(Software) Side Channel Attacks

Winter 2021

David Kohlbrenner

dkohlbre@cs.washington.edu

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
Admin

• **Lab3 due** Next Friday

• **Homework 3 due** Monday

• **Last extra credit** reading due next Thursday
  • No late days

• **Final project due** next 03/16
  • No late days
  • Make sure you:
    • Include references
    • Include at least one legal/ethics discussion slide
    • Create original content
    • Go beyond class materials (if it’s a topic we also covered)
Side-channels: conceptually

• A program’s implementation (that is, the final compiled version) is different from the conceptual description

• Side-effects of the difference between the implementation and conception can reveal unexpected information
  • Thus: Side-channels
Detour: Covert-channels

• We’ll see many unusual ways to have information flow from thing A to thing B

• If this is an *intentional* usage of side effects, it is a covert channel

• *Unintentional* means it is a side-channel

• The same *mechanism* can be used as a covert-channel, or abused as a side-channel
Side Channel Attacks

• Most commonly discussed in the context of cryptosystems
• But also prevalent in many contexts
  • E.g., we discussed the TENEX password implementation
  • E.g., we discussed browser fingerprinting
Why should we care about side-channels?

• Compromises happen via ‘simple’ methods
  • Phishing
  • Straight-forward attacks

• Embedded systems *do* see side-channel attacks

• “High Security” systems *do* see side-channel attacks
Timing Side-Channels

- Duration of a program (or operation) reveals information
- TENEX case
  - We... lied, sorry
  - Its... more complicated
TENEX attack (for real)

• TENEX had an early *memory paging system*

• The original attack used page faults, not timing
  • Timing would’ve also worked 😊
Timing side-channels: round 2

• Cryptographic implementations fall down
  • #1 target for timing attacks
  • Extremely common to find vulnerabilities

• Why?
  • Pollev.com/dkohlbre
Attacking cryptographic with side-channels

• ANY leakage is bad
  • E.g. 1 bit of key leaking is ‘catastrophic’

• Cryptographic implementations are complex
  • Many layers of protocols
Example Timing Attacks

• **RSA:** Leverage key-dependent timings of modular exponentiations
  • https://www.rambus.com/timing-attacks-on-implementations-of-diffie-hellman-rsa-dss-and-other-systems/

• **Block Ciphers:** Leverage key-dependent cache hits/misses
How odd can this get?
The SVG-filter **pixel-stealing** timing attack

- **Attacker:**
  - Hosts webpage
- **Victim:**
  - Visits attacker
  - Logged into target
- **Target:**
  - Website hosting private visual information
The SVG-filter **pixel-stealing** timing attack

- **Attacker:**
  - Hosts webpage

- **Victim:**
  - Visits attacker
  - Logged into target

- **Target:**
  - Website hosting private visual information

![Diagram showing attacker.com and targeted.com iframe with profile of David K]
The SVG-filter \textit{pixel-stealing} timing attack

- **Attacker:**
  - Hosts webpage

- **Victim:**
  - Visits attacker
  - Logged into target

- **Target:**
  - Website hosting private visual information

Diagram:

```
attacker.com

targeted.com
iframe
???
```
The SVG-filter **pixel-stealing** timing attack

- **Attacker:**
  - Hosts webpage
- **Victim:**
  - Visits attacker
  - Logged into target
- **Target:**
  - Website hosting private visual information
SVG-filter Pixel-stealing attack overview

1. iframe of target page
2. Target pixel in red
3. Pixel multiplication `<div>`
4. SVG Filter
5. Filtered rendering
6. Target pixel white
   Target pixel black
SVG-filter Pixel-stealing attack overview

1. iframe of target page
2. Target pixel in red
3. Pixel multiplication <div>
4. SVG Filter
5. Filtered rendering
6. Target pixel white
   - Target pixel black
SVG-filter Pixel-stealing attack overview

1. iframe of target page
2. Target pixel in red
3. Pixel multiplication <div>
4. SVG Filter
5. Filtered rendering
6. Target pixel white
   Target pixel black
SVG-filter Pixel-stealing attack overview

(1) iframe of target page

(2) Target pixel in red

(3) Pixel multiplication <div>

(4) SVG Filter

(5) Filtered rendering

(6) Target pixel white

Target pixel black
SVG-filter Pixel-stealing attack overview

1. iframe of target page
2. Target pixel in red
3. Pixel multiplication <div>
4. SVG Filter
5. Filtered rendering
6. Target pixel white
7. Target pixel black
SVG-filter Pixel-stealing attack overview
SVG-filter Pixel-stealing attack overview

(1) iframe of target page

(2) Target pixel in red

(3) Pixel multiplication `<div>`

(4) SVG Filter

(5) Filtered rendering

(6) Target pixel white

Target pixel black
How?
if (x == rect.x || xExt[0] <= startX || xExt[1] <= startX ||
    [...]
} else { // We only need to look at the newest column
    for (PRUint32 y1 = startY; y1 <= endY; y1++) {
        [...]
    }
}
if (x == rect.x || xExt[0] <= startX || xExt[1] <= startX || xExt[2] <= startX || xExt[3] <= startX) {
    [...]
} else { // We only need to look at the newest column
    for (PRUint32 y1 = startY; y1 <= endY; y1++) {
        [...]
    }
}
if (x == rect.x || xExt[0] <= startX || xExt[1] <= startX || xExt[2] <= startX || xExt[3] <= startX) {
    [...]
} else { // We only need to look at the newest column
    for (PRUint32 y1 = startY; y1 <= endY; y1++) {
        [...]
    }
}
// Constant-time max and min functions for unsigned arguments
static inline unsigned umax(unsigned a, unsigned b)
{
    return a - ((a - b) & -(a < b));
}

static inline unsigned umin(unsigned a, unsigned b)
{
    return a - ((a - b) & -(a > b));
}
Implemented with *Floating-point math*
Variable time instructions?
<table>
<thead>
<tr>
<th>Cycle count</th>
<th>0.0</th>
<th>1.0</th>
<th>1e10</th>
<th>1e+200</th>
<th>1e-300</th>
<th>1e-42</th>
<th>256</th>
<th>257</th>
<th>1e-320</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>6.59</td>
<td>6.56</td>
<td>6.59</td>
<td>6.58</td>
<td>6.58</td>
<td>6.57</td>
<td>6.58</td>
<td>6.59</td>
<td>6.57</td>
</tr>
<tr>
<td>256</td>
<td>6.58</td>
<td>6.53</td>
<td>6.56</td>
<td>6.54</td>
<td>6.56</td>
<td>6.56</td>
<td>6.58</td>
<td>6.57</td>
<td>130.94</td>
</tr>
<tr>
<td>1e-320</td>
<td>6.59</td>
<td>130.90</td>
<td>130.92</td>
<td>130.94</td>
<td>6.59</td>
<td>6.58</td>
<td>130.95</td>
<td>130.91</td>
<td>6.56</td>
</tr>
</tbody>
</table>
# Intel i5-4460 Double-Precision Floating-Point Multiply

<table>
<thead>
<tr>
<th></th>
<th>0.0</th>
<th>1.0</th>
<th>1e10</th>
<th>1e+200</th>
<th>1e-300</th>
<th>1e-42</th>
<th>256</th>
<th>257</th>
<th>1e-320</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>6.59</td>
<td>6.56</td>
<td>6.59</td>
<td>6.58</td>
<td>6.58</td>
<td>6.57</td>
<td>6.58</td>
<td>6.59</td>
<td>6.57</td>
</tr>
<tr>
<td>1.0</td>
<td>6.57</td>
<td>6.59</td>
<td>6.55</td>
<td>6.57</td>
<td>6.57</td>
<td>6.56</td>
<td>6.56</td>
<td>6.56</td>
<td>130.89</td>
</tr>
<tr>
<td>1e10</td>
<td>6.55</td>
<td>6.55</td>
<td>6.56</td>
<td>6.58</td>
<td>6.56</td>
<td>6.56</td>
<td>6.56</td>
<td>6.57</td>
<td>130.92</td>
</tr>
<tr>
<td>256</td>
<td>6.58</td>
<td>6.53</td>
<td>6.56</td>
<td>6.54</td>
<td>6.56</td>
<td>6.56</td>
<td>6.58</td>
<td>6.57</td>
<td>130.95</td>
</tr>
<tr>
<td>257</td>
<td>6.59</td>
<td>6.57</td>
<td>6.60</td>
<td>6.56</td>
<td>6.58</td>
<td>6.56</td>
<td>6.57</td>
<td>6.59</td>
<td>130.91</td>
</tr>
<tr>
<td>1e-320</td>
<td>6.59</td>
<td>130.90</td>
<td>130.92</td>
<td>130.94</td>
<td>6.59</td>
<td>6.58</td>
<td>130.95</td>
<td>130.91</td>
<td>6.56</td>
</tr>
</tbody>
</table>
## Intel i5-4460 double-precision floating-point multiply

<table>
<thead>
<tr>
<th>Secret Value</th>
<th>0.0</th>
<th>1.0</th>
<th>1e10</th>
<th>1e+200</th>
<th>1e-300</th>
<th>1e-42</th>
<th>256</th>
<th>257</th>
<th>1e-320</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>6.59</td>
<td>6.56</td>
<td>6.59</td>
<td>6.58</td>
<td>6.58</td>
<td>6.57</td>
<td>6.58</td>
<td>6.59</td>
<td>6.57</td>
</tr>
<tr>
<td>1.0</td>
<td>6.57</td>
<td>6.59</td>
<td>6.55</td>
<td>6.57</td>
<td>6.57</td>
<td>6.56</td>
<td>6.56</td>
<td>6.56</td>
<td>130.89</td>
</tr>
<tr>
<td>1e10</td>
<td>6.55</td>
<td>6.55</td>
<td>6.56</td>
<td>6.56</td>
<td>6.57</td>
<td>6.55</td>
<td>6.55</td>
<td>6.57</td>
<td>130.95</td>
</tr>
<tr>
<td>256</td>
<td>6.58</td>
<td>6.53</td>
<td>6.56</td>
<td>6.54</td>
<td>6.56</td>
<td>6.58</td>
<td>6.58</td>
<td>6.57</td>
<td>130.94</td>
</tr>
<tr>
<td>1e-320</td>
<td>6.59</td>
<td>130.90</td>
<td>130.92</td>
<td>130.94</td>
<td>6.59</td>
<td>6.58</td>
<td>130.95</td>
<td>130.91</td>
<td>6.56</td>
</tr>
</tbody>
</table>
Intel i5-4460 double-precision floating-point multiply

<table>
<thead>
<tr>
<th></th>
<th>0.0</th>
<th>1.0</th>
<th>1e10</th>
<th>1e+200</th>
<th>1e-300</th>
<th>1e-42</th>
<th>256</th>
<th>257</th>
<th>1e-320</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>6.59</td>
<td>6.56</td>
<td>6.59</td>
<td>6.58</td>
<td>6.58</td>
<td>6.57</td>
<td>6.58</td>
<td>6.59</td>
<td>6.57</td>
</tr>
<tr>
<td>1.0</td>
<td>6.57</td>
<td>6.59</td>
<td>6.55</td>
<td>6.57</td>
<td>6.57</td>
<td>6.56</td>
<td>6.56</td>
<td>6.56</td>
<td>130.89</td>
</tr>
<tr>
<td>1e10</td>
<td>6.55</td>
<td>6.55</td>
<td>6.56</td>
<td>6.58</td>
<td>6.56</td>
<td>6.56</td>
<td>6.56</td>
<td>6.57</td>
<td>130.95</td>
</tr>
<tr>
<td>256</td>
<td>6.58</td>
<td>6.53</td>
<td>6.56</td>
<td>6.54</td>
<td>6.56</td>
<td>6.56</td>
<td>6.58</td>
<td>6.57</td>
<td>130.94</td>
</tr>
<tr>
<td>1e-320</td>
<td>6.59</td>
<td>130.90</td>
<td>130.92</td>
<td>130.94</td>
<td>6.59</td>
<td>6.58</td>
<td>130.95</td>
<td>130.91</td>
<td>6.56</td>
</tr>
</tbody>
</table>
TOP SECRET\SI\ORCON\NOFORN
Analysis follows

Attacker’s Web Server
Pixel stealing takeaways

- Combines web security, hardware knowledge, and software design
- Side-channels are real, and viable 😊
Power-side channels

- The amount of *power* used by a computer is related to what it is doing
- How can you use this?
- Canvas
Cache side-channels

• **Idea:** The cache’s current state implies something about prior memory accesses

• **Insight:** Prior memory accesses can tell you a lot about a program!
Cache Basics

- **Cache lines**: fixed-size units of data
- **Cache set**: holds multiple cache lines
- **Set index**: assigns cache line to cache set
- **Eviction**: removing cache lines to make room
- **L1, L2, L3**: different levels of cache
- **Inclusive**: lines in L1/L2 must also be in L3

Many thanks to Craig Disselkoen for the animations.
Cache Attacks: Structure

Pre-Attack

Active Attack

Analysis

Many thanks to Craig Disselkoen for the animations.
Timing threshold
Eviction set

Active Attack
Prime targeted set
Wait
[Timed] Prime targeted set
Victim accesses targeted set

Analysis
Victim access if time > threshold

**PRE-ATTACK**

**ACTIVE ATTACK**

**ANALYSIS**

PRIME+PROBE
FLUSH+RELOAD

Cache set 0
Cache set 1 (requires shared memory)
Cache set 2

Pre-existing data  Attacker’s data  Victim’s data

Many thanks to Craig Disselkoen for the animations.
T-Table AES

- Tested against OpenSSL’s T-Table implementation of AES
- Traditional target for cache attacks
- No longer used in practice, but useful for comparison
- Chosen plaintext, key recovery
T-Table AES

**PRIME+PROBE**

![Graph of PRIME+PROBE](image)

**PRIME+ABORT**

![Graph of PRIME+ABORT](image)

Disselkoen et al. 2017