Intel(R) TXT Overview:
==========

Intel's technology for safer computing, Intel(R) Trusted Execution Technology (Intel(R) TXT), defines platform-level enhancements that provide the building blocks for creating trusted platforms.

Intel TXT was formerly known by the code name LaGrande Technology (LT).

Intel TXT in Brief:
o Provides dynamic root of trust for measurement (DRTM)
o Data protection in case of improper shutdown
o Measurement and verification of launched environment

Intel TXT is part of the vPro(TM) brand and is also available some non-vPro systems. It is currently available on desktop systems based on the Q35, X38, Q45, and Q43 Express chipsets (e.g. Dell Optiplex 755, HP dc7800, etc.) and mobile systems based on the GM45, PM45, and GS45 Express chipsets.

For more information, see http://www.intel.com/technology/security/. This site also has a link to the Intel TXT MLE Developers Manual, which has been updated for the new released platforms.

Intel TXT has been presented at various events over the past few years, some of which are:
LinuxTAG 2008:
http://www.linuxtag.org/2008/en/conf/events/vp-donnerstag/details.html?talkid=110
TRUST2008:
http://www.trust2008.eu/downloads/Keynote-Speakers/3_David-Grawrock_The-Front-Door-of-Trusted-Computing.pdf
IDF 2008, Shanghai:
http://inteldeveloperforum.com.edgesuite.net/shanghai_2008/aep/PROS003/index.html
IDFs 2006, 2007 (I'm not sure if/where they are online)

Trusted Boot Project Overview:
==========

Trusted Boot (tboot) is an open source, pre- kernel/VMM module that uses Intel TXT to perform a measured and verified launch of an OS kernel/VMM.

It is hosted on SourceForge at http://sourceforge.net/projects/tboot. The mercurial source repo is available at http://www.bughost.org/repos.hg/tboot.hg.

Tboot currently supports launching Xen (open source VMM/hypervisor w/ TXT support since v3.2), and now Linux kernels.

Value Proposition for Linux or "Why should you care?"
==========

While there are many products and technologies that attempt to

measure or protect the integrity of a running kernel, they all assume the kernel is "good" to begin with. The Integrity Measurement Architecture (IMA) and Linux Integrity Module interface are examples of such solutions.

To get trust in the initial kernel without using Intel TXT, a static root of trust must be used. This bases trust in BIOS starting at system reset and requires measurement of all code executed between system reset through the completion of the kernel boot as well as data objects used by that code. In the case of a Linux kernel, this means all of BIOS, any option ROMs, the bootloader and the boot config. In practice, this is a lot of code/data, much of which is subject to change from boot to boot (e.g. changing NICs may change option ROMs). Without reference hashes, these measurement changes are difficult to assess or confirm as benign. This process also does not provide DMA protection, memory configuration/alias checks and locks, crash protection, or policy support.

By using the hardware-based root of trust that Intel TXT provides, many of these issues can be mitigated. Specifically: many pre-launch components can be removed from the trust chain, DMA protection is provided to all launched components, a large number of platform configuration checks are performed and values locked, protection is provided for any data in the event of an improper shutdown, and there is support for policy-based execution/verification. This provides a more stable measurement and a higher assurance of system configuration and initial state than would be otherwise possible. Since the tboot project is open source, source code for almost all parts of the trust chain is available (excepting SMM and Intel-provided firmware).

How Does it Work?
=================

- Tboot is an executable that is launched by the bootloader as the "kernel" (the binary the bootloader executes).
- It performs all of the work necessary to determine if the platform supports Intel TXT and, if so, executes the GETSEC[SENTER] processor instruction that initiates the dynamic root of trust.
  - If tboot determines that the system does not support Intel TXT or is not configured correctly (e.g. the SINIT AC Module was incorrect), it will directly launch the kernel with no changes to any state.
  - Tboot will output various information about its progress to the terminal, serial port, and/or an in-memory log; the output locations can be configured with a command line switch.
- The GETSEC[SENTER] instruction will return control to tboot and tboot then verifies certain aspects of the environment (e.g. TPM NV lock, e820 table does not have invalid entries, etc.).
- It will wake the APs from the special sleep state the GETSEC[SENTER] instruction had put them in and place them into a wait-for-SIPI state.
  - Because the processors will not respond to an INIT or SIPI when in the TXT environment, it is necessary to create a small VT-x guest for the APs. When they run in this guest, they will

simply wait for the INIT-SIPI-SIPI sequence, which will cause VMEXITs, and then disable VT and jump to the SIPI vector. This approach seemed like a better choice than having to insert special code into the kernel's MP wakeup sequence.
o Tboot then applies an (optional) user-defined launch policy to verify the kernel and initrd.
  - This policy is rooted in TPM NV and is described in the tboot project. The tboot project also contains code for tools to create and provision the policy.
  - Policies are completely under user control and if not present then any kernel will be launched.
  - Policy action is flexible and can include halting on failures or simply logging them and continuing.
o Tboot adjusts the e820 table provided by the bootloader to reserve its own location in memory as well as to reserve certain other TXT-related regions.
o As part of its launch, tboot DMA protects all of RAM (using the VT-d PMRs). Thus, the kernel must be booted with 'intel_iommu=on' in order to remove this blanket protection and use VT-d's page-level protection.
o Tboot will populate a shared page with some data about itself and pass this to the Linux kernel as it transfers control.
  - The location of the shared page is passed via the boot_params struct as a physical address.
o The kernel will look for the tboot shared page address and, if it exists, map it.
o As one of the checks/protections provided by TXT, it makes a copy of the VT-d DMARs in a DMA-protected region of memory and verifies them for correctness. The VT-d code will detect if the kernel was launched with tboot and use this copy instead of the one in the ACPI table.
o At this point, tboot and TXT are out of the picture until a shutdown (S<n>)
o In order to put a system into any of the sleep states after a TXT launch, TXT must first be exited. This is to prevent attacks that attempt to crash the system to gain control on reboot and steal data left in memory.
  - The kernel will perform all of its sleep preparation and populate the shared page with the ACPI data needed to put the platform in the desired sleep state.
  - Then the kernel jumps into tboot via the vector specified in the shared page.
  - Tboot will clean up the environment and disable TXT, then use the kernel-provided ACPI information to actually place the platform into the desired sleep state.
  - In the case of S3, tboot will also register itself as the resume vector. This is necessary because it must re-establish the measured environment upon resume. Once the TXT environment has been restored, it will restore the TPM PCRs and then transfer control back to the kernel's S3 resume vector.
    In order to preserve system integrity across S3, the kernel provides tboot with a set of memory ranges (RAM and RESERVED_KERN in the e820 table, but not any memory that BIOS might alter over the S3 transition) that tboot will calculate a MAC (message authentication code) over and then seal with the TPM. On resume and once the measured environment has been re-established, tboot

will re-calculate the MAC and verify it against the sealed value. Tboot's policy determines what happens if the verification fails. Note that the c/s 194 of tboot which has the new MAC code supports this.

That's pretty much it for TXT support.

Configuring the System:
==========

This code works with 32bit, 32bit PAE, and 64bit (x86_64) kernels.

In BIOS, the user must enable: TPM, TXT, VT-x, VT-d. Not all BIOSes allow these to be individually enabled/disabled and the screens in which to find them are BIOS-specific.

grub.conf needs to be modified as follows:

```
        title Linux 2.6.29-tip w/ tboot
          root (hd0,0)
                kernel /tboot.gz logging=serial,vga,memory
                module /vmlinuz-2.6.29-tip intel_iommu=on ro
                       root=LABEL=/ rhgb console=ttyS0,115200 3
                module /initrd-2.6.29-tip.img
                module /Q35_SINIT_17.BIN
```

The kernel option for enabling Intel TXT support is found under the Security top-level menu and is called "Enable Intel(R) Trusted Execution Technology (TXT)". It is marked as EXPERIMENTAL and depends on the generic x86 support (to allow maximum flexibility in kernel build options), since the tboot code will detect whether the platform actually supports Intel TXT and thus whether any of the kernel code is executed.

The Q35_SINIT_17.BIN file is what Intel TXT refers to as an Authenticated Code Module. It is specific to the chipset in the system and can also be found on the Trusted Boot site. It is an (unencrypted) module signed by Intel that is used as part of the DRTM process to verify and configure the system. It is signed because it operates at a higher privilege level in the system than any other macrocode and its correct operation is critical to the establishment of the DRTM. The process for determining the correct SINIT ACM for a system is documented in the SINIT-guide.txt file that is on the tboot SourceForge site under the SINIT ACM downloads.