SeM: A CPU Architecture Extension for Secure Remote Computing

Ofir Shwartz, Yitzhak Birk

CATC, 2017
Motivation

• Clouds are promising:
  ▫ Pay per use
  ▫ No overhead costs
  ▫ Establish and discard resources on the fly

• Security limits adoption:
  ▫ Software: other users (competitors), OS, hypervisor, VMM
  ▫ Privileged attacker: exploit bugs, cloud owner
  ▫ Hardware: physical attacks
Threat Model

- **Platform software - Untrusted**
  - Hypervisor, VMM, OS
  - Any management software

- **Platform hardware**
  - Memory, network, board signals are untrusted
  - CPU is trusted – not internally snooped or modified

- **An attacker has full control of the machine**
  - Can implant software or hardware before or during the operation of the program
Secure Execution - Goals

- Keep confidentiality and integrity of:
  - Data: input, temporary, output
  - Code
  - State of execution

- While also:
  - Support existing applications (binaries)
  - Support conventional systems: multi-tasking, interrupts, signals, system calls, etc.
  - High performance execution
  - Low power / area overheads
Previous Works - Hardware Based

- Commonly: only the CPU is trusted.
- Many do not support existing binaries, and performance is low
- Intel SGX
  - Only matches programs developed for it
  - Limited performance
- Software on top of SGX:
  - E.g., Haven, PANOPLY, Graphene, SCONE, Eleos, …
  - Support for some applications, still major performance issues
Secure Machine - SeM
Flow
Flow

1) Take a program binary and prepare it
   - Manipulate / add instructions (if needed)
   - Encrypt and sign
Flow

1) Take a program binary and prepare it
   - Manipulate / add instructions (if needed)
   - Encrypt and sign
Flow

2) Communicate with the remote machine’s CPU
   - Setup execution: securely store necessary metadata
Flow

3) Send the result of (1) for remote secure execution
Flow
Flow
Flow

- 4) Collect results and error status
Flow

• 4) Collect results and error status

• (Similar to what is done in cloud deployment)
Single-Core Single-Thread

- Challenge #1: Protect code and data
- Challenge #2: Protect state and flow
- Challenge #3: Allow the use of untrusted code
Secure Machine (SeM) Arch. Ext.

SMU: Automatically acts on events and performs simple instructions
#1: Protect Code and Data

- Memory Encryption (commonly used)
  - Code and data: signed and encrypted when in untrusted memory, unencrypted when in cache
    - Counter mode encryption (e.g., GCM)
    - Signatures (e.g., GHASH) and an integrity tree

- Key Storage: securely store secret keys in the SMU
  - Per process, or group of processes
  - Keys: write-only, forming a Key Entry
    - Securely done using public key cryptography
  - Upon program start, attach with the process ID(s)

- But what about cached (unencrypted) data?
  - Secure Access: couple instructions and data by a security domain
Memory ⇔ Cache

- **On cache miss**: fetch block from memory
  - Decrypt and validate
  - If validated correctly, fetch decrypted block;
  - else, fetch original
  - To cache: data, Auth (true/false), owner ID (*ID in the Key Storage*)

- **Cache blocks**:
  - Each block also has Auth bit and OID
  - \{Auth, OID\} serves as the Security Domain of the block

<table>
<thead>
<tr>
<th>Cache</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>data</td>
<td>true</td>
</tr>
<tr>
<td>metadata</td>
<td>data</td>
<td>false</td>
</tr>
</tbody>
</table>
Cache $\Rightarrow$ Memory

- Upon eviction:
  - If Auth = $t$, sign and encrypt
    - Using the keys in the Key Storage (for owner ID)
    - Also update the integrity structure
  - Else: evict as is
Cache ⇔ Exec Unit

- On instruction fetch, also fetch Auth and OID

Secure Access
- On each cache access (memory instruction - load, store,..)
- If \( \text{inst}\{ \text{Auth}, \text{OID}\} = \text{data}\{ \text{Auth}, \text{OID}\} \)
- allow access
- else
- reject
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Untrusted Memory

### Execution Unit

SeM: CPU Arch. Ext. for Secure Computing
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Untrusted Memory

Instruction Fetch

Execution Unit

SeM: CPU Arch. Ext. for Secure Computing
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cache Miss!

Instruction Fetch

Untrusted Memory

Execution Unit

SeM: CPU Arch. Ext. for Secure Computing
Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cache Miss!

Instruction Fetch

SMU Validate

Untrusted Memory

Execution Unit
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
</tbody>
</table>

- **Untrusted Memory**
- **Execution Unit**
- **Instruction Fetch**

SeM: CPU Arch. Ext. for Secure Computing
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
</tbody>
</table>

Load R1,[20] (Auth=true, OID=30)
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
</tbody>
</table>

### Untrusted Memory

### Execution Unit

SeM: CPU Arch. Ext. for Secure Computing
# Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
</tbody>
</table>

Untrusted Memory

Execution Unit

Load R1,[20] (Auth=true, OID=30)
## Secure Access

### Table

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
</tbody>
</table>

Cache Miss!

Load R1,[20] (Auth=true, OID=30)

**Untrusted Memory**

**Execution Unit**
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
</tbody>
</table>

Cache Miss!

SMU Validate

Untrusted Memory

Load R1,[20] (Auth=true, OID=30)

Execution Unit
Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0xABCD</td>
<td>True</td>
<td>30</td>
</tr>
</tbody>
</table>

Load R1,[20] (Auth=true, OID=30)

Untrusted Memory

Execution Unit
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0xABCD</td>
<td>True</td>
<td>30</td>
</tr>
</tbody>
</table>

Grant!

Load R1,[20] (Auth=true, OID=30)

Untrusted Memory

Execution Unit

Ofir Shwartz, Electrical Eng. Department

SeM: CPU Arch. Ext. for Secure Computing
Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0xABCD</td>
<td>True</td>
<td>30</td>
</tr>
</tbody>
</table>

0xABCD

Untrusted Memory

Execution Unit
Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0xABCD</td>
<td>True</td>
<td>30</td>
</tr>
</tbody>
</table>

Untrusted Memory

Execution Unit
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0xABCD</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0x5678</td>
<td>false</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0x1122</td>
<td>true</td>
<td>35</td>
</tr>
</tbody>
</table>

Untrusted Memory

Execution Unit

SeM: CPU Arch. Ext. for Secure Computing
Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1, [20]</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0xABCD</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0x5678</td>
<td>false</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0x1122</td>
<td>true</td>
<td>35</td>
</tr>
</tbody>
</table>

Load R1, [50] (Auth=true, OID=30)

Untrusted Memory

Execution Unit

SeM: CPU Arch. Ext. for Secure Computing
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0xABCD</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0x5678</td>
<td>false</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0x1122</td>
<td>true</td>
<td>35</td>
</tr>
</tbody>
</table>

Load R1,[50] (Auth=true, OID=30)

Reject
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0xABCD</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0x5678</td>
<td>false</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0x1122</td>
<td>true</td>
<td>35</td>
</tr>
</tbody>
</table>
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0xABCD</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0x5678</td>
<td>false</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0x1122</td>
<td>true</td>
<td>35</td>
</tr>
</tbody>
</table>

Load R1,[60] (Auth=true, OID=30)
## Secure Access

<table>
<thead>
<tr>
<th>Metadata</th>
<th>Data</th>
<th>Auth</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata</td>
<td>Load R1,[20]</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0xABCD</td>
<td>True</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0x5678</td>
<td>false</td>
<td>30</td>
</tr>
<tr>
<td>metadata</td>
<td>0x1122</td>
<td>true</td>
<td>35</td>
</tr>
</tbody>
</table>

Reject

Load R1,[60]
(Auth=true, OID=30)

Untrusted Memory

Execution Unit
Secure Access: Benefits

- **Safe**: foreign code cannot validate correctly
  - Even if privileged
  - Must be validated to access validated (protected) data

- **Automatic** boundary between trusted and untrusted worlds
  - Unmodified code cannot expose memory data or import unauthorized memory data by mistake (unintentionally)

- **Performance**: adversarial blocks co-reside in the cache
  - No added evictions on top of a regular machine
  - The system matches the performance the Memory Encryption subsystem in use (encrypt and sign) (~2%)
#2: Protect State and Flow

- **State:** register values; **Flow:** seq. of instructions

- **Example:** Interrupt issues an untrusted instruction unexpectedly
  - Register values are exposed (*secret context*)
  - When back, need to enforce correct register values and correct instruction
Security Modes (Cache ↔ Exec Unit)

- Work in two modes: **Trusted** and **Untrusted**
  - Trusted mode: only runs validated (Auth=t) instructions
  - Untrusted mode: only runs non-validated instructions
  - Switch automatically
- If **Trusted** and inst{Auth} = false
  - Store register values in SMU Sealed Storage and clear *(secret context)*, keep the next legal entry point (LEP)
  - Change to **Untrusted** mode
- If **Untrusted** and inst{Auth} = true, and the process has a secret context (and inst{address} == LEP)
  - Restore the secret context
  - Change to **Trusted** mode
Parallel Execution - Settings

- Multiple threads on a single core
- Multi-core (CMP)
- Multiple compute nodes
Parallel Secure Machine (ParSeM)

Diagram showing the architecture of ParSeM with a trusted area (TA) overlaying the CPU architecture. The trusted area contains a security management unit (SMU) with cache and register units. The untrusted area, encrypted region, includes swap disk and main memory. The diagram illustrates the flow of data between the trusted and untrusted areas.
But More Is Needed

• Challenge #4: Thread management

and for multi-node:

• Challenge #5: Secure multi-node communication
  ▫ Secure Distributed Shared Memory (SDSM)
  ▫ High performance, thousands of nodes

• Challenge #6: Multi-node integrity
  ▫ Distributed Memory Integrity Trees
#4: Thread Management

- Multi-threaded applications: user vs. kernel threads
  - Only kernel threads benefits from parallel hardware
    - Multi-core, multi-CPU, multi-machine

- Yet it requires the *untrusted* OS to assist in thread creation (system call). Possible threats:
  - Spawning extra threads violates correctness
  - New thread’s register values must be set correctly
Hardware Assisted Secure Thread Creation

• Invoking a thread uses a special instruction (trusted) to prepare a pending context
  ▫ Ensure that only intended threads are created
    • Defend against replay attacks
  ▫ Pending context is securely stored
    • Correct register values are enforced

• This special instruction is automatically added before each thread creation syscall (Static binary instrumentation)

• Similarly: secure thread migration and termination
  ▫ not in this presentation
Thread Creation

... 
new_stack=malloc(...) // non secure stack 
...

SCID=SMU_NewThread()

clone(flags, new_stack) ← (system call)

if (rax>0) // return value
    rsp = malloc(..) // secure stack, including secure init
    push(thread_func_addr) // normally done by clone
else if (rax<0) // clone return value
    SMU_NewThreadDelete(SCID)

...

New code
SeM-Prepare

• Input: a compiled binary
• Static instrumentation, preparing it for the cloud (deployment)
  ▫ Statically embed shared libraries
  ▫ Attach itself with the Key Storage entry
  ▫ Allocate and initialize the secure stack
  ▫ Initialize memory on allocation
  ▫ Replace syscall instructions with syscallX
  ▫ IO accesses: enc and dec by software (wrap syscalls)
• When done, encrypt and sign
Evaluation

- SPEC CPU and PARSEC benchmark suites, prepared by SeM-Prepare
- Evaluated by SeM-Simulator
  - Memory Encryption
  - Secure Access enforcement
  - Security modes and register switch
  - Support new SeM instructions: memory, system calls.
  - Multi-node computation:
    - Hardware Assisted Secure Threading, Secure Distributed Shared Memory (SDSM), and Distributed Integrity Trees

- Purpose: prove applicability and measure performance
SPEC CPU - Performance Reduction

- %Reduction with Mem Enc
- %Reduction without Mem Enc

- aset ar
- bzip2
- gcc
- h264ref
- hmer
- libquantum
- mcf
- perbench
- sjeng
- xalanomk
- omnetpp
- Average

Ofir Shwartz, Electrical Eng. Department
SeM: CPU Arch. Ext. for Secure Computing
SeM’s Serial performance overhead is <2%, where ~95% of it is for memory encryption.
PARSEC - Performance Reduction

Performance Overhead (%)

- %Reduction, 32 nodes
- %Reduction, 64 nodes
- %Reduction, 128 nodes
- %Reduction, 256 nodes
- %Reduction no Mem Enc, 32 nodes
- %Reduction no Mem Enc, 64 nodes
- %Reduction no Mem Enc, 128 nodes
- %Reduction no Mem Enc, 256 nodes

Ofir Shwartz, Electrical Eng. Department
SeM: CPU Arch. Ext. for Secure Computing
PARSEC - Performance Reduction

SeM’s Parallel performance overhead is ~2%
# of IOs and Memory Allocations (PARSEC)

![Bar graph showing the number of IOs and memory allocations for different PARSEC benchmarks. The graph indicates that for most benchmarks, the number of IOs ranges from 30 to 40.]
Cost

- Negligible additional area (~0.01%), performance (~2%), and power (<3%) overheads
Putting It All Together

• **SeM** supplies *single-core* secure capabilities
  ▫ **SMU** protected cache access
    • Coupling data and instructions in a security domain
    • Automatic register hide and restore
    • Special care for system calls
  
• **ParSeM**: Hardware Assisted Thread Management extending that to *multi-core*

• **SDSM** and **Distributed Integrity Trees** enable **ParSeM** to use *multi-CPU* and *multi-machine*
Conclusions

• We have presented SeM, a CPU architecture extension that provides a practical, backwards compatible, easy to apply, high performance and low area solution for secure computing.

Thank You!