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

Meni Orenbach
Yan Michalevsky
Christof Fetzer
Mark Silberstein
The night is dark and full of terrors
The night is dark and full of terrors
The cloud is dark and full of terrors
The cloud is dark and full of terrors
Yet, hardware enclaves can help
Enclaves shield applications from privileged adversaries

- Confidentiality
- Integrity
- Untrusted OS
Enclaves shield applications from privileged adversaries

- Confidentiality
- Integrity
- Untrusted OS

Enclave

SQLite

Untrusted OS
Enclaves shield applications from privileged adversaries

- Confidentiality
- Integrity

Untrusted OS

Enclave

SQLite

$\%\%$

$\%\%$

$\%\%$

USENIX ATC 2019

Meni Orenbach
Run unmodified applications inside enclaves

Enclave

SQLite
LibOS
Run unmodified applications inside enclaves

Request OS to read

Enclave

SQLite

LibOS
Run unmodified applications inside enclaves

Request OS to read

Decrypt and verify request

Enclave

LibOS

SQLite

$\%\%$

$\%\%$

$\%\%$

User

SQLite

Library

Enclave

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$

$\%\%$
Talk focus: Current SGX Enclaves
Integrated into Intel CPUs

Can we execute any x86 application inside enclaves?
Talk focus: Current SGX Enclaves

Integrated into Intel CPUs

Can we execute any x86 application inside enclaves?
Memory-mapped files in SGX?

```c
fast_read_db():
ptr = mmap(...);
data = *ptr;
```
Memory-mapped files in SGX?

Enclave

```
fast_read_db():
ptr = mmap(...);
data = *ptr;
```
Memory-mapped files in SGX?

Enclave

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

Fault handler
Memory-mapped files in SGX?

Enclave

SQLite

Fault handler customization is a powerful tool
- File mapping
- Disaggregated memory
- Compressed memory
and more

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

USENIX ATC 2019

Meni Orenbach
Memory-mapped files in SGX?

Enclave

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

Handler customization is a powerful tool
- File mapping
- Disaggregated memory
- Compressed memory
and more

Efficient and requires no modifications
Applications rely on this abstraction
Memory-mapped files in SGX?

Handler customization is a powerful tool
- File mapping
- Disaggregated memory
- Compressed memory
and more

Efficient and requires no modifications
Applications rely on this abstraction

Its Not Possible in SGX Today!
Memory-mapped files in SGX?

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

Implementation options:
- Use fault handler
- Use SGX demand-paging instructions
- Use in-enclave handler
Memory-mapped files in SGX?

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

Implementation options:
- Use fault handler
- Use SGX demand-paging instructions
- Use in-enclave handler
Memory-mapped files in SGX?

**Enclave**

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

**Implementation options:**
- Use fault handler
- Use SGX demand-paging instructions
- Use in-enclave handler
Inefficient: in-enclave handler

```c
fast_read_db():
ptr = mmap(...);
data = *ptr;
```
Inefficient: in-enclave handler

```c
fast_read_db():
ptr = mmap(...);
data = *ptr;
```
Inefficient: in-enclave handler

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

Enclave

Fault handler
Inefficient: in-enclave handler

Enclave

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

Exception_handler():
....

Fault handler
Inefficient: in-enclave handler

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

exception_handler():
......
Inefficient: in-enclave handler

Enclave

SQLite

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

Exception handler:
...

Fault handler
Inefficient: in-enclave handler

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

**exception_handler():**
```
....
```

---

Enclave

SQLite

Fault handler
Inefficient: in-enclave handler

Enclave

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

exception_handler():
....

Empty fault handler:
6x compared to regular #PF penalty
Insecure: in-enclave handler

```
Enclave

SQLite

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

exception_handler():
....
```
Insecure: in-enclave handler

Enclave

SQLite

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

exception_handler():
....
Insecure: in-enclave handler

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

exception_handler():
....
Insecure: in-enclave handler

Enclave

SQLite

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

exception_handler():
....

Fault handler
Insecure: in-enclave handler

Enclave

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

exception_handler():
    ....

Fault handler
Insecure: in-enclave handler

Enclaves are missing an OS abstraction!

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

exception_handler():
  ....

Fault handler
Previous works that sidestep the lack of secure page faults

- In-enclave paging
- Oblivious page accesses
- Secure access to remote memory

Ad-hoc solutions
Requires applications modifications
What if we want to use more than one?
Previous works that sidestep the lack of secure page faults

- In-enclave paging
- Oblivious page accesses
- Secure access to remote memory

Ad-hoc solutions
Requires applications modifications
What if we want to use more than one?
Efficient page fault customization **missing**
We cannot rely on **hardware**
We cannot **change applications**

So what can we do?
Efficient page fault customization missing
We cannot rely on hardware
We cannot change applications

So what can we do?

Automatically change applications with memory instrumentation
Agenda

- Motivation
- CoSMIX
- Evaluation
CoSMIX

Efficient *Instrumenting* compiler and runtime system
What to instrument?

- **Observation:**
  - No need to instrument every memory access

- **Instrumentation policy:**
  - Annotations on memory allocations
  - Automatically inferring memory accesses to be instrumented
How to express memory behavior

Interface for custom page fault handlers
How to express memory behavior

Interface for custom page fault handlers

Software abstraction Needed!
Memory Stores (Mstores)

Another layer of virtual memory on top of an abstract backing store
Memory Stores (Mstores)

Another layer of virtual memory
on top of an abstract backing store

foo():

```c
val = *ptr;
```
Memory Stores (Mstores)

Another layer of virtual memory
on top of an abstract backing store

```c
foo():
    val = *ptr;
```
Memory Stores (Mstores)

Another layer of **virtual memory**
on top of an abstract **backing store**

foo():

```c
val = *ptr;
```
Memory store interface

- Allocation
- Deallocation
- Address translation
- Paging system
Memory store interface

- Allocation
- Deallocation
- Address translation
- Paging system

Instrument memory allocations
Memory store interface

- Allocation
- Deallocation
- Address translation
- Paging system

Instrument memory allocations

Instrument memory accesses
Software address translation

Memory store address     Backing store address

More flexible than page fault handler customization
Memory store interface

- Allocation
- Deallocation
- Address translation
- Paging system
Use case: Oblivious RAM (ORAM)

- Preserves input-output behavior
- Obfuscates distribution of memory accesses
Direct-access memory store

\[
\begin{align*}
\text{val} &= \ast\text{ptr}; \\
\ast\text{ptr} &= \text{val};
\end{align*}
\]
Direct-access memory store

Previous works required manual modifications to use ORAM on every access
Memory store interface

- Allocation
- Deallocation
- Address translation
- Paging system
Cached memory stores

val = *ptr;

*ptr = val;

Fault handler():

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

Instrument

Backing store

Page cache

Custom logic...
Use case: In-enclave demand-paging

- Maintains SGX enclaves security
- Removes costly enclave exits
  - Boost performance
- Previous work required manual modifications
Cached memory stores:
In-enclave demand-paging

Fault handler():

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

val = *ptr;

*ptr = val;

Instrument
Cached memory stores: In-enclave demand-paging

```c
val = *ptr;

*ptr = val;
```

**Fault handler():**

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

**Decrypt & validate**

**Untrusted memory**
Software encryption

**Enclave Page cache**
Hardware protected
Cached memory stores: In-enclave demand-paging

val = \*ptr;

*ptr = val;

Fault handler():

<table>
<thead>
<tr>
<th>Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backing store</td>
</tr>
<tr>
<td>0x1000</td>
</tr>
</tbody>
</table>

Decrypt & validate

Instrument

Untrusted memory
Software encryption

Enclave Page cache
Hardware protected
Cached memory stores: In-enclave demand-paging

val = *ptr;

*ptr = val;

Fault handler():

<table>
<thead>
<tr>
<th>Page Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backing store</td>
</tr>
<tr>
<td>0x1000</td>
</tr>
</tbody>
</table>

Decrypt &

No need for: copy, decrypt, integrity validate on each access

Enclave Page cache
Hardware protected

Untrusted memory
Software encryption
Memory stores are easy

- C/C++
- Small and simple interface: Function callbacks!
- Common building blocks provided

It took a week to implement memory-mapped files mstore
CoSMIX: end-to-end usage

CoSMIX Compiler (LLVM pass)

Binary

Enclave

LibOS
SCONE/Anjuna/
Graphene-SGX

Annotated Source Code

Memory Stores & Runtime
CoSMIX: end-to-end usage

Annotate memory allocations with memory stores

LibOS
SCONE/Anjuna/
Graphene-SGX

Enclave
CoSMIX: end-to-end usage

- Annotated Source Code
- Memory Stores & Runtime

CoSMIX Compiler (LLVM pass)

LibOS
SCONE/Anjuna/
Graphene-SGX

Binary

Enclave
CoSMIX: end-to-end usage

- Allocations instrumented to infer memory stores’ callbacks for each access

CoSMIX Compiler (LLVM pass)

LibOS
SCONE/Anjuna/
Graphene-SGX

Allocated Source Code
Memory Stores & Runtime

Binary
Enclave
**CoSMIX: end-to-end usage**

- CoSMIX Compiler
  - Annotated Source Code
  - Memory Stores & Runtime
  - CoSMIX Compiler
  - Binary

Allocations instrumented to infer memory stores' callbacks for each access

Mix different memory stores!
Memory stores are stackable

\[ \text{val} = *\text{ptr}; \]
Memory stores are stackable

\[
\text{val} = \ast \text{ptr};
\]
Memory stores are stackable

val = *ptr;

Mstore 1
annotate

val = *ptr;

Mstore 2
annotate
Memory stores are stackable

val = *ptr;

Mstore 1
Mstore2
annotate

val = *ptr;

Mstore 1
annotate

val = *ptr;

Mstore 1
Mstore 2
Memory stores are stackable

val = *ptr;

Vertical stacking of Memory stores

val = *ptr;

Mstore1
annotate

Mstore 1

Mstore2
annotate

Mstore 2

Mstore1
annotate

Mstore1
annotate

Mstore2
annotate

Mstore2
annotate
Stacking ORAM on in-enclave paging

- ORAM performs multiple memory accesses to hide memory access patterns
- ORAM data structures increase its memory footprint
- ORAM mstore in SGX may cause memory thrashing
Stacking ORAM on in-enclave paging

val = *ptr;
Stacking ORAM on in-enclave paging

```c
val = *ptr;
```
Stacking ORAM on in-enclave paging

```
val = *ptr;
```

```
val = *ptr;
```

ORAM

annotate

Self-paging

annotate

ORAM
Stacking ORAM on in-enclave paging

```c
val = *ptr;
```

Self-paging annotate

ORAM

annotate

Self-paging

ORAM

annotate
Stacking ORAM on in-enclave paging

val = *ptr;

val = *ptr;

val = *ptr;

No code changes required
CoSMIX

**Efficient** Instrumenting compiler and runtime system
CoSMIX

Efficient Instrumenting compiler and runtime system

20% overhead Phoenix suite
Efficient instrumentation

- **Selective** instrumentation with **pointer analysis**
- Temporal access locality
  - Software TLB: **Cache address translations**
  - Selective instrumentation: **TLB can be small**
- Intensive memory accesses in loops
  - **Hoist** instrumentation for loop-invariant accesses
Agenda

- Motivation
- CoSMIX
- Evaluation
# Workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>LOC</th>
<th>Added logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>16k</td>
<td>In-enclave paging</td>
</tr>
<tr>
<td>REDIS</td>
<td>124k</td>
<td>In-enclave paging</td>
</tr>
<tr>
<td>SQLITE</td>
<td>135k</td>
<td>Secured file mapping</td>
</tr>
<tr>
<td>Face Verification</td>
<td>700</td>
<td>Oblivious page access (ORAM)</td>
</tr>
</tbody>
</table>
## Workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>LOC</th>
<th>Added logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memcached</td>
<td>16k</td>
<td>In-enclave paging</td>
</tr>
<tr>
<td>REDIS</td>
<td>124k</td>
<td>In-enclave paging</td>
</tr>
<tr>
<td>SQLITE</td>
<td>135k</td>
<td>Secured file mapping</td>
</tr>
<tr>
<td>Face Verification</td>
<td>700</td>
<td>Oblivious page access (ORAM)</td>
</tr>
</tbody>
</table>
Memcached (600 MB)
Random access 1KB 10% SET

![Graph showing latency (msec) vs. throughput (kreq/sec) for Memcached with SGX.]
Memcached (600 MB)
Random access 1KB 10% SET

![Graph showing latency (msec) vs. throughput (kreq/sec) for SGX and CoSMIX. SGX has a latency of 5.5 msec at 40 kreq/sec, while CoSMIX has a latency of 2.5 msec at 100 kreq/sec. The graph highlights a 2.2x improvement in throughput for CoSMIX. There is also a note indicating that 1 LOC is annotated.]
Memcached (600 MB)
Random access 1KB 10% SET

![Graph showing latency vs. throughput for different methods: SGX, CoSMIX, and Eleos. The graph indicates that SGX has a latency of 2.2x at a throughput of 40 kreq/sec, while CoSMIX has a latency of 1.07x at a throughput of 100 kreq/sec. Manual modifications are shown as an increase in performance.](image)
SQLite Encrypted DB file

kvtest Random access

Lower is better!
SQLite Encrypted DB file kvtest Random access

Lower is better!

4KB item read latency (usec)

No page cache  CoSMIX mmap (internal page cache)

Meni Orenbach

2 LOC annotated
SQLite Encrypted DB file kvtest Random access

Lower is better!

4KB item read latency (usec)

No page cache

CoSMIX mmap (internal page cache)

4.4x faster due to internal Mstore caching!

2 LOC annotated

SQLite Encrypted DB file kvtest Random access

Lower is better!

4KB item read latency (usec)

No page cache

CoSMIX mmap (internal page cache)

4.4x faster due to internal Mstore caching!

2 LOC annotated
SQLite Encrypted DB file
kvtest Random access

Lower is better!
SQLite Encrypted DB file
kvtest Random access

Lower is better!

No page cache

CoSMIX mmap (internal page cache)

With page cache

Manually optimized page cache only 27% faster
Biometric Identity checking server
Biometric Identity checking server

Camera + ID = [Question mark]
Biometric Identity checking server

Meni Orenbach, Technion
Biometric Identity checking server

Use ORAM to maintain travelers’ privacy
ORAM notoriously inefficient

61x slowdown for memory accesses
Face verification

Max Throughput

SGX

ORAM

Higher is better!

1 LOC annotated
Face verification

Higher is better!

Max Throughput

<table>
<thead>
<tr>
<th></th>
<th>SGX</th>
<th>ORAM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>200</td>
<td>10</td>
</tr>
</tbody>
</table>

Just 8.6x slowdown due to selective instrumentation

1 LOC annotated
Face verification

Higher is better!
Face verification

Max Throughput

SGX

ORAM

ORAM Stacked on In-enclave paging

Just \textbf{5.8x} slowdown

Higher is better!
Conclusion

- Hardware enclaves improve the security of applications in public cloud environments
- Hardware \textbf{limits the use} of a trusted page fault handler
- \textbf{CoSMIX} provides trusted page fault handling \textit{today}
  - Efficient instrumentation makes this a feasible approach

\textbf{Thank You!}

https://github.com/acsl-technion/cosmix