Distributed Operating Systems

Side-Channels

Marcus Hähnel (marcus.haehnel@kernkonzept.com)

2023-07-03
What is a Side-Channel?
Introduction

Internal Attack Vectors

External Attack Vectors

Data remanence

Defense

Conclusion

What is a Side-Channel?

Visual side-channel

Which call has a positive connotation?
Definition

A side-channel is an *unintended* information source which enables the *extraction* of information that is processed through a means of communication or computation.

1. The presence of the side-channel does not depend on the presence of bugs.
A side-channel is an *unintended* information source which enables the *extraction* of information that is processed through a means of communication or computation.

**Phone example**

- **Primary source**: Audio signal
- **Unintended source**: Visual information (e.g. facial expression, lip movement)

1. The presence of the side-channel does not depend on the presence of bugs
2001: A Space Odysee — Video
Covert channels?

Definition: Side-Channel
A side-channel is an unintended information source which enables the extraction of information that is processed through a means of communication or computation.

Definition: Covert-Channel
A covert-channel is an unintended means of communication between two cooperating programs or systems.
Definition: Side-Channel

A side-channel is an unintended information source which enables the extraction of information that is processed through a means of communication or computation.
Definition: Side-Channel
A side-channel is an unintended information source which enables the extraction of information that is processed through a means of communication or computation.

Definition: Covert-Channel
A covert-channel is an unintended means of communication between two cooperating programs or systems.
Side-Channel usage

Introduction

Internal Attack Vectors

External Attack Vectors

Data remanence

Defense

Conclusion

DOS - Side-Channels

Introduction

Side-Channel usage
Malicious

Extracting ...

- ... other customers data across virtual machines
Side-Channel usage

Malicious

Extracting ...

- ... other customers data across virtual machines
- ... crypto keys from applications in different address spaces

Benign

... detecting rootkits
... detecting hardware trojans
Side-Channel usage

Malicious

Extracting ...
- ... other customers data across virtual machines
- ... crypto keys from applications in different address spaces
- ... data from inaccessible processors
Side-Channel usage

**Malicious**

- Extracting ...
  - ... other customers data across virtual machines
  - ... crypto keys from applications in different address spaces
  - ... data from inaccessible processors

**Benign**

- ... detecting rootkits
Side-Channel usage

Malicious

- Extracting ...
  - ... other customers data across virtual machines
  - ... crypto keys from applications in different address spaces
  - ... data from inaccessible processors

Benign

- ... detecting rootkits
- ... detecting hardware trojans
Typical Side-Channels

What is a suitable side-channel

Any measurable parameter of the system and of its individual operations that changes depending on the processed data.

Example parameters:

- Time (Duration)
- Error behavior (Out of memory? No more file handles?)
- Microarchitectural state
- Power usage
- Radiation (Heat, EM-Radiation)
- Unexpected persistence of data (Cold-boot, memory re-use)
Typical Side-Channels

What is a suitable side-channel

Any measurable parameter of the system and of its individual operations that changes depending on the processed data.

Example parameters:
- Time (Duration)
- Error behavior (Out of memory? No more file handles?)
- Microarchitectural state
- Power usage
- Radiation (Heat, EM-Radiation)
- Unexpected persistence of data (Cold-boot, memory re-use)
Typical Side-Channels

What is a suitable side-channel

Any measureable parameter of the system and of its individual operations that changes depending on the processed data.
Typical Side-Channels

What is a suitable side-channel

Any measurable parameter of the system and of its individual operations that changes depending on the processed data.

Example parameters

- Time (Duration)
Typical Side-Channels

What is a suitable side-channel

Any measureable parameter of the system and of its individual operations that changes depending on the processed data.

Example parameters

- Time (Duration)
- Error behavior (Out of memory? No more file handles?)
Typical Side-Channels

What is a suitable side-channel
Any measureable parameter of the system and of its individual operations that changes depending on the processed data.

Example parameters
- Time (Duration)
- Error behavior (Out of memory? No more file handles?)
- Microarchitectural state
Typical Side-Channels

What is a suitable side-channel

Any measureable parameter of the system and of its individual operations that changes depending on the processed data.

Example parameters

- Time (Duration)
- Error behavior (Out of memory? No more file handles?)
- Microarchitectural state
- Power usage
Typical Side-Channels

What is a suitable side-channel
Any measureable parameter of the system and of its individual operations that changes depending on the processed data.

Example parameters
- Time (Duration)
- Error behavior (Out of memory? No more file handles?)
- Microarchitectural state
- Power usage
- Radiation (Heat, EM-Radiation)
What is a suitable side-channel

Any measurable parameter of the system and of its individual operations that changes depending on the processed data.

Example parameters

- Time (Duration)
- Error behavior (Out of memory? No more file handles?)
- Microarchitectural state
- Power usage
- Radiation (Heat, EM-Radiation)
- Unexpected persistence of data (Cold-boot, memory re-use)
Timing Channels

Attack vector

The duration of an attacker observable operation depends on the data processed by the victim.
Timing Channels

**Attack vector**

The duration of an attacker observable operation depends on the data processed by the victim.

**Example - Graphics Processing**

Holidays
Day 1
Timing Channels

**Attack vector**

The duration of an attacker observable operation depends on the data processed by the victim

**Example - Graphics Processing**

Holidays
Day 1
Timing Channels

Attack vector

The duration of an attacker observable operation depends on the data processed by the victim

Example - Graphics Processing

Holidays
Day 1

Convert to png: 1s vs. 17s
**Cache Side-Channel**

- **CPU**
  - Core 1
    - Thr 1
    - L1I
    - L1D
    - L2 Cache
  - Core 2
    - Thr 1
    - L1I
    - L1D
    - L2 Cache

**Levels**

- **L1D**: 32 KiB, 4 cycles
- **L1I**: 32 KiB, 4 cycles
- **L2**: 256 KiB, 12 cycles
- **L3**: 3 MiB, 36 cycles
- **DRAM**: Large, 250 cycles

---

**DOS - Side-Channels**

- **Internal Attack Vectors**
- **Timing Channels**
- **Cache Side-Channel**
Introduction

Internal Attack Vectors

External Attack Vectors

Data remanence

Defense

Conclusion

2023-07-04

DOS - Side-Channels

- Internal Attack Vectors
- Timing Channels
- Cache Side-Channel
Prime & Probe

Concept

- Fill cache with known data (Prime)
- Repeatedly measure how long it takes to access this data
- Longer duration means cache-line was "stolen"
Internal Attack Vectors

Example (Victim)

```c
struct Person {
    char name[56];
    double account;
} Alice, Bob;

void transact(Person& p) {
    p.account += 4000;
}

transact(Alice);
```

DOS - Side-Channels

- Internal Attack Vectors
- Timing Channels
- Prime & Probe
Prime & Probe

Example (Victim)

```c
struct Person {
    char name[56];
    double account;
} Alice, Bob;
```

L1D 8-way set cache

<table>
<thead>
<tr>
<th>Tag (20)</th>
<th>Set Index (6)</th>
<th>Offset (6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Alice)</td>
<td>0</td>
<td>56</td>
</tr>
<tr>
<td>(Bob)</td>
<td>1</td>
<td>56</td>
</tr>
</tbody>
</table>

DOS - Side-Channels

- Internal Attack Vectors
- Timing Channels
- Prime & Probe
Prime & Probe

Example (Victim)

```c
struct Person {
    char name[56];
    double account;
} Alice, Bob;
```

L1D 8-way set cache

<table>
<thead>
<tr>
<th>Tag (20)</th>
<th>Set Index (6)</th>
<th>Offset (6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Alice)</td>
<td>0</td>
<td>56</td>
</tr>
<tr>
<td>(Bob)</td>
<td>1</td>
<td>56</td>
</tr>
</tbody>
</table>

Attacker

Prime

Way
Set Index

---

DOS - Side-Channels
- Internal Attack Vectors
- Timing Channels
- Prime & Probe
Prime & Probe

Example (Victim)

```c
struct Person {
    char name[56];
    double account;
} Alice, Bob;
```

L1D 8-way set cache

<table>
<thead>
<tr>
<th>Tag (20)</th>
<th>Set Index (6)</th>
<th>Offset (6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Alice)</td>
<td>0</td>
<td>56</td>
</tr>
<tr>
<td>(Bob)</td>
<td>1</td>
<td>56</td>
</tr>
</tbody>
</table>

Attacker

Prime, Probe
Prime & Probe

Example (Victim)

```c
struct Person {
    char name[56];
    double account;
} Alice, Bob;
```

L1D 8-way set cache

<table>
<thead>
<tr>
<th>Tag (20)</th>
<th>Set Index (6)</th>
<th>Offset (6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Alice)</td>
<td>0</td>
<td>56</td>
</tr>
<tr>
<td>(Bob)</td>
<td>1</td>
<td>56</td>
</tr>
</tbody>
</table>

Attacker

Prime, Probe, Detect
Results of prime-probe observations for 20 distinct processed text words (rows). Darker fields indicate more evicted ways within an 8-way associativity set. Vertical lines identify cache addresses evicted in every observation.
Evict & Time

Prime & Probe shortcomings
- Hard with smart caches

Prime & Probe shortcomings
- Hard with smart caches

Alternative: Evict & Time
Possible if execution of victim code is under attacker control
Evict cache (by filling with known data)
Run victim and measure runtime
Evict most of the cache
Run victim again and measure time
Time difference tells if victim used non-evicted cache-line
Evict & Time

Prime & Probe shortcomings

- Hard with smart caches
- Probing is prone to many false positives
Evict & Time

Prime & Probe shortcomings

- Hard with smart caches
- Probing is prone to many false positives

Alternative: Evict & Time

- Possible if execution of victim code is under attacker control
Evict & Time

Prime & Probe shortcomings
- Hard with smart caches
- Probing is prone to many false positives

Alternative: Evict & Time
- Possible if execution of victim code is under attacker control
- Evict cache (by filling with known data)
Evict & Time

Prime & Probe shortcomings
- Hard with smart caches
- Probing is prone to many false positives

Alternative: Evict & Time
- Possible if execution of victim code is under attacker control
- Evict cache (by filling with known data)
- Run victim and measure runtime
Evict & Time

Prime & Probe shortcomings
- Hard with smart caches
- Probing is prone to many false positives

Alternative: Evict & Time
- Possible if execution of victim code is under attacker control
- Evict cache (by filling with known data)
- Run victim and measure runtime
- Evict most of the cache
Evict & Time

Prime & Probe shortcomings
- Hard with smart caches
- Probing is prone to many false positives

Alternative: Evict & Time
- Possible if execution of victim code is under attacker control
- Evict cache (by filling with known data)
- Run victim and measure runtime
- Evict most of the cache
- Run victim again and measure time
Evict & Time

Prime & Probe shortcomings
- Hard with smart caches
- Probing is prone to many false positives

Alternative: Evict & Time
- Possible if execution of victim code is under attacker control
- Evict cache (by filling with known data)
- Run victim and measure runtime
- Evict most of the cache
- Run victim again and measure time
- Time difference tells if victim used non-evicted cache-line
Challenges

Smart Caches

Smart Caches "reserve" parts of the L3 cache for individual cores. This makes priming hard.
Challenges

Smart Caches
Smart Caches "reserve" parts of the L3 cache for individual cores. This makes priming hard.

Prefetchers
Detect access patterns. Probing may cause prefetch of evicted line leading to false-negative.
Challenges

Smart Caches
Smart Caches "reserve" parts of the L3 cache for individual cores. This makes priming hard.

Prefetchers
Detect access patterns. Probing may cause prefetch of evicted line leading to false-negative.
Challenges

Smart Caches
Smart Caches "reserve" parts of the L3 cache for individual cores. This makes priming hard.

Prefetchers
Detect access patterns. Probing may cause prefetch of evicted line leading to false-negative.
Challenges

**Smart Caches**

Smart Caches "reserve" parts of the L3 cache for individual cores. This makes priming hard.

**Prefetchers**

Detect access patterns. Probing may cause prefetch of evicted line leading to false-negative.
Challenges

Smart Caches
Smart Caches "reserve" parts of the L3 cache for individual cores. This makes priming hard.

Prefetchers
Detect access patterns. Probing may cause prefetch of evicted line leading to false-negative.
Challenges

Smart Caches
Smart Caches "reserve" parts of the L3 cache for individual cores. This makes priming hard.

Prefetchers
Detect access patterns. Probing may cause prefetch of evicted line leading to false-negative.

Scheduling
May evict primed data leading to 'blind times'
# Pagefault Side-Channel

## Assumption

Removing the OS from the TCB
Assumption
Removing the OS from the TCB

Scenario: Shielding Systems
- InkTag: Hypervisor / paging based isolation between OS and Application
Pagefault Side-Channel

Assumption
Removing the OS from the TCB

Scenario: Shielding Systems
- InkTag: Hypervisor / paging based isolation between OS and Application
- Intel SGX: Hardware-based isolation through read-protected memory
Assumption
Removing the OS from the TCB

Scenario: Shielding Systems
- InkTag: Hypervisor / paging based isolation between OS and Application
- Intel SGX: Hardware-based isolation through read-protected memory

Vulnerability
- These systems don’t trust OS but use it to configure hardware
- OS makes a powerful adversary
Controlled Channel Attacks

First attack vector against Intel SGX

Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems

Yuanzhong Xu, Weidong Cui, and Marcus Peinado, MSR

System Model

- OS cannot directly observe memory or registers of application
- OS controls virtual memory

DOS - Side-Channels
- Internal Attack Vectors
- Fault Channels
- Controlled Channel Attacks
Example: string length

```
// str on heap
int strlen(char* str) {
    int len = 0; // Stack
    while (*str++ != '\0') len ++;
    return len;
}
```
Example: string length

Example (Source, simplified)

```c
int strlen(char* str) {
    int len = 0; // Stack
    while (*str != '\0') len ++;
    return len ;
}
```

- Heap not present
- Stack not present
Example (Source, simplified)

```c
// str on heap
int strlen(char* str) {
    int len = 0; // Stack
    while (*str++ != '\0') len++;
    return len;
}
```

- Heap not present
- Stack not present

<table>
<thead>
<tr>
<th></th>
<th>Phys-Addr</th>
<th>other Flags</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>Stack</td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

Attackers Knowledge
Length = 0
Example (Source, simplified)

```c
// string on heap
int strlen(char* str) {
    int len = 0; // Stack
    while (* (str++) != '\0') len ++;
    return len;
}
```

- Heap not present
- Stack not present

<table>
<thead>
<tr>
<th></th>
<th>Phys-Addr</th>
<th>other Flags</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>Stack</td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

Attackers Knowledge

Length = 0
Example: string length

```c
// str on heap
int strlen(char* str) {
    int len = 0; // Stack
    while (*(str++) != '\0') len++;
    return len;
}
```

- Heap not present
- Stack not present

<table>
<thead>
<tr>
<th></th>
<th>Phys-Addr</th>
<th>Other Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>Stack</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

Attackers Knowledge
Length = 0
Example: string length

Example (Source, simplified)

```c
// str on heap
int strlen(char* str) {
    int len = 0; // Stack
    while (*str++) != '\0')
        len++;
    return len;
}
```

- Heap not present
- Stack not present

<table>
<thead>
<tr>
<th></th>
<th>Phys-Addr</th>
<th>other Flags</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>...</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>Stack</td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

Attackers Knowledge

Length = 1
Example: string length

```c
int strlen(char *str) {
    int len = 0; // Stack
    while (*((str++) != '\0') {
        len++;
    }
    return len;
}
```

- Heap not present
- Stack not present

<table>
<thead>
<tr>
<th></th>
<th>Phys-Addr</th>
<th>other Flags</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>Stack</td>
<td>...</td>
<td>...</td>
<td>1</td>
</tr>
</tbody>
</table>

Attackers Knowledge

Length = 1
Example: string length

Example (Source, simplified)

```c
// str on heap
int strlen(char* str) {
    int len = 0; // Stack
    while (* (str++) != '\0') len ++;
    return len;
}
```

- Heap not present
- Stack not present

<table>
<thead>
<tr>
<th></th>
<th>Phys-Addr</th>
<th>other Flags</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>Stack</td>
<td>...</td>
<td>...</td>
<td>1</td>
</tr>
</tbody>
</table>

Attackers Knowledge
Length = 1
Example: string length

Example (Source, simplified)

```c
// str on heap
int strlen(char* str) {
    int len = 0; // Stack
    while (*(str++) != '\0')
        len++;
    return len;
}
```

- Heap not present
- Stack not present

<table>
<thead>
<tr>
<th></th>
<th>Phys-Addr</th>
<th>other Flags</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>...</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>Stack</td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

Attackers Knowledge

Length = 1
Example: string length

Example (Source, simplified)

```c
// str on heap
int strlen(char* str) {
    int len = 0; // Stack
    while (* (str++) != '\0')
        len ++;
    return len;
}
```

- Heap not present
- Stack not present

<table>
<thead>
<tr>
<th></th>
<th>Phys-Addr</th>
<th>other Flags</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>...</td>
<td>...</td>
<td>1</td>
</tr>
<tr>
<td>! Stack</td>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

Attackers Knowledge

Length = 2

2023-07-04

DOS - Side-Channels
- Internal Attack Vectors
- Fault Channels
- Example: string length
Example: string length

**Example (Source, simplified)**

```c
// str on heap
int strlen(char* str) {
  int len = 0; // Stack
  while (*str++ != '\0') len++;
  return len;
}
```

- **Heap not present**
- **Stack not present**

<table>
<thead>
<tr>
<th></th>
<th>Phys-Addr</th>
<th>other Flags</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>...</td>
<td>..</td>
<td>0</td>
</tr>
<tr>
<td>Stack</td>
<td>...</td>
<td>..</td>
<td>1</td>
</tr>
</tbody>
</table>

**Attackers Knowledge**

Length = 2

---

**Internal Attack Vectors**

**External Attack Vectors**

**Data remanence**

**Defense**

**Conclusion**

---

**Example: string length**

DOS - Side-Channels

- Internal Attack Vectors
- Fault Channels
- Example: string length

2023-07-04
Example Results (PF vs. Cache Channel)

1. IDCT (inverse discrete cosine transformation)
2. Index in array ≈ 8kB big

DOS - Side-Channels
- Internal Attack Vectors
  - Fault Channels
  - Example Results (PF vs. Cache Channel)
Example Results (PF vs. Cache Channel)

1. IDCT (inverse discrete cosine transformation)
2. Index in array ≈ 8 kB big
Example Results (PF vs. Cache Channel)

1. IDCT (inverse discrete cosine transformation)
2. Index in array $\approx 8 \text{ kB big}$
<table>
<thead>
<tr>
<th>Microarchitectural Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leaking speculative CPU-state to attackers</td>
</tr>
<tr>
<td>Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg</td>
</tr>
<tr>
<td>Examples and figures taken from the Meltdown paper</td>
</tr>
</tbody>
</table>

**Meltdown**

- Leaking speculative CPU-state to attackers
- Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg
- Examples and figures taken from the Meltdown paper

**Spectre**

- Leaking speculative CPU-state to attackers
- Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg
- Examples and figures taken from the Meltdown paper
Side-Effects of Out-of-Order execution

Toy Example

```
raise_exception();
// the line below is never reached
access(probe_array[data*4096]);
```
Side-Effects of Out-of-Order execution

**Toy Example**

```python
raise_exception();
// the line below is never reached
access(probe_array[data*4096]);
```

Constraints

- Raising the exception should be slow
- Accessing the array should be fast
Side-Effects of Out-of-Order execution

Toy Example

```python
raise_exception();
// the line below is never reached
access(probe_array[data*4096]);
```

Constraints
- Raising the exception should be slow
- Accessing the array should be fast
Introduction

Internal Attack Vectors

External Attack Vectors

Data remanence

Defense

Conclusion

DOS - Side-Channels

--- Internal Attack Vectors

--- Microarchitectural Channels

1. Retry needed because exception handling zeroes registers
2. No evicted cache line is considered zero
3. Exception can be prevented (amongst others) using TSX

---

Meltdown example code

; rcx = kernel address
; rbx = probe array
retry:
    MOV AL, byte [RCX]
    SHL RAX, 12
    JZ retry
    MOV RBX, qword [RBX + RAX]
Power channels

Features

- Requires no capability to run code
- Hard to detect
- In theory usable remotely
Power channels

**Features**
- Requires no capability to run code
- Hard to detect
- In theory usable remotely

**Requirements**
- (very) high-resolution power measurement
- Physical access to power supply
- Detailed knowledge about exact processor used
Example (Square-And-Multiply)

```c
int exp(int base, int e) {
    int res = 1;
    while (e != 0) {
        res *= res; // square
        if (e & 1) res *= base; // multiply
        e >>= 1;
    }
    return res;
}
```

Source: https://commons.wikimedia.org/wiki/File:Power_attack.png
Example (Square-And-Multiply)

```c
int exp(int base, int e) {
    int res = 1;
    while (e != 0) {
        res *= res; // square
        if (e & 1) res *= base; // multiply
        e >>= 1;
    }
    return res;
}
```

Source: https://commons.wikimedia.org/wiki/File:Power_attack.png
Acoustic channels

Features
- Requires no capability to run code
- Hard to detect
- Usable remotely, bugs
Acoustic channels

Features
- Requires no capability to run code
- Hard to detect
- Usable remotely, bugs

Requirements
- Good audio equipment
- Reliable audio filters
- Knowledge about typing style
- Knowledge about hardware used
Example

Password typing attack

Keyboard Acoustic Emanations Revisited
Li Zhuang, Feng Zhou, J. D. Tygar
University of California, Berkeley
Example

Password typing attack

Keyboard Acoustic Emanations Revisited
Li Zhuang, Feng Zhou, J. D. Tygar
University of California, Berkeley
Password typing attack

Keyboard Acoustic Emanations Revisited
Li Zhuang, Feng Zhou, J. D. Tygar
University of California, Berkeley
Results

DOS - Side-Channels
- External Attack Vectors
  - Acoustic and Radiation
  - Results
Results

- Acoustic and Radiation

- External Attack Vectors

- Data remanence

- Defense

- Conclusion

- External Attack Vectors

- DOS - Side-Channels

- Results

- Internal Attack Vectors

- Data remanence

- Defense

- Conclusion

2023-07-04
Electro Magnetic (EM) Radiation

**Features**
- Requires no capability to run code
- Hard to detect
- No “wire-cutting” needed

**Requirements**
- Expensive detection equipment (antenna, scope)
- Detailed knowledge about hardware used

---

**DOS - Side-Channels**
- External Attack Vectors
  - Acoustic and Radiation
  - Electro Magnetic (EM) Radiation
Electro Magnetic (EM) Radiation

Features

- Requires no capability to run code
- Hard to detect
- No “wire-cutting” needed

Requirements

- Expensive detection equipment (antenna, scope)
- Detailed knowledge about hardware used
**Warning**

- Not a classical side-channel
- No indirect observance of data → direct
Data Remanence

Warning

- **NOT** a classical side-channel
- no indirect observance of data $\rightarrow$ direct
- is still interesting

Access to data you thought is gone

Usually if you get data it is pretty good
Data Remanence

Warning
- NOT a classical side-channel
- no indirect observance of data → direct
- is still interesting

Features
- Access to data you thought is gone
- Usually if you get data it is pretty good
Example (Your friend, the compiler)

```c
void secret() {
    char *buf = (char*)malloc(1024);
    // put sth. secret into buf
    free(buf);
}
```
Example (Your friend, the compiler)

```c
void secret() {
    char* buf = (char*)malloc(1024);
    // put sth. secret into buf
    free(buf);
}
```

Problem
Examples / Software

Example (Your friend, the compiler)

```c
void secret() {
    char* buf = (char*)malloc(1024);
    // put sth. secret into buf
    memset(buf, '\0', 1024);
    free(buf);
}
```

Problem

What if someone gets the same memory?
Example (Your friend, the compiler)

```c
void secret() {
    char* buf = (char*)malloc(1024);
    // put sth. secret into buf
    memset(buf, '\0', 1024);
    free(buf);
}
```

Problem
Example (Your friend, the compiler)

```c
void secret() {
    char* buf = (char*)malloc(1024);
    // put sth. secret into buf
    memset(buf, '\0', 1024);
    free(buf);
}
```

Problem

The compiler could optimize the memset out
Lest We Remember: Cold Boot Attacks on Encryption Keys

J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten
Princeton University, Electronic Frontier Foundation, Wind River Systems

Cold Boot
Performance

<table>
<thead>
<tr>
<th>Seconds w/o power</th>
<th>Error % at operating temp.</th>
<th>Error % at -50 °C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>60 41 (no errors)</td>
<td>0.000095</td>
</tr>
<tr>
<td></td>
<td>300 50</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>360 50 (no errors)</td>
<td>0.000036</td>
</tr>
<tr>
<td></td>
<td>600 50</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>120 41 0.00105</td>
<td></td>
</tr>
<tr>
<td></td>
<td>360 42 0.00144</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>40 50 0.025</td>
<td></td>
</tr>
<tr>
<td></td>
<td>80 50 0.18</td>
<td></td>
</tr>
</tbody>
</table>
Performance

<table>
<thead>
<tr>
<th></th>
<th>Seconds</th>
<th>Error % at operating temp.</th>
<th>Error % at -50 °C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>60</td>
<td>41</td>
<td>(no errors)</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>50</td>
<td>0.000095</td>
</tr>
<tr>
<td>B</td>
<td>360</td>
<td>50</td>
<td>(no errors)</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>50</td>
<td>0.000036</td>
</tr>
<tr>
<td>C</td>
<td>120</td>
<td>41</td>
<td>0.00105</td>
</tr>
<tr>
<td></td>
<td>360</td>
<td>42</td>
<td>0.00144</td>
</tr>
<tr>
<td>D</td>
<td>40</td>
<td>50</td>
<td>0.025</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>50</td>
<td>0.18</td>
</tr>
</tbody>
</table>
Results

Image after 5, 30, 60 and 300 seconds
Defense mechanisms

Approach

Make all behavior that is observable independent of the input data

Caveat

Complete independence is not always achievable

(Algorithmic requirements, some channels hard to control)

Alternative

Remove ability to observe the given aspect
Defense mechanisms

**Approach**

Make all behavior that is observable independent of the input data

**Caveat**

Complete independence is not always achievable
(Algorithmic requirements, some channels hard to control)
## Defense mechanisms

### Approach
Make all behavior that is observable independent of the input data

### Caveat
Complete independence is not always achievable
(Algorithmic requirements, some channels hard to control)

### Alternative
Remove ability to observe the given aspect
Blinding

- Modify data computed on in such a way that operation always takes equal time
- Requires inverse unblinding that can be performed after the operation
- Noise injection

Timing channels

Blinding

Modify data computed on in such a way that operation always takes equal time
Requires inverse unblinding that can be performed after the operation
Noise injection

Noise injection
Timing channels

**Blinding**
- Modify data computed on in such a way that operation always takes equal time
- Requires inverse unblinding that can be performed after the operation
- Noise injection

**Branch elimination/equalisation**
Removes changes in runtime due to different operations depending on data
Example: Move different data processed in different branch targets to same cacheline

Prevent statistical analysis
Avoid running the same algorithm on attacker observable data multiple times.
Challenge-response is prone to this!
**Blinding**
- Modify data computed on in such a way that operation always takes equal time
- Requires inverse unblinding that can be performed after the operation
- Noise injection

**Branch elimination/equalisation**
Removes changes in runtime due to different operations depending on data
Example: Move different data processed in different branch targets to same cacheline

**Prevent statistical analysis**
Avoid running the same algorithm on attacker observable data multiple times.
Challenge-response is prone to this!
Page-Fault Channel / Fault channels

Detection

- Given a reliable time-source constant page-faults can be detected as unusually long program runtime
- SGX v2 can notify the protected program of page-faults. It may choose not to compute on secret data if such page-faults come unexpectedly

Prevention

- Don’t use paging. Require all memory to be mapped
- Avoid dynamic allocation of shared resources

DOS - Side-Channels

Defense

Page-Fault Channel / Fault channels
Detection
- Given a reliable time-source constant page-faults can be detected as unusually long program runtime
- SGX v2 can notify the protected program of page-faults. It may chose not to compute on secret data if such page-faults come unexpected

Prevention
- Don’t use paging. Require all memory to be mapped
- Avoid dynamic allocation of shared resources
Meltdown / Spectre

### Meltdown
- KPTI - Kernel Page Table Isolation
- HW: Don’t speculate across protection boundaries

---

DOS - Side-Channels

- Defense

---

Meltdown / Spectre
Meltdown / Spectre

**Meltdown**
- KPTI - Kernel Page Table Isolation
- HW: Don’t speculate across protection boundarys

**Spectre**
- Speculation fences
Internal Attack Vectors

External Attack Vectors

Data remanence

Defense

Conclusion

Power / Acoustic / EM

Power Channel

- Use internal power source or high-capacitance in power path for sensitive instructions (low pass effect)
- Use same-complexity instructions for input-dependent code (mul instead of shift)
**Power / Acoustic / EM**

**Power Channel**
- Use internal power source or high-capacitance in power path for sensitive instructions (low pass effect)
- Use same-complexity instructions for input-dependent code (mul instead of shift)

**Acoustic**
- Counter-noise to mask real typing
- Avoid typing sensitive information (on-screen keyboard)
Power Channel
- Use internal power source or high-capacitance in power path for sensitive instructions (low pass effect)
- Use same-complexity instructions for input-dependent code (mul instead of shift)

Acoustic
- Counter-noise to mask real typing
- Avoid typing sensitive information (on-screen keyboard)

Electro Magnetic Radiation
- Use EM shielding on chips
- Use EM shielding for case
Data remanence

Zero memory

- Like really zero it! (memset_s for C11, SecureZeroMemory for Windows)
Data remanence

Zero memory

- Like really zero it! (memset_s for C11, SecureZeroMemory for Windows)
- Remember copies of the data! (Stack? Heap?)
Data remanence

Zero memory
- Like really zero it! (memset_s for C11, SecureZeroMemory for Windows)
- Remember copies of the data! (Stack? Heap?)
- Not all copies are immediately obvious! Compilers may create additional ones
Data remanence

Zero memory

- Like really zero it! (memset for C11, SecureZeroMemory for Windows)
- Remember copies of the data! (Stack? Heap?)
- Not all copies are immediately obvious! Compilers may create additional ones
- And of course you remembered the XMM registers, right?

Cold Boot

Combined with the above very hard! Use shut down and not hybernate / suspend.

After a few seconds you should be fine.

Idea: Write secret data to physical 0x7c00 - 0x7dFF! MBR is loaded there :)
Data remanence

Zero memory

- Like really zero it! (memset_s for C11, SecureZeroMemory for Windows)
- Remember copies of the data! (Stack? Heap?)
- Not all copies are immediately obvious! Compilers may create additional ones
- And of course you remembered the XMM registers, right?

Cold Boot

Combined with the above very hard! Use shut down and not hybernate / suspend. After a few seconds you should be fine.

Idea: Write secret data to physical 0x7c00 - 0x7dFF! MBR is loaded there :)
Zero memory
- Like really zero it! (memset for C11, SecureZeroMemory for Windows)
- Remember copies of the data! (Stack? Heap?)
- Not all copies are immediately obvious! Compilers may create additional ones
- And of course you remembered the XMM registers, right?

Cold Boot
- Combined with the above very hard! Use shut down and not hybernate / suspend. After a few seconds you should be fine.
- Idea: Write secret data to physical 0x7c00 - 0x7dFF! MBR is loaded there :)
Sidechannels

... are unintended information sources for extracting secret data
Summary

Sidechannels

... are unintended information sources for extracting secret data

Attacks

There are a plethora of side-channels in every normal system! We only touched on a few methods! Your imagination is the limit.
Sidechannels
... are unintended information sources for extracting secret data

Attacks
There are a plethora of side-channels in every normal system! We only touched on a few methods! Your imagination is the limit.

Defense
... is very hard. The best way is to design algorithms from the ground up with side-channels in mind!
Overview

Cache Side-Channels

Page-fault Channel

Microarchitectural Channels

Acoustic Channels

References and Related Work
<table>
<thead>
<tr>
<th>Cold Boot</th>
<th>Remanence</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://www.semanticscholar.org/paper/Software-mitigations-to-hedge-AES-against-cache-Brickell-Graunke/11c6fddeff9e2f95c8cf238ea9f12f8f8ae7cf8c.pdf">Link</a></td>
<td></td>
<td><a href="https://www.cc.gatech.edu/~slee3036/papers/shih:tsgx.pdf">PDF</a></td>
</tr>
</tbody>
</table>