CE 874 - Secure Software Systems

Secure Architecture III

Mehdi Kharrazi
Department of Computer Engineering
Sharif University of Technology

Acknowledgments: Some of the slides are fully or partially obtained from other sources. A reference is noted on the bottom of each slide, when the content is fully obtained from another source. Otherwise a full list of references is provided on the last slide.
Secure Architecture

- How to come up with a secure architecture?
- What design principals is should be followed?
- What are the available mechanisms?
- How do you trust the code getting executed?

xkcd.com
Bootstrapping Trust in Commodity Computers,
A Travel Story

Your 30 day trial of KeyLogger Express has expired.

Please enter your registration code:

```
[ ]-[ ]-[ ]-[ ]
```

or visit our website:


to purchase a license

[OK] [Cancel]
Trust is Critical

Will I regret having done this?

[Image: shopping cart, computer, medical symbol]
Does program P compute F?
Is F what the programmer intended?

Bootstrapping Trust
What F will this machine compute?

X_{Other} → Y_{Other}
X_{Alice} → Y_{Alice}

[Parno’10]
Bootstrapping Trust is **Hard!**

- Challenges:
  - Hardware assurance
  - Ephemeral software
  - User Interaction

Safe? Yes!

---

Spring 1398

Ce 874 - Secure Architecture III

[Parno’10]
Bootstrapping Trust is **Hard**!

- Challenges:
  - Hardware assurance
  - Ephemeral software
  - User Interaction

Yes!

Yes!

[Parino’10]
In the paper...

- Bootstrapping foundations
- Transmitting bootstrap data
- Interpretation
- Validation

- Applications

- Human factors
- Limitations

- Future directions

- ... and much more!

[Spring 1398] Ce 874 - Secure Architecture III

[Parno’10]
1) Establish Trust in Hardware

- Hardware is durable

Open Question: Can we do better? [Parno’10]
2) Establish Trust in Software

- Software is ephemeral
- We care about the software currently in control
- Many properties matter:
  - Proper control flow
  - Type safety
  - Correct information flow

Which property matters most?

[Parno’10]
A Simple Thought Experiment

- Imagine a perfect algorithm for analyzing control flow
  - Guarantees a program always follows intended control flow
  - Does this suffice to bootstrap trust? No!

We want code *identity*

[Reference: Parno’10]
What is Code Identity?

- An attempt to capture the behavior of a program
- Current state of the art is the collection of:
  - Program binary
  - Program libraries
  - Program configuration files
  - Initial inputs
- Often condensed into a hash of the above

\[ f \text{ Inputs to } f \]

\[ \text{Function } f \]
Code Identity as Trust Foundation

• From code identity, you may be able to infer:
  • Proper control flow
  • Type safety
  • Correct information flow
  ...
• Reverse is not true!
What Can Code Identity Do For You?

- Research applications
  - Secure the boot process
  - Count-limit objects
  - Improve security of network protocols
  - Thwart insider attacks
  - Protect passwords
  - Create a Trusted Third Party

- Commercial applications
  - Secure disk encryption (e.g., Bitlocker)
  - Improve network access control
  - Secure boot on mobile phones
  - Validate cloud computing platforms

[Parno’10]
Establishing Code Identity

\[ F \]

\( X_{Other} \)

\( Y_{Other} \)

\( X_{Alice} \)

\( Y_{Alice} \)

[Parno’10]
Establishing Code Identity

\[ f_1 \rightarrow f_2 \rightarrow \ldots \rightarrow f_N \]
Establishing Code Identity

### Root of Trust

- **Software 1**
- Software N-1
- Software N

### Chain of Trust
Trusted Boot: Recording Code Identity

Root of Trust

Software 1

Software N-1

Software N

SW 1

SW 2

SW N-1

SW N

[Parno’10]
Attestation: Conveying Records to an External Entity

Software 1 \(\text{random #}\) Software \(N-1\)

\[
\text{Sign}_{K_{\text{priv}}} (\text{random #})
\]

Controls \(K_{\text{priv}}\)

[Paro’10]
Interpreting Code Identity

**App 1...N**

**Drivers 1...N**

**OS**

**Bootloader**

**Option ROMs**

**BIOS**

**Traditional**

[Gasser et al. ‘89], [Sailer et al. ‘04]

**Policy Enforcement**

[Marchesini et al. ‘04], [Jaeger et al. ‘06]
Interpreting Code Identity

**Traditional**

[Gasser et al. ‘89], [Sailer et al. ‘04]

**Policy Enforcement**

[Marchesini et al. ‘04], [Jaeger et al. ‘06]

**Virtualization**

[England et al. ‘03], [Garfinkel et al. ‘03]
## Interpreting Code Identity

### Traditional
[Gasser et al. ‘89], [Sailer et al. ‘04]

### Policy Enforcement
[Marchesini et al. ‘04], [Jaeger et al. ’06]

### Virtualization
[England et al. ‘03], [Garfinkel et al. ‘03]

### Late Launch
[Kauer et al. ‘07], [Grawrock ‘08]

<table>
<thead>
<tr>
<th>BIOS</th>
<th>Bootloader</th>
<th>Option ROMs</th>
<th>Virtual Machine</th>
<th>VMM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Marchesini et al. ‘04], [Jaeger et al. ’06]

[England et al. ‘03], [Garfinkel et al. ‘03]

[Kauer et al. ‘07], [Grawrock ‘08]
Interpreting Code Identity

- **Traditional**
  - [Gasser et al. ‘89], [Sailer et al. ‘04]
- **Policy Enforcement**
  - [Marchesini et al. ‘04], [Jaeger et al. ‘06]
- **Virtualization**
  - [England et al. ‘03], [Garfinkel et al. ‘03]
- **Late Launch**
  - [Kauer et al. ‘07], [Grawrock ‘08]
- **Targeted Late Launch**
  - [McCune et al. ‘07]

**Notes:**
- **Late Launch**: Flicker
- **Attested**: Flicker

[Spring 1398] [Ce 874 - Secure Architecture III] [Parno’10]
Interpreting Code Identity

- App 1...N
- Drivers 1...N
- OS
- Bootloader
- Option ROMs
- BIOS

S
Flicker
Load-Time vs. Run-Time Properties

- Code identity provides load-time guarantees
- What about run time?
- Approach #1: Static transformation

[Run-Time Policy]

Code → Compiler → Code’

[Attested]

[Erlingsson et al. ‘06]

[Parno’10]
Load-Time vs Run-Time Properties

- Code identity provides load-time guarantees
- What about run time?

**Open Question:**
How can we get complete run-time properties?

[Attested]

[Code]

[Enforcer]

[Run Time]

[Load Time]

[Reference: Erlingsson et al. '06, Haldar et al. '04, Kil et al. '09, Parno'10]
Open Question:
What functionality do we need in hardware?

- [Weingart ‘87]
- [White et al. ‘91]
- [Yee ‘94]
- [Smith et al. ‘99]
- [ARM TrustZone ‘04]
- [TCG ‘04]
- [Zhuang et al. ‘04]
- [Chun et al. ‘07]
- [Levin et al. ‘09]
- [Spinellis et al. ‘00]
- [Seshadri et al. ‘05]
- [ARM TrustZone ‘04]
- [TCG ‘04]
- [Zhuang et al. ‘04]
- [Chun et al. ‘07]
- [Levin et al. ‘09]
- [Spinellis et al. ‘00]
- [Seshadri et al. ‘05]

...
Open Questions:

- How should $SW_1$ be communicated to Alice?
- What does Alice do with a failed attestation?
- How can Alice trust her device?
Conclusions

- **Code identity** is critical to bootstrapping trust
- Assorted hardware **roots of trust** available
- Many **open questions** remain!
A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping, Seunghun Han, Wook Shin, Jun-Hyeok Park, and HyoungChun Kim, Usenix Security 2018
Trusted Computing Group (TCG)

- Defines global industry specifications and standards
- Is supportive of a hardware root of trust
  - Trusted Platform Module (TPM) is the core technology
  - TCG technology has been applied to Unified Extensible Firmware Interface (UEFI)
Trusted Platform Module (TPM) (1)

- Is a tamper-resistant device
- Has own processor, RAM, ROM, and non-volatile RAM
  - It has own state separated from the system
- Provides cryptographic and accumulating measurements functions
  - Measurement values are accumulated to Platform Configuration Registers (PCR #0~#23)
Trusted Platform Module (TPM) (2)

• Is used to determine the trustworthiness of a system by investigating the values stored in PCRs
  • A local verification or remote attestation can be used
• Is used to limit access to secret data based on specific PCR values
  • “Seal” operation encrypts secret data with the PCRs of the TPM
  • “Unseal” operation can decrypt the sealed data only if the PCR values match the specific values

[Han’18]
Root of Trust for Measurement (RTM)

- Sends integrity-relevant information (measurements) to the TPM
  - TPM accumulates the measurements to a PCR with the previously stored value in the PCR
    - **Extend:** \( \text{PCR}_{\text{new}} = \text{Hash}(\text{PCR}_{\text{old}} \ || \ \text{Measurement}_{\text{new}}) \)
  - The CPU controlled by Core RTM (CRTM)
    - The CRTM is the first set of instructions when a new chain of trust is established
Static and Dynamic RTM (SRTM and DRTM)

- SRTM is started by static CRTM (S-CRTM) when the host platform starts at POWER-ON or RESTART
- DRTM is started by dynamic CRTM (D-CRTM) at runtime WITHOUT platform RESET
- They extend measurements (hashes) of components to PCRs BEFORE passing control to them
Static Root of Trust for Measurement

BIOS/UEFI firmware

S-CRTM → BIOS/UEFI Code → Bootloader → Kernel → User Applications

TPM

- : Extend a hash of next code to TPM
- : Execute next code

Power On/Restart

Dynamic Root of Trust for Measurement

(Intel Trusted Execution Technology)

Untrusted Code → D-CRTM (DCE) → tboot (DLME) → Bootloader Kernel → User Applications

TPM

DL Event: Dynamic Launch Event
DCE: DRTM Configuration Environment
DLME: Dynamically Launched Measured Environment
PCR Protection

- PCRs contain measurement results of a system.
- They **MUST NOT** be reset by disallowed operations.
  - Static PCRs (PCR #0~#15) can be reset only if the host resets.
  - Dynamic PCRs (PCR #17~#19) can be reset only if the host initializes the DRTM.
- If PCRs are reset by attackers, they can reproduce specific PCR values by replaying hashes.
  - They can steal the secret and deceive the local and remote verification.
- verification.
PCR protection mechanisms work properly until yesterday.
Assumptions and Threat Model

- The system measures boot components using the SRTM and DRTM
  - The measurement results stored in PCRs are verified by a remote verifier
  - The modifications of boot components are detected
- The attackers already gain a root privilege and try to compromise the whole system
  - They try to hide the breach and retain the root privilege
  - They cannot access the system circuit physically
  - They cannot flash the firmware with arbitrary code
Advanced Configuration and Power Interface (ACPI)

- Defines power states and hardware register sets
  - Global states
    - G0 (Working), G1 (Sleeping), G2 (Soft-off), G3 (Mechanical-off)
  - Sleeping states
    - S0 and S1: Working and Power on Suspend
    - S2: Same as S1, CPU is powered off
    - S3: Sleep, All devices are powered off except RAM
    - S4: Hibernation, All devices are powered off
ACPI Sleep Process with TPM

1. Request to save a state
2. Request to enter sleep
3. Sleep
4. Wake up
5. Request to restore a state
6. Resume OS
The Grey Area vulnerability (CVE-2018-6622)
The Grey Area Vulnerability (CVE-2018-6622)

What is the “corrective action”?

If the TPM receives Startup(STATE) that was not preceded by Shutdown(STATE), then there is no state to restore and the TPM will return TPM_RC_VALUE. The CRTM is expected to take corrective action to prevent malicious software from manipulating the PCR values such that they would misrepresent the state of the platform. The CRTM would abort the Startup(State) and restart with Startup(CLEAR).

This means “reset the TPM”

The startup behavior defined by this specification is different than TPM 1.2 with respect to Startup(STATE). A TPM 1.2 device will enter Failure Mode if no state is available when the TPM receives Startup(STATE). This is not the case in this specification. It is up to the CRTM to take corrective action if it the TPM returns TPM_RC_VALUE in response to Startup(STATE).

Trusted Platform Module Library Part1: Architecture
Exploit of the Grey Area Vulnerability

1. BIOS/UEFI
2. Compromised Software Stack
3. Compromised State
   - Sleep without saving the TPM state
4. Sleep
   - Wake up
5. Compromised Software Stack
6. Faked State

- Leaves normal hashes in event logs
- Extract and calculate the normal hashes
- Store the normal hashes in RAM
- Reset the TPM and replay the normal hashes

[Han’18]
## Evaluation – The Grey Area Vulnerability

<table>
<thead>
<tr>
<th>PC No.</th>
<th>Vendor</th>
<th>CPU (Intel)</th>
<th>PC and mainboard model</th>
<th>BIOS Ver. and release date</th>
<th>TPM Ver.</th>
<th>TPM vendor and firmware Ver.</th>
<th>SRTM attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Intel</td>
<td>Core i5-5300U</td>
<td>NUC5i5MYHE</td>
<td>MYBDEWi5v.86A, 2017.11.30</td>
<td>2.0</td>
<td>Infineon, 5.40</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>Intel</td>
<td>Core m5-6Y57</td>
<td>Compute Stick STK2mv64CC</td>
<td>CCSKLm5v.86A.0054, 2017.12.26</td>
<td>2.0</td>
<td>NTC, 1.3.0.1</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>Dell</td>
<td>Core i5-6500T</td>
<td>Optiplex 7040</td>
<td>1.8.1, 2018.01.09</td>
<td>2.0</td>
<td>NTC, 1.3.2.8</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>GIGABYTE</td>
<td>Core i7-6700</td>
<td>Q170M-MK</td>
<td>F23c, 2018.01.11</td>
<td>2.0</td>
<td>Infineon, 5.51</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>GIGABYTE</td>
<td>Core i7-6700</td>
<td>H170-D3HP</td>
<td>F20e, 2018.01.10</td>
<td>2.0</td>
<td>Infineon, 5.61</td>
<td>Y</td>
</tr>
<tr>
<td>6</td>
<td>ASUS</td>
<td>Core i7-6700</td>
<td>Q170M-C</td>
<td>3601, 2017.12.12</td>
<td>2.0</td>
<td>Infineon, 5.51</td>
<td>Y</td>
</tr>
<tr>
<td>7</td>
<td>Lenovo</td>
<td>Core i7-6600U</td>
<td>X1 Carbon 4th Generation</td>
<td>N1FET59W (1.33), 2017.12.19</td>
<td>1.2</td>
<td>Infineon, 6.40</td>
<td>N</td>
</tr>
<tr>
<td>8</td>
<td>Lenovo</td>
<td>Core i5-4570T</td>
<td>ThinkCentre m93p</td>
<td>F8KTCPA, 2017.12.29</td>
<td>1.2</td>
<td>STMicroelectronics, 13.12</td>
<td>N</td>
</tr>
<tr>
<td>9</td>
<td>Dell</td>
<td>Core i5-6500T</td>
<td>Optiplex 7040</td>
<td>1.8.1, 2018.01.09</td>
<td>1.2</td>
<td>NTC, 5.81.2.1</td>
<td>N</td>
</tr>
<tr>
<td>10</td>
<td>HP</td>
<td>Xeon E5-2690 v4</td>
<td>z840</td>
<td>M60 v02.38, 2017.11.08</td>
<td>1.2</td>
<td>Infineon, 4.43</td>
<td>N</td>
</tr>
<tr>
<td>11</td>
<td>GIGABYTE</td>
<td>Core i7-6700</td>
<td>H170-D3HP</td>
<td>F20e, 2018.01.10</td>
<td>1.2</td>
<td>Infineon, 3.19</td>
<td>N</td>
</tr>
</tbody>
</table>
Countermeasures – The Grey Area Vulnerability

• **Disable S3 sleeping state** option in BIOS menu
  • Brutal, but simple and effective
• Revise TPM 2.0 specification to **enter failure mode** if there is no state to restore
• Revise TPM 2.0 specification to **define “corrective action” in detail**
  • A long time to revise and apply to the TPM or BIOS/UEFI firmware, but fundamental solutions
Countermeasures – The Lost Pointer Vulnerability

- **Apply our patch to tboot**
  - https://sourceforge.net/p/tboot/code/ci/521c58e51eb5be105a29983742850e72c44ed80e/

- **Update tboot to the latest version**
Conclusion

- Two vulnerabilities that can subvert the TPM using S3 sleeping state were found
  - The Grey Area Vulnerability: CVE-2018-6622
  - The Lost Pointer Vulnerability: CVE-2017-16837
- Attackers can deceive the local and remote verification with the vulnerabilities
  - They also can unseal the seal secret and steal it
- We have contacted manufacturers and contributed a patch to tboot project to solve the vulnerabilities
Acknowledgments/References


• [Han’18] A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping, Seunghun Han, Wook Shin, Jun-Hyeok Park, and HyoungChun Kim, Slides, Usenix Security 2018