MoLE: Mitigation of Side-channel Attacks against SGX via Dynamic Data Location Escape

Fan Lang$^{1,2}$, Wei Wang$^1$, Lingjia Meng$^{1,2}$, Jingqiang Lin$^3$, Qiongxiao Wang$^1$, Linli Lu$^1$

1 State Ley Laboratory of Information Security, Institute of Information Engineering, China Academy of Sciences, Beijing 100089, China
2 School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100089, China
3 School of Cyber Security, University of Sciences and Technology of China, Hefei 230027, Anhui, China
Abstract

Software Guard eXtension (SGX) is a set of instructions and mechanisms for memory accesses added to Intel architecture processors, which aim to provide integrity and confidentiality assurance.

Unfortunately, the security mechanism still has weakness, such as side-channel attacks (SCAs).

Therefore, we propose MoLE, a dynamic data location randomization scheme to defend against SCAs and transient execution attacks that target sensitive data within enclaves. By continuously obfuscating the location of sensitive data at runtime, MoLE prevents the adversary from directly obtaining or disclosing data based on data access patterns.
SGX (Software Guard Extensions)

- SGX can provide applications trusted execution environments, called enclave
  - Isolated from the OS by the hardware processor
  - Encrypted using processor-specific keys
  - Ensure the confidentiality and integrity of data and code in enclave

- Vulnerable to SCA
Cache SCA

- Characteristics of cache side-channel attacks
  - No need to access the code or data in the isolated execution environment
  - By analyzing the cache side-channel information to indirectly obtain the key information

- Such as, Prime + Probe
Characteristics of page table side-channel attacks

- No need to access the target pages
- By analyzing the page table side-channel information to indirectly obtain the key information

Such as,
Controlled-Channel

<table>
<thead>
<tr>
<th>Word</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>word1</td>
<td>A, D</td>
</tr>
<tr>
<td>word2</td>
<td>B, D</td>
</tr>
<tr>
<td>word3</td>
<td>A, E</td>
</tr>
<tr>
<td>word4</td>
<td>B, D, F</td>
</tr>
</tbody>
</table>
Meltdown-type Attack

- Characteristics of page table side-channel attacks
  - Getting unauthorized access to the data with transient execution driven by fault or assist
  - Combining Flush + Reload Cache SCA

- Such as, Foreshadow
Basic idea

- Constantly changing location of sensitive data at runtime to obfuscate the data access patterns

Challenge

- How to achieve untraceable data obfuscation against privileged attackers with single-step capability?

- How to design a defense that is generalizable and relatively transparent to the target application?
Technology

➢ TSX (Transactional Synchronization Extension)

➢ TSX is Intel’s implementation of hardware transactional memory (HTM)
  ➢ Operations within a transaction are atomic
  ➢ An interruption or exception causes a rollback
  ➢ XBEGIN, XEND, XABORT, and XTEST

➢ Support for multi-threading
➢ Resistant to single-step behavior
Threat Model

- Attackers have full control over the OS
  - Generate interruptions to single-step the target application
  - Block, replay, read, and modify all message outside enclaves

- Attackers have the ability to launch the following attacks
  - Cache SCAs
  - Page table SCAs
  - Meltdown-type transient execution attacks

- Do not consider LLC SCAs
MoLE defeats attacks by altering the sensitive data’s location dynamically at runtime, which we call dynamic data location *Escape*.

- **Tunnels**
- **Escape function**

- Embraced in a single transaction
A designated portion of the enclave heap to store the sensitive data

A page list

Several doubly-linked lists

The Tunnels has $l$ (Each L1d cache set consists of $l$ cache lines) items in the page list
Escape Function

- Computing new addresses for data *Escape*

---

**Algorithm 1** The process of *Escape.*

**Input:** The page list of MoLE $PL$, $PL$ has $l$ items; The last location of data $L_O \in PL[i]$, $0 \leq i \leq l$ and the size of data $Size_{Data}$; The random numbers stored in RCX and RDX, $R_C$, $R_D$;

**Output:** The new location of data, $L_N$; The location of unrelated data, $L_U$;

1. $L_B = [L_O \oplus R_C]_{NearBlock}$; $L_B$ is the nearest free block to the result;
2. $L_N = L_B + R_D \mod (Size_B - Size_{Data})$, $L_N \in PL[i]$, $0 \leq j \leq l$; $Size_B$ is the size of $L_B$;
3. for $k = 0 \rightarrow l - 1$ do
4. $L_U[k] = PL[k] + (L_N - PL[j] + k \times Size_{Data}) \mod (Size_{PL[j]} - Size_{Data})$, $k \neq i, k \neq j$;
5. end for

- Feint accesses
## Implementation

➢ **Runtime Library**

<table>
<thead>
<tr>
<th>Interface</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>int MoLEInit()</td>
<td>(ECALL) Allocate memory for the <strong>Tunnels</strong> and organize it, RET: the <strong>Tunnels</strong> status</td>
</tr>
<tr>
<td>void* MoLEMalloc(uint64_t ptr_size)</td>
<td>Allocate buffer from the <strong>Tunnels</strong>, RET: allocation pointer, ptr_size: allocation size</td>
</tr>
<tr>
<td>uint64 MoLEMallocSize(void* ptr)</td>
<td>Get allocation buffer size, RET: buffer size, ptr: buffer pointer</td>
</tr>
<tr>
<td>void MoLEEscape(void* ptr)</td>
<td>If the buffer access reaches the access threshold, data <strong>Escapes</strong>, ptr: buffer pointer</td>
</tr>
<tr>
<td>void* MoLERealloc(void* ptr, uint64_t ptr_size)</td>
<td>Expand buffer size, RET: new buffer pointer, ptr: original pointer, ptr_size: new buffer size</td>
</tr>
<tr>
<td>void MoLEFree(void* ptr)</td>
<td>Release buffer to the <strong>Tunnels</strong>, ptr: buffer pointer</td>
</tr>
</tbody>
</table>

➢ **Encapsulating the operations for the **Tunnels** and the **Tunnels** data structure**
LLVM Pass

%4 = alloca [64 x i8], align 4  
%5 = tail call noalias i8* @malloc(i8 1024)  
...  
store i8* %5, i8** %23, align 8  
%24 = bitcast [64 x i8]* %4 to i8*  
...  
call void @free(i8* %4)  
call void @free(i8* %6)

%4 = tail call noalias i8* @MoLEMalloc(i8 64)  
%5 = tail call noalias i8* @MoLEMalloc(i8 1024)  
...  
%23 = getelementptr inbounds i8*, i8** %5, i32 0  
%24 = bitcast i8** %23 to i8*call void @MoLEEscape(i8* %24)  
store i8* %5, i8** %25, align 8  
%26 = getelementptr inbounds i8*, i8** %4, i32 0  
%27 = bitcast i8** %26 to i8*call void @MoLEEscape(i8* %27)  
%28 = bitcast [64 x i8]* %4 to i8*  
...  
call void @MoLEFree(i8* %4)  
call void @MoLEFree(i8* %6)

Instrumenting the application enclave code about the annotation variables at the intermediate representation (IR) level
Taint Tracking

There may be taint propagation when accessing sensitive data by some instructions

- load
- call
Analysis

- Cache SCA
  - Prime + Probe
    - *Escape* evicts the preloaded data of various cache lines
- Page table SCA
  - *Escape* would allow sensitive data to be located on any page in the *Tunnels*
- Meltdown-type attack
  - *Escape* make it difficult to get the exact location of the data
- Single-step tracking
  - TSX will roll operations within a transaction back to invalidate single-step tracking
Evaluation

➢ Cache SCA

➢ As the access threshold $T$ decreases, the cache access pattern becomes chaotic

(a) Without Escape
(b) The threshold of Escape $T = 256$
(c) The threshold of Escape $T = 128$
Evaluation

➢ Page table SCA

➢ With the *Escape*, the patterns of these two variables become confusing

(a) Without *Escape*  (b) The threshold of *Escape* $T = 256$  (c) The threshold of *Escape* $T = 128$
Evaluation

- Meltdown-type attack
- Foreshadow
- The amount of stolen data decreases as the threshold $T$ decrease
Evaluation

Nbench

When $T \in \{4, 8, 16, 32, 64, 128\}$, the overhead is $7.89 \times$, $4.78 \times$, $3.14 \times$, $2.29 \times$, $1.89 \times$, and $1.48 \times$. 
Evaluation

- Nbench
- Data size
- Multi-threading
Related Work

- Limiting concurrency
  - Usage restrictions (multi-threading)
- Obfuscation
  - Weak for single-step tracking

<table>
<thead>
<tr>
<th>Defensive concept</th>
<th>Scheme</th>
<th>Protection scope</th>
<th>Muti-threading</th>
<th>Single-step tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limit interruption or concurrency</td>
<td>Déjà Vu [14]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Racing [13]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>VARYS [32]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Isolate or obfuscate shared resources</td>
<td>T-SGX [39]</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Cloak [19]</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>KLOTSKI [48]</td>
<td>✓</td>
<td>x</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>OBFSUCRO [5]</td>
<td>✓</td>
<td>x</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>DR.SGX [6]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>MoLE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- Limited defense range
Thank you!