CPU as the New Perimeter
Attestation and Memory Encryption Protect Sensitive Data in the Cloud

Oded Horovitz
Co-Founder & CEO
PrivateCore Inc
Computation

Input: 10010010101

Processing: $f(x) = y$

Internal State: 10010010101
01001001001
00100100010
00100100010

Output: 10010010101
Processing composition

Input: 10010010101

Processing: f(x) = y

Internal State: 10010010101
01001001001
00100100010
01001001001
00100100010

Output: 10010010101

Composed of:
- Hardware
- Software
- Policy

Mutable
Internal Storage hierarchy

Input
10010010101

Processing
f(x) = y

Internal State
10010010101
01001001001
00100100010

Output
10010010101

Storage Hierarchy
- HDD
- Flash
- RAM
- CPU Cache
- Registers
- Persistent
Hacking, exploits existing vulnerabilities

\[ f(x) = y \]
Physical attack, walking with the data

Input
10010010101

Output
10010010101

Processing
f(x) = y

Internal State
10010010101
01001001001
00100100010

Out of bound
Let's add Operations

User I/O

10010010101

Developers

New code

IT

Settings

Processing

f(x) = y

f-upgrade(x)

f-config(x)

Internal State

10010010101
01001001001
00100100010
Admin hack - self provision access

User I/O
10010010101

Developers
New code

IT
Settings

Processing
f(x) = y
f-upgrade(x)
f-config(x)

Internal State
10010010101
01001001001
00100100010
Developer hack – Introducing backdoors

User I/O
10010010101

Developers
Backdoor

IT
Settings

Processing
f(x) = y
f-backdoor(x)
f-upgrade(x)
f-config(x)

Internal State
10010010101
01001001001
00100100010

I/O
Add risk of public communications

User I/O

Internet

f(x) = y

Processing

Internal State

10010010101
00100100010

10010010101
01001001001
00100100010
Also, real systems show complex composition

Up the stack!
Still, real systems show complex composition

Down the stack!
Sample attacks at the IaaS level

Integrity attacks

- SMM infection
- HDD firmware infection
  - injected kernel arguments

Physical attacks

- Grabbing clear private SSH keys
- Cold-boot

Logical access attacks

- Inception
- DMA capture of mysql records
- Malicious device I/O
SMM Infection, execution integrity forever lost

Status: Released / Deployed. Ready for Immediate Delivery

Unit Cost: $0
HDD firmware infection, WYSI\textsuperscript{N}WYG
Injected kernel argument & SSH key grab

http://youtu.be/6C0b3nMXeGU
Cold-boot attack, grabbing memory

http://youtu.be/5SKq9o0Luy0
Inception rewriting your memory

http://youtu.be/wki66w1iJHA
DMA IaaS (Inception-as-a-Service)

http://youtu.be/AI-XbzKO7HM
Malicious device I/O

OS Developers are not writing defensive device drivers...

In response for our submitted drivers vulnerabilities:

"These are lengths written by hardware, so will only be wrong if the hardware is broken. If the hardware is broken (or replaced by something malicious) then it can do anything it likes. Invalid values in ring entries are the least of your worries."
So how do we protect against such attacks?
IT Security Job I: Prevent physical grab

User I/O

10010010101

Developers
New code

IT
Settings

Processing

f(x) = y
f-upgrade(x)
f-config(x)

Internal State
10010010101
01001001001
00100100010

Physical security control
IT Security Job II: Check system integrity & lockdown

Processing

f(x) = y
f-upgrade(x)
f-config(x)

User

10010010101

Dev

New code

IT

Settings

Internal State

10010010101
01001001001
00100100010

Hardware

Software

Policy

OK OK OK
IT Security Job III: Secure logical access

- **User**: 10010010101
- **Dev**: New code
- **IT**: Settings

**Processing**
- $f(x) = y$
- $f$-upgrade($x$)
- $f$-config($x$)

**Internal State**
10010010101
010010010001
00100100010
IT Security Job IV: Encrypt public I/O

User I/O

10010010101

Internet

f(x) = y

Processing

Physical security control

Internal State

10010010101
01001001001
00100100010
00100100010

Internet

10010010101

Physical security control

10010010101
01001001001
00100100010
00100100010
So how do we protect against such attacks?

Integrity attacks
- SMM infection
- HDD firmware infection
- injected kernel arguments

Physical attacks
- Grabbing clear private SSH keys
- Cold-boot

Logical access attacks
- Inception
- DMA capture of mysql records
- Malicious device I/O

Hardware: OK
Software: OK
Policy: OK

Physical security control
Encrypt elsewhere

Logical Security Control (IO-MMU)
# The Cloud Challenge

<table>
<thead>
<tr>
<th>Service Type</th>
<th>Component</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaaS</td>
<td>App</td>
<td>How can a tenant verify integrity? Who defines an “OK” stack?</td>
</tr>
<tr>
<td></td>
<td>VM</td>
<td>What’s a good physical perimeter?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The data-center?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cage?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Server?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPU?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Encryption depends on the above question)</td>
</tr>
<tr>
<td>IaaS</td>
<td>KVM</td>
<td></td>
</tr>
<tr>
<td>Bare-Metal-aaS</td>
<td>Server</td>
<td>Should IaaS CSPs take more responsibility? Or give more control to customer?</td>
</tr>
</tbody>
</table>

**Diagram:**

- **SaaS**
  - App
  - VM

- **IaaS**
  - KVM
  - Host OS

- **Bare-Metal-aaS**
  - Server
Our mantra for secure IaaS (in x86 world)

1. Enable TPM & TXT

2. Choose a policy for hypervisor (i.e. “below the VM”) secure configuration. Tip: Consider stateless hypervisors.

3. Verify than trust. Give no secrets to unverified systems

4. Decide on physical perimeter
   - Best – CPU
   - Good – The server
   - Risky – Data-center

5. Encrypt outside your chosen perimeter! (storage & network)
PrivateCore vCage Host

The CPU as the perimeter of computation
PrivateCore vCage Host

The CPU as the perimeter of computation

• Physical security is the CPU package itself
PrivateCore vCage Host

The CPU as the perimeter of computation

- Physical security is the CPU package itself
- Loading stateless image into CPU cache
PrivateCore vCage Host

The CPU as the perimeter of computation

- Physical security is the CPU package itself
- Loading stateless image into CPU cache
- Test system integrity via Intel TXT
PrivateCore vCage Host

The CPU as the perimeter of computation

- Physical security is the CPU package itself
- Loading stateless image into CPU cache
- Test system integrity via Intel TXT
- Provision secrets (keys)
PrivateCore vCage Host

The CPU as the perimeter of computation

- Physical security is the CPU package itself
- Loading stateless image into CPU cache
- Test system integrity via Intel TXT
- Provision secrets (keys)
- Add logical security
  - DMA protection
  - Filter device IO
PrivateCore vCage Host

The CPU as the perimeter of computation

- Physical security is the CPU package itself
- Loading stateless image into CPU cache
- Test system integrity via Intel TXT
- Provision secrets (keys)
- Add logical security
  - DMA protection
  - Filter device IO
- Encrypt anything outside the CPU
<table>
<thead>
<tr>
<th></th>
<th>Registers</th>
<th>CPU Cache</th>
<th>RAM</th>
<th>DISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrivateCore</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CARMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frozen Cache</td>
<td>Pin</td>
<td>Exposed</td>
<td>Encr</td>
<td></td>
</tr>
<tr>
<td>Tresor</td>
<td>Exposed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cryptkeeper</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status quo</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- PrivateCore is Pinned with Encrypted
- CARMA is Pinned with Disabled
- Frozen Cache is No-fill with Exposed Encrypted
- Tresor is Exposed with Encrypted
- Cryptkeeper is Encrypt with Encrypted
- Status quo is Encrypted with Encrypted

The diagram shows the encryption status for different components (Registers, CPU Cache, RAM, DISK) for various systems (PrivateCore, CARMA, Frozen Cache, Tresor, Cryptkeeper, Status quo).
A reasonable performance tradeoff

The CPU & DRAM as the perimeter of computation

- Encrypt anything outside the CPU & DRAM
- Cons: Vulnerable to “cold-boot”, “malicious DIMM” & bus analysis
- Pro: High integrity without the performance penalties
- Ideal for public cloud environments
Biggest challenges

• Squeeze the Linux kernel into < 10MB while
  – Keeping all virtualization features
  – Keeping it stable (No OOM allowed)

• Keep CPU cache under our control

• Performance work
  – Squeeze different data structure to reduce working set
  – Identify new hot-paths in the kernel
  – Utilize AESNI capabilities
What’s coming?

Offensive

Deeper down the stack we go!
Sniffing and MITM any bus
facedancer – USB hacking in python! 55$

Defensive

Intel SGX – A huge step toward CPU as physical perimeter
More Open Source software & hardware
Q & A

Oded Horovitz
oded@privatecore.com