: 13adbf4309bd82709c8cd54f316ed...
...asymmetric: Microsoft Windows Production PCA 2011: a92902398e16c49778cd90f99e...
...asymmetric: Red Hat Enterprise Linux kernel signing key: 4249689eefc77e95880b...
...asymmetric: Red Hat Enterprise Linux kpatch signing key: 4d38fd864ebe18c5f0b7...
```

The above output shows the addition of two keys from the UEFI Secure Boot “db” keys plus the “Red Hat Secure Boot (CA key 1)” which is embedded in the shim.efi boot loader. You can also look for the kernel console messages that identify the keys with an UEFI Secure Boot related source (i.e. UEFI Secure Boot db, embedded shim, and MOK list).

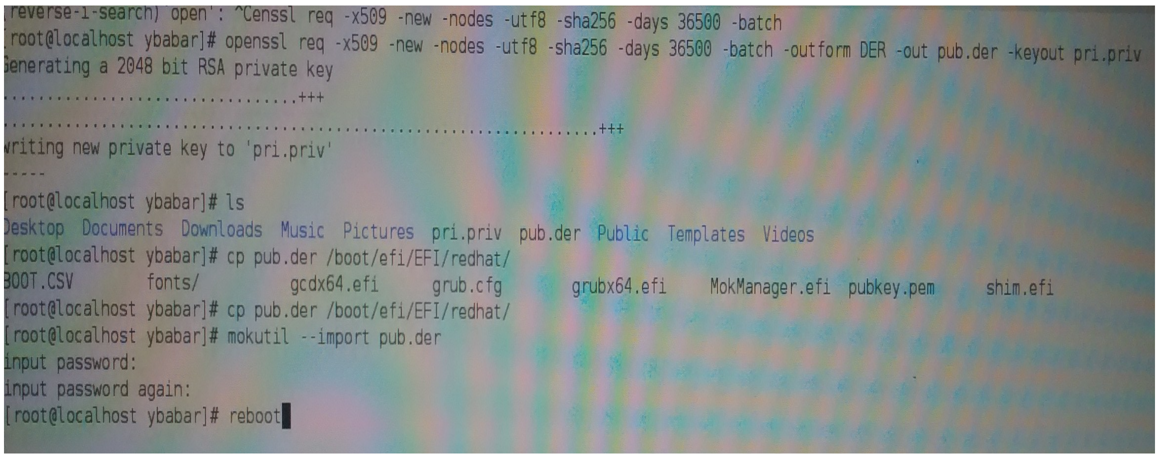
```
# dmesg | grep 'EFI: Loaded cert'
[5.160660] EFI: Loaded cert 'Microsoft Windows Production PCA 2011: a9290239...
[5.160674] EFI: Loaded cert 'Microsoft Corporation UEFI CA 2011: 13adbf4309b...
[5.165794] EFI: Loaded cert 'Red Hat Secure Boot (CA key 1): 4016841644ce3a8...
```

MANUALLY ADDING A PUBLIC KEY TO THE MOK (THE MACHINE OWNER KEY)

The Machine Owner Key (MOK) facility is a feature that is supported by RHEL 7 and can be used to augment the UEFI Secure Boot database. When RHEL 7 boots on a UEFI-enabled system with Secure Boot enabled, the keys on the MOK list are also added to the system key ring in addition to the database keys.

Here are the steps that can be used to add your public key to the MOK list:

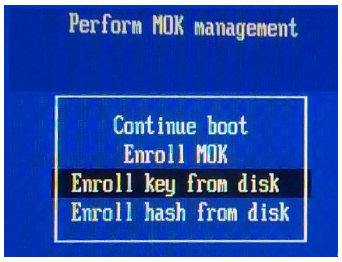
1. Create a Public/Private key pair and copy your public key to the /boot/efi/EFI/redhat directory. Then enroll that key to the UEFI firmware with the help of the mokutil utility:

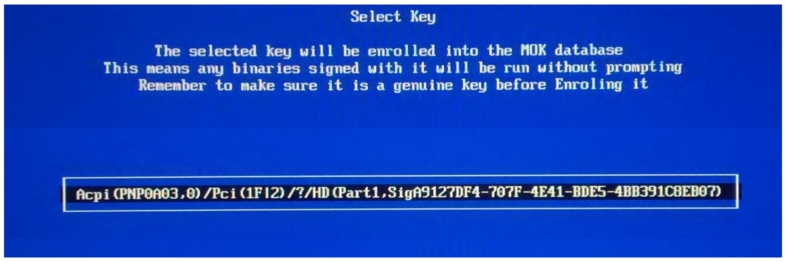


```
reverse-i-search)`open': ^Censsl req -x509 -new -nodes -utf8 -sha256 -days 36500 -batch
root@localhost ybabar]# openssl req -x509 -new -nodes -utf8 -sha256 -days 36500 -batch -outform DER -out pub.der -keyout pri.priv
enerating a 2048 bit RSA private key
...............................+++
..............................................................+++
riting new private key to 'pri.priv'
-----
[root@localhost ybabar]# ls
esktop  Documents  Downloads  Music  Pictures  pri.priv  pub.der  Public  Templates  Videos
[root@localhost ybabar]# cp pub.der /boot/efi/EFI/redhat/
OOT.CSV        fonts/        gcdx64.efi     grub.cfg       grubx64.efi    MokManager.efi  pubkey.pem     shim.efi
[root@localhost ybabar]# cp pub.der /boot/efi/EFI/redhat/
[root@localhost ybabar]# mokutil --import pub.der
input password:
input password again:
[root@localhost ybabar]# reboot
```

2. Reboot the system.

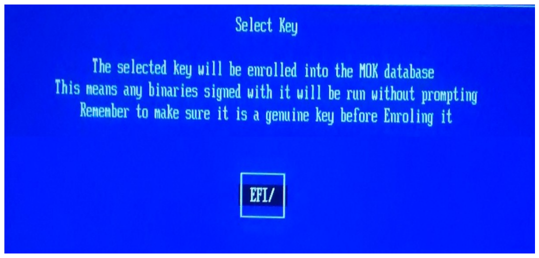
3. The pending MOK key enrollment request will be noticed by shim.efi and will launch MokManager.efi to let you complete the enrollment from the UEFI console. You need to enter the password you associated with this request earlier and confirm the enrollment. Your public key is added to the MOK list, which is persistent. The following screen shots illustrate that process:



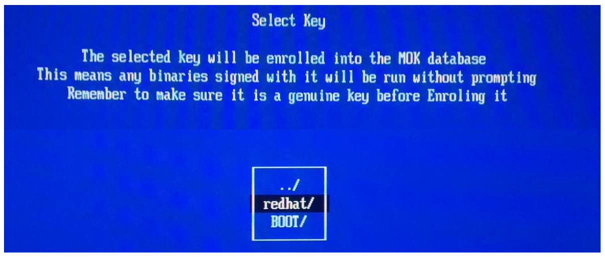

Select Key

The selected key will be enrolled into the MOK database
This means any binaries signed with it will be run without prompting
Remember to make sure it is a genuine key before Enroling it

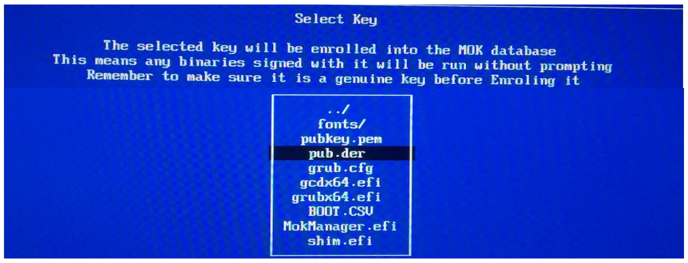
```
Acpi(PNP0A03,0)/Pci(1F|2)/?/HD(Part1,SigA9127DF4-707F-4E41-BDE5-4BB391C8EB07)
```


Select Key

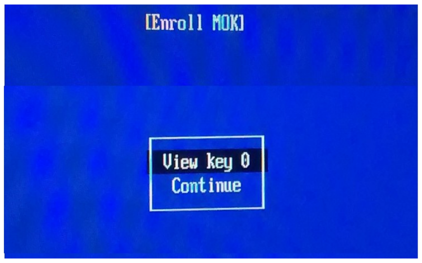
The selected key will be enrolled into the MOK database
This means any binaries signed with it will be run without prompting
Remember to make sure it is a genuine key before Enroling it

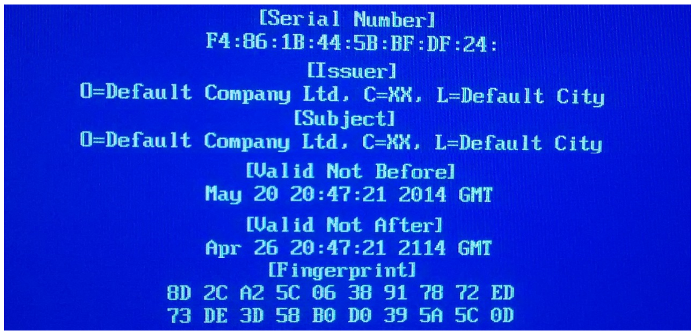

Select Key

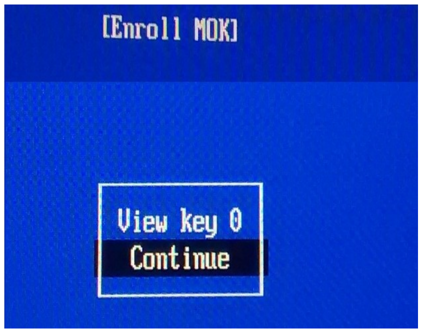
The selected key will be enrolled into the MOK database
This means any binaries signed with it will be run without prompting
Remember to make sure it is a genuine key before Enroling it

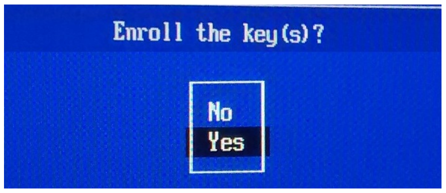

Select Key

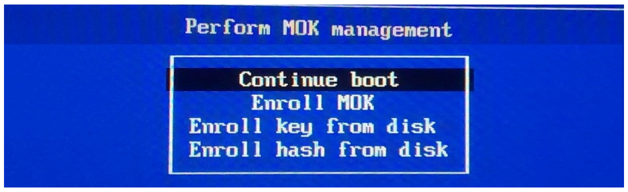
The selected key will be enrolled into the MOK database
This means any binaries signed with it will be run without prompting
Remember to make sure it is a genuine key before Enroling it




```
[Serial Number]
F4:86:1B:44:5B:BF:DF:24:
[Issuer]
O=Default Company Ltd, C=XX, L=Default City
[Subject]
O=Default Company Ltd, C=XX, L=Default City
[Valid Not Before]
May 20 20:47:21 2014 GMT
[Valid Not After]
Apr 26 20:47:21 2114 GMT
[Fingerprint]
8D 2C A2 5C 06 38 91 78 72 ED
73 DE 3D 58 B0 D0 39 5A 5C 0D
```


[Enroll MOK]
View key 0
[✓ Continue]


Enroll the key(s)?
No
[✓ Yes]


Perform MOK management
[✓ Continue boot]
Enroll MOK
Enroll key from disk
Enroll hash from disk