SGXBounds
Memory Safety for Shielded Execution

Dmitrii Kuvaiskii†, Oleksii Oleksenko †, Sergei Arnautov †, Bohdan Trach †, Pramod Bhatotia *, Pascal Felber ‡, Christof Fetzer †

† TU Dresden, * The University of Edinburgh, ‡ University of Neuchâtel
Security in the Cloud

- **Security** is a key barrier to adoption of cloud computing

[Logos of Google Cloud Platform, Amazon Web Services, and Microsoft Azure]
Security in the Cloud

- **Security** is a key barrier to adoption of cloud computing
- Attackers compromise **confidentiality** and **integrity**
Security in the Cloud

• **Security** is a key barrier to adoption of cloud computing

• Attackers compromise **confidentiality** and **integrity**
  ➔ Malicious host (e.g., cloud provider)
  ➔ Software vulnerabilities
• Security is a key barrier to adoption of cloud computing
• Attackers compromise confidentiality and integrity
  ➞ Malicious host (e.g., cloud provider)
  ➞ Software vulnerabilities
Security in the Cloud

- **Security** is a key barrier to adoption of cloud computing
- Attackers compromise **confidentiality** and **integrity**
  - Malicious host (e.g., cloud provider)
  - **Software vulnerabilities**

Virtual Address Space

Shielded execution (SGX Enclave)
Security in the Cloud

• **Security** is a key barrier to adoption of cloud computing

• Attackers compromise **confidentiality** and **integrity**
  - Malicious host (e.g., cloud provider)
  - **Software vulnerabilities**

![Virtual Address Space](image)

- Shielded execution (SGX Enclave)
- Heartbleed
- Cloudbleed
SGX Enclave
(malicious host)
Protecting against Attacks

SGX Enclave  
(malicious host)  

+  

Memory safety  
(vulnerabilities)
Protecting against Attacks

SGX Enclave (malicious host) + Memory safety (vulnerabilities)

AddressSanitizer (software-based) Intel MPX (hardware-based)
Protecting against Attacks

SGX Enclave (malicious host) + Memory safety (vulnerabilities)

State-of-the-art memory-safety mechanisms are inefficient!

- AddressSanitizer (software-based)
- Intel MPX (hardware-based)
State-of-the-Art: SQLite example
State-of-the-Art: SQLite example

![Graph showing execution time and memory usage for different working set items. The graph compares four different methods: SGX, ASan, MPX, and SGXBounds. The x-axis represents working set items, and the y-axis represents execution time (in $10^3$ s) and memory used (in GB). The graph indicates a lower execution time and better memory usage for lower working set items.](graph.png)
State-of-the-Art: SQLite example

![Graph showing execution time and memory usage for different working set items]
State-of-the-Art: SQLite example

Execution time ($\times 10^3$ s) vs. Working set items

- SGX
- ASan
- MPX
- SGXBounds

Memory used (GB) vs. Working set items

- SGX
- ASan
- MPX
- SGXBounds

3.1x lower better
State-of-the-Art: SQLite example

- **Execution time**: The graph shows the execution time across different working set sizes. MPX crashes at a lower point compared to the other methods (SGX, ASan, SGXBounds).
- **Memory used**: The graph illustrates the memory usage across different working set sizes. There is a significant difference in memory used between MPX and the other methods, with MPX consuming significantly more memory, up to 57 times more at certain working set sizes.

Overall, the data suggests that MPX has lower performance and higher memory consumption compared to SGX and ASan, especially as the working set size increases.
How to make it efficient?
State-of-the-Art: SQLite example

 Execution time ($\times 10^3$ s)

 Memory used (GB)

 Working set items

 SGX  ASan  MPX  SGXBounds

 MPX crashes

 lower better

 3.1x

 57x

 3.1x
State-of-the-Art: SQLite example

SGXBounds is practical
– Motivation
– Constraints of SGX enclaves
– Design of SGXBounds
– Implementation of SGXBounds
– Evaluation
- Motivation
- Constraints of SGX enclaves
- Design of SGXBounds
- Implementation of SGXBounds
- Evaluation
Why AddressSanitizer and Intel MPX perform poorly under SGX?

Virtual Address Space

Shielded execution (SGX Enclave)
Constraints of SGX Enclaves

Why AddressSanitizer and Intel MPX perform poorly under SGX?

😊 Increased latency of memory accesses
Constraints of SGX Enclaves

Why **AddressSanitizer** and Intel MPX perform poorly under SGX?

😊 **Increased latency** of memory accesses
Constraints of SGX Enclaves

Why AddressSanitizer and Intel MPX perform poorly under SGX?

 Exiting latency of memory accesses

Virtual Address Space

Shielded execution (SGX Enclave)

Physical Address Space

MEE encryption (1-12x)

CPU Cache (8MB)

Enclave Page Cache (94MB)

DRAM (64GB)
Why **AddressSanitizer** and Intel MPX perform poorly under SGX?

😊 *Increased latency* of memory accesses
Constraints of SGX Enclaves

Why AddressSanitizer and **Intel MPX** perform poorly under SGX?

- **Increased latency** of memory accesses
- **Limited** enclave memory (4GB)

![Diagram of Memory Spaces]

- Virtual Address Space
  - Shielded execution (SGX Enclave)
  - DRAM (64GB)
  - Enclave Page Cache (94MB)
- Physical Address Space
  - EPC paging (2-2000x)
  - MEE encryption (1-12x)
  - CPU Cache (8MB)
  - DRAM (64GB)
Assumptions of AddressSanitizer and Intel MPX violated in SGX!
Assumptions of AddressSanitizer and Intel MPX violated in SGX!
**State-of-the-Art: Metadata Layout**

**Assumptions** of AddressSanitizer and Intel MPX **violated** in SGX!

- 😞 Fast accesses to metadata
- 😞 Almost endless memory

---

**AddressSanitizer**

- shadow object
  - red zone
  - object
  - red zone
  - pointer

---

**Intel MPX**

- Bounds Table 1
- Bounds Directory
- object
- pointer

---
**Assumptions** of AddressSanitizer and Intel MPX violated in SGX!

- Fast accesses to metadata ≠ increased latency
- Almost endless memory ≠ limited enclave memory
State-of-the-Art: Metadata Layout

Assumptions of AddressSanitizer and Intel MPX violated in SGX!

😊 Fast accesses to metadata ≠ increased latency
😊 Almost endless memory ≠ limited enclave memory

Inefficient!
- Motivation
- Constraints of SGX enclaves
- Design of SGXBounds
- Implementation of SGXBounds
- Evaluation
Memory constraints of SGX dictated design of SGXBounds
Memory constraints of SGX dictated design of SGXBounds

😊 Increased latency → minimize accesses to metadata
SGXBounds: Metadata Layout

Memory constraints of SGX dictated design of SGXBounds

😊 Increased latency → **minimize accesses** to metadata
😊 Limited enclave memory → **minimize space** of metadata
SGX Bounds: Metadata Layout

Memory constraints of SGX dictated design of SGX Bounds

😊 Increased latency → **minimize accesses** to metadata
😊 Limited enclave memory → **minimize space** of metadata
Memory constraints of SGX dictated design of SGXBounds

😊 Increased latency → **minimize accesses** to metadata
😊 Limited enclave memory → **minimize space** of metadata

[Diagram showing the SGXBounds data structure with fields for object and pointer]
Memory constraints of SGX dictated design of SGXBounds

- 😊 Increased latency → **minimize accesses** to metadata
- 😊 Limited enclave memory → **minimize space** of metadata
SGXBounds: Metadata Layout

Memory constraints of SGX dictated design of SGXBounds

😊 Increased latency → **minimize accesses** to metadata
😊 Limited enclave memory → **minimize space** of metadata

![Diagram of SGXBounds]

- Upper bound (UB) in pointer
- Lower bound (LB) per object
Memory constraints of SGX dictated design of SGXBounds

😊 Increased latency → **minimize accesses** to metadata
😊 Limited enclave memory → **minimize space** of metadata

- Upper bound (UB) in pointer
- Lower bound (LB) per object
- Out-of-the-box **multithreading** (unlike MPX)
How SGXBounds detects vulnerabilities like Heartbleed?
How SGXBounds **detects vulnerabilities** like Heartbleed?

![Diagram showing SGXBounds structure with labels: LB, password, object, UB, pointer.](image-url)
How SGXBounds detects vulnerabilities like Heartbleed?

😊 Data leak through write(socket, pointer, objlen)
SGXBounds: Detecting Vulnerabilities

How SGXBounds **detects vulnerabilities** like Heartbleed?

😊 Data leak through **write(socket, pointer, objlen)**

---

**SGXBounds**

- **LB**
- **password**
- **LB**
- **object**
- **UB**
- **pointer**
How SGXBounds detects vulnerabilities like Heartbleed?

😊 Data leak through `write(socket, pointer, objlen)`

😊 Protect using efficient *bounds checks*

**Bounds-check** before each memory access: $\text{LB} \leq \text{pointer} \leq \text{UB}$
How SGXBounds \textbf{detects vulnerabilities} like Heartbleed?

-Headers

- Data leak through \texttt{write(socket, pointer, objlen)}

- Protect using efficient \textbf{bounds checks}

**SGXBounds**

\textbf{Bounds-check} before each memory access:

\[ LB \leq \text{pointer} \leq UB \]

embedded in tagged pointer
How SGXBounds detects vulnerabilities like Heartbleed?

Data leak through `write(socket, pointer, objlen)`
Protect using efficient bounds checks

Bounds-check before each memory access:

```
LB ≤ pointer ≤ UB
```

loaded from memory based on UB

embedded in tagged pointer
– Motivation
– Constraints of SGX enclaves
– Design of SGXBounds
– Implementation of SGXBounds
– Evaluation

Eurosys 2017
SGXBounds: Implementation

SGXBounds (LLVM pass)
SGXBounds: Implementation

Source code -> SGXBounds (LLVM pass) -> Shielded app (e.g., SCONE)
SGXBounds: Implementation

Source code → SGXBounds (LLVM pass) → Shielded app (e.g., SCONE) → Operating System → CPU SGX RAM
Advanced features:
SGXBounds: Implementation

Advanced features:

- **Tolerating** errors with **boundless memory**
- **Metadata management** support
- Compile-time **optimizations**
SGXBounds: Implementation

Advanced features:

- **Tolerating** errors with **boundless memory**
- **Metadata management** support
- **Compile-time** **optimizations**

See paper for details
– Motivation
– Constraints of SGX enclaves
– Design of SGXBounds
– Implementation of SGXBounds
– Evaluation
- Motivation
- Constraints of SGX enclaves
- Design of SGXBounds
- Implementation of SGXBounds

- Evaluation
  - Benchmark suites
  - Case studies
  - Security
### Benchmark Suites

<table>
<thead>
<tr>
<th></th>
<th>ASan</th>
<th>MPX</th>
<th>SGXBounds</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>
</tbody>
</table>
### Benchmark Suites

<table>
<thead>
<tr>
<th></th>
<th>ASan</th>
<th>MPX</th>
<th>SGXBounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phoenix</td>
<td>1.41</td>
<td>2.27</td>
<td>1.13</td>
</tr>
</tbody>
</table>
### Benchmark Suites

<table>
<thead>
<tr>
<th></th>
<th>ASan</th>
<th>MPX</th>
<th>SGXBounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phoenix</td>
<td>1.41</td>
<td>2.27</td>
<td>1.13</td>
</tr>
<tr>
<td>PARSEC</td>
<td>1.60</td>
<td>1.43*</td>
<td>1.20</td>
</tr>
</tbody>
</table>

* some programs failed due to insufficient memory
## Benchmark Suites

<table>
<thead>
<tr>
<th></th>
<th>ASan</th>
<th>MPX</th>
<th>SGXBounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phoenix</td>
<td>1.41</td>
<td>2.27</td>
<td>1.13</td>
</tr>
<tr>
<td>PARSEC</td>
<td>1.60</td>
<td>1.43*</td>
<td>1.20</td>
</tr>
<tr>
<td>SPEC</td>
<td>1.76</td>
<td>1.52*</td>
<td>1.41</td>
</tr>
</tbody>
</table>

* some programs failed due to insufficient memory
Case Studies

☹ MPX: EPC thrashing on Memcached
Case Studies

☹ **MPX**: EPC thrashing on Memcached

☹ **ASan**: metadata overload on Nginx
Case Studies

……………

☹ MPX: EPC thrashing on Memcached
☹ ASan: metadata overload on Nginx
☺ SGXBounds: no corner cases
Security guarantees

😊 **RIPE** synthetic benchmark:

→ **Similar guarantees** as ASan and MPX
Security guarantees

😊 RIPE synthetic benchmark:
  ➔ Similar guarantees as ASan and MPX

😊 Real-world vulnerabilities detected and tolerated:
  ➔ Memcached denial-of-service
  ➔ Nginx stack buffer overflow
  ➔ Apache Heartbleed
- Motivation
- Constraints of SGX enclaves
- Design of SGXBounds
- Implementation of SGXBounds
- Evaluation
Conclusion

• **Security** is barrier to adoption of cloud computing
  ➔ Use **shielded execution** with Intel SGX
• **Security** is barrier to adoption of cloud computing
  ➞ Use *shielded execution* with Intel SGX

• Insufficient to protect against **SW vulnerabilities**
  ➞ Use *memory defense* like ASan and MPX
Conclusion

• **Security** is barrier to adoption of cloud computing
  ➡️ Use *shielded execution* with Intel SGX

• Insufficient to protect against **SW vulnerabilities**
  ➡️ Use *memory defense* like ASan and MPX

• ASan and MPX **perform poorly** in SGX enclaves
  ➡️ Their memory assumptions are **violated in SGX**
Conclusion

• Security is barrier to adoption of cloud computing
  ➡️ Use shielded execution with Intel SGX

• Insufficient to protect against SW vulnerabilities
  ➡️ Use memory defense like ASan and MPX

• ASan and MPX perform poorly in SGX enclaves
  ➡️ Their memory assumptions are violated in SGX

• SGXBounds: memory safety for shielded execution
Conclusion

• **Security** is barrier to adoption of cloud computing
  ➞ Use **shielded execution** with Intel SGX

• Insufficient to protect against **SW vulnerabilities**
  ➞ Use **memory defense** like ASan and MPX

• ASan and MPX **perform poorly** in SGX enclaves
  ➞ Their memory assumptions are **violated in SGX**

• **SGXBounds**: memory safety for shielded execution

Thank you!

dmitrii.kuvaiskii@tu-dresden.de
https://github.com/tudinfse/sgxbounds
Backup slides
Intel SGX and SCONE

Virtual Address Space

SGX Enclave (legacy app with SCONE)

- Asynchronous syscalls

SCONE

- Application
- Libraries
  - Network shield
  - FS shield
  - M:N user threading
  - Modified Musl C library

-V. Costan, S. Devadas. „Intel SGX Explained“. IACR Cryptology ePrint Archive '16
-S. Arnautov et al. „SCONE: Secure linux containers with Intel SGX“. OSDI'16

Eurosys 2017
SGXBounds: Implementation

Native

\[ a = \text{add} \ x, i \]

\[ \text{store} \ 42, a \]

SGXBounds

\[ x = \text{specify\_bounds}(x, x+N) \]

\[ a = \text{add} \ x, i \]

\[ \text{aptr} = \text{extract\_ptr}(a) \]

\[ \text{UB} = \text{extract\_upper\_bound}(a) \]

\[ \text{LB} = \text{load\_lower\_bound}(\text{UB}) \]

\[ \text{if} (\text{aptr} < \text{LB} \ \text{or} \ \text{aptr} \geq \text{UB}) : \]

\[ \text{handle\_error}(\text{aptr}) \]

\[ \text{store} \ 42, a \]
## Related Work (SPEC CPU2006 outside of SGX enclave)

<table>
<thead>
<tr>
<th></th>
<th>Perf</th>
<th>Mem</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel MPX</td>
<td>146%</td>
<td>116%</td>
<td>FP/FN for multithreaded</td>
</tr>
<tr>
<td>AddressSanitizer</td>
<td>38%</td>
<td>292%</td>
<td>–</td>
</tr>
<tr>
<td>BaggyBounds(^1)</td>
<td>70%</td>
<td>12%</td>
<td>Not publicly available</td>
</tr>
<tr>
<td>Low-Fat Pointers(^2)</td>
<td>54%</td>
<td>12%</td>
<td>Not publicly available</td>
</tr>
<tr>
<td>SGXBounds</td>
<td>55%</td>
<td>0%</td>
<td>(this work)</td>
</tr>
</tbody>
</table>

\(^1\) P. Akritidis et al. „Baggy Bounds Checking: An efficient and backwards-compatible defense against out-of-bounds errors“. Usenix Security'09

\(^2\) G. Duck et al. „Stack Bounds Protection with Low Fat Pointers“. NDSS'17
Instrumentation:

data: lower bound metadata after each allocated object
pointers: upper bound metadata in each data pointer
code: bounds-check before each memory access
## Security guarantees

<table>
<thead>
<tr>
<th></th>
<th>MPX</th>
<th>ASan</th>
<th>SGXBounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIPE benchmark</td>
<td>2/16</td>
<td>8/16</td>
<td>8/16</td>
</tr>
<tr>
<td>Memcached CVE-2011-4971</td>
<td>D (T)</td>
<td>D (T)</td>
<td>D (T)</td>
</tr>
<tr>
<td>Nginx CVE-2013-2028</td>
<td>D (T)</td>
<td>D (T)</td>
<td>D (T)</td>
</tr>
<tr>
<td>Apache Heartbleed</td>
<td>D (T)</td>
<td>D (T)</td>
<td>D (T)</td>
</tr>
</tbody>
</table>

D detected?  
T tolerated?
Case Studies: Full Picture

(a) Memcached key-value store

(b) Apache web server

(c) Nginx web server

Latency (ms)

Throughput ($\times 10^3$ msg/s)
## Classes of Defenses against Attacks

<table>
<thead>
<tr>
<th></th>
<th>CF</th>
<th>DO</th>
<th>IL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Flow Integrity [27, 39, 52, 84]</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Code Pointer Integrity [46]</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Address Space Randomization [45, 48, 50, 68, 70]</td>
<td>✔️*</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Data Integrity [16]</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>Data Flow Integrity [29]</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>Software Fault Isolation [39, 79]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Data Space Randomization [24, 28]</td>
<td>✔️*</td>
<td>✔️*</td>
<td>✔️*</td>
</tr>
<tr>
<td>Memory safety [9, 17, 20, 26, 35, 55, 58, 69]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

*SGX enclaves do not provide sufficient bits of entropy in random offsets/masks

**CF** – control flow hijack, **DO** – data-only attack, **IL** – information leak
SGXBounds and Boundless Memory

![Diagram showing memory bounds and cache structures](image)

Lower Bound (LB)  Upper Bound (UB)

mapping: aligned(p) -> chunk

LRU cache

chunk  chunk  chunk

\[ (p + \text{offset}) < \text{LB} \]  \( (p + \text{offset}) \geq \text{UB} \}

out-of-bounds

\( *(p + \text{offset}) \)

\[ \text{out-of-bounds access} \]

\[ \text{redirect access} \]

1 M. Rinard et al. „A dynamic technique for eliminating buffer overflow vulnerabilities (and other memory errors)“. ACSAC'04
SGXBounds: Outside of Enclaves

The graph shows normalized runtime (w.r.t. native) for various applications under different conditions:

- **MPX**
- **AddressSanitizer**
- **SGXBounds**

The graph includes applications such as:
- astar
- bzip2
- gobmk
- h264
- hmmr
- libm
- libq
- mcf
- mic
- namd
- sjeng
- sphinx3
- xalanc
- gmean

The x-axis represents different applications, and the y-axis shows normalized runtime. The graph also highlights a significant increase of 15x for an unknown application, marked as 'xalanc'.