Intel® Trusted Execution Technology
Best Practices for Linux on PRIMERGY

This white paper provides a comprehensible step-by-step tutorial how to set up Intel® TXT with TPM 1.2 on Fujitsu PRIMERGY Servers under Linux to maximum effect with moderate amount of administrative effort, explains concepts of TXT, and compares TXT to other similar technologies.

Intel® Trusted Execution Technology

Intel® Trusted Execution Technology (TXT) is a hardware-based technology to reduce the attack surface of servers [1]. Properly configured, TXT can verify that the kernel image (and, in the case of Xen, hypervisor image), its command line, and the initial RAM disk image of your Linux server haven't been tampered with by an attacker. Sensitive data can be "sealed" and encrypted such that they only become accessible if the validation has succeeded.
TXT works by setting up a "Chain of Trust", where every piece of software performs a "measurement" of the next piece of code to be executed by the CPU. The measurement is done by calculating a cryptographic hash of the code and feeding it into PCR registers of the TPM ("extending" the PCR registers with it). After boot, the PCR register carry a fingerprint of the data extended to them that can be used to verify system integrity at every step in the Chain of Trust. The "Root of Trust", i.e. the beginning of the chain, is the so-called ACM (Authenticated Code Module), a piece of software with a signature from Intel. In theory, the ACM can be executed at any time; in practice, it is started shortly before booting the hypervisor or kernel. This is called a "Dynamic Root of Trust" model because pieces of software that executed before the ACM don't have to be measured; the ACM takes care that code executed afterwards is protected from them by hardware. See [1] and [2] for more detailed explanations of the internals of TXT.

Purpose of this White Paper

While TXT is a powerful technology, its deployment and configuration under Linux remains difficult. The available documentation is split over various independent documents, not well-organized, and aimed at a very highly skilled technical audience. This paper aims to provide a tutorial which can be understood by Linux administrators on an intermediate skill level. It provides a walkthrough towards a TXT configuration that offers a maximum of trust benefit at a reasonable administrative cost, and tries to explain the concepts both behind the technology itself and the "Best Practices" recommendations.

Hardware Requirements on Fujitsu PRIMERGY servers

Fujitsu offers TXT support in the PRIMERGY servers of the "M2" generation with Intel® "Broadwell" CPUs: RX2540 M2, RX2530 M2, etc. Consult your server's data sheet to find out if your particular hardware supports TXT. For TXT with Linux, you need to have a system with a Trusted Platform Module (TPM) v1.2 installed [3]. As of early 2016, no enterprise Linux distribution supports the current TPM 2.0 hardware yet. Consequently, TXT with TPM 2.0 isn't supported, either. **This white paper covers TXT with TPM 1.2 only.**

Enabling TXT in the BIOS SETUP

To activate TXT on a Fujitsu PRIMERGY Server that supports it, proceed as follows:
- Under Advanced → "Trusted Computing", make sure the TPM Status is "Enabled" and "Activated".
- The TPM should be "UnOwned". If it is "Owned", you (any only you) should possess the TPM Owner password. To reset TPM ownership, clear the TPM by setting "Pending TPM Operation" to "TPM Clear", saving BIOS settings, and rebooting. This will delete all keys and other secrets stored in the TPM.
- If the "TPM State" is set to "Disabled", set it to "Enabled", save BIOS settings, and reboot.
- Under Advanced → "CPU Configuration", set "Intel TXT support" to "Enabled". Save BIOS settings and reboot.

Choosing and Installing a Linux Operating System

Choose a Linux distribution that supports TXT. Please check the release documents for your PRIMERGY server for supported OS versions on your hardware. Out of the Enterprise Linux distributions that Fujitsu supports:
- Red Hat Enterprise Linux (RHEL) supports TXT in legacy mode since RHEL 6.2 and TXT in UEFI mode since 7.2.
- SUSE Linux Enterprise Server (SLES) supports TXT in legacy mode since SLES 11 SP2. TXT in UEFI mode will be supported in SLES 12 SP2.

Linux Software Packages for TPM and TXT

The following software packages are necessary to work with TXT under Linux. They are available on all major Linux distributions.

- trousers: An Open Source implementation of the TSS (Trusted Software Stack) v1.2. It contains the `tcsd` daemon which controls user-space access to TPM functionality under Linux.
- tboot:: Trusted Boot [4], an open-source MLE (Measured Launch Environment). It comes with a set of tools for creating TXT policies (`lcp_crtXXX`, `tb_polgen`) that will be used in this tutorial.
- tpm-tools: A set of basic tools for working with TPM hardware, such as `tpm_takeownership` and `tpm_sealdata`.

Under RHEL 7.2 with UEFI, install the **grub2-efi-modules** package as well. It is available from the "RHEL Server Optional" channel on Red Hat Network.

Basic Operating System Setup for TXT

Install the Operating System as usual. After the installation has finished, install possibly missing packages (see above) and make sure the service for the TSS daemon `tcsd` is running:

```
### RHEL7, SLES12 (systemd)
systemctl enable tcsd
systemctl start tcsd

### RHEL 6, SLES 11 (SysV init)
chkconfig --level 2345 tcsd on
/etc/init.d/tcsd start
```

Add a tboot entry to the boot loader configuration. For systems using GRUB2 (RHEL 7, SLES12), this is most easily achieved by running the `grub2-mkconfig` command after installing the tboot package. It will create tboot boot entries for Linux and Xen.

```
### RHEL 7 and SLES 12 (legacy)
grub2-mkconfig -o /boot/grub2/grub.cfg
### RHEL 7 (UEFI)
grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
cp -a /usr/lib/grub/x86_64-efi /boot/efi/EFI/redhat
```

On systems Using GRUB1 (RHEL 6, SLES 11), a tboot section must be created manually in the configuration file `/boot/grub/menu.lst`.
Hint: Simply copy the existing normal boot entries, changing "`kernel`" to "`module`", and insert a "`kernel tboot.gz ...`" line.

```
title Linux with tboot
       root (hd0,0)
       kernel /boot/tboot.gz logging=serial,vga,memory pcr_map=da
       module /boot/vmlinuz-... [parameters...]
       module /boot/initrd-...

title Xen with tboot
       root (hd0,0)
       kernel /boot/tboot.gz logging=serial,vga,memory pcr_map=da
       module /boot/xen.gz iommu=required [...]
       module /boot/vmlinuz-... [parameters...]
       module /boot/initrd-...
```

If the `/boot` file system is on a separate partition on your system, remove the leading "`/boot`" in the paths of the kernel and modules.
Caution for SLES12: tboot 1.8.3 has a bug that causes a TXT Reset if memory logging is used. Therefore you must use `logging=serial` on the tboot command line. This has the side effect that it won't be possible to examine the memory logs with the `txt-stat` tool after boot.

Booting into "Basic" TXT Environment

If you reboot now, the boot loader should offer you a "tboot" boot option. Select it and boot the system. The system is now running in basic TXT mode. You can verify this by checking the PCR registers of the TPM:

```
### RHEL 7
cat /sys/class/tpm/tpm0/device/pcrs

### RHEL 6, SLES 11, SLES 12
cat /sys/class/misc/tpm0/device/pcrs
```

If you see anything other than "`FF FF FF … FF`" in the output lines for PCR 17 and 18, TXT has been successfully activated. You can use the command `txt-stat` to examine the logs of tboot in detail.

If TXT Boot Fails

Booting into TXT can fail for various reasons. If an error occurs in the Authenticated Code Module (ACM), the system will simply reset spontaneously; this is called a "TXT Reset". If errors occur later in the boot sequence, depending on configuration, the server may execute a "TXT

Shutdown", or may boot up "successfully" without activating TXT. For debugging TXT issues, it is recommended to watch the tboot output on the serial console (available e.g. from a ssh connection to the server's iRMC).
If a TXT Reset happens in the basic configuration, most probably something is wrong with your TPM setup. Check if your TPM is correctly "provisioned",i.e. the TPM NV has been prepared for TXT. The command `tpmnv_getcap` should print both 0x50000001 and 0x50000003 in the last line of output. The `txt-stat` command output should contain the line "`TBOOT: TPM nv_locked: TRUE`". If it doesn't, please report back to Fujitsu.
It is normal that system appears "frozen" for about a minute after booting into tboot. The ACM needs to perform various measurements of code and modules using the TPM. This takes a long time, because the TPM is a rather slow device.

"Basic" TXT: What Benefit Does It Provide?

If you have come to this point, you have successfully booted into a Trusted Execution Environment. However, it isn't advisable to stop here, because you haven't achieved much for practical security yet. Only the default "Platform Supplier" (PS) policy by Fujitsu is active. Because Fujitsu has no knowledge what environment you are going to run on your system, it can only enforce the weakest policy, so-called "ANY" policy. The ACM has booted and started the MLE (tboot). The values of TPM PCRs 17-19 represent measurements made by the ACM of the MLE and other components [2]. **These PCR values have not been compared with "known good" values, and they are not stable across OS updates.**

The Remedy: TXT Best Practices

Fortunately, it is possible to change the configuration such that the problems mentioned in the previous paragraph are overcome. The Best Practices described here will make sure that kernel and initial RAM disk are validated against previously "known good" hashes, and provide a method that allows encrypting data against PCR values which is stable across OS updates. In the following, we shall offer a step-by-step guide on how to put these recommendations in place.

TXT Best Practices

1. Create a Verified Launch Policy (VLP) for tboot of type "halt" that validates hypervisor, kernel, and the initial RAM disk against "known good" values.
2. Create a Platform Owner Launch Control Policy (LCP) as a signed list policy containing at least an MLE element and a CUSTOM element with the VLP.
3. Operate tboot with the "Details / Authorities" PCR mapping.
4. Seal data against PCR 18, the "Authorities" PCR.
5. Optionally, put further, kernel-level trust measurements in place.

Preparing a Signing Authority

This should be done in a safe environment, not on the server to be protected. It can be done by any tool that supports generation of RSA keys and creating signed SHA-1 digests with it. The examples here use the **openssl** utility, which is available on all major operating systems.

```
### Create 2048bit RSA key
openssl genrsa -des3 -out txt-priv.pem 2048

### Export the RSA public key
openssl rsa -in txt-priv.pem -out txt-pub.pem -pubout
```

Of course, the private key `txt-priv.pem` should be protected as well as your organization is able to.

Preparing the TPM for Platform Owner Policy

Unless you've already done so, take ownership of the TPM now. It is recommended to set the password of the Storage Root Key (SRK) to the "well-known secret" but use a safe TPM Owner password.

```
tpm_takeownership -z
```

Create the NVM space for tboot error code (0x20000002) and for the PO policy (0x40000001). The VLP policy index 0x20000001 is not necessary in our recommended setup, because we will use a LCP CUSTOM element instead to pass our VLP policy to the MLE (see below).

```
### LCP "platform owner" (PO) policy index, using OWNERWRITE permissions (-pv 2).
 tpmnv_defindex -i 0x40000001 -s 54 -pv 2 -p $OWNER_PASSWORD

### tboot error index, with restricted read/write localities
 tpmnv_defindex -i 0x20000002 -s 8 -pv 0 -rl 0x07 -wl 0x07 -p $OWNER_PASSWORD
```

Double-Checking Command Lines in the Boot Loader Configuration

tboot's Verified Launch Policy (VLP) rules check not only the image to be started, but also the command line arguments passed to it by the boot loader. Single-character, even whitespace, differences may cause the test (and thus, the TXT boot), to fail. Therefore, the command lines need to be crafted meticulously before putting them into a policy. The goal is to be able to copy/paste the command lines from the boot loader configuration file to the commands for generating the policy. Typing errors in the kernel command line are a common cause of TXT Boot failures. Here are a few hints to avoid common pitfalls:

- Make sure the parameter lists for hypervisor and kernel are correct and complete (Xen needs `iommu=required`, for example).
- Whitespace matters. When parameters are separated by whitespace, make sure this whitespace consists of a single space character only. Look for the "`ro`" kernel parameter: it is often followed by two space characters. (The reason for this rule is that GRUB2 collapses series of whitespace characters to a single space during boot – if you copy a command line with series of spaces to `tb_polgen`, booting will fail).
- `module` or `module2` lines generated by `grub2_mkconfig` for tboot+Xen contain the string "`placeholder`". Delete it in `grub.cfg`. It should only be used for Xen configurations without tboot. (The reason for this rule is that tboot always drops the first command line argument it is passed, assuming that it is the image name. GRUB1 passed the image name as first parameter, but GRUB2 does not. Therefore one (but only one!) "dummy" argument has to be passed to tboot as first argument in GRUB2 setups. This could be "`placeholder`" or the image name again, but not both).

Warning: On GRUB2 systems, the `grub2_mkconfig` command creates a new configuration file with new tboot entries at every invocation, destroying all fixes just made. To avoid this, copy the configuration entries for tboot you just edited into the file "`/etc/grub.d/40_custom`".

Creating a Verified Launch Policy

Verified Launch Policy (VLP) is not part of the TXT technology itself. Rather, it is a specific feature of the tboot MLE. However, it is indispensable to carry the chain of trust forward from the MLE to the kernel and other parts of the OS. Although it comes last in the boot sequence, we start with creating the VLP, because it has to be included in the LCP later.

Different types of VLP exist, with different behaviour in case of failure. "nonfatal" policy always boots the system and just extends PCRs, whereas "halt" policy executes a "TXT Shutdown" if the data seen doesn't match "known good" hash values. "halt" policy offers better protection and is recommended as Best Practice. The following command creates an empty "halt" policy:

```
tb_polgen --create --type halt vlp.pol
```

The VLP must contain an entry for every tboot "module" (not to be confused with kernel modules) in the boot loader configuration: hypervisor (if present), kernel, and initial RAM disk. VLP works by calculating hashes of the module images and their command line in the boot loader configuration. Only modules with matching hashes will be loaded by the MLE at boot time. The following example adds two kernel images, one for a RHEL 7.2 GA kernel and one for an errata kernel, with different command lines. Add more lines for each kernel/command line combination you wish to allow.

```
tb_polgen --add vlp.pol --num 0 --pcr 19 --hash image --image /boot/vmlinuz-3.10.0-327.el7.x86_64 \
   --cmdline 'root=UUID=... ro crashkernel=auto rhgb quiet noefi module.sig_enforce=1'

tb_polgen --add vlp.pol --num 0 --pcr 19 --hash image --image /boot/vmlinuz-3.10.0-327.4.5.el7.x86_64 \
   --cmdline 'root=UUID=... ro crashkernel=auto debug noefi module.sig_enforce=1'
```

For each allowed kernel image, a matching initial RAM disk must be added to the policy as well:

```
tb_polgen --add vlp.pol --num 1 --pcr 20 --hash image \
   --image /boot/initramfs-3.10.0-327.el7.x86_64.img --cmdline ''

tb_polgen --add vlp.pol --num 1 --pcr 20 --hash image \
   --image /boot/initramfs-3.10.0-327.4.5.el7.x86_64.img --cmdline ''
```

Hints to avoid common pitfalls:

- Copy the command line from the boot loader configuration to the `tb_polgen` command line by copy/paste, bearing the following in mind:
  - Never include the image name ("`/boot/vmlinuz-...`") in the command line passed to `tb_polgen`. (In GRUB2 setups, the image name appears twice on the "`module2`" lines, making it appear as if the image name was used as the first boot parameter. This is not the case.)
  - The "`showopts`" parameter is stripped from the kernel command line by GRUB. Don't include it in the `tb_polgen` command line.
- The tboot module number (indicated by the `--num` parameter of `tb_polgen)` is counted in the order the "module" lines appear in the boot loader configuration, starting from 0. The above example is for a case without the Xen hypervisor. For Xen setups, the Xen hypervisor is module 0, the kernel is module 1 and the initial RAM disk is module 2.
- Don't use PCR 18 with the `--pcr` parameter; this would make it loose its stability against software updates. Use PCR 19, 20, or 21.

It is recommended to record the command lines used to create a VLP and store them somewhere along with the policy file in order to be able to reproduce or modify it later. The created policy can be examined with

```
tb_polgen --show vlp.pol
```

The command shows a policy entry for every module, and one hash line for every allowed variant of each entry. When the policy is updated later, obsolete entries can be removed as follows:

```
tb_polgen --del --num 1 --pos 0 vlp.pol
```

This example would remove the first hash `(--pos 0)` of module 1 `(--num 1)`.

Creating the Launch Control Policy Elements

To have the created VLP verified by the ACM at boot time, convert it into a CUSTOM Launch Policy (LCP) element:

```
lcp_crtpolelt --create --type custom --uuid tboot --out vlp.elt vlp.pol
```

The second mandatory policy element for the LCP is an MLE element to validate the Measured Launch Environment (the tboot image) itself and its command line. The procedure is similar as for VLP. It is possible to allow different command lines and even different MLE images (tboot versions) in a single element. Again, it is recommended to take notes of the command lines used.

```
### Calculate the hash of the MLE and the tboot command line. Allow different command lines for tboot.
lcp_mlehash -c 'logging=serial,vga,memory pcr_map=da' /boot/tboot.gz >mle.txt
lcp_mlehash -c 'logging=serial pcr_map=da' /boot/tboot.gz >>mle.txt

### Create a policy element from the hash
lcp_crtpolelt --create --type mle --out mle.elt mle.txt
```

Optionally, one can create a "PCONF" policy element to restrict the allowed configurations. This re-introduces parts of the "Static Root of Trust" concept into the Dynamic TXT scheme: PCRs 00-07 represent the state of various boot-time components such as BIOS and Option ROMs [5]. PCONF elements bound to these indices need to be updated when the BIOS is updated. Do not create PCONF elements for the TXT PCRs 17-19!

```
### Read out the current PCR state, and create a PCONF element from PCRs 00-07
egrep 'PCR-0[0-7]:' /sys/class/tpm/tpm0/device/pcrs >pconf.txt
lcp_crtpolelt --create --type pconf --out pconf.elt pconf.txt
```

In the following, we will assume that a PCONF element is used. If you don't use it, just leave it out.

Creating the Signed Launch Control Policy

The following commands create an LCP from the elements created above. We will create a signed policy to avoid update problems later. This must be run on the system where the signing authority (see above) has stored its keys.

```
### Combine the previously created elements into a policy list
lcp_crtpollist --create --out unsigned.lst mle.elt pconf.elt vlp.elt
cp unsigned.lst signed.lst

### Convert the list to signed list type and add the public key (no signature yet)
lcp_crtpollist --sign --nosig --pub txt-pub.pem --out signed.lst

### Run openssl dgst to create a signature for the policy list
openssl dgst -sha1 -sign txt-priv.pem -out signed.lst.sig signed.lst

### Add the signature to the policy list file
lcp_crtpollist --addsig --sig signed.lst.sig --out signed.lst

### Finally, create the LCP from the signed list
lcp_crtpol2 --create --type list --minver 0x30 --pol lcp.pol --data lcp.dat signed.lst
```

It would be possible to add "`unsigned.lst`" at the end of the last command, creating an LCP with two policy lists, one signed and one unsigned. But that doesn't provide an actual benefit because the two policies are identical. Later on, only the signed policy lists can be updated hassle-free, so the unsigned policy list is really just a useless burden.
The `--minver` option to `lcp_crtpol2` denotes the minimum acceptable version of the ACM. The value 0x30 is valid for PRIMERGY M2 servers. To obtain the correct value for your platform, read it from the Platform Supplier policy with the following commands:

```
lcp_readpol -i 0x50000001 -f /tmp/ps.pol
lcp_crtpol2 --show /tmp/ps.pol | grep sinit_min_version:
```

Installing the Launch Control Policy

The results of the above procedure are the policy file `lcp.pol` and the policy data file `lcp.dat`. They are distributed by the signing authority to the servers that need them. The policy data file must be installed under `/boot`; here we assume the path `/boot/lcp.dat`.
The policy data file must be added to the boot loader configuration as an additional module. For GRUB2, this looks as follows:

```
menuentry 'Linux with tboot' {
    multiboot2 /boot/tboot.gz /boot/tboot.gz logging=serial,memory pcr_map=da
    module2 /boot/vmlinuz-3.10.0-327.el7.x86_64 /boot/vmlinuz-3.10.0-327.el7.x86_64 [...]
    module2 /boot/initramfs-3.10.0-327.el7.x86_64.img /boot/initramfs-3.10.0-327.el7.x86_64.img
    module2 /boot/lcp.dat
}
```

For GRUB1, proceed analogously.
The policy itself is written to the TPM NVM. This step requires the TPM owner password.

```
lcp_writepol -i 0x40000001 -f lcp.pol -p $OWNER_PASSWORD
```

Updating the Policy

When the kernel, hypervisor, or tboot itself is updated, or when the initial RAM disk is recreated, the policy (more precisely, the policy data file containing information on both LCP and VLP) needs to be updated. In the data centre, this is typically be done by the same authority that eventually adds the signature, and updated policy is distributed together with kernel updates and pre-built initial RAM disks to all servers in a deployment. The signing authority creates a new policy data file as described above and installs it as `/boot/lcp.dat` on all systems. The policy itself doesn't need to be reinstalled to TPM NVM. Therefore, entering the TPM Owner password on each system is not required.

"Details / Authorities" PCR Mapping

TXT knows two ways to map measurements to PCRs, "legacy" and "Details / Authorities" (DA) [2]. The latter is preferable, because in this scheme PCR 18 is only extended with "Authorities" such as signing keys and policies, and not with the actual measured images. The measurements of all "details" go into PCR 17, and we configured VLP such that its measurements don't affect PCR 18. PCR 18 in the DA scheme is unaffected by updates of the initial RAM disk, kernel, hypervisor, and even tboot itself.
DA is activated by adding the parameter "`pcr_map=da`" to the tboot command line. This was done already in the examples above.
Caution: Once DA has been activated, it's important to stick with it; otherwise the PCR values after boot won't match. On GRUB2 systems, the `grub2-mkconfig` utility regenerates the boot loader configuration file on every invocation, deleting the `pcr_map=da` parameter again. See "Double-Checking the Boot Loader Configuration" above for a workaround.

Booting into "Best Practices" TXT Environment

When you reboot your system after applying the procedure above, you will have a fully featured TXT system. PCR 18, which is only valid after a successful TXT boot, can be used to seal sensitive data such that it will be visible only in a secure system, without risking data loss due to authorized software updates. The LCP and "halt" type VLP make sure that the system fails to boot if any changes are made to the MLE, kernel, hypervisor, their command lines, or the initial RAM disks.
As a general warning, keep in mind that all code measurements are carried out before executing the respective images. The Linux kernel is extended by loading modules at run-time, and even contains self-modifying code. Therefore, the initial measurement, valuable as it may be to rule out boot-time malware infection, says little about the state of the kernel at any later time – you have to trust the OS vendor, too.

Sealed and Encrypted Block Device Example

Here comes a simple example how to use a sealed password for an encrypted block device. First, create a random password and seal it against PRC 18:

```
od -An -t x8 -N32 -w80 /dev/urandom  | tr -d ' ' | tpm_sealdata -z -p 18 >key.asc
```

This string will be used as a passphrase for an encrypted disk. In the following example, `$BLOCKDEVICE` could be a disk, partition, logical volume, or loop device:

```
### Prepare the block device with random data
dd if=/dev/urandom bs=1M of=$BLOCKDEVICE

### Format the device with the sealed passphrase. Note the dash "-" at the end of the line
tpm_unsealdata -z -i key.asc | cryptsetup -v luksFormat $BLOCKDEVICE –

### Open the device with the sealed passphrase
tpm_unsealdata -z -i key.asc  | cryptsetup luksOpen --key-file - /dev/loop0 decrypted

### Format the decrypted device with the file system of your choice
mkefs.xfs /dev/mapper/decrypted

### Close the device
crtyptsetup luksClose decrypted
```

The sealed and encrypted file system is now set up. To use the file system, proceed as follows:

```
tpm_unsealdata -z -i key.asc  | cryptsetup luksOpen --key-file - /dev/loop0 decrypted
mount -t xfs /dev/mapper/decrypted $MOUNTPOINT

### [... Work with data ...]

umount $MOUNTPOINT
cryptsetup luksClose decrypted
```

If PCR 18 is changed (or FF FF FF… in non-TXT environment), this file system will be inaccessible.
The procedure outlined here needs fine-tuning for practical purposes. For safety, it is recommended to create a second pass phrase for the LUKS device and store it offline, for data recovery e.g. in case the TPM hardware ceases to function.

Tightening Further

Using the Linux kernel parameter `module.sig_enforce=1`, and enforcing its use with the VLP, causes the kernel to refuse to load any module which isn't signed by a known authority (in practice, the OS vendor). This increases the integrity of the kernel by extending the Chain of Trust to all kernel modules. **Using this option will prevent loading third-party modules, including Fujitsu-provided modules**, so don't use it if your system needs 3rd party drivers. `module.sig_enforce=1` is supported under RHEL 7, SLES 11 SP4 and SLES 12.
On SLES, `module.sig_enforce=1` also puts other protections in place, such as disabling `kexec/kdump` and preventing write access to physical or I/O memory from user space. This enforces effectively the same kernel protection as UEFI Secure Boot (see below). Under RHEL 7, the same effect can be obtained by running the following command very early in the boot sequence (from initial RAM disk scripts, right after mounting sysfs):

```
echo 1 >/sys/kernel/security/securelevel
```

Additional integrity can be asserted by deploying the Linux Integrity Measurement Architecture (IMA) [6]. IMA is out of scope for this white paper.

TXT and UEFI Secure Boot

UEFI Secure Boot [7] is an independently developed technology with a similar purpose than TXT: Reducing the attach surface of the system. **It is impossible to activate TXT and UEFI Secure Boot in parallel on the same system.** The reason for that is the different role of the BIOS. Under UEFI secure boot, the BIOS is considered the "Root of Trust" and used to obtain public keys to verify software against. TXT, on the contrary, views the BIOS as a possible source of malware. TXT therefore disables EFI runtime support in the kernel, which UEFI secure boot depends on.
Users wishing to secure their systems must therefore choose between the two technologies. The following table highlights similarities and differences, and should help you determine which technology is better suited for you security demands. See Glossary below for acronyms.

	UEFI Secure Boot	Trusted Execution Technology
Origin	UEFI Interface Forum, an industry consortium	Intel
Root of Trust	The UEFI BIOS, provided by the PS.	The ACM, provided and signed by Intel® and validated by hardware.
Authentication	Hashes or keys stored in UEFI variables "db", "dbx" and provided by the PS.	Hashes or keys provided by the PO (sometimes, the PS) and verified against hashes stored in the TPM.
Primary purpose	Protecting the base platform software (UEFI BIOS)	Providing a trusted environment for applications and virtual machines.
Needs hardware support?	No.	Yes, TPM and certain CPU/Chip set features.
Linux OS support	SLES 11 SP4, SLES12, RHEL 7.0 and newer	SLES 11 SP2, SLES 12, RHEL 6.2, RHEL 7.0 and newer (legacy); RHEL 7.2, SLES 12 SP2 (UEFI)
Failure mode	Halt at launch of boot loader / kernel	TXT Reset, TXT Shutdown.
Allows "lockout" of PO so that PO can't install/modify OS?	Yes (but disabled on current x86 systems). In practice, Microsoft is in charge of issuing / revoking valid code certificates worldwide.	No, PO can always override PS settings.
Protects kernel from user space?	Not by definition, but Linux activates `module.sig_enforce=1` automatically when secure boot is detected.	No, but this can be configured with `module.sig_enforce=1`.
Allows addition of PS/PO keys for validation of additional components in the chain of trust?	Yes, via db/dbx for PS and via MOK for PO. This makes it possible to run 3rd party drivers and custom-built kernels.	No.
Can sensitive data be sealed (made available only after verified boot)?	No.	Yes, as described in this paper.
Works with Network (PXE) boot?	No (on current PRIMERGY systems).	Yes.
Stable against software updates?	Yes, out-of-the-box.	Yes (if properly configured as described here).
Works in UEFI mode?	Yes.	Yes (only with latest Linux distributions).
Works in Legacy mode?	No.	Yes.
Ease of administration	Works out-of-the-box with all major Linux distributions. Only MOK installation, which is rarely needed, needs manual configuration.	Needs manual setup. No high-level tools are currently available (2016).

Glossary

- ACM: Authenticated Code Module. A small piece of code authenticated by Intel® that represents the "Root of Trust" in a TXT setup. The ACM is entered with the GETSEC(SENTER) CPU instruction. It is responsible for securing the system, validating the MLE and possibly other data, and launching the MLE. In modern servers, the ACM is part of the BIOS; earlier CPUs needed a separate piece of software.
- DA: "Details / Authorities", a scheme of assigning boot-time measurements by the ACM to PCRs.
- LCP: Launch Control Policy, a set of rules that determine the behaviour of the ACM, and data to be validated by the ACM.
- MLE: Measured Launch Environment. The second part in the Chain of Trust after the ACM, responsible for setting up the hardware and booting the kernel. Under Linux, the "tboot" MLE is commonly used.
- MOK: Machine Owner Key, a Linux community concept used for adding custom user certificates under UEFI Secure Boot.
- NVM: Non-volatile Memory. The TPM has a limited space of Non-volatile memory available. This memory is divided into spaces of variable length also referred to as "NVM indices". Several NVM indices are used for TXT to store important information such as policy hashes.
- PCR: Platform Configuration Register, a special register in the TPM that can't be written to. PCRs can only be "extended" with data. The PCR value represents the SHA-1 hash of all data "extended" into it since the last system reset. The PCR value is a "computationally unique" indicator of the components (such as MLE, kernel, etc) it has been extended with, meaning that it's computationally infeasible to create a different set of (tampered) data resulting in the same PCR value.
- PO: Platform Owner, the owner (user / administrator) of the system that has the password for the TPM "Owner" key.
- PS: Platform Supplier, the system manufacturer / OEM.
- RHEL: Red Hat Enterprise Linux
- SLES: SUSE Linux Enterprise Server
- TPM: Trusted Platform Module, the piece of hardware responsible for cryptographic operations, specified by the Trusted Computing Group.
- TSS: The "Trusted Software Stack" specified by the Trusted Computing Group.
- TXT: Trusted Execution Technology, an Intel® technology that combines a TPM with CPU, chip set, and OS features to provide a secure computing environment.
- UEFI: Unified Extensible Firmware Interface, a standard for a BIOS environment as opposed to "legacy" x86 PC BIOS. PRIMERGY servers
- VLP: Verified Launch Policy, a set of rules for the tboot MLE governing the launch of the kernel.
- Xen: An open-source hypervisor, supported under SLES 11 and SLES 12.

References

[1] D. Mulnix, „Intel(r) Trusted Execution Technology (Intel(r) TXT) Enabling Guide," Intel Corporation, 2014. [Online]. Available: https://software.intel.com/en-us/articles/intel-trusted-execution-technology-intel-txt-enabling-guide.

[2] Intel Corporation, „Intel(R) Trusted Execution Technology - Software Develoment Guide (315168-12)," 2015. [Online]. Available: http://www.intel.com/content/www/us/en/software-developers/intel-txt-software-development-guide.html.

[3] Trusted Computing Group, "Trusted Platform Module (TPM) Summary," 2008. [Online]. Available: http://www.trustedcomputinggroup.org/files/resource_files/4B55C6B9-1D09-3519-AD916F3031BCB586/Trusted%20Platform%20Module%20Summary_04292008.pdf.

[4] Various, "Trusted Boot (tboot) home page," 2009-2015. [Online].

[5] Wikimedia project, "Wikipedia: Trusted Execution Technology," 2005-2015. [Online]. Available: https://en.wikipedia.org/wiki/Trusted_Execution_Technology. [Accessed 2015].

[6] Various, "Linux Integrity Measurement Architecture (IMA)," 2009-2015. [Online]. Available: https://sourceforge.net/p/linux-ima/wiki/Home/.

[7] R. Wilkins and B. Richardson, "UEFI Secure Boot in Modern Computer Security Solutions," 2013. [Online]. Available: http://www.uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf.

[8] Unified EFI, Inc., „UEFI Specification v2.3.1D," 2013. [Online]. Available: http://www.uefi.org/sites/default/files/resources/2.3.1_D.pdf.

[9] Various, "Intel (R) TXT Overview (Linux kernel documentation)," 2009. [Online]. Available: https://www.kernel.org/doc/Documentation/intel_txt.txt.

[10] Trusted Computing Group, "TPM Main Specification Level 2 Version 1.2," 2011. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm_main_specification.

Revision History

1.0	2016-02-22	Initial revision
1.1	2016-02-23	Minor corrections

Formatiert: Schriftart: 8 Pt.
Formatierte Tabelle
Formatiert: FS Body copy black light
Formatiert: Schriftart: 8 Pt.
Formatiert: FS Body copy black light, Zentriert
Formatiert: FS Body copy black light, Zentriert
Formatiert: Schriftart: 8 Pt.
Formatiert: FS Body copy black light
Formatiert: Schriftart: 8 Pt.
Formatiert: Schriftart: 8 Pt.