Security Engineering

Hardware Security 2: More on side channels and enclaves. Spectre, Rowhammer, Plundervolt. Codesign for security e.g. CHERI, MTE
Hardware Security 2

• Today we’ll focus more on more complex systems, e.g. your phone or laptop, or a data centre

• Many of these can be triggered without physical access!

• Attacks we’ll look at: Rowhammer, Cache side channels incl. Spectre/Meltdown, Plundervolt

• We’ll also look (briefly) at hardware defence mechanisms: TPMs, Enclaves, Physically Unclonable Function.

• We’ll also look at techniques to make correct software easier to write/debug: CHERI and MTE
Rowhammer

<table>
<thead>
<tr>
<th>I</th>
<th>S</th>
<th>_</th>
<th>A</th>
<th>D</th>
<th>M</th>
<th>I</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rowhammer

<table>
<thead>
<tr>
<th>I</th>
<th>S</th>
<th>_</th>
<th>A</th>
<th>D</th>
<th>M</th>
<th>I</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeatedly Activate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The diagram illustrates the concept of Rowhammer, a type of memory attack. It shows a repeated activation process that can lead to security vulnerabilities.
Rowhammer

Repeatedly Activate
**Rowhammer**

Repeatedly Activate

<table>
<thead>
<tr>
<th>I</th>
<th>S</th>
<th>_</th>
<th>A</th>
<th>D</th>
<th>M</th>
<th>I</th>
<th>N</th>
</tr>
</thead>
</table>

---

Rows of memory addresses are highlighted.
## Rowhammer

Repeatedly Activate

<table>
<thead>
<tr>
<th>I</th>
<th>S</th>
<th>_</th>
<th>A</th>
<th>D</th>
<th>M</th>
<th>I</th>
<th>N</th>
</tr>
</thead>
</table>

**Diagram:**

- Repeatedly Activate
- Colored cells indicate affected memory locations.
Side Channels, Spectre and Meltdown
Covert Channels

• E.g. Bell LaPadula
Covert Channels

• E.g. Bell LaPadula
Side Channels

Secret Data (e.g. Key)

Attacker

Victim
Example Channels

• Cache
• Disk timing
• Instruction timing
• CPU utilisation
• Clock frequency
• Power consumption
• Even erroneous behaviour e.g. differential analysis
• RF emissions (e.g. from monitors)
Cache Hierarchy

Main memory

L2 Cache

L1 Cache

CPU
Prime and Probe Cache Attack
Prime and Probe Cache Attack
Prime and Probe Cache Attack
Prime and Probe Cache Attack
### Example: AES Attack

- **Read in S-Box from Memory**

<table>
<thead>
<tr>
<th>AES S-Box</th>
</tr>
</thead>
<tbody>
<tr>
<td>X0</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0X</td>
</tr>
<tr>
<td>1X</td>
</tr>
<tr>
<td>2X</td>
</tr>
<tr>
<td>3X</td>
</tr>
<tr>
<td>4X</td>
</tr>
<tr>
<td>5X</td>
</tr>
<tr>
<td>6X</td>
</tr>
<tr>
<td>7X</td>
</tr>
<tr>
<td>8X</td>
</tr>
<tr>
<td>9X</td>
</tr>
<tr>
<td>aX</td>
</tr>
<tr>
<td>bX</td>
</tr>
<tr>
<td>cX</td>
</tr>
<tr>
<td>dX</td>
</tr>
<tr>
<td>eX</td>
</tr>
<tr>
<td>fX</td>
</tr>
</tbody>
</table>
Instructions can leak as well!

- E.g. $1 \times 1$ vs $2338572 \times 2314908$
- Arm v8.4-A Data Independent Timing flag
- No branching on secrets, either
Optimising Compilers

{
Load Sbox[0] – Sbox[128];
Return Sbox[Secret];
}
Optimising Compilers

{  
  Load Sbox[0] – Sbox[128];
  Return Sbox[Secret];
}

Optimising Compilers

{
    Return Sbox[Secret];
}

Hardware AES

• x86: Advanced Encryption Standard New Instructions – AES-NI

• Arm: Cryptographic Extensions in Aarch64, and/or separate crypto accelerators.
Speculative Side Channel Attacks

• E.g. Meltdown / Spectre
• Not the program that leaks anymore – it’s the speculative execution the processor does!
Speculative Execution

- Branch Predictor
- Branch Target Buffer
Speculative Execution
Speculative Execution

- Branch Predictor
- Branch Target Buffer
- Fetch & Decode
- Reorder Buffer
Speculative Execution

- Branch Predictor
- Branch Target Buffer
- Fetch & Decode
- Reorder Buffer
- Commit
Speculative Execution

- Branch Predictor
- Branch Target Buffer
- Fetch & Decode
- Reorder Buffer
- Commit

In Order | Out of Order | In Order
Speculative Execution

100s of Instructions

In Order

Out of Order

In Order
Meltdown

Flush (array);
Meltdown

Flush (array);

Try {
    Int x = *secret_banned_data;
    Int y = array[x];
}

Catch (Exception E) {
    printf("the above never happened, right?");
}
Meltdown

Flush (array);

Try {
    Int x = *secret_banned_data;
    Int y = array[x];
} Catch (Exception E) {
    printf("the above never happened, right?");
}
Meltdown

Flush (array);

Try {
    Int x = *secret_banned_data;
    Int y = array[x];
} Catch (Exception E) {
    printf("the above never happened, right?");
}
Flush (array);

Try {
    Int x = *secret_banned_data;
    Int y = array[x];
}

} Catch (Exception E) {
    printf("the above never happened, right?");
}

Actual
Meltdown

Flush (array);

Try {
    Int x = *secret_banned_data;
    Int y = array[x];
} Catch (Exception E) {
    printf(“the above never happened, right?”);
}
Flush (array);
Try {
    Int x = *secret_banned_data;
    Int y = array[x];
} Catch (Exception E) {
    printf(“the above never happened, right?”);
}
for(int z=0; z<size; z++) {
    time(array[z]);
}
Meltdown

• Effectively a bug
• Fixed using Kernel Page Table Isolation
Spectre v1

Int x = index_of_secret_out_of_bounds_data;
If(x < array_size) {
    y = array[x];
    z = array2[y];
}
Spectre v1

Int x = index_of_secret_out_of_bounds_data;
If (x < array_size) {
    y = array[x];
    z = array2[y];
}

X = 928309183902
Array_size = 100
Spectre v1

Int x = index_of_secret_out_of_bounds_data;
If(x < array_size) {
    y = array[x];
    z = array2[y];
}

True execution: No

X = 928309183902
Array_size = 100
Spectre v1

Int x = index_of_secret_out_of_bounds_data;
If(x < array_size) {
    y = array[x];
    z = array2[y];
}

True execution: No
Branch prediction: Yes

X = 928309183902
Array_size = 100
Spectre v1

Int x = index_of_secret_out_of_bounds_data;
If(x < array_size) {
  y = array[x];
  z = array2[y];
}

X = 928309183902
Array_size = 100
Leaks (Partial) Contents of Y

True execution: No
Branch prediction: Yes
Spectre v2

Int x = index_of_secret_out_of_bounds_data;
Call_safe_function();
Spectre v2

Int x = index_of_secret_out_of_bounds_data;
Call_safe_function();
Spectre v2

Int x = index_of_secret_out_of_bounds_data;
Call_safe_function();

If(x < array_size) {
    y = array[x];
    z = array2[y];
}
How can we use Spectre?

- Sandbox Escape
- Inter Process Communication
Conclusions

• Security isn't just limited to the “programmer's model”

• Don’t “roll your own crypto”

• Bugs can be around for a long time before they are discovered

• Make sure you’re aware of what the hardware is doing underneath your code!
What can hardware provide for YOU?
What can hardware provide for YOU?

• Hardware AES
• Trusted Platform Modules
• Enclaves
• Physically Unclonable Functions
• Codesign for Software Security: CHERI and MTE
Trusted Platform Module

- IO
- Crypto Hardware
- Persistent Keystore
- Memory
Chain of Verification

From google.github.io/tpm-js/
Enclaves (Trustzone, SGX, SEV)
Issues: Side Channels

- Shared Cache
  - Untrusted Code
    - AES Encryption
    - Enclave
    - Private Data
Issues: Side Channels

- Shared Cache
- Enclave 1: Secret Timing, Code
- Enclave 2: AES Encryption, Private Data
Issues: Am I really in an Enclave (SEV)?

1. Downgrade Firmware
2. Leak Root Key through Signature Check Vulnerability
3. Profit
Issues: Am I really in an Enclave (SEV)?

1. Downgrade Firmware
2. Leak Root Key through Signature Check Vulnerability
3. Profit

Hash( )

You
Issues: Am I really in an Enclave (SEV)?

1. Downgrade Firmware
2. Leak Root Key through Signature
   Check Vulnerability
3. Profit

Migrate

Real Enclave

Fake Enclave

Hash( )

You
Plundervolt

Enclave

Root-user Undervolt

Extract keys of AES, or redirect pointers to Outside enclave
Codesign for Security: CHERI vs MTE
MTE (Memory Tagging Extension)

| Unused (16 bits) | Virtual Address (48 bits) |
MTE (Memory Tagging Extension)

- Tag (4 bits)
- Virtual Address (48 bits)
MTE (Memory Tagging Extension)
MTE (Memory Tagging Extension)
MTE (Memory Tagging Extension)

[Diagram showing MTE with hexadecimal values and a shadow memory region marked with '13' and '7']
MTE (Memory Tagging Extension)

Shadow Memory
MTE (Memory Tagging Extension)
CHERI Capabilities

 Permissions (64 bit)
 Length (64 bit)
 Base (64 bit)
 Address (64 bit)
CHERI Capabilities
CHERI Capabilities

Permissions  Length  Base  Address (64 bit)
Yes  Yes  No  No

Shadow Memory
<table>
<thead>
<tr>
<th></th>
<th>CHERI</th>
<th>MTE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+ Guaranteed mitigation of spatial safety bugs (in pure-cap mode)</td>
<td>- Probabilistic mitigation of temporal/spatial bugs</td>
</tr>
<tr>
<td></td>
<td>+ 1-bit/128 shadow space</td>
<td>- 4-bit/128 shadow space</td>
</tr>
<tr>
<td></td>
<td>- 128-bit pointers</td>
<td>+ 64-bit pointers</td>
</tr>
<tr>
<td></td>
<td>- Standards-incompatible</td>
<td>+ Standards-compatible</td>
</tr>
</tbody>
</table>
Further Reading

- Security Engineering Chapters 18, 19, 5, 27
- https://meltdownattack.com/
- https://plundervolt.com/
- https://google.github.io/tpm-js/
- What you get is what you C: https://www.cl.cam.ac.uk/~rja14/Papers/whatyouc.pdf