Distributed Operating Systems
Side-Channels

Marcus Hähnel

29.05.2017
What is a Side-Channel?
What is a Side-Channel?

Visual side-channel

Which call has a positive connotation?
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.

Phone example

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

Malicious

Extracting ...

- ... other customers data across virtual machines
### Malicious

Extracting ...

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

<table>
<thead>
<tr>
<th>Malicious</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Extracting ...</td>
<td>- other customers data across virtual machines</td>
</tr>
<tr>
<td></td>
<td>- crypto keys from applications in different address spaces</td>
</tr>
<tr>
<td></td>
<td>- data from inaccessible processors</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benign</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- detecting rootkits</td>
</tr>
<tr>
<td></td>
<td>- detecting hardware trojans</td>
</tr>
</tbody>
</table>
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?)
- 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 measureable parameter of the system and of its individual operations that changes depending on the processed data.

Example parameters

- Time (Duration)
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?)
- 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?)
- Power usage
- Radiation (Heat, EM-Radiation)
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?)
- Power usage
- Radiation (Heat, EM-Radiation)
- Unexpected persistence of data (Cold-boot, memory re-use)
Attack vector

The duration of an attacker observable operation depends on the data processed by the victim
Introduction
Common Attack Vectors
Defense
Conclusion

Timing Channels
Fault Channels
Power channels
Acoustic and Radiation
Data remanence

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

Example - Graphics Processing

Holidays
Day 1
Attack vector

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

Example - Graphics Processing

Holidays Day 1
**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: 1 s vs. 17 s
Cache Side-Channel

Diagram of a CPU with various cache levels:
- Core 1: L1I, L1D, Thr 1, Thr 2, L2 Cache
- Core 2: L1I, L1D, Thr 1, Thr 2, L2 Cache
- L3 Cache
- DRAM Memory

Table of 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
Cache Side-Channel

**Level** | **Size** | **Cycles**
--- | --- | ---
L1D | 32 KiB | 4
L1I | 32 KiB | 4
L2 | 256 KiB | 12
L3 | 3 MiB | 36
DRAM | large | 250
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"
Prime & Probe

Example (Victim)

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

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

transact(Alice);
```

L1D 8-way set cache

<table>
<thead>
<tr>
<th>Tag (20)</th>
<th>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>
Prime & Probe

Example (Victim)

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

Attacker

<table>
<thead>
<tr>
<th>Tag (20)</th>
<th>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>

L1D 8-way set cache

Set

Indices
Prime & Probe

Example (Victim)

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

Attacker

Prime

<table>
<thead>
<tr>
<th>L1D 8-way set cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag (20)</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>(Alice)</td>
</tr>
<tr>
<td>(Bob)</td>
</tr>
</tbody>
</table>
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>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;
```

Attacker
Prime, Probe, Detect

L1D 8-way set cache

<table>
<thead>
<tr>
<th>Tag (20)</th>
<th>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>
Figure: Results of prime-probe observations for 20 distinct 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

Alternative: Evict & Time
Evict & Time

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

Alternative: Evict & 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 & 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
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.

Set

Indices
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'
Assumption

Removing the OS from the TCB
Pagefault Side-Channel

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
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

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
Example: string length

```
// str on heap

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

- Heap not present
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
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>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>0</td>
</tr>
<tr>
<td>Stack</td>
<td>...</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 = 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

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

```
//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>

**Attacker’s 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>Phys-Addr</th>
<th>other Flags</th>
<th>P</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>

Attacker’s Knowledge
Length = 2
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>
<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>

**Attacker's Knowledge**

Length = 2
Example Results (PF vs. Cache Channel)
Example Results (PF vs. Cache Channel)
Example Results (PF vs. Cache Channel)
Example Results (PF vs. Cache Channel)
Example Results (PF vs. Cache Channel)
Example Results (PF vs. Cache Channel)
Example Results (PF vs. Cache Channel)
Example Results (PF vs. Cache Channel)
Example Results (PF vs. Cache Channel)
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

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;
}
```
Example

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;
}
```

Figure: Source: [https://commons.wikimedia.org/wiki/File:Power_attack.png](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
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

![Graph showing sample value and sum of FFT coefficients for keystrokes start positions](image)
Results

- **Final Recognition Rate**
  - Word correct rate
  - Char correct rate

- **Cumulative Distribution Function**
  - Password length = 5
  - Password length = 8
  - Password length = 10

<table>
<thead>
<tr>
<th>Length of Recording</th>
<th>Number of Trials Needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>90</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>12</td>
<td>90</td>
</tr>
<tr>
<td>14</td>
<td>80</td>
</tr>
</tbody>
</table>

Cumulative Distribution Function for different password lengths.
Electro Magnetic (EM) Radiation

Features

- Requires no capability to run code
- Hard to detect
- No "wire-cutting" needed
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
Warning

- **Not** a classical side-channel
- no indirect observance of data → direct
- is still interesting
**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

What if someone gets the same memory?
Examples

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
Cold Boot

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
Performance

<table>
<thead>
<tr>
<th></th>
<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</td>
<td>41</td>
<td>(no errors)</td>
</tr>
<tr>
<td>A</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>B</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>C</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>D</td>
<td>80</td>
<td>50</td>
<td>0.18</td>
</tr>
</tbody>
</table>
Performance

<table>
<thead>
<tr>
<th></th>
<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</td>
<td>41</td>
<td>(no errors) 0.000095</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>360</td>
<td>50</td>
<td>(no errors) 0.000036</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>50</td>
<td></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

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

Approach

Make all behavior that is observable independent of the input data
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
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
## 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
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!
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
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
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 / 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)

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_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?
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?
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 :)

Cold Boot
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.
**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.

**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

### Acoustic Channels
- [http://people.eecs.berkeley.edu/~tygar/papers/Keyboard_Acoustic_Emanations_Revisited/ccs.pdf](http://people.eecs.berkeley.edu/~tygar/papers/Keyboard_Acoustic_Emanations_Revisited/ccs.pdf)
<table>
<thead>
<tr>
<th>Cold Boot</th>
<th><a href="https://www.usenix.org/event/sec08/tech/full_papers/halderman/halderman.pdf">https://www.usenix.org/event/sec08/tech/full_papers/halderman/halderman.pdf</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Remanence</td>
<td><a href="http://www.daemonology.net/blog/2014-09-04-how-to-zero-a-buffer.html">http://www.daemonology.net/blog/2014-09-04-how-to-zero-a-buffer.html</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://www.daemonology.net/blog/2014-09-06-zeroing-buffers-is-insufficient.html">http://www.daemonology.net/blog/2014-09-06-zeroing-buffers-is-insufficient.html</a></td>
</tr>
<tr>
<td></td>
<td><a href="https://www.semanticscholar.org/paper/Software-mitigations-to-hedge-AES-against-cache-Brickell-Graunke/11c6fddeff9e2f95c8cf238ea9f12f8ffae7cf8c/pdf">https://www.semanticscholar.org/paper/Software-mitigations-to-hedge-AES-against-cache-Brickell-Graunke/11c6fddeff9e2f95c8cf238ea9f12f8ffae7cf8c/pdf</a></td>
</tr>
</tbody>
</table>