Trusted Computing Challenges

Leendert van Doorn
Sr. Fellow
Security at Work

November, 2007
Outline

• Trusted Computing Technology
• Attestation Usage Scenarios
  • Static Root of Trust
    *IBM’s Integrity Measurement Architecture (IMA)*
  • Dynamic Root of Trust
    *Microsoft’s Next Generation Secure Computing Base (NGSCB)*
  • Open Challenges
• Root of all Evil: Hierarchical Trust Dependencies
  • Thought experiment
• Summary

*The views and opinions expressed in this talk are strictly those of the author.*
Trusted Computing Technology
Trusted Platform Module (TPM)

- Trusted Computing in today’s world is largely synonymous with a use that involves the Trusted Platform Module (TPM).

- TPM is a passive storage device that has some interesting properties:
  - You cannot remove data once you’ve written it to the TPM.
  - You can retrieve an aggregate of the data from the TPM that is signed by that TPM’s unique key.
  - The TPM provides sealed storage.
  - Storage root key protection.

Winbond
Infineon
Atmel
TPM – Trusted Platform Module

- Non-Volatile Storage
- Platform Configuration Register (PCR)
- Attestation Identity Key (AIK)
- Endorsement Key (EK)
- Random Number Generator
- SHA-1 Engine
- Key Generation
- RSA Engine
- Opt-In
- Exec Engine

LPC bus

I/O

Trusted Platform Module (TPM)

Tamper-Protected Packaging
Basic TPM properties

• Program Configuration Registers
  – Extend \((\text{PCR}_n, H)\): \(\text{PCR}_n = f (\text{PCR}_n || H)\), where \(f\) is a SHA-1
  – Quote \((\text{PCR}_n, \text{AIK}_i)\): return PCR\(_n\) encrypted under AIK\(_i\)

• Sealed Storage
  – Encrypt data under a specific PCR value
  – Can only be release if PCR has that value

• Non-Volatile Storage
  – Storage Root Key
Attest To The Integrity of a System

- Prove to a remote party what software/configuration is running on the target system

- Good applications
  - Bank only allows access with up to date software patches
  - Network Admission Control
  - Games

- Digital Right Management
Other than for Windows Vista™ Bitlocker, the commercial software impact of TCG technology has been negligible so far.
TCG Software Enablement Challenges

- Current applications use the TPM mostly as a smartcard
  - PKCS11 interface
  - Store browser certificates
  - Encrypted disk, password store, etc.

- TCG attestation concepts do not scale

- TCG technology needs an isolation mechanism that doesn’t exist in today’s software
  - All commercial software is monolithic
  - BitLocker & NAP/TNC only attest the lowest layers
Static Root of Trust

*IBM’s Integrity Measurement Architecture (IMA)*
TCG Static Root of Trust

“Measure before Load”

BIOS

ROT

CRTM

POST

Bootloader

GRUB

Stage1 (MBR)

GRUB Stage1.5

GRUB Stage2

PCR01-07

PCR04-05

PCR08

PCR10

Operating System

Linux Kernel

/bin/ls

/usr/sbin/httpd

November, 2007 Trusted Computing Challenges
TPM-Based Integrity Measurement Architecture

• Achievement of the Integrity Measurement Architecture
  – Extend TPM-based attestation into the system runtime

• IMA-Guarantees
  – Non-intrusive (not changing system behavior)
  – Load-guarantees for code loaded into the system
  – Detects systems cheating with the measurement list

• Goals
  – Negligible overhead on attested system
  – Usable
Example: Web Server

- **Executables (Program & Libraries)**
  - apachectl, httpd, java, ..
  - mod_ssl.so, mod_auth.so, mod_cgi.so,
  - libc-2.3.2.so libjvm.so, libjava.so

- **Configuration Files**
  - httpd.conf, html-pages
  - httpd-startup, catalina.sh, servlet.jar
Integrity Measurement Architecture (IMA)

Attesting System

Measurements

Verifying System

Deduce System Properties

Inferred System

TPM-Signed PCR Integrity Value

Analysis

Known Fingerprints

(1) Measurement    (2) Attestation    (3) Verification

November, 2007    Trusted Computing Challenges
Phase I: IMA Measurement Process

Measurement list aggregation:

- **Compute** 160bit-SHA1 over the contents of the data (measurement)
- **Adjust** Protected hw Platform Configuration Register (PCR) to maintain measurement list integrity value
- **Add** measurement to ordered measurement list

→ **Executable content is recorded before it impacts the system**

→ **That is, before it can corrupt the system**
Phase I: IMA Measurements Induced by the Kernel

Include a description of the diagram:
- /bin/bash
- Execve (*file)
- Integrity Value
- Measurement List (Kernel-held)
- SHA1
- Memory Map
- Schedule

Linux Security Module

Traditional execution path
Phase II: Attestation Protocol

1. Create unpredictable Nonce

2. Nonce (160bit)

3. \(\text{Sig}\{\text{Nonce, PCR}\}_\text{TPM}\) \(\rightarrow\) Measurement List

4. Check TPM Signature & Nonce

5. Validate Measurement List against PCR Integrity Value

6. Validate individual Measurements using Fingerprint DB
Phase II: Example from our RSA Demonstration

1. Submit Request and Nonce

2. Receive: \( \text{Sig(Nonce, PCR)} \) Measurement List

3. Check: Signature Nonce

4. Validate: PCR Value

5. Evaluate: Individual Measurements

6. Infer: High-Level System Properties

November, 2007  Trusted Computing Challenges
### Phase III: Verification Example

<table>
<thead>
<tr>
<th>CLIENT Measurement List</th>
<th>Fingerprint DB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>#000: BC55F0AFE013C3402F00E0AA11EE6CFAA2B4D2AB</td>
<td>boot_aggregate (bios + grub stages)</td>
</tr>
<tr>
<td>#001: A8A865C7203F2565DDEB511480B0A2289F7D035B</td>
<td>grub.conf (boot configuration)</td>
</tr>
<tr>
<td>#002: 1238AD50C652C88D139EA2E9987D06A99A2A22D1</td>
<td>vmlinuz-2.6.5-bk2-lsmtcg</td>
</tr>
<tr>
<td>#003: 84ABD2960414CA4A448E0D2C9364B4E1725BDA4F</td>
<td>init (first process)</td>
</tr>
<tr>
<td>#004: 9ECF02F90A2EE2080D4946005DE47968C8A1BE3D</td>
<td>ld-2.3.2.so (dynamic linker)</td>
</tr>
<tr>
<td>#110: F969BD9D27C2CC16BC668374A9FBA9D35B3E1AA2</td>
<td>syslogd</td>
</tr>
</tbody>
</table>

**a) Earlier: THE GOOD CASE**

...  

#110: F969BD9D27C2CC16BC668374A9FBA9D35B3E1AA2 | syslogd  

...  

#525: 4CA3918834E48694187F5A4DAB4EECD540AA8EA2 | syslogd (ROOTKIT !)

**b) Later: The Linux Root Kit 5 Compromise!**
IMA Overhead

- Attested System
  - Implementation: ~ 5000 LOC (LSM kernel module)
  - About 400-600 measurements for Fedora C2, Apache, Jakarta Tomcat, etc.
  - Measurement Overhead

<table>
<thead>
<tr>
<th></th>
<th>Kernel</th>
<th>Application</th>
<th>Likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean Hit</td>
<td>~ 0.1 μs</td>
<td>~ 5 μs</td>
<td>&gt;&gt; 99 %</td>
</tr>
<tr>
<td>New (TPM)</td>
<td>~ 5 ms</td>
<td>~ 5ms</td>
<td>&lt;&lt; 1 %</td>
</tr>
<tr>
<td>Measurement</td>
<td>+ SHA1 (~80MB/s)</td>
<td>+ SHA1 (~80MB/s)</td>
<td></td>
</tr>
</tbody>
</table>

- Attestation service
  - Known Fingerprint DB ~ 20 000 Fingerprints (RedHat 9.0, Fedora, ES3)
  - Attestation: 1-2 second “latency” (unoptimized demonstration)
IMA Challenges

- IMA is an example of pushing the static root of trust measurement concepts to the extreme
  - In doing so, it needs to measure **everything**!
  - Since there are no real isolation boundaries, anything is a potential threat
    - Shell, perl, python scripts, emacs macros, excel spreadsheets, word files, etc.
    - You also need to measure configuration files, environment variables, etc. All things that influence execution behavior

- IMA gives you load-time guarantees, not runtime

- IMA measurements are potentially unbounded, especially on multi-user and/or development systems
  - IMA can work for application specific servers that have a static configuration
Dynamic Root of Trust

Microsoft’s Next Generation Secure Computing Base (NGSCB)
Secure Kernel Initialization

- A lot can happen before the Kernel gets to run
  - BIOS, extension BIOSes, bootstrap loaders, etc.
  - Do we really have to trust this?

- Dynamic Root of Trust is used to alleviate this
  - It enables a secure **late launch** of the operating system
  - Unknown state → known secure state

- AMD has been shipping this capability since 2006 as part of its virtualization extensions

- Intel has been shipping this since 2007 as part of their LaGrande Technology
New AMD64 Instruction: SKINIT

- **Objective**: Known code executing in known (quiet) environment

- **Result**: Forms the basis for establishing trust in the platform environment

- SKINIT instruction behavior:
  - Performs an “INIT” of the CPU
  - Resets the architecturally visible CPU state to known values
  - Removes Microcode patches
  - Activates special DMA protection over the Secure Loader (SL) code
  - Multi-Processor “Safety Check”
  - SL code is copied to the TPM using SKINIT-only special cycles
  - Unconditional Jump to entry point in the SL code
Secure Hypervisor

- Combine this with hardware virtualization capabilities (isolation) and that enables a hypervisor-based security solution

- AMD Virtualization™ provides strong Memory/IO isolation
  - Guest to Hypervisor
  - Guest to Guest

- Microsoft’s NGSCB uses DRTM and virtualization
Microsoft’s 2003 NGSCB Version

Source: Microsoft’s WinHEC 2004 presentation
Microsoft’s 2004 NGSCB Version

Source: Microsoft’s WinHEC 2004 presentation

• Great device diversity
  • Thousands of drivers
  • MLOC

• Little device diversity
  • Only a few drivers
  • KLOC
Attestation in NGSCB

• The Nexus uses late launch to securely bootstrap itself from Windows®

• The use of isolation allows you to scale attestation statements
  – For example, in NGSCB 2003 you have to attest to the
    – NAL, Nexus, NCA RT, TSP, TUE and a specific NCA
    – or, if you trust the Nexus you only have to attest that and believe its answers
  – This is much smaller (around 100KLOC) than a whole windows kernel

• Isolation also helps SRTM in reducing the attestation requirements
DRTM Challenges

• Microsoft shelved NGSCB in 2004
  – There is currently no commercial consumer of DRTM technologies

• Bringing a system from an unknown state into a known secure mode is non-trivial
  – Microcode patches
  – System Management Mode (SMM)

• How much do you attest to versus trust your security kernel?
Open Challenges
Attestation Infrastructure

Trusted 3rd party

- SHA1(Boot Process)
- SHA1(Kernel)
- SHA1(Kernel Modules)
- SHA1(Program)
- SHA1(Libraries)
- SHA1(Configurations)
- SHA1(Structured data)
- ...

Attested System

Verifier

Attest

AIKi⁻¹

OK | BAD
Attestation Infrastructure (cont’d)

• This is PKI’s scalability problem squared
  – PKI’s don’t scale world-wide for socio-economic reasons
  – Attestation measurement lists don’t scale
    Too many elements
    Changing too rapidly

• It may work within an enterprise
  – PKI’s are directly related to HR data
  – They typically have a limited number of approved software configurations that are centrally managed and distributed

• We need more research into different attestation models
## TPM 1.2 Features Used In Current Commercial Software

### Used
- SRTM
- Sealing
- Secure Storage
- Random Number

### Not Used Yet
- *Attestation/quote*
- *EK / AIK*
- DAA
- DRTM
- Locality
- Monotonic Counters
- Transports

---

*Microsoft, Juniper, TCG recently announce NAP, TNC interoperability*
Root of All Evil: Hierarchical Trust Dependencies
Hierarchical Trust Dependencies

- Hardware
  - BIOS/SMM
  - Operating System
    - Middleware
      - glibc, ...
    - JVM
    - dbms
  - browser
  - app
  - japp

- 1 MLOC (glibc)
- 8 MLOC (linux 2.6.22.6)
- 2 MLOC (popular BIOS)
Hierarchical Trust Dependencies (cont’d)

• Just a simple C program depends on about 11 MLOC
  – According to Wietse Venema’s metrics that is 11K security bugs (1 security bug per 1 KLOC)
  – This assumes a simple linear dependency, but …

• Other programs may have access to the kernel’s address space and tamper with it
  – /dev/kmem
  – System call argument checking errors
  – So the actual dependency graph is much bigger
Thought Experiment

• Some Webservices architects assume that the platform starts at the application layer (JVM) and everything else on the system is secure.

• What would it take to build an architecture that guarantees that integrity/confidentiality guarantees start at the application layer?

• *Thought experiment:* Breaking the hierarchical trust dependencies
  - This is a hard problem as Adrian Perrig (CMU) and I found out, especially when you want to stay backward compatible, so let’s ignore that.
  - Following are three guiding principles
Application Resource Ownership

- Applications own their resources and explicitly grant access to their resources.

- The kernel sets up an application context:
  - Once the context is activated the kernel loses access to the content of the pages.
  - For example, the kernel can reclaim pages but they will always be filled with zeroes.

- Kernel manages thread of control, but thread state is saved in a application owned page (hardware does the saving).

- An application can explicitly grant access to its resources (for example set up a mailbox with the kernel).

- These principles guarantee integrity and confidentiality, they do not prevent denial of service or side channel analysis.

- It breaks existing kernels.
Application Authentication

- Applications need to prove to themselves and others that they are running in this environment
  - Executable content can be encrypted with some hardware accessible secret to prevent tampering
  - Or use DRTM / sealed storage like mechanism to attest itself / unlock application data
  - Attestation obviously scales much better with just an application
Application Trusted I/O

- I/O is complicated because of its device/interface diversity
  - Why not introduce a common I/O abstraction like mainframe channels?
  - Applications interact directly with channels through channel “programs” (control blocks), no kernel is necessary

- The channels are end-to-end authenticated and interact directly with the hardware / other access points
Summary

- After 8 years the commercial impact of TCG technology has been negligible
  - It is used as a smartcard
  - The interesting use cases need more research
  - Fortunately, there is a vibrant and growing TC research community

- Attestation does not scale well
  - Isolation improves scaling, but only to enterprises with established PKI infrastructures
  - We need more research into different attestation models

- Call to arms: Should we change our architecture?
Acknowledgements

- IMA slides are taken, with permission, and/or paraphrased from Reiner Sailer (IBM)
- NGSCB pictures are taken, with permission, from Peter Biddle’s (Microsoft) WinHec 2004 presentation
- The architectural ideas in the last section are based on discussions with Adrian Perrig (CMU)
Trademark Attribution

AMD, the AMD Arrow logo, AMD Virtualization, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Windows Vista and Windows are trademarks of Microsoft Corporation in the United States and other jurisdictions. Linux is a registered trademark of Linus Torvalds. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

©2006-2007 Advanced Micro Devices, Inc. All rights reserved.