CoSMIX: A Compiler-based System for Secure Memory Instrumentation and Execution in Enclaves

Meni Orenbach (Technion), Yan Michalevsky (Anjuna), Christof Fetzer (TU Dresden, Scone), Mark Silberstein (Technion)

Published in USENIX ATC’19
Speaker bio

- Yan Michalevsky
  - Co-founder and CTO of Anjuna Security (www.anjuna.io)
  - Phd from Stanford University (applied security and cryptography)
  - B.Sc from Technion (EE)
  - Speaker at BlackHat, RSA Conference
  - Research featured in BBC, Wired, Engadget, ArsTechnica and more
Enclaves

- Confidentiality
- Integrity
- Assume an untrusted operating system
- Recent advancements in Library OS and unikernel-based approaches enable execution of entire applications
Motivation: missing OS abstractions, performance and side-channel protection

- **Features**
  - Memory-mapping

- **Performance**
  - Secure User-managed Virtual Memory (SUVM) [Orenbach et al. ’17 (Eleos)]

- **Side-channel protection**
  - Transparent Oblivious RAM for enclaved applications protects against controlled side-channel attacks

- And much more (custom memory backends…)
Memory-mapping: missing construct in enclaves

Enclave

fast_read_db():
ptr = mmap(...);
data = *ptr;

Fault handler
Page-fault handling with SGX

6x the latency of signal handling without SGX
Prior work

• Sidestep the lack of secure page faults by customizing applications
  • Eleos (SUVM) [Orenbach et al. ’17]
  • ZeroTrace (ORAM) [Sasy et al. ’18]

• Require specialized handling of memory accesses

• Reference implementations are language-specific
  • Eleos implementation is not suitable for high-level languages
CoSMIX

• Compiler + runtime

• Automatic and transparent customization of memory accesses and page-fault handling

  • Automatic inference of pointer types via pointer-analysis
  
  • Locality-optimized translation caching

• Selective instrumentation of memory accesses

  • Guided by annotations of memory allocation
  
  • Automatic inference of related memory accesses
Memory Store (mStore)

- *mStore* — a software abstraction of memory access behavior
  - An additional *virtual memory* layer on top of a *backing store*
- Handles
  - Allocation
  - Deallocation
  - Address translation
  - Paging
Direct-access memory store

```c
val = *ptr;

*ptr = val;
```

Diagram:

1. **Mstore** Instrument
   - `val = *ptr;`
   - `*ptr = val;`

2. **Mstore_Fetch()**
   - `ORMA`

3. **Mstore_Writeback()**
   - `ORMA`

4. Memory Access
   - `??`
Cached memory store

```
val = *ptr;

*ptr = val;
```

Fault handler():

```
<table>
<thead>
<tr>
<th>Backing store</th>
<th>Page cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1000</td>
<td>0x5000</td>
</tr>
</tbody>
</table>
```

Custom logic...

Back ing store

Page cache
Use-case: Secure User-managed Virtual Memory (SUVM)

- Proposed by Orenbach et al. ’17 (Eleos)
- Motivation: avoid costly enclave transitions to handle demand paging
- Provides the same confidentiality and integrity guarantees as the EPC
- Caches pages in the EPC
- Can boost performance by ~2x compared to regular execution in SGX
Use case: Oblivious RAM (ORAM)

Controlled side-channel attacks can recover quite a bit of information by examining memory access patterns

[Xu et al. 2015]
Use case: Oblivious RAM (ORAM)

- Preserves I/O behavior
- Obfuscates memory access patterns
CoSMIX end-to-end

Annotated Source Code

Memory Stores & Runtime

CoSMIX Compiler (LLVM pass)

LibOS
SCONE/Anjuna/Graphene-SGX

Binary

Enclave
CoSMIX end-to-end

Annotate memory allocations with memory stores to use
CoSMIX end-to-end

Proper memory access instrumentation is inferred based on allocation annotations.

Annotate memory allocations with memory stores to use.
Stacking mStore-s

- ORAM → SUVM (Valid)
- SUVM → ORAM (Invalid)
Evaluation

Workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>LOC</th>
<th>Changed LOC</th>
<th>mstore</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcached [35]</td>
<td>15,927</td>
<td>1</td>
<td>SUVM</td>
</tr>
<tr>
<td>Redis [8]</td>
<td>123,907</td>
<td>0</td>
<td>SUVM</td>
</tr>
<tr>
<td>SQLite [61]</td>
<td>134,835</td>
<td>2</td>
<td>secure mmap</td>
</tr>
<tr>
<td>Phoenix suite [65]</td>
<td>1,064</td>
<td>1/bench</td>
<td>SUVM</td>
</tr>
<tr>
<td>Face verification [60]</td>
<td>700</td>
<td>1</td>
<td>SUVM → ORAM</td>
</tr>
</tbody>
</table>

Fetching a 4 KB page

<table>
<thead>
<tr>
<th>Data size</th>
<th>Baseline</th>
<th>SUVM</th>
<th>ORAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 MB</td>
<td>0.7μsec</td>
<td>0.9μsec (−1.28×)</td>
<td>32.3μsec (−46.1×)</td>
</tr>
<tr>
<td>256 MB</td>
<td>14.4μsec</td>
<td>1.5μsec (9.6×)</td>
<td>590.6μsec (−41×)</td>
</tr>
<tr>
<td>1GB</td>
<td>19.9μsec</td>
<td>1.6μsec (12.4×)</td>
<td>1.23msec (−61.8×)</td>
</tr>
</tbody>
</table>
Memcached

600 MB dataset
Random access to 1KB objects. 90% get / 10% set

![Chart showing latency vs. throughput for SGX and CoSMIX. SGX has a higher latency at lower throughputs but reaches a peak and drops significantly at higher throughputs. CoSMIX maintains a lower and more consistent latency across all throughputs. The chart highlights a 2.2x improvement in throughput for CoSMIX compared to SGX at the same latency. There is a 1 LOC annotated note.](image)
Memcached

600 MB dataset
Random access to 1KB objects. 90% get / 10% set
<table>
<thead>
<tr>
<th></th>
<th>Native SGX</th>
<th>ORAM</th>
<th>ORAM $\rightarrow$ SUVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput (req/sec)</td>
<td>203.1</td>
<td>23.4</td>
<td>34.7</td>
</tr>
<tr>
<td>Slowdown</td>
<td>8.6$\times$</td>
<td>5.8$\times$</td>
<td></td>
</tr>
</tbody>
</table>
Summary

• Compiler-based approach to memory instrumentation and SW page-fault handling

• Conveniently addresses
  • Lacking functionality
  • Performance
  • Security against certain side-channels

• Extensible
Thank You. Questions?

www.anjuna.io
yan@anjuna.io