High Resolution Side Channels for Untrusted Operating Systems

Marcus Hähnel ¹  Marcus Peinado ²  Weidong Cui ²

¹TU Dresden
²Microsoft Research

2017-07-13
Reasons to distrust the OS
Reasons to distrust the OS
Reasons to distrust the OS

Large code bases, security bugs
Reasons to distrust the OS

Large code bases, security bugs
Reasons to distrust the OS

- Large code bases, security bugs
- Rootkit
Shielding Systems
Shielding Systems

Removing the OS from the trusted computing base
Shielding Systems

Removing the OS from the trusted computing base

Hypervisor-based
- Overshadow [ASPLOS’08]
- InkTag [ASPLOS’13]
Shielding Systems

Removing the OS from the trusted computing base

Hypervisor-based
- Overshadow [ASPLOS’08]
- InkTag [ASPLOS’13]

Intel SGX-based
- Haven [OSDI’14]
- VC3 [Oakland’15]
- SCONE [OSDI’16]
- Glamdring [ATC’17]
Shielding Systems

Removing the OS from the trusted computing base

Hypervisor-based
- Overshadow [ASPLOS’08]
- InkTag [ASPLOS’13]

Intel SGX-based
- Haven [OSDI’14]
- VC3 [Oakland’15]
- SCONES [OSDI’16]
- Glamdring [ATC’17]
But how well do these solutions protect the application?
Controlled Channels

OS still manages shielded applications

- Control over page tables
- ... and thus over page faults 😊

---

1Xu, Yuanzhong, Weidong Cui, and Marcus Peinado. "Controlled-channel attacks: Deterministic side channels for untrusted operating systems.", Oakland 2015
Controlled Channels

OS still manages shielded applications
- Control over page tables
- ... and thus over page faults 😊

Data dependent control flow

```c
// @ Page 1
void processData(bool secret) {
    if (secret) {
        secretData(); // @ Page 2
    } else {
        publicData(); // @ Page 3
    }
}
```

1Xu, Yuanzhong, Weidong Cui, and Marcus Peinado. “Controlled-channel attacks: Deterministic side channels for untrusted operating systems.”, Oakland 2015
Controlled Channels

OS still manages shielded applications

- Control over page tables
- ... and thus over page faults

Data dependent control flow

```c
void processData(bool secret) {
    if (secret) {
        secretData(); // @ Page 2
    } else {
        publicData(); // @ Page 3
    }
}
```

Page faults serve as de facto *breakpoints* and reveal memory access patterns

---

1 Xu, Yuanzhong, Weidong Cui, and Marcus Peinado. "Controlled-channel attacks: Deterministic side channels for untrusted operating systems." Oakl...
Controlled Channels

OS still manages shielded applications

- Control over page tables
- ... and thus over page faults 😊

Retrieved

- outlines of images
- text from font rendering
- text from spell checking

Data dependent control flow

```c
// @ Page 1
void processData(bool secret) {
    if (secret) {
        secretData(); // @ Page 2
    } else {
        publicData(); // @ Page 3
    }
}
```

Page faults serve as de facto breakpoints and reveal memory access patterns

---

¹Xu, Yuanzhong, Weidong Cui, and Marcus Peinado. "Controlled-channel attacks: Deterministic side channels for untrusted operating systems.", Oakland 2015
### Contributions

**Page-Fault Channel Limitations**

- limited to page granular memory observation
## Contributions

Page-Fault Channel Limitations

- limited to page granular memory observation

- requires *page toggling*
Contributions

Page-Fault Channel Limitations

- limited to page granular memory observation

- requires *page toggling*

- is *only* means to set breakpoint (may be detectable)
Contributions

Page-Fault Channel Limitations

- limited to page granular memory observation
- requires \textit{page toggling}
- is \textit{only} means to set breakpoint (may be detectable)

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXINFO</td>
<td>0</td>
<td>Report page fault and general protection exception info inside an enclave</td>
</tr>
<tr>
<td>Reserved</td>
<td>31:1</td>
<td>Reserved (0).</td>
</tr>
</tbody>
</table>
Contributions

Page-Fault Channel Limitations
- limited to page granular memory observation
  ⇒ Increase spatial resolution
- requires page toggling
  ⇒ Improve temporal resolution
- is only means to set breakpoint (may be detectable)

### Table 2-4. Bit Vector Layout of MISCELECT Field of Extended Information

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXINFO</td>
<td>0</td>
<td>Report page fault and general protection exception info inside an enclave</td>
</tr>
<tr>
<td>Reserved</td>
<td>31:1</td>
<td>Reserved (0).</td>
</tr>
</tbody>
</table>
Contributions

Page-Fault Channel Limitations
- limited to page granular memory observation
  ⇒ Increase spatial resolution
- requires page toggling
  ⇒ Improve temporal resolution
- is *only* means to set breakpoint (may be detectable)

Show more code than previously thought is vulnerable

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXINFO</td>
<td>0</td>
<td>Report page fault and general protection exception info inside an enclave</td>
</tr>
<tr>
<td>Reserved</td>
<td>31:1</td>
<td>Reserved (0).</td>
</tr>
</tbody>
</table>
Contributions

Page-Fault Channel Limitations
- limited to page granular memory observation
  ⇒ Increase spatial resolution
- requires page toggling
  ⇒ Improve temporal resolution
- is only means to set breakpoint (may be detectable)
  ⇒ Other ways to step through the application

Show more code than previously thought is vulnerable

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXINFO</td>
<td>0</td>
<td>Report page fault and general protection exception info inside an enclave</td>
</tr>
<tr>
<td>Reserved</td>
<td>31:1</td>
<td>Reserved (0).</td>
</tr>
</tbody>
</table>
Assumptions
Assumptions

Working shielding system
... protects integrity and security of applications’ memory against direct access
Assumptions

Working shielding system

... protects integrity and security of applications’ memory against direct access

Commodity OS

... is still responsible for:

- Memory management
- Scheduling
- Hardware Configuration
System Model

Assumptions

Working shielding system
... protects integrity and security of applications’ memory against direct access

Commodity OS
... is still responsible for:
- Memory management

OS

Protected Application

Hypervisor or SGX CPU
**System Model**

**Assumptions**

**Working shielding system**
- protects integrity and security of applications’ memory against direct access

**Commodity OS**
- is still responsible for:
  - Memory management
  - Scheduling

[Diagram showing a protected application with OS and Hypervisor or SGX CPU]
**System Model**

**Assumptions**

**Working shielding system**

... protects integrity and security of applications’ memory against direct access

**Commodity OS**

... is still responsible for:

- Memory management
- Scheduling
- Hardware Configuration
New Attack Tools
Timer-based Attacks

Scheduling

The OS has control over scheduling ... and thus over timers 😊
Timer-based Attacks

**Scheduling**

The OS has control over scheduling ... and thus over timers 😊

**Challenges**

- 25 MHz LAPIC Timer vs. 4 GHz CPU clock
Timer-based Attacks

Scheduling

The OS has control over scheduling ... and thus over timers 😏

Challenges

- 25 MHz LAPIC Timer vs. 4 GHz CPU clock
- No page fault address
Timer-based Attacks

Scheduling

The OS has control over scheduling ... and thus over timers 😁

Challenges

- 25 MHz LAPIC Timer vs. 4 GHz CPU clock
- No page fault address

Figure: Accessed & Dirty bits in PTE
### Example

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

```c
const char* s = "The";
int l = strlen(s);
```

<table>
<thead>
<tr>
<th>*str</th>
<th>Attacker count</th>
</tr>
</thead>
<tbody>
<tr>
<td>'T'</td>
<td>0</td>
</tr>
</tbody>
</table>
Example

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

const char* s = "The";
int l = strlen(s);
```

<table>
<thead>
<tr>
<th>*str</th>
<th>Attacker count</th>
</tr>
</thead>
<tbody>
<tr>
<td>'T'</td>
<td>0</td>
</tr>
</tbody>
</table>
### Example

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

```c
const char* s = "The";
int l = strlen(s);
```

<table>
<thead>
<tr>
<th>*str</th>
<th>Attacker count</th>
</tr>
</thead>
<tbody>
<tr>
<td>'T'</td>
<td>0</td>
</tr>
</tbody>
</table>
```c
size_t strlen(const char* str) {
    size_t len = 0;
    while (*str != '\0') {
        str++;
        len++;
    }
    return len;
}

const char* s = "The";
int l = strlen(s);
```

<table>
<thead>
<tr>
<th>*str</th>
<th>Attacker count</th>
</tr>
</thead>
<tbody>
<tr>
<td>'T'</td>
<td>0</td>
</tr>
</tbody>
</table>
Example

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

const char* s = "The";
int l = strlen(s);
```

<table>
<thead>
<tr>
<th>*str</th>
<th>Attacker count</th>
</tr>
</thead>
<tbody>
<tr>
<td>'T'</td>
<td>1</td>
</tr>
</tbody>
</table>
Example

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

const char* s = "The";
int l = strlen(s);
```

<table>
<thead>
<tr>
<th>*str</th>
<th>Attacker count</th>
</tr>
</thead>
<tbody>
<tr>
<td>'T'</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XD</th>
<th>Ignored</th>
<th>Rsvd.</th>
<th>Address of 4KB page frame</th>
<th>Ign.</th>
<th>PAT</th>
<th>PDA</th>
<th>PPU</th>
<th>CM/</th>
<th>TSW</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

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

const char* s = "The";
int l = strlen(s);
```

<table>
<thead>
<tr>
<th>*str</th>
<th>Attacker count</th>
</tr>
</thead>
<tbody>
<tr>
<td>'T'</td>
<td>1</td>
</tr>
<tr>
<td>'h'</td>
<td>1</td>
</tr>
</tbody>
</table>
```c
size_t strlen(const char* str) {
    size_t len = 0;
    while (*str != '\0') {
        str++;
        len++;
    }
    return len;
}

const char* s = "The";
int l = strlen(s);
```

<table>
<thead>
<tr>
<th>*str</th>
<th>Attacker count</th>
</tr>
</thead>
<tbody>
<tr>
<td>'T'</td>
<td>1</td>
</tr>
<tr>
<td>'h'</td>
<td>2</td>
</tr>
<tr>
<td>'e'</td>
<td>3</td>
</tr>
<tr>
<td>'\0'</td>
<td>4</td>
</tr>
</tbody>
</table>

```
XD | Ignored | Rsvd. | Address of 4KB page frame | Ign. | PAD | T | P | P | U | R | D | T | S | W
---|---------|------|---------------------------|------|-----|---|---|---|---|---|---|---|---|---|---
---|---------|------|---------------------------|------|-----|---|---|---|---|---|---|---|---|---|---
```
**Results**

**STRLLEN function**

- 99.98% of string lengths detected correctly
- Can *effectively* single-step through the application
- Works where Page-Fault Channel fails
- Can replace page-fault based break points
- Requires fine-tuning for correct timing
**Prime & Probe**

**Traditional Cache Side-Channel**

- Unprivileged attacker and victim on same machine share cache
- Attacker can indirectly observe victims memory access
Prime & Probe

Traditional Cache Side-Channel

- Unprivileged attacker and victim on same machine share cache
- Attacker can indirectly observe victims memory access
Prime & Probe

**Traditional Cache Side-Channel**
- Unprivileged attacker and victim on same machine share cache
- Attacker can indirectly observe victims memory access

Attacker: Prime
Prime & Probe

Traditional Cache Side-Channel

- Unprivileged attacker and victim on same machine share cache
- Attacker can indirectly observe victims memory access

Attacker: Prime

Victim: Run
Prime & Probe

Traditional Cache Side-Channel

- Unprivileged attacker and victim on same machine share cache
- Attacker can indirectly observe victims memory access
Prime & Probe

Traditional Cache Side-Channel
- Unprivileged attacker and victim on same machine share cache
- Attacker can indirectly observe victims memory access

Attacker: Prime

Victim: Run

Attacker: Probe
But we are not an unprivileged attacker, but the OS

Noise reduction by
- Targeted Breakpoints
- Preventing other applications from being scheduled
- Turn off prefetching
Results

The chart illustrates the relationship between array access and associativity set. The x-axis represents the associativity set, while the y-axis shows the array access. The data points indicate varying levels of interaction within the system model, providing insights into the effectiveness of attack tools and the evaluation process.
Evaluation

Victims

- libjpeg: image decoding
- VC3: map-reduce framework for SGX
libjpeg: High resolution image extraction
libjpeg: High resolution image extraction
libjpeg: High resolution image extraction
Why is attacking VC3 interesting

- First/only realistic shielding system for Hadoop
- Protects mapper and reducer applications and their data from the OS/cloud
- Uses SGX (Enclaves)
VC3

Why is attacking VC3 interesting

- First/only realistic shielding system for Hadoop
- Protects mapper and reducer applications and their data from the OS/cloud
- Uses SGX (Enclaves)
Why is attacking VC3 hard

- Only attack framework; not user’s secret mappers and reducers
- Framework is small (only 13 code pages)
- Framework does not know application semantics
Attack Overview

Why is attacking VC3 hard

- Only attack framework; not user’s secret mappers and reducers
- Framework is small (only 13 code pages)
- Framework does not know application semantics

Can this leak information?
Attack Overview

Why is attacking VC3 hard

- Only attack framework; not user’s secret mappers and reducers
- Framework is small (only 13 code pages)
- Framework does not know application semantics

Can this leak information?

Map/Reduce spec

"The MapReduce library \textit{groups together} all intermediate values associated with the same intermediate key \(i\) and passes them to the \textit{Reduce} function" \(^2\)

VC3 implements grouping using a hash table

Attack: Information Gathering Phase

**Victim**

Concrete attack here: *WordCount* (or similar: e.g., *inverted index*)
Attacker: Information Gathering Phase

**Victim**

Concrete attack here: *WordCount* (or similar: e.g., *inverted index*)

- Word \( w \)
Concrete attack here: *WordCount* (or similar: e.g., *inverted index*)

- **Word** $w$
- $\text{hash}(w)$
- $h(w)$
Concrete attack here: *WordCount* (or similar: e.g., *inverted index*)

\[
\text{Word } w \xrightarrow{\text{hash}(w)} h(w) \xrightarrow{\text{map}[h(w)]}
\]
Attack: Information Gathering Phase

Concrete attack here: *WordCount* (or similar: e.g., *inverted index*)

**Victim**

- **Word** \( w \)
- **hash** \( (w) \)
- **\( h(w) \)**
- **map** \[ h(w) \]

**word length**

**timer attack**

**HashMap**

- \( h(\text{keya}) \)
- \( h(\text{keyb}) \)
- ...
Attack: Information Gathering Phase

Concrete attack here: *WordCount* (or similar: e.g., *inverted index*)

- Word $w$
- $\text{hash}(w)$
- $h(w)$
- $\text{map}[h(w)]$
- word length timer attack
- hash(word) cache side-channel

<table>
<thead>
<tr>
<th>HashMap</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>$h(\text{key}a)$</td>
</tr>
<tr>
<td>$h(\text{key}b)$</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Attack: Text Recovery Phase

Using the gathered information to re-construct the document

\[
\begin{align*}
\ldots \\
(len_{n-4}, hash_{n-4}) \\
(len_{n-3}, hash_{n-3}) \\
(len_{n-2}, hash_{n-2}) \\
(len_{n-1}, hash_{n-1}) \\
(len_n, hash_n) \\
(len_{n+1}, hash_{n+1}) \\
(len_{n+2}, hash_{n+2}) \\
(len_{n+3}, hash_{n+3}) \\
(len_{n+4}, hash_{n+4}) \\
\ldots
\end{align*}
\]
Attack: Text Recovery Phase

Using the gathered information to re-construct the document

\[
\begin{array}{l}
\text{Candidate List} \\
\text{all words with length } l_n \\
\text{and hash } h_n \\
\end{array}
\]

\[
\begin{array}{l}
\text{Dictionary} \\
\text{...} \\
(l_{n-4}, h_{n-4}) \\
(l_{n-3}, h_{n-3}) \\
(l_{n-2}, h_{n-2}) \\
(l_{n-1}, h_{n-1}) \\
(l_{n}, h_{n}) \\
(l_{n+1}, h_{n+1}) \\
(l_{n+2}, h_{n+2}) \\
(l_{n+3}, h_{n+3}) \\
(l_{n+4}, h_{n+4}) \\
\text{...} \\
\end{array}
\]
**Attack: Text Recovery Phase**

Using the gathered information to re-construct the document

<table>
<thead>
<tr>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\text{len}<em>{n-4}, \text{hash}</em>{n-4})$</td>
</tr>
<tr>
<td>$(\text{len}<em>{n-3}, \text{hash}</em>{n-3})$</td>
</tr>
<tr>
<td>$(\text{len}<em>{n-2}, \text{hash}</em>{n-2})$</td>
</tr>
<tr>
<td>$(\text{len}<em>{n-1}, \text{hash}</em>{n-1})$</td>
</tr>
<tr>
<td>$(\text{len}_n, \text{hash}_n)$</td>
</tr>
<tr>
<td>$(\text{len}<em>{n+1}, \text{hash}</em>{n+1})$</td>
</tr>
<tr>
<td>$(\text{len}<em>{n+2}, \text{hash}</em>{n+2})$</td>
</tr>
<tr>
<td>$(\text{len}<em>{n+3}, \text{hash}</em>{n+3})$</td>
</tr>
<tr>
<td>$(\text{len}<em>{n+4}, \text{hash}</em>{n+4})$</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

**Dictionary**

**Candidate List**

all words with length $l_n$ and hash $h_n$

**Prune and order candidate list**
Attack: Text Recovery Phase

Using the gathered information to re-construct the document

<table>
<thead>
<tr>
<th>Candidate List</th>
<th>Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>all words with length $l_n$ and hash $h_n$</td>
<td></td>
</tr>
</tbody>
</table>

Prune and order candidate list

$word_{n_1}, word_{n_2}, ...$
THE WONDERFUL WIZARD OF OZ

The Cyclone

Dorothy lived in the midst of the great Kansas prairies with Uncle Henry who was a farmer and Aunt Em who was the Their house was small for the family to build it had to be carried by many There were four walls a floor and a roof which made one and this room contained a big cooking cookstove a cupboard for the those a table three or four chairs and the little Henry and Aunt Em had a big bed in one corner and Dorothy a little bed in another There was no plate at all and no small hole dug in the ground called a cellar where the family could go in case one of those great whirlwinds arose mighty enough to crush any building in its It was reached by a trap door in the middle of the floor from which a ladder came down into the small dark

When Dorothy stood in the doorway and looked around she could see nothing but the great gray prairie on every Not a tree nor a house broke the broad sweep of flat country that reached to the edge of the sky in all The sun had turned the sky into a gray mass with little marks running through Even the grass was not green for the sun had burned the tops of the long blades until they were the same gray color to be seen Once the house had been painted but the sun Blasted the paint and the rain washed it away and now the house was as dull and gray as everything

When Aunt Em came there to live she was a young pretty She had taken the sun from her eyes and left them a sober They had taken the red from her cheeks and lips and they were gray She was thin and pale and never smiled When Dorothy who was always first came to her Aunt Em had been so startled by the laughter that she would scream and press her hand upon her heart whenever merry voice reached her and she still looked at the little girl with wonder that she could find anything to laugh
Conclusion

Enhanced Side-Channels

- memory access detection at higher *spatial* resolution (64 byte vs. 4kB granularity)
- fine-granular *breakpoints* through timers
- low-noise cache side-channel with single execution
## Conclusion

### Enhanced Side-Channels
- memory access detection at higher *spatial* resolution (64 byte vs. 4kB granularity)
- fine-granular *breakpoints* through timers
- low-noise cache side-channel with single execution

### Results
- High resolution image extraction from libjpeg
- Document extraction from map/reduce
Conclusion

Enhanced Side-Channels
- memory access detection at higher spatial resolution (64 byte vs. 4kB granularity)
- fine-granular breakpoints through timers
- low-noise cache side-channel with single execution

Results
- High resolution image extraction from libjpeg
- Document extraction from map/reduce

Mitigations
Are increasingly important
- T-SGX, Intel Taint Analysis Tool, Trusted Schedulers