SANDIA REPORT

SAND2011-0475
Unlimited Release
January 2011

Trusted Computing Technologies, Intel®Trusted Execution Technology

Jeremy Daniel Wendt and Max Joseph Guise

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

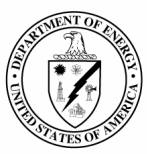
Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 6056900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods/asp?loc=7-4-0#online



SAND2011-0475
Unlimited Release
January 2011

Trusted Computing Technologies, Intel®Trusted Execution Technology

Jeremy Daniel Wendt and Max Joseph Guise
Information Systems Analysis Center
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS1073

Abstract

We describe the current state-of-the-art in Trusted Computing Technologies – focusing mainly on Intel's Trusted Execution Technology (TXT). This document is based on existing documentation and tests of two existing TXT-based systems: Intel's *Trusted Boot* and Invisible Things Lab's *Qubes OS*. We describe what features are lacking in current implementations, describe what a mature system could provide, and present a list of developments to watch.

Contents

List of Figures

1 Introduction

Critical systems perform operation-critical computations on high importance data. In such systems, the inputs, computation steps, and outputs may be highly sensitive. Sensitive components must be protected from both unauthorized release, and unauthorized alteration: Unauthorized users should not access the sensitive input and sensitive output data, nor be able to alter them; the computation contains intermediate data with the same requirements, and executes algorithms that the unauthorized should not be able to know or alter.

Due to various system requirements, such critical systems are frequently built from commercial hardware, employ commercial software, and require network access. These hardware, software, and network system components increase the risk that sensitive input data, computation, and output data may be compromised.

Problem Statement: Given a computing system built from commercial hardware, using commercial software, on an untrusted network: Protect sensitive inputs, computations, and outputs from unauthorized release and alteration.

This document describes an investigation into how "Trusted Computing Technologies" and specifically Intel®Trusted Execution Technology (TXT) can reduce the risk of critical-system compromise. Section 2 defines how the word "trust" is used in trusted computing. Section 3 describes non-Intel-based trusted computing solutions. Section 4 describes Intel's Trusted Execution Technology. Section 5 describes what tests were performed on actual computing resources. Section 6 describes further tests which could be performed to further validate TXT-generated security improvements as claimed by Intel. Section 7 lists Intel's acknowledged limits when creating TXT as well as publicized flaws in TXT's design or implementation. Section 8 summarizes and discusses all previous sections. Section 9 discusses known developing technologies and improvements that extend the current state-of-the-art. Section 10 provides the reasons for and defines the bounds of what was researched.

2 Trust

In his Turing Award Lecture, Ken Thompson demonstrated ways to hide malicious code within visually harmless source code [Thompson, 1984]. He concluded with the following: "You can't trust code that you did not totally create yourself." He included language compilers, assemblers, and hardware microcode as components that would need to be self-created to be completely trustworthy.

However, the modern computing environment contains many complicated components – hardware, operating system, compiler, application software, and network capabilities. This set of components is called the "trusted computing base" (TCB), as a vulnerability or bug in any component could jeopardize the security of the whole [Wikipedia, 2010c]. A small, trusted group of engineers designing and implementing each would likely never finish their project. Any system so-built would still be susceptible to attack due to any design or implementation errors or oversights.

Therefore, using any modern computing environment requires trusting some set of externally developed hardware, software, or network resources. To permit the use of externally developed tools, the Trusted Computing Group (TCG – an initiative formed by various technical companies, including Intel, AMD, Microsoft, and IBM) defines trust as follows: "Trust is the expectation that a device [or component] will behave in a particular manner for a specific purpose" [Trusted Computing Group, 2010]. For any component (hardware, software, or network), the user has an expectation of what action it will or will not perform in response to some user input. By the TCG's definition, a trusted component is one that the user (or some entity trusted by the user) has ensured behaves as expected.

Comparing these two definitions of trust is instructive. Both Thompson's and the TCG's definitions of trust require considerable effort. Thompson's trust requires all components be built by oneself – or a small number of trusted individuals. The TCG's trust requires that each component be thoroughly evaluated. Both definitions require that no external entity has altered any of the components or their configuration since the trusted system was established.

By either definition of trust, creating a trusted system requires a time consuming and costly process. In fact, recreating or thoroughly validating all components of a modern computing system is infeasible. Therefore, the set of all trusted components (TCB) must be decreased as much as possible. Not only does a smaller TCB make trusting the system more possible, it also generally leads to a more secure system [Shinagawa et al., 2009]. The following two sections describe different solutions developed to shrink the untested TCB.

3 Other Trusted Computing Solutions

3.1 The SSP

The Sandia Secure Processor (SSP)[1] was created to be a fully trusted processor for embedded systems (by Thompson's definition of trust) [Wickstrom et al., 2004a]. The hardware was developed at Sandia National Labs by a trusted team of engineers. They also created a trusted class loader to ensure that programs loaded into the SSP will run precisely as coded. Both the hardware and class loader were tested rigorously by different teams using various techniques to ensure correctness. Further publications discuss more details of the class loader [Wickstrom et al., 2004b, Winter et al., 2005].

The SSP's stated purpose (embedded computing) does not match this paper's stated problem (protecting sensitive inputs, computations, and outputs on commodity hardware, software, and networks). Due to this mismatch, we do not discuss it further herein.

3.2 Anti-Tamper

Anti-Tamper (AT) techniques are "intended to prevent and/or delay exploitation of critical technologies" of all kinds [Department of Defense, 2010]. AT attempts to prevent one or more of unauthorized access, reverse engineering, and violating code integrity (e.g., inserting malware). Therefore, AT techniques could be used to protect either Thompson- or TCG-based trusted systems. The following are common computer-based AT techniques [Atallah et al., 2004]:

- **Hardware-supported protections**: These protections are built into the computing system itself. Intel®'s TXT employs many of AT's hardware-based protections. In short, these protections include measuring programs before execution to ensure they are unchanged, and hardware-based encryption. Such techniques' strengths and weaknesses are further discussed throughout this document.

- **Encryption Wrappers**: Portions of the code are encrypted separately from each other, with each decrypted only when in use. In this way, the entire program is never decrypted at once. While this does not prevent attackers from obtaining an unencrypted version of the entire program, obtaining an unencrypted version requires running the program, taking multiple memory images, analyzing each image to identify unencrypted portions, and then merging these images.

- **Code obfuscation**: After compilation, the code's layout, data, and control statements are obfuscated to hinder reverse engineering. Care must be taken to make deobfuscation difficult, and to ensure that obfuscated portions still appear to be standard program executions and do not execute significantly slower than the original codes.

- **Watermarking and fingerprinting**: Watermarks and fingerprints are unique messages inserted into each instance of a piece of software to identify the original purchaser, the legality of the program, and/or the integrity of the program (that it is unchanged). Alone, they do

[1] This device is still being developed, although now called the "Score Processor".

not prevent tampering, but provide evidence of it. To properly function, watermarks and fingerprints must cover all secured portions of the code, and be difficult to reproduce.

- **Guarding**: Several software guards can be inserted into code that monitor if code is being used incorrectly, and may alter or hinder the code from functioning if misuse is identified. Guards should be hidden in the code so that they are hard to detect and remove.

No software-based AT technique will completely prevent unauthorized access, reverse engineering, or integrity violation: Each serves only to increase the attack's difficulty. It is recommended that several AT techniques be used together to create a more robust defense. However, each technology increases development effort, and frequently decreases execution efficiency. Furthermore, encryption wrappers, code obfuscation, watermarking, and guarding all require the ability to alter the codebase – preventing the use of many commercial codes.

4 Intel®Trusted Execution Technology

Intel®Trusted Execution Technology (TXT) is Intel's implementation of the Trusted Computing Group's specification for trusted computing hardware. The TCG proposed six technology concepts required to create a system that fulfills their definition of "trusted" [Grawrock, 2008, Wikipedia, 2010b]:

1. **Endorsement key**: An unalterable RSA public/private key pair, written in the computer hardware. The private key is kept private even to software on the computer – it never leaves the chip. These keys are used to create trusted digital signatures and perform RSA encryption.

2. **Secure input and output**: Information transmitted between computers (via "channels") and information transmitted between the user and the computer (via "paths") must be protected from unauthorized reading or altering to be trusted.

3. **Memory curtaining / protected execution**: To trust a program's execution, its data and instructions must be secured. Both are stored in computer memory during execution. Therefore, a securely launched program's memory is protected from all external entities, including the Operating System and Direct-Memory-Access-enabled devices such as network cards. Protected execution also includes isolating all internal CPU state from external entities.

4. **Sealed storage**: This technique binds private information to specific computing system signatures – hardware and software signatures. This means that data on a hard drive cannot be read by removing the disk from one machine and making it a slave to another machine.

5. **Remote attestation**: When a computer system requests results or data from another system, the requesting system needs to know that the second system is trustworthy. Remote attestation is the mechanism whereby a computing system demonstrates to the requesting system that it is using trustable hardware in a trusted configuration.

6. **Trusted Third Party (TTP)**: The signatures produced by these features could be used to specifically identify machines and (by proxy) users. Many features of the current internet benefit from user anonymity. A Trusted Third Party could validate a machine as trusted, supply an unforgeable credential to the validated machine that the validated machine could present to avoid self-identifying. If the third party is truly trusted by both parties, then the requesting system can know that the computing system is trustworthy, and the computing system can know that its identy is secure.

For this document's problem – a critical system operating on high sensitivity data using third-party technologies – there are two principle TXT features: static Roots of Trust for Measurement (RTM) and dynamic RTM. In short, both RTM features measure some program or set of programs before starting them to ensure they have not changed from a previously identified trusted configuration.

Static RTM begins at system boot. The TXT-enabled computer measures each boot component via cryptographic hash before allowing it to execute. Each component's hash is combined with the previous component's hash to create a new value. Each value is compared with a trusted boot policy (the measurements produced by an earlier boot sequence that was specifically marked as trusted). A static RTM begins by measuring the most fundamental components (SMM, BIOS,

VMM, OS, etc.). These measurements ensure two things: First, they ensure that each program is unchanged from the previously identified trusted boot. Second, since each component's hash is combined with that of the previous component, and since the combined value is compared to the trusted value, the measurements ensure that the boot order has not been changed: Only the trusted components have been started as part of the boot process. Since static RTM begins at system boot, it guarantees that no rootkit has installed itself below the operating system, and that the operating system itself has not been corrupted by unwanted start-on-boot programs.

Dynamic RTM begins upon user or program request at any time during or after boot. The TXT-enabled computer will measure the about-to-start program(s) in a similar fashion to the static RTM – ensuring that the started programs have not been altered since a previously identified trusted configuration. Furthermore, after a dynamic RTM completes, the resulting program can be run with secure input and output, and memory curtaining / protected execution. By measuring all programs that are launched into a secure I/O and memory/execution protected partition, TXT guarantees that none of the programs in the trusted partition have been altered and protects their data and execution from corruption while running.

4.1 Flicker

Flicker is an early technology built using AMD's implementation of the TCG's recommendation called "Secure Virtual Machine" (SVM). In short, Flicker removes the trust-requiring components from a larger program and quickly runs those trust-requiring components as needed in their own dynamic RTM [McCune et al., 2008]. In this way, the TCB is reduced to a minimum: One need trust only the computer hardware and the extracted software component.

Unfortunately, Flicker would not solve this document's problem for the following reasons: First, Flicker requires altering the program by extracting trust-requiring components from the rest of the program. Our critical system employs sensitive data and operations continually during execution. Second, Flicker halts execution of all other processes on the computer (including the operating system) when the critical code runs. Flicker's authors do not state if this was a decision they made, or if AMD's early SVM technology required this. However, this would prevent the system user from doing two sensitive operations (or sensitive and normal operations) at the same time unless both were within the same program.

5 Implemented Tests

To validate Intel®'s claims and to provide us experience with TXT, we purchased a laptop with all required TXT hardware and tested both currently existing TXT-based software systems: Intel®'s Trusted Boot (tboot), and Invisible Things Lab's Qubes OS.

In both systems, our goal was to test TXT's ability to secure the path from the keyboard to an application. Our motivation was protecting a password or secret as it is typed into one program from being recorded by a malware keylogger. A dynamic-RTM-launched application with partitioned memory and protected execution should protect the typed password or secret from a keylogger not in the secured partition.

5.1 tboot

Intel released the first software support for their TXT hardware, called Trusted Boot (tboot) [Intel, 2010]. Tboot is a package which can be added to a Linux distribution to enable various TXT features.

Unfortunately, Intel has provided little documentation on setting up and using tboot. What little documentation we found (cited at the end of this section) was spread throughout the internet – and was difficult to find. Generally, to find instructions on how to overcome a problem, we had to attempt to employ some TXT technique, receive an error message, and search the internet for that specific error message. As most attempts included minimally rebooting the machine and often required installing a different version of the operating system, this was a painstakingly slow process.

After significant setup and configuration work, we stopped examining tboot in order to examine Qubes OS before the deadline. Before stopping tboot work, we had configured the laptop as follows: Fedora 13 ("Xenified" Linux kernel 2.6.32.13) on the Xen Hypervisor (version 3.4.3). After installing Fedora and Xen, we installed tboot, and successfully took control of the TPM (a fundamental piece of the TXT hardware), stored measurements of the Xen Hypervisor and components of Fedora, and attempted a dynamic RTM (beginning with Xen boot) with a tboot-provided boot policy. The RTM did not complete the boot process. We were able to see the system attempt to execute the fundamental RTM operation (`GETSEC[SENTER]`), but then the laptop failed with a white screen and no visible debugging information. We were unable to identify the error messages because the hardware did not support external logging, and the internal memory logging did not work.

Helpful resources: The information necessary to understand and use Intel's TXT via tboot were spread across various sources. Intel Senior Principal Engineer David Grawrock's text provides details of TXT architecture and argues for why TXT is a complete solution, but lacks specific details on how to use TXT and tboot [Grawrock, 2008]. Another Intel Technologist – Joseph Cihula – created slides on how to use tboot [Cihula, 2007]. Invisible Things Lab has successfully attacked TXT, and their write-ups contain useful information on the steps tboot must take to startup [Wojtczuk and Rutkowska, 2009, Wojtczuk and Rutkowska, 2009]. Various internet websites provided other useful answers during debugging and setup: tboot-focused (`http://blog.gmane.org/gmane.comp.boot-loaders.tboot.devel` and `http://lwn.net/Articles/382077/`), Xen and Fedora focused

(http://fclose.com/b/2405/setting-up-stable-xen-dom0-with-fedora-xen-3-4-3-with-xenified-linux-kernel-2-6-32-13-in-fedora-12/), and an overview of TXT hardware (http://www.linuxjournal.com/article/6633?page=0,0).

5.2 Qubes OS

Invisible Things Lab (ITL) is a group of world-class cybersecurity researchers who frequently demonstrate some of the most groundbreaking security breaches at BlackHat conferences. They have discovered all known TXT breaches (see below). They have also created Qubes OS – a Linux-based, cyber-security-focused Operating System that employs various TXT principles [Rutkowska and Wojtczuk, 2010, Rutkowska, 2010]. It is currently an Alpha release – meaning the software is feature complete for their first planned release, but has several bugs. Qubes's distinguishing characteristic is that, to better protect system resources, most operations and programs are launched into their own Virtual Machines (VMs). Although the current system only supports launching Linux-based VMs, they plan later releases supporting Windows- and Macintosh-based VMs.

Qubes OS employs various components of Intel TXT technologies:

- The harddisk, network, and user I/O controllers are partitioned from each other.
- The system boot code is protected.[2]
- A root of trust is created on startup – after the bootloader.

We found installing and running Qubes OS to be straightforward. We then performed the following tests:

- **Downloaded malware**: Qubes OS launches most applications into their own separate VMs. We attempted to install a keylogger into one of these application VMs to see which parts of the system were vulnerable should such an attack succeed. In the limited time we had, we were unable to install a successful keylogger to an application VM for two Qubes-OS-security reasons: First, Qubes employs Intel VT-d (one TXT technology) to partition the VMs from unnecessary I/O devices. This partitioning prevented us from transferring the keylogger to the application VM via CD ROM or memory stick. Second, had we successfully imported the keylogger into the application VM, the application VM lacks sufficient privileges to install our keylogger.

  We were able to download the keylogger via FTP into a networking VM (a VM that sandboxes networking capaiblities). However, Qubes VMs use a different set of interrupts than the Dom0 VM to handle keystrokes. This difference meant that our keylogger would not work for this test. The project funding ended before we were able to modify the software to correctly monitor keystrokes in the VM. We believe that a successfully installed keylogger would only have been able to monitor keystrokes typed into the same VM as it is loaded into – missing all keystrokes typed into other VMs.

[2]Their documentation doesn't specify precisely how it is protected, but we guess the boot code is encrypted through the TPM. While an attacker could destroy this code and make the system unbootable, he could not alter the boot code.

- **Corrupted Qubes OS Dom0**: In Qubes OS, user I/O functions – including keyboard handling – are performed by the administrative domain (Dom0). Keystrokes are passed from Dom0 to the appropriate application VM. This design indicates that a Dom0-installed keylogger would capture all keystrokes. To verify this, we installed a keylogger into Qubes OS's Dom0. We verified this keylogger could see all keystrokes. This problem is mitigated somewhat by the difficulty of installing a keylogger into Dom0 over the network without user consent. Qubes OS sandboxes all networking capabilities in unprivileged VMs – code sent to these VMs cannot write to Dom0.

Although all TXT hardware features were enabled in the BIOS, we are not certain that Qubes OS was properly configured to leverage each of them – the documentation does not specify how to enable them in the OS, nor how to determine if they are being used. ITL should soon release further Qubes OS documentation.

6 Further Possible Testing

Due to this project's constraints (Section 10), we were unable to perform any comprehensive tests of Intel's claims of TXT's protections. If these constraints were removed, we would perform at least the following tests:

- **Complete Qubes OS tests**: We would like to verify our suspicion that a downloaded keylogger would only be able to monitor keystrokes typed in the infected VM. Furthermore, we would like to search for any ways that a downloaded malware could circumvent the unprivileged VM and install itself into Dom0.

- **Other common attacks**: The Qubes-OS-based keyboard logger test we performed tested the secure path from the keyboard to the program. However, this still leaves two areas that non-TXT systems leave open to attack: the secure channel from the program to the screen, and the curtained memory / protected execution. Screen scrapers can read the memory used for the frame buffer and store any data presented there to the user. Priviledged ring 0 codes have access to other code's memory and CPU state. Intel claims that TXT prevents these attacks from succeeding, but we would like to validate these claims.

- **Known successful TXT attacks**: *Invisible Things Lab* (ITL) has demonstrated three different ways to corrupt TXT-enabled compute systems: an Intel BIOS bug (see demo advertised in [Invisible Things Lab, 2009b]), two System Managment Mode (SMM) memory vulnerabilities[3] [Wojtczuk and Rutkowska, 2009, Invisible Things Lab, 2009b], and a bug in the SINIT module [Wojtczuk et al., 2009, Invisible Things Lab, 2009a]. As all but the SMM vulnerabilities[4] are supposed to be patched [Intel, 2008, Intel, 2009], we would expect the BIOS and SINIT bugs to fail on a current system.

  Although ITL has found all known TXT exploits, ITL is quick to emphasize that TXT is a major step forward [Invisible Things Lab, 2009a].

[3] ITL's vulnerabilities make it possible to inject an SMM rootkit [Embleton et al., 2008].

[4] Intel says they have a design for a "SMM Transfer Monitor" (STM) that should eliminate the SMM vulnerabilities. These STMs are not yet released in any system.

7 TXT's Limitations

Other than the successful attacks discussed previously, TXT still has several limitations.

Poor external support: Principal among these is poor external support. The only two existing operating systems supporting TXT are Linux variants: Xen/Linux with Intel's tboot, and ITL's Qubes OS. Windows's only TXT use is BitLocker Drive Encryption [Microsoft, 2010, Wikipedia, 2010a]. We know of no TXT use by Mac OS X. Since creating a dynamic RTM requires ring 0 (or OS-level) privileges and static RTM requires measurements on startup, until Microsoft and Apple add TXT support, TXT is unusable on their operating systems.

We believe that this poor external support is caused by two related problems. First, TXT is a relatively new technology: TXT-supporting hardware is only a few years old. Since TXT requires OS support, until operating systems are released with TXT support, TXT-feature-leveraging applications cannot be created. Second, since the principles behind TXT are new, implementations differ. Since Intel's and AMD's solutions do not support the exact same features, software companies are slower to support it. They don't want to invest considerable energy into supporting a technology that is only available to a fraction of their users without strong customer demand.

Time costs: Another limitation is the time cost required for a complete static RTM. In creating a static RTM, every piece of code, and every code dependency must be examined via crytographic hash before that program is executed. This means that before booting a VMM (such as Xen) in a static RTM, the TXT hardware must perform a cryptographic hash on several GBs of files – slowing the boot time substantially.

Limited measurements: Currently, it seems that RTMs only measure up to the VMM that launches each VM. This means that the VM, its OS, and any applications launched into it are not individually measured and ensured to be unchanged. This means that any of these could be corrupted without TXT alerting the user. We believe that this limitation applies only to current implementations: TXT hardware could be used to allow further measurements through further OS software support.

User burden: TXT also increases the system administration burden. TXT's measurements ensure that a program has not changed since a user specifically marked that program as "trusted". However, security patches are valid reasons for the trusted program to change. This means that after the update has completed, the user (or some system administrator) would have to explicitly mark the new versions as "trusted". If this step were somehow made implicit, what would prevent malware from exploiting the "implicit trust feature" to mark itself as trusted?

After-startup attacks: Both static and dynamic RTMs measure the launched environment. Dynamic RTMs also provide several useful security features (curtained memory, protected execution, etc.). However, once the program is launched there is no way to identify if the executing program has been compromised. Therefore, a program could be corrupted after execution begins and the attack would go undetected until the next time the program is launched ["University of Michigan", 2010]. The longer a dynamic RTM is left open, the greater the chance of a successful attack.

Identifies corruption only: Furthermore, TXT only identifies a corrupted system and prevents it from executing. While this is very useful, fixing a corrupted system is still the system administrator's job and requires other tools.

Trusted path not complete: A trusted path ensures that no input from the user or output to the user can be read or altered by any untrusted party. TXT prevents several trusted-path attacks. However, since TXT features are purely within the chipset, there are still available attacks:

- Hardware keyboard loggers can be inserted between the keyboard's plug and the computer's socket. These hardware loggers store all keyboard data – awaiting physical retrieval.
- A similar device could be created to store display output to the monitors.
- A truly desperate attacker could simply video record the keyboard or monitors themselves.

Hardware attacks: The trusted path attacks described above are hardware attacks: They require the attacker to be physically present at the machine at some time to add the sniffing hardware. Intel TXT was not intended to prevent hardware attacks, and Grawrock lists several other hardware attacks that would succeed to varying degrees (see Chapter 17, [Grawrock, 2008]).

Hardware damage can cause data loss: TXT permits encrypting data on the hard drive using the encryption keys bound to a specific machine's hardware. This prevents theives from extracting data by simply pulling the hard drive from a machine. However, if that machine's TXT hardware were destroyed (by power surge or other accident), any TXT-encrypted data would be permanently lost to the system owners as well.

TCB still requires trusting others: When launching a static or dynamic RTM, TXT measures the current software's harddisk image against a previously "trusted" image. However, the "trusted" image is likely simply the image received from the manufacturer. Marking that image as "trusted" implies that it has no exploitable bugs, no intentional back doors, and that it does not leak information (intentionally or not) over any network connections. For many professional applications, many of these statements are known to be false. What TXT provides is the knowledge that the software has not changed from the last approved state, and that software packages launched in separate dynamic RTMs are reasonably secured from the rest of the system and each other.

Preventing user control: Finally, many free-software advocates see TXT and the larger Trusted Computing movement as a threat to the end user's control over his own system. Many of the features described herein that prevent malware from altering a computer's state could also prevent users from controlling their own system. For instance remote attestation and sealed storage could be used to ensure that a legally purchased DRM-protected song could only be played on one computer by one program.[5] Although we don't specifically know how decreased user control could affect a critical system calculating against critical data, we believe some problems could arise from these changes.

[5]More examples can be found by internet searching for "Treacherous Computing" or "Richard Stallman".

8 Discussion

TXT is extremely new: TXT is a new technology that could drastically alter the way computing is performed: It permits measuring that software is unchanged from a previous "trusted" version. It can protect software from several common software-based attacks. It can also decrease the user's control over his own data. Its newness leaves open many of the details of how it will affect computing.

This newness leads to several issues as well: Current TXT-enabling software systems are hard to use and don't supply all TXT features. Precise documentation is limited.

However, TXT's newness leaves open a valuable window: If we were to dedicate resources to researching and using TXT now – before it is widely accepted – we could shape the details of how TXT affects computing. Opening a relationship with Intel could provide answers to many of the questions that current documentation does not answer.

Merging solutions: We believe properly used, fully mature TXT-based systems could reduce a critical system's vulnerability – even in an otherwise untrusted environment. Critical data could be protected before and after program execution through hard drive encryption. During program execution, the program could be launched into a trusted, partitioned dynamic RTM that would protect user input, output, and program computation and data from loss or corruption from other system components. However, the program itself could still be corrupted due to errors in its own programming (e.g., buffer overflow attacks).

To best protect a critical system from attack, we recommend a holistic approach in which TXT is only a component. Other than employing other well-known techniques (such as Anti-Tamper), we believe TXT's newness provides another opportunity. Intel and AMD currently provide disparate implemenations of the TCG's recommended Trusted Computing Platform. If a system were built for each of Intel's and AMD's solutions, an interesting system could be built by combining both. If Intel's and AMD's systems were run on the same problem concurrently – with both being measured on startup – corruption in one system could be identified at runtime as it would differ from the other system. In order for an attack to succeed and not be quickly detected, the attack would have to succeed against two different technologies at the same time – a much harder feat.

Supply-chain-based hardware attacks: TXT prevents many common software attacks. TXT's stated goal is to force all viable attacks to be hardware attacks [Grawrock, 2008]. Intel asserts that such attacks require physical presence. While altering hardware configurations of a functioning system requires physical presence, this alteration could happen before the system arrives at its final destination [Adee, 2008, Derene and Pappalardo, 2009, Markoff, 2009]. The effect of a hardware-based attack would depend on which component is compromised: If a peripheral (e.g., network card) were compromised, any data which passed through that device could be compromised; if a TXT-critical component (e.g., the TPM) were compromised, all TXT-provided guarantees could be nullified. Detecting compromised hardware is a very complex problem – and an active research area [Adee, 2008, Cooper, 2009].

TXT's help to the problem: This document focused on Intel's Trusted Execution Technology and how it could help prevent sensitive data loss or corruption on a system built of commercial components on an untrusted network. In short, a fully realized TXT-based system can decrease

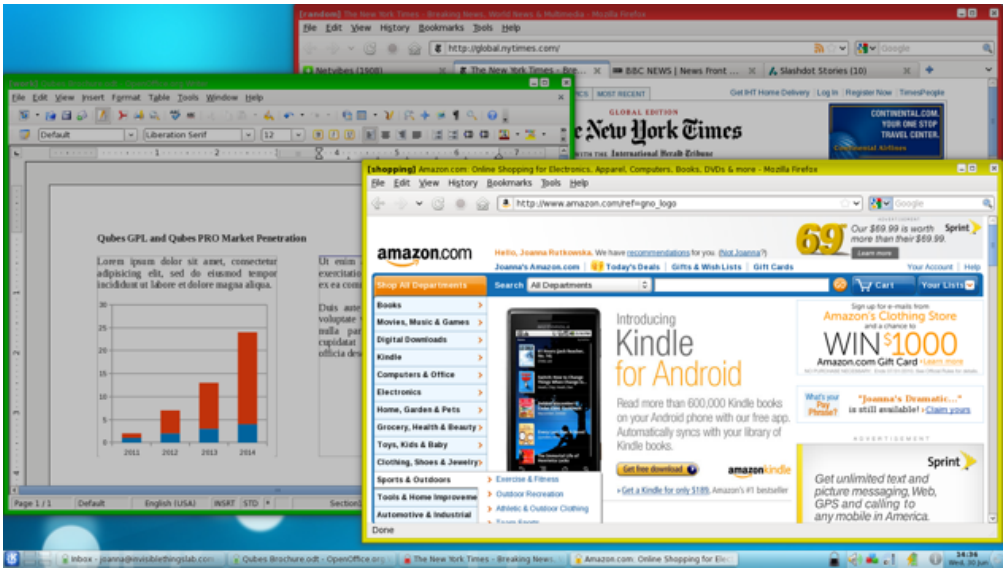


Figure 1: **Security levels in Qubes OS**: Qubes OS creates VMs with different security levels. Red is least trusted: Browse any website, but don't enter any secure data. Yellow is somewhat trusted: Browse only trusted websites, and enter limited secure data. Green is most trusted: Avoid nearly all web use, enter any secure data.

the portion of the system that must be trusted, but does not create a fully secure system from untrusted components. Through partitioning and protected execution, TXT can protect the critical components from other, untrusted components. Through pre-launch measurements, TXT can ensure that the critical components have not changed from a previously identified "trusted" state. Through hardware-hidden keys, TXT can encrypt data such that external entities can only decrypt them through brute force.

These three features – partitioning, measurements, and hidden encryption keys – could considerably improve security of critical computing resources. If the critical-computation program does not require access to the network, or a storage device (e.g., USB memory stick), the program can be loaded into a VM that lacks the unnecessary devices. This would prevent a program that does not require network access from leaking data over the network, and prevents network-based attack. Thus, executing software can be made more secure. Qubes OS provides some of these features (Figure 1).

Qubes OS demonstrates another benefit of partitioning: disposable VMs (Figure 2). Malware is frequently unwittingly downloaded and executed as part of an otherwise useful attachment. Disposable, unprivileged VMs can be quickly generated to permit reading common file formats (Adobe PDF, Microsoft PowerPoint or Word Documents, etc.), but contain any potential malware in a no-network, no-storage VM that cannot change any machine state (including disposable VM settings) or send any external messages. After the document is read, the VM is closed, and all contents are permanently deleted. Thus, a common malware attack vector is closed.

With measured launches of key programs, an engineer can know that his critical programs have not been altered by malware since he identified them as trusted. Although this still requires creating or verifying software as trusted (both very difficult tasks), it protects such trusted software. Without measured launches, the trusted software may be invisibly corrupted after its first use; with measured launches, any corrupted once-trusted software is identified during its first post-corruption launch.

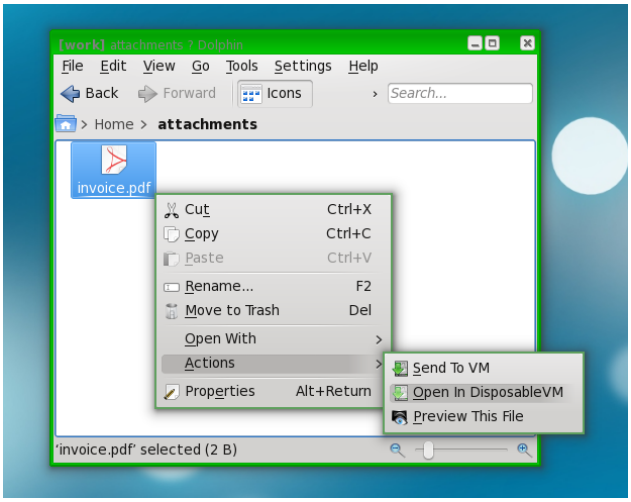


Figure 2: **Disposable VMs in Qubes OS**: Qubes OS allows creating disposable VMs that allow viewing untrusted documents in an environment that any hidden malware cannot affect external state, and all contents are deleted after closing.

Thus, successful measured launches guarantee pristine, uncorrupted codes.

Hardware-protected, software-invisible encryption keys prevent attackers from obtaining usable data from a hard drive separated from the encryption-key containing hardware device. Thus, stealing an encrypted file over the network will provide no useful data: The necessary decryption key is only available on the machine that generated the file. Furthermore, it is impossible to invisibly alter a hardware-encrypted file: Any altered portion will not match the encrypted portions, and will be quickly identified after hardware-based decryption. Thus, hardware-protected encryption keys protect critical data.

While TXT provides many protections, privilege-requiring software must still be trustworthy. TXT cannot protect data if the critical-calculation-performing software requires network access and has an intentional backdoor, or can be hacked during runtime.

9 Future Developments Watchlist

Since it's a new technology, there are many exciting developments that could improve or change how well Trusted Computing technologies are accepted.

First, both Intel and AMD need to complete and debug their trusted computing solutions. AMD's SVM does not supply all features required by the Trusted Computing standard. As previously mentioned, Intel has announced an STM unit that should fix ITL's successful SMM attacks. However, the design has not yet been disclosed, and no hardware with an STM has been announced.

Second, operating systems need to further their support for Trusted Computing technologies. Until Windows and Mac OS X provide support for static and dynamic RTMs, Trusted Computing will continue to be a niche technology: There are so many applications that could benefit from a secured execution environment that can only run on Windows or OS X. Qubes OS's alpha versions look promising. However, until Qubes Pro is released with Windows and OS X VMs, it will also lack critical applications.

Sandia need not idly wait for these events to occur. Sandia could seek industry partnerships to create a higher security system with the tool-required features. From external evidence, it seems that Qubes OS was developed using Xen, Fedora, and tboot by two or three expert engineers over the course of less than one year. We believe that if Sandia dedicated ten operating system and computer security engineers for approximately one year, they could develop an OS similar to Qubes – but with more of the necessary features (better OS support, graphics drivers, etc.).

10 Limitations on Research Performed

Several factors limited the scope and type of the research performed in creating this report:

- **Time**: Work on this project began less than eleven weeks before the final report was due.
- **Money**: This project's funding provided for only one testing machine, and less than 50% time for two technical staff.
- **Existing Knowledge Base**: The trusted computing technologies discussed herein are new. This leads to few technical staff with knowledge of them, few available good documentation sources, and few existing supporting technologies.

Due to these limitations, research was precisely focused. We focused principally on Intel®Trusted Execution Technology (TXT): reading Intel documentation, reading reports from non-Intel technologists, purchasing a TXT-capable system, and testing the few existing TXT-supporting technologies (tboot and Qubes OS). Section 6 presents other attacks we think should be attempted. To present an overview of other proprosed critical system safeguards, we read technical reports and discussed with knowledgable technical staff about various other systems (reported in Section 3). We did not test or personally experience any of these systems.

References

[Adee, 2008] Adee, S. (2008). The hunt for the kill switch. http://spectrum.ieee.org/semiconductors/design/the-hunt-for-the-kill-switch.

[Atallah et al., 2004] Atallah, M. J., Bryant, E. D., and Stytz, M. R. (2004). A survey of anti-tamper technologies. http://www.stsc.hill.af.mil/crosstalk/2004/11/0411Atallah.html.

[Cihula, 2007] Cihula, J. (2007). Trusted boot: Verifying the xen launch. http://xen.org/files/xensummit_fall07/23_JosephCihula.pdf.

[Cooper, 2009] Cooper, N. (2009). Engineer to develop methods to detect tampering in computer chips. http://advance.uconn.edu/2009/090521/09052106.htm.

[Department of Defense, 2010] Department of Defense (2010). DoD anti-tamper. http://at.dod.mil/.

[Derene and Pappalardo, 2009] Derene, G. and Pappalardo, J. (2009). Counterfeit chips raise big hacking, terror threats, experts say. http://www.popularmechanics.com/technology/gadgets/news/4253628.

[Embleton et al., 2008] Embleton, S., Sparks, S., and Zou, C. (2008). SMM rootkits: A new breed of OS independent malware. In *Proceedings of the 4th international conference on Security and Privacy in Communication Networks*.

[Grawrock, 2008] Grawrock, D. (2008). *Dynamics of a Trusted Platform: A building block approach*. Intel Press.

[Intel, 2008] Intel (2008). Intel desktop and intel mobile boards privilege escalation. http://security-center.intel.com/advisory.aspx?intelid=INTEL-SA-00017&languageid=en-fr.

[Intel, 2009] Intel (2009). SINIT misconfiguration allows for privilege escalation. http://security-center.intel.com/advisory.aspx?intelid=INTEL-SA-00021&languageid=en-fr.

[Intel, 2010] Intel (2010). Trusted boot. http://sourceforge.net/projects/tboot/.

[Invisible Things Lab, 2009a] Invisible Things Lab (2009a). Press release – another way to circumvent Intel trusted execution technology. http://www.invisiblethingslab.com/press/itl-press-2009-04.pdf.

[Invisible Things Lab, 2009b] Invisible Things Lab (2009b). Press release – attacking Intel trusted execution technology. http://www.invisiblethingslab.com/press/itl-press-2009-02.pdf.

[Markoff, 2009] Markoff, J. (2009). Old trick threatens the newest weapons. http://www.nytimes.com/2009/10/27/science/27trojan.html.

[McCune et al., 2008] McCune, J. M., Parno, B., Perrig, A., Reiter, M. K., and Isozaki, H. (2008). Flicker: An execution infrastructure for TCB minimization. In *Proceedings of EuroSys '08*, pages 315–328.

[Microsoft, 2010] Microsoft (2010). Bitlocker. http://www.microsoft.com/windows/windows-7/features/bitlocker.aspx.

[Rutkowska, 2010] Rutkowska, J. (2010). Qubes, Qubes Pro, and the future... http://theinvisiblethings.blogspot.com/2010/09/qubes-qubes-pro-and-future.html.

[Rutkowska and Wojtczuk, 2010] Rutkowska, J. and Wojtczuk, R. (2010). Qubes OS architecture – version 0.3. http://qubes-os.org/files/doc/arch-spec-0.3.pdf.

[Shinagawa et al., 2009] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y., and Kato, K. (2009). BitVisor: A thin hypervisor for enforcing I/O device security. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2009)*, pages 121–130.

[Thompson, 1984] Thompson, K. (1984). Reflections on trusting trust. *Communications of the ACM*, 27(8):761–763.

[Trusted Computing Group, 2010] Trusted Computing Group (2010). Trusted computing group – developers – glossary. http://www.trustedcomputinggroup.org/developers/glossary/.

["University of Michigan", 2010] "University of Michigan" (2010). Intel trusted execution technology slides. http://www.slideshare.net/slkevin/txt-introduction. The slides do not list an author. The slideshare username "University of Michigan" uploaded the slides.

[Wickstrom et al., 2004a] Wickstrom, G. L., Davis, J., Morrison, S. E., Roach, S., and Winter, V. L. (2004a). The SSP: An example of high-assurance systems engineering. In *Proceedings of the Eigth IEEE International Symposium on High Assurance System Engineering (HASE'04)*.

[Wickstrom et al., 2004b] Wickstrom, G. L., Winter, V. L., Beranek, J., Roach, S., and Fraij, F. (2004b). An abstract class loader for the SSP and its implementation in TL. Unlimited Release 3225, Sandia.

[Wikipedia, 2010a] Wikipedia (2010a). Bitlocker drive encryption. http://en.wikipedia.org/wiki/BitLocker_Drive_Encryption.

[Wikipedia, 2010b] Wikipedia (2010b). Trusted computing. http://en.wikipedia.org/wiki/Trusted_Computing.

[Wikipedia, 2010c] Wikipedia (2010c). Trusted computing base. http://en.wikipedia.org/wiki/Trusted_computing_base.

[Winter et al., 2005] Winter, V. L., Beranek, J., Fraij, F., Roach, S., and Wickstrom, G. L. (2005). A transformational overview of the core functionality of an abstract class loader for the SSP. In *Tenth IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2005)*.

[Wojtczuk and Rutkowska, 2009] Wojtczuk, R. and Rutkowska, J. (2009). Attacking Intel trusted execution technology. In *Black Hat*.

[Wojtczuk et al., 2009] Wojtczuk, R., Rutkowska, J., and Tereshkin, A. (2009). Another way to circumvent Intel trusted execution technology. In *Black Hat*.

DISTRIBUTION

1	MS-0620	Daniel R. Cantu	5634	
1	MS-0620	Berlinda B. Eras	5644	
1	MS-0620	Rebecca Darnell Horton	5640	
1	MS-0620	Marian Chrisma Jackson	5625	
1	MS-0620	Edward J. Nava	5620	
1	MS-0620	Frederick W. Sexton	5622	
1	MS-0620	Mark Terhune	5630	
1	MS-0621	Kim M. Denton-Hill	5638	
1	MS-0621	Roxana M. Jansma	5631	
1	MS-0621	Leann Adams Miller	5636	
1	MS-0621	Dallas J. Wiener	5632	
1	MS-0627	Steven Rinaldi	5643	
1	MS-0671	Jennifer M. Depoy	5628	
1	MS-0671	Frederick Mitch McCrory	5627	
1	MS-0672	Jeffrey J. Danneels	5621	
1	MS-0672	Han Wei Lin	5629	
1	MS-0899	Technical Library	9536	(electronic copy)
1	MS-1027	Bernard P. Clifford	5633	
1	MS-1027	Curtis M. Johnson	5635	
2	MS-1069	Marion W. Scott	2300	
1	MS-1073	Benjamin K. Cook	5641	
1	MS-1221	Cindy L. Longenbaugh	5642	
1	MS-1221	James S. Peery	5600	
1	MS-1397	Bridget L. Rogers	5626	

Sandia National Laboratories