Secure Boot for Linux on HPE Servers

Enhanced security for your Linux environment

Contents

Technical white paper

Secure Boot for high performance computing software, as defined in the UEFI specification, provides an industry standard defense against potential malware attacks. Without Secure Boot, malware can attack systems during pre-boot by targeting the system-embedded firmware during the interval between BIOS initiation and operating system load. Malware inserted at this point compromises the security of the operating system, no matter how secure. Secure Boot protects the system by preventing the insertion of malware during the pre-boot process.

This technical white paper introduces Secure Boot technology and explains what it is, how it works and how to use it on UEFI based HPE servers running Linux®.

What is Secure Boot?

Secure Boot, a high performance computing software solution, is a method to restrict which binaries can be executed to boot the system. With Secure Boot, the system BIOS will only allow the execution of boot loaders that carry the cryptographic signature of trusted entities. In other words, anything run in the BIOS must be "signed" with a key that the system knows is trustworthy. With each reboot of the server, every executed component is verified. This prevents malware from hiding embedded code in the boot chain.

Secure Boot is:

- Intended to prevent boot-sector malware or kernel code injection.
- Hardware-based code signing.
- Extension of the UEFI BIOS architecture.
- Optional with the ability to enable or disable it through the BIOS.

For a more detailed description of what Secure Boot is and how it works, see the Resources section.

Chain of trust

SLES11 SP3, RHEL 7.0 and greater distributions support a chain of trust which goes down to the kernel module level. Loadable kernel modules must be signed with a trusted key or they cannot be loaded into the kernel.

The following trusted keys are stored in UEFI NVRAM variables:

- **Database (DB)**—Signature database that contains well know keys. Only binaries that can be verified against the DB are executed by the BIOS.
- **Forbidden Signature Database (DBX)**—Keys that are blacklisted. Attempting to load an object with a key that matches an entry in the DBX will be denied.
- **Machine Owner Key (MOK)**—User added keys for kernel modules they want to install.
- **Platform Key (PK)**—The key installed by the hardware vendor.
- **Key Exchange Key (KEK)**—The key required to update the signature database.

The user must have physical access to the system console to add/modify keys or enable/disable Secure Boot through the UEFI configuration menu.

The default boot loader on most UEFI enabled servers running Linux is **grub2** or **elilo**. With Secure Boot enabled, an additional “shim” boot loader is needed. When booting in Secure Boot mode, the **shim** loader is called first since it contains a trusted signature. The **shim** will then load **grub2** or **elilo** which loads the OS kernel which is also signed.

HPE Server support for Secure Boot

HPE Gen9 servers and greater with UEFI enabled will support Secure Boot. To determine whether Secure Boot is supported on a specific platform and enabled or disabled by default, please check the system specifications.

Some Linux distributions extend the chain of trust to the kernel module. Consult the distribution’s documentation for what level of Secure Boot support is provided.

Limitations of Secure Boot

With Secure Boot enabled, some actions on a Linux system are either limited or restricted.

- Hybrid ISO images are not recognized as bootable on UEFI systems. Thus, UEFI booting from USB devices is not supported.
- Boot loader, kernel, and kernel modules must be signed.
- kexec and kdump are disabled.
- Hibernation (suspend on disk) is disabled.
- Write access to /dev/kmem and /dev/mem is not possible, not even as root user.
- Access to I/O port is not possible, even as root user. All X11 graphical drivers must use a kernel driver.
- PCI BAR access through sysfs is not possible.
- custom_method in ACPI is not available.
- acpi_rsdp parameter does not have any effect on the kernel.

Secure Boot on HPE Servers

To determine if Secure Boot is enabled, run the following command in Linux:

```
$ od -An -t u1 /sys/firmware/efi/vars/SecureBoot-8be4df61-93ca-11d2-aa0d-00e098032b8c/data
```

The system will return a value of either 0 or 1:

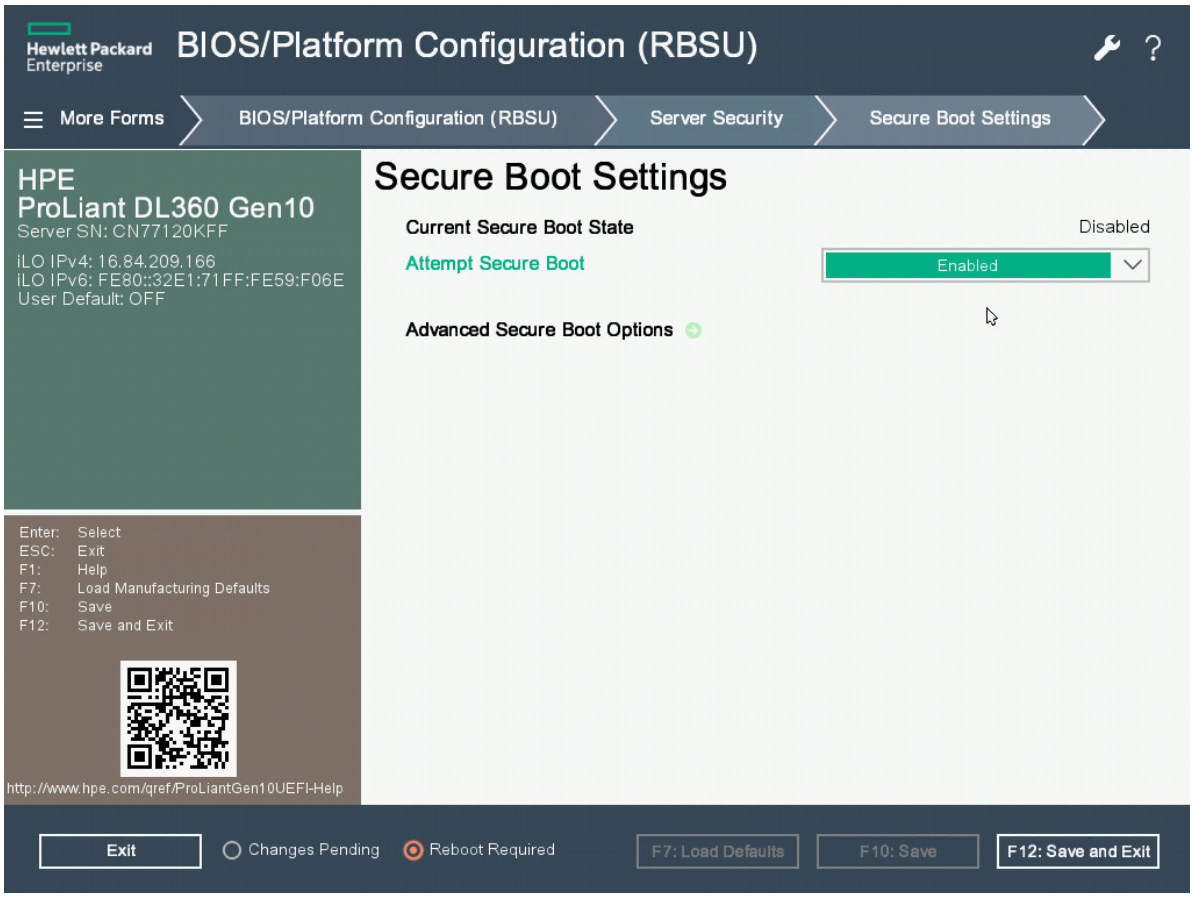
- 0 = secure boot is NOT enabled
- 1 = secure boot is enabled

Enabling/Disabling Secure Boot

To enable or disable Secure Boot do the following:

1. During system boot, press F9 to run the System Utilities.
2. Select “System Configuration”.
3. Select “BIOS/Platform Configuration (RBSU)”.
4. Select “Server Security”.
5. Select “Secure Boot Configuration”.
6. Select “Enable Secure Boot” to toggle the state on or off (see Figure 1).
7. Exit the all RBSU menu screens using the Esc key.

For more information, see the HPE UEFI System Utilities User Guide.


Hewlett Packard Enterprise
BIOS/Platform Configuration (RBSU)
More Forms
BIOS/Platform Configuration (RBSU)
Server Security
Secure Boot Settings

Secure Boot Settings

Current Secure Boot State Disabled
Attempt Secure Boot Enabled
Advanced Secure Boot Options

HPE
ProLiant DL360 Gen10
Server SN: CN77120KFF
iLO IPv4: 16.84.209.166
iLO IPv6: FE80::32E1:71FF:FE59:F06E
User Default: OFF

Enter:	Select
ESC:	Exit
F1:	Help
F7:	Load Manufacturing Defaults
F10:	Save
F12:	Save and Exit

http://www.hpe.com/qref/ProLiantGen10UEFI-Help

Figure 1. Enabling Secure Boot through the HPE Server BIOS

Note for SLES installations
If you want to use Secure Boot with SLES, it is advisable to make sure that Secure Boot is enabled prior to installing the operating system. If it is not enabled prior to the installation of SLES, the system may not include all the components needed for Secure Boot. RHEL installations do not have these limitations.

Signing a kernel module and loading the associated key in the MOK

A Machine Owner Key (MOK) is a type of user generated key that is used to "sign," or authenticate as trustworthy, an Extensible Firmware Interface (EFI) binary. MOK gives you ownership of the boot process by allowing you to run locally-compiled kernels or boot loaders not delivered with the Linux distribution. This means that you can install custom kernel or kernel modules that are compatible with UEFI Secure Boot. For more information, see the Administration Guide for your Linux distribution.

To authenticate a kernel module and load the associated key in the MOK, complete the following steps:

1. Generate a certificate/key pair with the following command:

```
# openssl req -new -x509 -newkey rsa:2048 -sha256 -keyout key.asc -out cert.der -outform
der -nodes -days 4745 -subj “yourname”
```

2. If necessary, sign the kernel module with a private key:

```
# /usr/src/linux/scripts/sign-file sha256 key.asc cert.der e1000e.ko
```

3. Move cert.der to the EFI partition

4. To load the public certificate into the MOK, reboot the system, go to the EFI Shell and run the following command:

```
:boot/efi/efi/hp # mokutil -import cert.der
```

Note

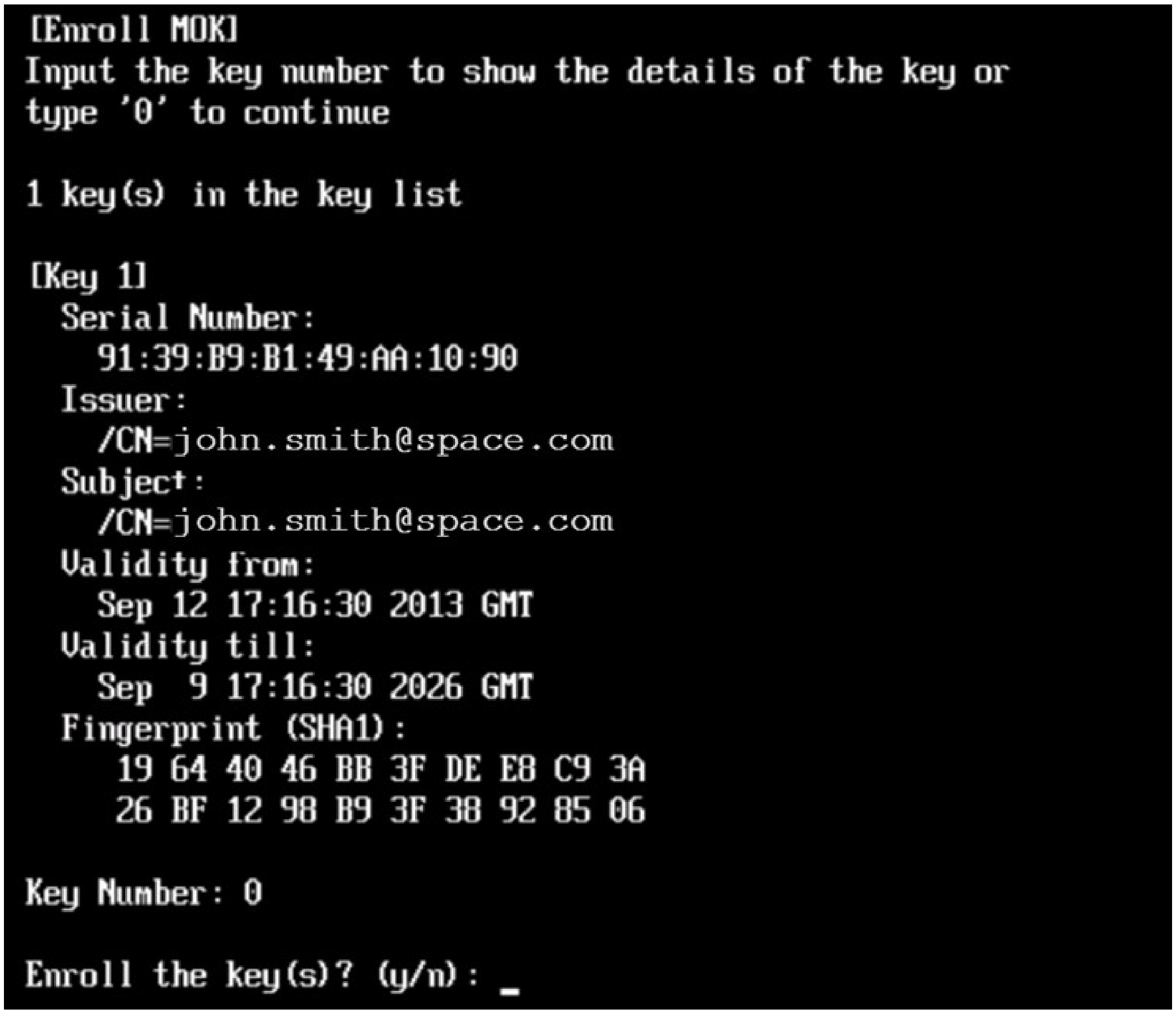
Some certificates require a password to allow them to be loaded, if so:

Input password:

Input password again:

---

5. Select “y” and the process is complete with the certificate loaded.



```
[Enroll MOK]
Input the key number to show the details of the key or
type '0' to continue

1 key(s) in the key list

[Key 1]
  Serial Number:
    91:39:B9:B1:49:AA:10:90
  Issuer:
    /CN=john.smith@space.com
  Subject:
    /CN=john.smith@space.com
  Validity from:
    Sep 12 17:16:30 2013 GMT
  Validity till:
    Sep  9 17:16:30 2026 GMT
  Fingerprint (SHA1):
     19 64 40 46 BB 3F DE E8 C9 3A
     26 BF 12 98 B9 3F 38 92 85 06

Key Number: 0

Enroll the key(s)? (y/n): _
```

Figure 2. Loading a public certificate into the MOK

Building and Booting a Custom Kernel

To boot a custom compiled kernel with Secure Boot enabled, it must be signed with a certificate known to the BIOS. The following SLES11SP3 example illustrates how this is done.

1. Enable Secure Boot on the system (see earlier section).
2. Go to the kernel source directory:
```
$ cd /usr/src/linux
```
3. Get a copy of the current configuration file:
```
$ RELEASE=`uname -r`
$ cp /boot/config-${RELEASE} .config
```
4. Add automatic module signing to the configuration file:

   Reference: wiki.gentoo.org/wiki/Signed_kernel_module_support

   Either edit the .config file manually and add the following: CONFIG_MODULE_SIG_ALL=y

   Or use "make menuconfig" and check the "Automatically sign all modules":
```
 --- Enable loadable module support
[*]  Forced module loading
[*]  Module unloading
[*]  Forced module unloading
[*]  Module versioning support
[*]  Source checksum for all modules
[*]  Module signature support
[ ]  Check module signatures by default
[ ]  Require modules to be validly signed
[*]  Automatically sign all modules
[*]  Support for blacklisting module signature certificates
[*]  Allow modules signed with certs stored in UEFI
```
5. Create your own personal key:

   Reference: en.opensuse.org/openSUSE:UEFI see "Booting a Custom Kernel"
```
$ cd /usr/src/linux
```

Note
You only need to create and register a key once. Multiple kernels can be signed with the same key.

6. Create a custom X.509 key and certificate used for signing:
```
$ openssl req -new -x509 -newkey rsa:2048 -sha256 -keyout key.asc -out cert.der -outform der -nodes
-days 4745 -subj "/CN=$USER/"
```

7. Package the key and certificate as PKCS#12 structure:

```
$ openssl pkcs12 -export -inkey key.asc -in cert.pem -name kernel_cert -out cert.p12
```

8. Generate the NSS database for use by **pesign**:

```
$ certutil -d . -N
```

9. Import the key and certificate contained in PKCS#12 into the NSS database:

```
$ pk12util -d . -i cert.p12
```

10. To allow unsupported modules to load, edit the /etc/modprobe.d/unsupported-modules file and add "allow_unsupported_modules 1":

```
$ vi /etc/modprobe.d/unsupported-modules
```

11. It is advisable to modify the "Makefile" and add something to "EXTRAVERSION =" so your new kernel is installed beside the existing kernel instead of overwriting it.

```
$ vi Makefile
```

12. Build the kernel/modules and install them:

```
$ make
$ make modules_install
$ make install
```

13. Get the version of your new kernel:

```
$ RELEASE=`cd /usr/src/linux ; make kernelrelease` ; echo $RELEASE
```

14. Create initramfs for the new kernel:

```
$ /sbin/mkinitrd -k /boot/vmlinuz-${RELEASE} -i initrd-${RELEASE}
```

15. Setup bootloader configuration file:

```
$ /sbin/update-bootloader --name ${RELEASE} --image /boot/vmlinuz-${RELEASE} --initrd
/boot/initrd-${RELEASE} --add --force
```

16. Manually sign the kernel:

```
$ pesign -n . -c kernel_cert -i arch/x86/boot/bzImage -o vmlinuz.signed -s
$ mv vmlinuz.signed /boot/vmlinuz-${RELEASE}
```

17. List the signatures for the kernel image:

```
$ pesign -n . -S -i /boot/vmlinuz-${RELEASE}
```

18. Convert the certificate to DER format for import into the UEFI BIOS or MOK:

```
$ openssl x509 -in cert.pem -outform der -out cert.der
```

19. Copy the certificate to the EFI partition so you can import it:

```
$ cp cert.der /boot/efi/cert.der
```

Note
The SLES "mokutil" utility can also be used to queue up the inclusion of a new key. It will cause "MokManager.efi" to be run automatically.

20. Reboot the system to the UEFI shell.

21. Add the new key to MOK database:
    a. Run the “MokManager.efi” utility;
    b. Scroll down to “Enroll key from disk” and hit RETURN;
    c. Navigate to the “cert.der” file you copied to disk earlier, select it and hit RETURN;
    d. Follow the directions to enroll the key; and
    e. Choose “Continue boot” to exit.
22. Run “grub.efi” and boot your new kernel as normal.

Resources

- Using MOK and UEFI Secure Boot with SUSE Linux
- UEFI home page where you can find the current UEFI specifications
- Blog posts by Olaf Kirch and Vojtěch Pavlík (the chapter above is heavily based on these posts):
  - suse.com/blogs/uefi-secure-boot-plan/
  - suse.com/blogs/uefi-secure-boot-overview/
  - suse.com/blogs/uefi-secure-boot-details/
- UEFI with openSUSE
- SUSE Linux Enterprise Server 11 SP4 Administrator Guide
- UEFI Secure boot with Red Hat®
- Fedora UEFI Secure Boot Guide

Make the right purchase decision. Click here to chat with our presales specialists.

Sign up for updates

4AA5-4496ENW, September 2017, Rev. 1