“TRUSTED” COMPUTING

DISTRIBUTED OPERATING SYSTEMS

HERMANN HÄRTIG, SUMMER 2018
Lecture Goals

Understand principles of:

- Authenticated booting, relation to (closed) secure booting
- Remote attestation
- Sealed memory
- Dynamic root of trust, late launch
- Protection of applications from the OS
- Point to implementation variants (TPM, iSGX, ARM-TZ)
Non-Goal:

- Lots of TPM, TCG, Trustzone, SGX details → read the documents once needed
Some Terms

- Secure Booting
- Authenticated Booting
- (Remote) Attestation
- Sealed Memory
- Late Launch / dynamic root of trust
- Trusted Computing (Group) / Trusted Computing Base

- Beware of terminology chaos!
Trusted Computing Base (TCB)

- The set off all components, *hardware, software, procedures*, that must be relied upon to enforce a security policy.

Trusted Computing (TC)

- A particular technology comprised of authenticated booting, remote attestation and sealed memory.
TC Key Goals

- Can running certain Software be prevented?
- Which computer system do I communicate with?
- Which stack of Software is running?
  - In front of me?
  - On my server somewhere?
- Restrict access to certain secrets (keys) to certain software?
- Protect an application against the OS
Usage Examples (1)

Digital Rights Management:

- Provider sells content
- Provider creates key, encrypts content
- Client downloads encrypted content, stores on disk
- Provider sends key, but needs to ensure that only specific SW can use it
- Has to work also when client is off line
- PROVIDER DOES NOT TRUST CUSTOMER
Virtual machine provided by cloud

- Client buys Cycles + Storage (Virtual machine)
- Client provides its own operating system
- Needs to ensure that provided OS runs
- Needs to ensure that provider cannot access data
- CUSTOMER DOES NOT TRUST PROVIDER
Industrial Plant Control (Uranium enrichment)

- Remote Operator sends commands, keys
- Local operator occasionally has to run test SW, update to new version, ...
- Local technicians are not Trusted
Anonymity Service

- Intended to provide anonymous communication over internet
- Legal system can request introduction of trap door (program change)
- Anonymity-service provider not trusted
Trusted Computing Terminology

**Measuring**
- “process of obtaining metrics of platform characteristics”
- example for metric: Hash- Codes of SW

**Attestation**
- “vouching for accuracy of information”

**Sealed Memory**
- binding information to a configuration
Notation

- $H(M)$
  Collision-Resistant Hash Function $H$
  applied to content $M$

- $S_{\text{pair}}: S_{\text{priv}} \quad S_{\text{pub}}$
  Asymmetric key pair of entity $S$
  used to conceal or sign some content
  $S_{\text{pub}}$ is published, $S_{\text{priv}}$ must be kept secret

- $S_{\text{symm}}$
  symmetric key, must be kept secret ("secret key")


- "Digital Signature": \( \{ M \} S_{\text{priv}} \)
  
  \( S_{\text{pub}} \) can be used to verify that \( S \) has signed \( M \)
  
  is short for: \( ( M, \text{encrypt}(H(M), S_{\text{priv}}) ) \)
  
  \( S_{\text{pub}} \) is needed and sufficient to check signature

- "Concealed Message": \( \{ M \} S_{\text{pub}} \)
  
  Message concealed for \( S \)
  
  \( S_{\text{priv}} \) is needed to unconceal \( M \)
Identification of Software

Program vendor: Foosoft FS

Two ways to identify Software: Hash / Public Key

- \( H(\text{Program}) \)

- \( \{\text{Program, ID- Program}\}FS^{\text{priv}} \)
  use \( FS^{\text{pub}} \) to check
  the signature must be made available,
  e.g. shipped with the Program

The „ID“ of SW must be known.
\( FS^{\text{pub}} \) can serve as ID as well.
Tamperresistant Black Box (TRB)

- **CPU**
- **Memory**
- **Non-Volatile Memory (NVM)**
- **Platform Configuration Regs (PCR)**

Conceptional View
Ways to “burn in” the OS or “Secure Booting”

- Read-Only Memory (Flash)
- H(OS) in NVM preset by manufacturer
  - load OS- Code
  - compare H(loaded OS code) to preset H(OS)
  - abort if different
- $FS_{pub}$ in NVM preset by manufacturer
  - load OS- Code
  - check signature of loaded OS-Code using $FS_{pub}$
  - abort if check fails
Authenticated Booting, using HASH

Steps:
A. Preparation by TRB and OS Vendors
B. Booting & “Measuring”
C. Remote attestation
Tamperresistant Black Box (TRB)

- CPU
- Memory
- TRB
- Conceptional View
- NVM:
- PCR:
Tamperresistant Black Box (TRB)

NVM:

PCR:
TRB generates key pair: „Endorsement Key“ $E_{K}^{\text{pair}}$
stores $E_{K}^{\text{priv}}$ in TRB NVM
publishes $E_{K}^{\text{pub}}$
TRB and OS vendor

- TRB vendor certifies:
  \{"a valid EK", EK^{\text{pub}}\}_{\text{TRB_VENDOR}^{\text{priv}}}

- OS-Vendor certifies:
  \{"a valid OS", H(OS)\}_{\text{OS_VENDOR}^{\text{priv}}}

- serve as identifiers:
  \text{EK}^{\text{pub}} \text{ and } H(OS)
Booting

TRB:
- resets TRB!
- measures OS code $H(\text{OS})$
- stores $H(\text{OS})$ in PCR

PCR not (directly) writable by OS
more later
Challenge:
send NONCE

Response:
{NONCE', PCR}E_{K_{priv}}
Problem

- boot Linux
  - challenge
  - response “Linux”

- reboot Windows
  - send data

add one step of indirection:
create keypairs at each reboot
Booting (Considering Reboot)

At booting, TRB:

- computes $H(\text{OS})$ and stores in PCR
- creates 2 keypairs for the booted, “active” OS (like “Session key”):
  - $\text{ActiveOSAuth}_{\text{pair}}$ /* for Authentication
  - $\text{ActiveOSCons}_{\text{pair}}$ /* for Concellation
- certifies:
  \[
  \{ \text{ActiveOSAuth}_{\text{pub}}, \text{ActiveOSCons}_{\text{Kpub}}, H(\text{OS}) \} \quad \text{EK}_{\text{priv}}
  \]
- hands over ActiveOSKeys to booted OS
Remote Attestation:

- Challenge: nonce
- Active OS generates response:
  \[
  \{ \text{ActiveOSCons}^{\text{pub}}, \text{ActiveOSAuth}^{\text{pub}}, H(\text{OS}) \} EK^{\text{priv}}
  \]
  /* see previous slide
  \[
  \{ \text{nonce}' \} \text{ActiveOSAuth}^{\text{priv}}
  \]

Secure channel:

\[
\{ \text{message } \} \text{ActiveOSCons}^{\text{pub}}
\]
Assumptions

- TRB can protect: $E_{K^{\text{priv}}}$, PCR
- OS can protect: “Active OS keys”
- Rebooting destroys content of
  - PCR
  - Memory Holding “Active OS keys”
Software Stacks and Trees
Software Stacks and Trees

2 Concerns:

- Very large Trusted Computing Base for Booting (including Device Drivers etc)
- Remote attestation of one process (leaf in tree)
“Extend” Operation:

- stack: $\text{PCR}_n = H(\text{PCR}_{n-1} \ || \ \text{next-component})$
- tree: difficult (hearsay, unpublished?)
Key pairs per step:

- OS controls applications → generate key pair per application
- OS certifies
  - \{ Application 1, App1Kpub \} ActiveOS_{\text{priv}}
  - \{ Application 2, App2Kpub \} ActiveOS_{\text{priv}}
Late Launch/Dynamic Root of Trust

Problem: huge Software to boot system !!!

- Use arbitrary SW to start system and load all SW
- provide specific instruction to enter “secure mode”
  - set HW in specific state (stop all processors, IO, …)
  - Measure “root of trust” SW and store in PCR

- AMD: “skinit” (Hash) arbitrary root of trust
- Intel: “senter” (must be signed by chip set manufacturer)
Sealed Memory

Goal:

- Send information using secure channels
- Bind that information to Software configuration
- Work offline:
  How to store information in the absence of communication channels?
- For example DRM:
  bind encryption keys to specific machine, specific OS
Sealed Memory Principle

Tamper-Resistant Black Box

<table>
<thead>
<tr>
<th>PCR: H(OS)</th>
<th>Win 7</th>
<th>SUSE-Linux</th>
<th>L4-Test-Version</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Add / delete entry
Read / write
Sealed Memory Principle

Tamper-Resistant Black Box

Add / delete entry
Read / write

PCR: $H(OS)$

Win 7
SUSE-Linux
L4-Test-Version

Win 7
SUSE-Linux
L4-Test-Version
Sealed Memory Principle

Tamper-Resistant Black Box

Add / delete entry
Read / write

PCR: \( H(\text{OS}) \)

Win 7
SUSE-Linux
L4-Test-Version
Sealed Memory Principle

Tamper-Resistant Black Box

<table>
<thead>
<tr>
<th>PCR: $H(OS)$</th>
<th>Win 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SUSE-Linux</td>
</tr>
<tr>
<td></td>
<td>L4-Test-Version</td>
</tr>
</tbody>
</table>

Add / delete entry
Read / write
Sealed Memory: Seal Operation

Tamper-resistant black box

PCR: $H(\text{Win-7})$

- Win 7
- SUSE-Linux
- L4-Test-Version

Sealed Message

Message
TRB generates symmetric Storage Key
never leaves chip
Sealed Memory

Seal(message):
encrypt("PCR, message", S) → "sealed_message";
emit sealed_message

Unseal(sealed_message):
decrypt(sealed_message, S) → "SealTime_PCR,message";
If SealTime_PCR == PCR
then emit message
else abort
Sealed Memory for future configuration

Seal(message, FUTURE_Config):

encrypt(“FUTURE_Config, message”, S) → “sealed_message”;
emit sealed_message

“seals” information such that it can be unsealed by a future configuration
(for example: future OS version)
Example

- **Win8:** Seal („SonyOS, Sony-Secret“)
  → SealedMessage (store it on disk)

- **L4:** Unseal (SealedMessage)
  → SonyOS, Sony-Secret
  → PCR#SonyOS
  → abort

- **SonyOS:** Unseal(SealedMessage)
  → SonyOS, Sony-Secret
  → PCR==SonyOS
Tamper Resistant Box?

Ideally, includes CPU, Memory, ...

Current practice

- Additional physical protection, for example IBM 4758 ...
  look it up in Wikipedia

- HW support:
  - TPM:
    separate “Trusted Platform Modules” (replacing BIOS breaks TRB)
  - Add a new privilege mode: ARM TrustZone
  - raise to user processes: Intel SGX
Protection of Application

Principle Method:
- separate critical Software
- rely on small Trusted Computing Base
  - Small OS kernels
    - micro kernels, separation kernels, ....
  - Hardware/Microcode Support
Small Trusted Computing Base

Hardware

App.

X11

Linux

APP!

TU Dresden: Hermann Härtig

Distributed Operating Systems, “Trusted Computing”, SS 2018
Small Trusted Computing Base

App.
X11
Linux
Mini OS
Hardware
APP!
Helper

TU Dresden: Hermann Härtig
Distributed Operating Systems, “Trusted Computing”, SS 2018
Small Trusted Computing Base

App.
X11
Linux

Helper

Hardware

APP!

Linux
X11
App.
Helper
APP!
TCG PC Platforms: “Trusted Platform Module” (TPM)
Small Trusted Computing Base

- App.
- X11
- Linux

- Mini OS
- Hardware

- APP!
- Helper
ARM TrustZone

Normal World | Secure World
---|---
PL0: User | PL0: User
PL1: Kernel | PL1: Kernel
PL2: Hypervisor | PL2: Hypervisor
Monitor | Trusted Services
| Trusted OS
intel SGX

App.
X11
Linux

Hardware

Helper

APP!

Intel
SGX
intel SGX

Conceptional View

CPU
Memory

Non-Volatile Memory (NVM): Platform Configuration Regs (PCR):
bound to application “enclaves”
“Enclaves” for Applications:

- established per special new instruction
- measured by HW
- provide controlled entry points
- resource management via untrusted OS
The iPhone architecture includes the following components:

- **Hardware**
- **IOS-Kernel**
- **GUI**
- **App.**

These components are protected and managed by the **Security CPU** and the **L4** level of the security architecture.
Important Foundational Paper:

Authentication in distributed systems: theory and practice
Butler Lampson, Martin Abadi, Michael Burrows, Edward Wobber
ACM Transactions on Computer Systems (TOCS)
More References

- TCG Specifications: https://www.trustedcomputinggroup.org/groups/TCG_1_3_Architecture_Overview.pdf
- ARM Trustzone & Intel SGX vendor sources