Time and Order: Towards Automatically Identifying Side-Channel Vulnerabilities in Enclave Binaries

Wubing Wang, Yinqian Zhang, and Zhiqiang Lin

Department of Computer Science and Engineering
The Ohio State University

RAID 2019
Objective
Intel SGX

Diagram showing the structure of Intel SGX components:
- Hardware
- VMM
- OS
- App
- App
Intel SGX side-channel attacks - Granularity

1. Different Granularities
2. Different Targets
Intel SGX side-channel attacks - Granularity

Program

Enclave

Secrets

Page-Level Attacks

Cache-Level Attacks

Basic Block-Level Attacks
Intel SGX side-channel attacks - Granularity

Program

Enclave

Secrets

Page-Level Attacks

Cache-Level Attacks

Basic Block-Level Attacks
Page-Level Attacks

1. Approaches to observe page-level pattern
2. The page-level vulnerability
Page-Level Attacks

- Present bit
- Reserve bit
- EPC page access
- Page Fault Exception
- NX bit

...
Page-Level Attacks

Barcode:
Page-Level Attacks

Barcode: [black line, white line, black line, black line, black line]
Page-Level Attacks

White line:

```cpp
1 void generate_barcode()
2  { for (i=0; i<barcode_size; i++)
3  if (x[i] == 0)
4  render_whiteline();
5  } else {
6  render_blackline();
7  }
8  }
9 }

10 int render_whiteline()
11  { ....
12  }

13 int render_blackline()
14  { ....
15  }
```

Black line:
Page-Level Attacks

Barcode: [black line]

Page Sequence:
  page 0, page 1, page 0
### Page-Level Attacks

**Barcode:**

[black line, white line]

**Page Sequence:**

page 0, page 1, page 0, page 2, page 0
Page-Level Attacks

Barcode: [black line, white line, black line]

Page Sequence:
page 0, page 1, page 0, page 2, page 0, page 1, page 0
Page-Level Attacks

Barcode: [black line, white line, black line, black line]

Page Sequence:
page 0, page 1, page 0, page 2, page 0, page 1,
page 0, page 1, page 0
Page-Level Attacks

Barcode: [black line, white line, black line, black line, black line]

Page Sequence:
page 0, page 1, page 0, page 2, page 0, page 1,
page 0, page 1, page 0, page 1, page 0
Cache-Level Attacks

Program

Enclave

Secrets

Page-Level Attacks

Cache-Level Attacks

Basic Block-Level Attacks
Cache-Level Attacks

1. Approaches to observe cache-level pattern
2. The cache-level vulnerability
# Cache-Level Attacks

## Prime + Probe

1. Occupy specific cache set  
2. Victim program is scheduled  
3. Check which cache sets are still occupied

## Flush + Reload

1. Map binary into address space  
2. Flush a cache line from the cache  
3. Victim program is scheduled  
4. Check Whether the flushed cache line has been reloaded
## Cache-Level Attacks

### Prime + Probe
1. Occupy specific cache set
2. Victim program is scheduled
3. Check which cache sets are still occupied

### Flush + Reload
1. Map binary into address space
2. Flush a cache line from the cache
3. Victim program is scheduled
4. Check Whether the flushed cache line has been reloaded

Not applicable: SGX do not share memory with external!
Cache-Level Attacks

page 0

4096 B

64 B

64

cache 0

... cache 63

64
Cache-Level Attacks

Program

Enclave

Secrets

Page-Level Attacks

Cache-Level Attacks

Basic Block-Level Attacks
Attack Targets

Program Inputs (e.g., Hunspell, Libjpeg, Freetype, Apache)
  Controlled-channel (S&P’15), Branch Shadowing (USENIX’17)

Encrypted Data (e.g., Padding Oracle attack & Bleichenbacher attack)
  Stacco (CCS’17)

Cryptography Key [e.g., RSA, DSA, AES]
  DATA (USENIX’18), MicroWalk (ACSAC’18), CacheD (USENIX’17)

Genomic sequences
  Software Grand Exposure(WOOT’17)
Motivations

The timing information is not thoroughly used

No automatic tool to detect the side-channel attack in general
Motivations

The timing information is not thoroughly used

“An analysis of covert timing channels” John C. Wray 1992:

Both storage nature (order) and timing nature are attributes of the channel, and a given channel may posses either or both.
Motivations

Storage nature (order):

Input 1: page 0, page 1, page 0, page 2

Input 2: page 0, page 2, page 0, page 1
### Motivations

**Timing nature:**

<table>
<thead>
<tr>
<th>Input 1:</th>
<th>page 0,</th>
<th>page 1,</th>
<th>page 0,</th>
<th>page 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 ns</td>
<td>5 ns</td>
<td>2 ns</td>
<td>4 ns</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input 2</th>
<th>page 0,</th>
<th>page 1,</th>
<th>page 0,</th>
<th>page 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 ns</td>
<td>50 ns</td>
<td>2 ns</td>
<td>4 ns</td>
</tr>
</tbody>
</table>
Motivations

Input - execution mapping
Motivations

Keys:
- Cryptography libraries

Texts:
- Freetype

Words:
- Hunspell

Jpeg Figures:
- Libjpeg
Motivations
Challenges

1. How to accurately measure the timing information
2. What is the relationship between each input with the whole input set and other inputs
ANABLEPS

Input Generation → Graph Generation → Vulnerability Detection
ANABLEPS

- Input Generation
- Graph Generation
- Vulnerability Detection
Input Generation

1. Program → Fuzzing → Seed inputs
Input Generation

1. Program
2. Fuzzing
3. Symbolic Execution
Input Generation

1. Program
2. Seed inputs
3. Symbolic Execution

Fuzzing
Input Generation
Input Space

- Program
- Symbolic Execution
- Constraints
- Solver
- Inputs
- ANABLEPS
- Evaluation
- Related Work
- Summary

Introduction
Motivations
ANABLEPS
Evaluation
Related Work
Summary
Input Space

Program

Symbolic Execution

Constraints

Solver

#Inputs
Input Space
Input Space

#Inputs

- $\#\text{Inputs} = 1$
- $\#\text{Inputs} > 1$

$== 1$ \rightarrow Full Leaked

$> 1$ \rightarrow Partial Leaked
Input Space

#Inputs

- **<= 1**
  - Full Leaked

- **> 1**
  - Partial Leaked
Dynamic Control-Flow Graph
Extended Dynamic Control-Flow Graph (ED-CFG)

Node 0

Order: [node 2, node 3]
Time: [(0.8 ± 0.1),
(0.7 ± 0.1)]

Node 1

Node 2

Node 3

Node 4
Extended Dynamic Control-Flow Graph (ED-CFG) Generation
Extended Dynamic Control-Flow Graph (ED-CFG) Generation

Input

Program

Intel PT

Execution Trace

Execution Time
Extended Dynamic Control-Flow Graph (ED-CFG) Generation

- Intel PT
  - Control-flow Reconstruct
  - Execution Timing Reconstruct
  - Low Performance Overhead
## Extended Dynamic Control-Flow Graph (ED-CFG) Generation

<table>
<thead>
<tr>
<th>Execution Trace</th>
<th>[0x400a08, 0x400c9b, 0x400ce0, ... ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time</td>
<td>[10, 23, 25, ...]</td>
</tr>
</tbody>
</table>
Extended Dynamic Control-Flow Graph (ED-CFG) Generation

Execution Time

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[10,</td>
<td>23,</td>
<td>25, ...</td>
</tr>
<tr>
<td>[11,</td>
<td>22,</td>
<td>25, ...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>[10,</td>
<td>21,</td>
<td>24, ...</td>
</tr>
<tr>
<td>[9,</td>
<td>23,</td>
<td>25, ...</td>
</tr>
</tbody>
</table>

Mean: [10, 12, 3, ...]

Std: [0.8, 0.9, 0.6, ...]
Extended Dynamic Control-Flow Graph (ED-CFG) Generation
Extended Dynamic Control-Flow Graph (ED-CFG) Generation

Order: \([cb1, cb2]\)
Time: \([(1.3 \pm 0.2), (1.2 \pm 0.1)]\)

Order: \([bb2, bb3]\)
Time: \([(0.8 \pm 0.1), (0.7 \pm 0.1)]\)

Order: \([pb1, pb1]\)
Time: \([(2.5 \pm 0.3), (2.3 \pm 0.2)]\)

Basic Block Level ED-CFG
- Instruction virtual addresses

Cache Level ED-CFG
- virtual address / cacheline size (64 B)

Page Level ED-CFG
- virtual address / page size (4096 B)
<table>
<thead>
<tr>
<th>Introduction</th>
<th>Motivations</th>
<th>ANABLEPS</th>
<th>Evaluation</th>
<th>Related Work</th>
<th>Summary</th>
</tr>
</thead>
</table>

**ANABLEPS**

1. **Input Generation**
2. **Graph Generation**
3. **Vulnerability Detection**
The vulnerability detection - order-based

ED-CFG 1

Node 0

Node 1

Node 2

Node 3

Node 4

Order: [node2, node3]
Time: [0.8 ± 0.7]

ED-CFG 2

Node 0

Node 1

Node 2

Node 3

Node 4

Order: [node3, node2]
Time: [0.5 ± 0.9]
The vulnerability detection - order-based

ED-CFG 1

Node 0

Node 1

Node 2

Node 3

Node 4

Order: [node2, node3]
Time: [0.8 ± 0.7]

ED-CFG 2

Node 0

Node 1

Node 2

Node 3

Node 4

Order: [node3, node2]
Time: [0.5 ± 0.9]
The vulnerability detection - time-based

ED-CFG 3

Order: [node4, node4]
Time: [(0.8 ± 0.1), (0.7 ± 0.1)]

ED-CFG 4

Order: [node4, node4]
Time: [(1.5 ± 0.3), (0.9 ± 0.1)]
The vulnerability detection - time-based
Evaluation

1. Detection Results
2. Case Studies
## Detection Results

<table>
<thead>
<tr>
<th>Programs</th>
<th>Functionalities Under Test</th>
<th>Cache Level</th>
<th>Page Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#Nodes</td>
<td>#Order-Based Vulnerable Nodes</td>
</tr>
<tr>
<td>Deep Learning</td>
<td>dA</td>
<td>69</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>SdA</td>
<td>109</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>DBN</td>
<td>126</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>RBM</td>
<td>68</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>LogisticRegression</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>gsl</td>
<td>Sort</td>
<td>31</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Permutation</td>
<td>99</td>
<td>30</td>
</tr>
<tr>
<td>Hunspell</td>
<td>Spell checking</td>
<td>302</td>
<td>48</td>
</tr>
<tr>
<td>PNG</td>
<td>PNG Image Render</td>
<td>640</td>
<td>170</td>
</tr>
<tr>
<td>Freetype</td>
<td>Character Render</td>
<td>1054</td>
<td>263</td>
</tr>
<tr>
<td>Bio-rainbow</td>
<td>Bioinfo Clustering</td>
<td>214</td>
<td>16</td>
</tr>
<tr>
<td>QRcodegen</td>
<td>Generatee QR</td>
<td>176</td>
<td>32</td>
</tr>
<tr>
<td>Genometools</td>
<td>bed to gff3 conversion</td>
<td>1901</td>
<td>231</td>
</tr>
</tbody>
</table>
Evaluation

1. Detection Results
2. Case Studies
## Detection Results - dA deep learning algorithm

<table>
<thead>
<tr>
<th>Programs</th>
<th>Functionalities Under Test</th>
<th>Cache Level</th>
<th>Page Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#Nodes</td>
<td>#Order-Based Vulnerable Nodes</td>
</tr>
<tr>
<td>Deep Learning</td>
<td>dA</td>
<td>69</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>SdA</td>
<td>109</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>DBN</td>
<td>126</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>RBM</td>
<td>68</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>LogisticRegression</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>gsl</td>
<td>Sort</td>
<td>31</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Permutation</td>
<td>99</td>
<td>30</td>
</tr>
<tr>
<td>Hunspell</td>
<td>Spell checking</td>
<td>302</td>
<td>48</td>
</tr>
<tr>
<td>PNG</td>
<td>PNG Image Render</td>
<td>640</td>
<td>170</td>
</tr>
<tr>
<td>Freetype</td>
<td>Character Render</td>
<td>1054</td>
<td>263</td>
</tr>
<tr>
<td>Bio-rainbow</td>
<td>Bioinfo Clustering</td>
<td>214</td>
<td>16</td>
</tr>
<tr>
<td>QRcodegen</td>
<td>Generatee QR</td>
<td>176</td>
<td>32</td>
</tr>
<tr>
<td>Genometools</td>
<td>bed to gff3 conversion</td>
<td>1901</td>
<td>231</td>
</tr>
</tbody>
</table>
Detection Results - dA deep learning algorithm

```c
1 int binomial(int n, double p){
2   ...
3   for (i=0; i<n; i++){  
4     r = rand() / (RAND_MAX + 1.0)
5     if (r < p) c++;
6   }
7   ....
8 }

9 void dA_get_corrupted_input(dA* this, int* x, int* tilde_x, double p){  
10   int i;
11   for (i=0; i<this->n_visible; i++){
12     if (x[i] == 0){
13       tilde_x[i] = 0;
14     } else {
15       tilde_x[i] = binomial(x[i], p);
16     }
17   }
18 }
```
Detection Results - dA deep learning algorithm

```c
int binomial(int n, double p) {
    ...
    for (i=0; i<n; i++) {
        r = rand() / (RAND_MAX + 1.0)
        if (r < p) c++;
    }
    ...
}

void dA_get_corrupted_input(dA* this, int* x, int* tilde_x, double p) {
    int i;
    for (i=0; i<this->n_visible; i++) {
        if (x[i] == 0) {
            tilde_x[i] = 0;
        } else {
            tilde_x[i] = binomial(x[i], p);
        }
    }
}
```
Detection Results - dA deep learning algorithm

```c
1 int binomial(int n, double p){
2    ...;
3    for (i=0; i<n; i++){
4        r = rand() / (RAND_MAX + 1.0)
5        if (r < p) c++;
6    }
7    ...;
8 }
9
10 void dA_get_corrupted_input(dA* this, int* x, int* tilde_x, double p){
11    int i;
12    for (i=0; i<this->n_visible; i++){
13        if (x[i] == 0){
14            tilde_x[i] = 0;
15        } else {
16            tilde_x[i] = binomial(x[i], p);
17        }
18    }
```
Detection Results - dA deep learning algorithm

```
1 int binomial(int i, double p) {
2   ...
3   for (i=0; i<i; i++){
4     r = rand() / (RAND_MAX + 1.0)
5     if (r < p) c++;
6   }
7   ....
8 }

9 void dA_get_corrupted_input(dA* this, int* x, int* tilde_x, double p) {
10   int i;
11   for (i=0; i<this->n_visible; i++){
12     if (x[i] == 0) {
13       tilde_x[i] = 0;
14     } else {
15       tilde_x[i] = binomial(x[i], p);
16     }
17   }
18 }
```
### Detection Results - Sorting algorithm

<table>
<thead>
<tr>
<th>Programs</th>
<th>Functionalities Under Test</th>
<th>Cache Level</th>
<th>Page Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Nodes</td>
<td>#Order-Based Vulnerable Nodes</td>
<td>#Nodes</td>
</tr>
<tr>
<td>Deep Learning</td>
<td>dA</td>
<td>69</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>SdA</td>
<td>109</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>DBN</td>
<td>126</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>RBM</td>
<td>68</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>LogisticRegression</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>gsl</td>
<td>Sort</td>
<td>31</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Permutation</td>
<td>99</td>
<td>30</td>
</tr>
<tr>
<td>Hunspell</td>
<td>Spell checking</td>
<td>302</td>
<td>48</td>
</tr>
<tr>
<td>PNG</td>
<td>PNG Image Render</td>
<td>640</td>
<td>170</td>
</tr>
<tr>
<td>Freetype</td>
<td>Character Render</td>
<td>1054</td>
<td>263</td>
</tr>
<tr>
<td>Bio-rainbow</td>
<td>Bioinfo Clustering</td>
<td>214</td>
<td>16</td>
</tr>
<tr>
<td>QRcodegen</td>
<td>Generatee QR</td>
<td>176</td>
<td>32</td>
</tr>
<tr>
<td>Genometools</td>
<td>bed to gff3 conversion</td>
<td>1901</td>
<td>231</td>
</tr>
</tbody>
</table>
Detection Results - Sorting algorithm

Page Node 0

Page Node 1
Detection Results - Sorting algorithm

Page Node 0

Page Node 1
Detection Results - Sorting algorithm
Detection Results - Sorting algorithm

Page Node 0

Page Node 1
Detection Results - Sorting algorithm

10 -> 9 -> 7 -> 6 -> 5 -> 4 -> 8 -> 3
Detection Results - Sorting algorithm

Time: [(19, 0.8)]
Detection Results - Sorting algorithm

Node 0

Page

Time: [(19, 0.8)]

Node 1

3
Detection Results - Sorting algorithm

Time: [(19, 0.8), (5, 0.7)]
Related Work

1. **Stacco**: Differentially Analyzing Side-Channel Traces for Detecting SSL/TLS Vulnerabilities in Secure Enclaves.
   Yuan Xiao, Mengyuan Li, Sanchuan Cheng, and Yinqian Zhang

   Jan Wichelmann, Ahmad Moghimi, Thomas Eisenbarth, and Berk Sunar

   Samuel Weiser, Andreas Zankl, Raphael Spreitzer, Katja Miller, Stefan Mangard, and Georg Sigl

4. **CacheD**: Identifying Cache-Based Timing Channels in Production Software.
   Shuai Wang, Pei Wang, Xiao Liu, Danfeng Zhang, and Dinghao Wu
Conclusion

1 New insights: With the time information, attacker could get more secret data than only order information.

2 New methods: Use the fuzzing and symbolic execution to generate inputs and quantify the leakage is a new attempt.

3 New tools: ANABLEPS is an automatically program analysis tool, and will be released to the community. [github.com/OSUSecLab/ANABLEPS](https://github.com/OSUSecLab/ANABLEPS)
Time and Order: Towards Automatically Identifying Side-Channel Vulnerabilities in Enclave Binaries

Wubing Wang, Yinqian Zhang, and Zhiqiang Lin

Department of Computer Science and Engineering
The Ohio State University

RAID 2019