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

Yuanzhong Xu, Weidong Cui, Marcus Peinado
The University of Texas at Austin, Microsoft Research

San Jose, CA
May 19, 2015
Applications run on top of OS
Reasons to **distrust** the OS

- Large code bases, security bugs
- rootkit
Reasons to **distrust** the OS

Hosted on untrusted cloud
“Shielding systems:”

remove OS from trusted computing base (TCB)

Use hypervisor or secure hardware to confine OS

- Hypervisor-based
  - Overshadow [ASPLOS ’08]
  - InkTag [ASPLOS ’13]
- Intel SGX-based
  - Haven [OSDI ’14]
“Shielding systems:”

remove OS from trusted computing base (TCB)

Hypervisor or SGX protects **application memory pages** from OS

- Secrecy
- Integrity
“Shielding systems:”

backward-compatibility & functionality

• Run *legacy applications*, on a *legacy OS* (with small or no modifications)

• OS is still needed
  • **Memory management**
  • Storage
  • Network
  • ...

Efficient memory management
“Shielding systems:”

*backward-compatibility & functionality*

- Run **legacy applications**, on a **legacy OS** (with small or no modifications)

- OS is still needed
  - Memory management
  - Storage
  - Network
  - ...

---

Protected app memory pages

Hypervisor or SGX CPU

Application page table

Integrity
Breaking the shield (1/3)

Hypervisor (InkTag) or SGX CPU (Haven)
Breaking the shield (2/3)

Font rendering (FreeType)

The treasure is buried ten feet north of the third palm tree

100% accuracy
• English letters
• Common punctuation marks

Hypervisor (InkTag) or SGX CPU (Haven)
Breaking the shield (3/3)

96% accuracy
• All words in novel “The Wizard of OZ”

Spell checking (Hunspell)

Folklore, legends, myths and fairy tales...

folklore *legend* myths and fairy *tale*...
How?

“Controlled-channel attacks”
– Side channel attacks from the OS

• Powerful due to the OS’ control over the applications
Side-channel attacks

Traditional side channels:
  - cache access
  - power consumption
  - network traffic...
Side-channel attacks

Traditional side channels:
cache access, power consumption, network traffic...

System noise: context switches, TLB flushes, exceptions, page faults
Side channels for the OS

Under attacker’s control

System noise: context switches, TLB flushes, exceptions, page faults

Victim

secret

Noise-free channel

OS
“Controlled-channel attacks”

- **Page faults** as a side channel

- **Deterministic:**
  - OS controls page tables
  - Set traps by making pages inaccessible
  - Do not need timing

Challenge: page-level granularity
Input-dependent memory access

- If (input)
- Page X
- Page Y
- Page Z
- Page fault sequence X, Y
- Page fault sequence X, Z
- func1()
- func2()
Input-dependent memory access

\[ \text{input} = n \]

Page X

read \[ \text{array}[n] \]

Page Y

Page Z

\( N \leq 3 \)
Page fault sequence:
\( X, Y \)

\( N \geq 4 \)
Page fault sequence:
\( X, Z \)
Hunspell

• Open-source spell checker, used in
Hunspell – hash table

Assumption: Dictionary is known

Load

Linked lists

Page faults

Page-level knowledge about layout

Separate chaining

Pointer array: index is hash code

PG 1  PG 2  PG 3  PG 4

PG 5  PG 6  PG 7

a
attack
change
channel
side
size
Hunspell – hash table lookup

Input: side channel attack

```c
while (word) {
    n = hash(word);
    listnode = table[n];

    while (listnode) {
        if (equal(listnode, word))
            break;
        listnode = listnode->next;
    }

    if (listnode) success(); else failure();
    word = get_next();
}
```
Hunspell – hash table lookup

Input: side channel attack

```c
while (word) {
    n = hash(word);
    listnode = table[n];

    while (listnode) {
        if (equal(listnode, word))
            break;
        listnode = listnode->next;
    }

    if (listnode) success(); else failure();
    word = get_next();
}
```

Page faults: 4
Hunspell – hash table lookup

Input: `side` channel attack

```c
while (word) {
    n = hash(word);
    listnode = table[n];

    while (listnode) {
        if (equal(listnode, word))
            break;
        listnode = listnode->next;
    }

    if (listnode) success();
    else failure();
    word = get_next();
}
```

Page faults: 4 7
Hunspell – hash table lookup

Input: side channel attack

```c
while (word) {
    n = hash(word);
    listnode = table[n];

    while (listnode) {
        if (equal(listnode, word))
            break;
        listnode = listnode->next;
    }

    if (listnode) success(); else failure();
    word = get_next();
}
```

Page faults:

4 7 1
Hunspell – hash table lookup

Input: side channel attack

```c
while (word) {
    n = hash(word);
    listnode = table[n];

    while (listnode) {
        if (equal(listnode, word))
            break;
        listnode = listnode->next;
    }

    if (listnode) success(); else failure();
    word = get_next();
}
```

Page faults:
4 7 15
Hunspell – hash table lookup

Input: side channel attack

```c
while (word) {
    n = hash(word);
    listnode = table[n];

    while (listnode) {
        if (equal(listnode, word))
            break;
        listnode = listnode->next;
    }

    if (listnode) success(); else failure();
    word = get_next();
}
```

Page faults:
4 7 1 5 6
Hunspell – hash table lookup

Input: side channel attack

```c
while (word) {
    n = hash(word);
    listnode = table[n];
    while (listnode) {
        if (equal(listnode, word))
            break;
        listnode = listnode->next;
    }
    if (listnode) success(); else failure();
    word = get_next();
}
```

Page faults:
4 7 1 5 6 3
Hunspell – hash table lookup

Input: side channel attack

while (word) {
    n = hash(word);
    listnode = table[n];

    while (listnode) {
        if (equal(listnode, word))
            break;
        listnode = listnode->next;
    }

    if (listnode) success(); else failure();
    word = get_next();
}
Hunspell – hash table lookup

Input: side channel attack

```
while (word) {
    n = hash(word);
    listnode = table[n];

    while (listnode) {
        if (equal(listnode, word))
            break;
        listnode = listnode->next;
    }

    if (listnode) success();
    else failure();
    word = get_next();
}
```

Page faults:

h 4 7 h 1 5 6 h 3 5
side channel attack
Hunspell – hash table lookup

Hash function **shuffles** the order

**Low correlation** between
- The address of the head pointer
- The addresses of the list nodes

**Low ambiguity**

---

Page faults:

```
4 7 1 5 6 3 5
```

side channel attack
Attack results

Implemented 3 attacks (Hunspell, FreeType, libjpeg) on both Haven and InkTag

• Hunspell (spell checking)
  • ~96% accuracy for novel “The Wizard of Oz”

• FreeType (font rendering)
  • 100% accuracy for any English input (letters and punctuation marks)
libjpeg results

<table>
<thead>
<tr>
<th>Original</th>
<th>Recovered</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Democratic Donkey" /></td>
<td><img src="image2" alt="Recovered Donkey" /></td>
</tr>
<tr>
<td><img src="image3" alt="Republican Elephant" /></td>
<td><img src="image4" alt="Recovered Elephant" /></td>
</tr>
<tr>
<td><img src="image5" alt="NPR Logo" /></td>
<td><img src="image6" alt="Recovered NPR Logo" /></td>
</tr>
<tr>
<td><img src="image7" alt="CNN Logo" /></td>
<td><img src="image8" alt="Recovered CNN Logo" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Original</th>
<th>Recovered</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image9" alt="Original Bird" /></td>
<td><img src="image10" alt="Recovered Bird" /></td>
</tr>
<tr>
<td><img src="image11" alt="Original Dragonfly" /></td>
<td><img src="image12" alt="Recovered Dragonfly" /></td>
</tr>
<tr>
<td><img src="image13" alt="Original Snail" /></td>
<td><img src="image14" alt="Recovered Snail" /></td>
</tr>
<tr>
<td><img src="image15" alt="Original Deer" /></td>
<td><img src="image16" alt="Recovered Deer" /></td>
</tr>
</tbody>
</table>
Performance overhead

• Non-trivial overhead due to expensive page fault handling

• The attacks are still realistic to execute
  • Hunspell:
    • 2.94s to check the entire novel (39,719 words)
  • FreeType:
    • 0.52s to render 5 KB text

• Small run time may be hidden in noises
Thoughts on potential mitigations

- No great off-the-shelf solution

- Applications: rewrite to hide access patterns
  - Manually (e.g., OpenSSL) or by compiler
  - High cost for complex legacy apps: engineering effort & performance overhead

- Shielding systems: more restrictions over the OS
  - Restrictions on memory management
  - May affect functionality

Rethink the challenges for Overshadow’s vision – protect legacy applications from legacy untrusted operating systems
Conclusion

• An untrusted OS can construct noise-free side channels.

• The page-fault side channel leaks large amounts of information about legacy applications.

• Controlled-channel attacks must be addressed in the design of shielding systems.