Software Techniques to Mitigate Hardware Transient Execution Attacks

Scott Constable
Research Scientist, Intel Labs

#IEEESecDev https://secdev.ieee.org/2020
Legal Disclaimers

• Intel provides these materials as-is, with no express or implied warranties.
• All products, dates, and figures specified are preliminary, based on current expectations, and are subject to change without notice.
• Intel processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.
• Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at http://intel.com.
• Some results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.
• Intel and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
• *Other names and brands may be claimed as the property of others.
© Intel Corporation 2020
Brief Bio

• Security researcher in Intel Labs
• Member of the Intel Product Security Incident Response Team (iPSIRT)

The latest security information on Intel® products.

Vulnerability Handling Guidelines

The Intel Product Security Incident Response Team (iPSIRT) proactively searches for and responds to reported security vulnerabilities in Intel products. Working with members of the security community, customers and end users, the iPSIRT works to best ensure that security vulnerabilities affecting Intel production products are documented and solutions are released in a responsible fashion. Intel is committed to rapidly addressing security vulnerabilities affecting our customers and providing clear guidance on the solution, impact, severity and mitigation.

January 3, 2018: Spectre and Meltdown

Spectre Attacks: Exploiting Speculative Execution

Paul Kocher\textsuperscript{1}, Jann Horn\textsuperscript{2}, Anders Fogh\textsuperscript{3}, Daniel Genkin\textsuperscript{4},
Daniel Gruss\textsuperscript{5}, Werner Haas\textsuperscript{6}, Mike Hamburg\textsuperscript{7}, Moritz Lipp\textsuperscript{8},
Stefan Mangard\textsuperscript{9}, Thomas Prescher\textsuperscript{9}, Michael Schwarz\textsuperscript{5}, Yuval Yarom\textsuperscript{8}
\textsuperscript{1} Independent (www.paulkocher.com), \textsuperscript{2} Google Project Zero,
\textsuperscript{3} G DATA Advanced Analytics, \textsuperscript{4} University of Pennsylvania and University of Maryland,
\textsuperscript{5} Graz University of Technology, \textsuperscript{6} Cyberus Technology,
\textsuperscript{7} Rambus, Cryptography Research Division, \textsuperscript{8} University of Adelaide and Data61

Meltdown: Reading Kernel Memory from User Space

Moritz Lipp\textsuperscript{1}, Michael Schwarz\textsuperscript{1}, Daniel Gruss\textsuperscript{1}, Thomas Prescher\textsuperscript{2},
Werner Haas\textsuperscript{6}, Anders Fogh\textsuperscript{3}, Jann Horn\textsuperscript{4}, Stefan Mangard\textsuperscript{9},
Paul Kocher\textsuperscript{2}, Daniel Genkin\textsuperscript{6,9}, Yuval Yarom\textsuperscript{7}, Mike Hamburg\textsuperscript{8}
\textsuperscript{1}Graz University of Technology, \textsuperscript{2}Cyberus Technology GmbH,
\textsuperscript{3}G-Data Advanced Analytics, \textsuperscript{4}Google Project Zero,
\textsuperscript{5}Independent (www.paulkocher.com), \textsuperscript{6}University of Michigan,
\textsuperscript{7}University of Adelaide & Data61, \textsuperscript{8}Rambus, Cryptography Research Division
Side-Channel Attacks: A Brief History
The First Side-Channel Attacks

Side Channel Cryptanalysis of Product Ciphers

John Kelsey¹, Bruce Schneier¹, David Wagner², and Chris Hall¹

¹ Counterpane Systems
{hall,kelsey,schneier}@counterpane.com
101 E. Minnehaha Pkwy
Minneapolis, MN 55419
(612) 823-1098

² U.C. at Berkeley
daw@cs.berkeley.edu
Soda Hall
Berkeley, CA 94720-1776

Abstract. Building on the work of Kocher [Koc96], we introduce the notion of side-channel cryptanalysis: cryptanalysis using implementation data. We discuss the notion of side-channel attacks and the vulnerabilities they introduce, demonstrate side-channel attacks against three product ciphers—timing attack against IDEA, processor-flag attack against RC5, and Hamming weight attack against DES—and then generalize our research to other cryptosystems.

“A side-channel attack occurs when an attacker is able to use some additional information leaked from the implementation of a cryptographic function to cryptanalyze the function”
Hardware Side Channels

Unconventional Side Channels

Reaction Attacks Against Several Public-Key Cryptosystems

Chris Hall¹, Ian Goldberg², and Bruce Schneier³

¹ Counterpane Systems
{hall,schneier}@counterpane.com
101 E. Minnehaha Pkwy
Minneapolis, MN 55419
(612) 832-1098

² U.C. at Berkeley
iang@cs.berkeley.edu
Soda Hall
Berkeley, CA 94720-1776

Abstract. We present attacks against the McEliece Public-Key Cryptosystem, the Ajtai-Dwork Public-Key Cryptosystem, and variants of those systems. Most of these systems base their security on the apparent intractibility of one or more problems. The attacks we present do not violate the intractibility of the underlying problems, but instead obtain information about the private key or plaintext by watching the reaction of someone decrypting a given ciphertext with the private key. In the case of the McEliece system we must repeat the attack for each ciphertext we wish to decrypt, whereas for the Ajtai-Dwork system we are able to recover the private key.
Cache-based Side Channels and Prime+Probe

Step 1: Adversary primes cache set
Step 2: Victim accesses secret-dependent address
Step 3: Adversary probes cache set and detects 1 miss

Prime+Probe attack (solid blue = victim data; hatched orange = adversary data)

Lampson’s Early Insight: Covert Channels

A Note on the Confinement Problem

Butler W. Lampson
Xerox Palo Alto Research Center

Covert channels: “those not intended for information transfer at all”

Lampson’s example: “By varying its ratio of computing to input/output or its paging rate, the service can transmit information which a concurrently running process can receive by observing the performance of the system.”

URL http://doi.acm.org/10.1145/362375.362389.
Is it a Side Channel or a Covert Channel?

According to the classical definitions:

• It’s a covert channel if the adversary controls both input and output

\[\text{Input} \rightarrow \text{Adversary} \rightarrow \text{Adversary} \rightarrow \text{Output}\]

• It’s a side channel if the victim is leaking data to the adversary

\[\text{Input} \rightarrow \text{Victim} \rightarrow \text{Adversary} \rightarrow \text{Output}\]
Is it a Side Channel or a Covert Channel?

Intel’s (new) definitions:

• Channels are of two sorts: legitimate channels are those intended by system designers for the transmission of information, and all other channels are incidental channels.

• An incidental channel is functioning as a covert channel if the adversary controls both the input and output.

• An incidental channel is serving as a side channel if the adversary cannot exert control over the input and is only able to read the output.

January 3, 2018: Spectre and Meltdown

// sandboxed Spectre gadget
if (idx < array1_bound) {
    long x = array1[idx];
    long y = array2[x * 4096];
}
// x <- Prime+Probe on array2

Microarchitectural Instruction Stream

1. Speculative Prediction

2. Mis-predicted Path (Transient Execution)

3. Correct Path

4. Receive data via cache
Is it a Side Channel or a Covert Channel?

// sandboxed Spectre gadget
if (idx < array1_bound) {
    long x = array1[idx];
    long y = array2[x * 4096];
}

// x <- Prime+Probe on array2

In-Domain Transient Execution Attack:
The disclosure gadget is in an adversary-controlled sandbox within the victim’s domain.

Let’s Change Gears...
Intel Software Guard Extensions (SGX)

- Trusted Execution Environment embedded in a process
- With its own code and data
- Confidentiality and integrity protected
- With controlled entry points
- Supporting multiple threads
- With full access to application memory
Load Value Injection (LVI)

Victim SGX Enclave Code

- Store <malicious> → [0xABCDE000]
- Load [0x12345000] → RAX
- Jump RAX 😈 Jumps to <malicious>

Memory

<table>
<thead>
<tr>
<th>&lt;malicious&gt;</th>
<th>0xABCDE000</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;benign&gt;</td>
<td>0x12345000</td>
</tr>
</tbody>
</table>

4K Alias

Store Buffer

<table>
<thead>
<tr>
<th>&lt;malicious&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAX</td>
</tr>
<tr>
<td>&lt;malicious&gt;</td>
</tr>
</tbody>
</table>
Hmm...
Mitigating LVI

Canonical form of an LVI Gadget:

- **Prime** + Load + Transmit/Branch
- <some loading/storing instr>
- Load <memory> → <register>
- Transmit/Branch <register>

Load <memory> → <register>
Transmit/Branch <register>

Load <memory> → <register>
Transmit/Branch <register>

LFENCE

<some loading/storing instr>
Load <memory> → <register>
Transmit/Branch <register>

Load <memory> → <register>
LFENCE
Transmit/Branch <register>
Mitigating LVI

Mitigation for potential LVI gadgets:

*For all Load+Transmit gadgets in each function, every path in the control flow graph (CFG) from Load to Transmit is “cut” by at least one LFENCE instruction.*
A Revelation: An Algorithm to Optimize Execution Order Enforcement

“Our fence insertion algorithm first identifies the execution orders that a given memory model enforces automatically, and then inserts fences that enforce the rest.”

An Optimized Mitigation Approach for LVI

• Analyze the CFG to find Load+Transmit LVI gadgets
• Minimize $\sum_{e \in E} c_e x_e$ ...
  • $x_e$ is 1 if edge $e$ is cut by an LFENCE; otherwise it is 0
  • $c_e$ is proportional to the loop depth of edge $e$
• ...subject to $\sum_{e \in P} x_e \geq 1$, for all paths $P$ from $L_i$ to $T_i$
• This is known as the **minimum multi-cut problem**, which can be solved by a **Mixed Integer Programming** solver
A Revelation? One Drawback...

Parry uses Clang to translate C/C++ into LLVM’s intermediate representation...Parry uses Python to orchestrate the three major tasks in fence insertion: control-flow graph generation, order elimination, and fence insertion...We use SAGE and the default solver GLPK to solve the ILP.”
Optimized Solution in LLVM with SYMPHONY

1. Build a graph
2. **Eliminate** gadgets that are mitigated by existing fences

SYMPHONY solver: https://github.com/coin-or/SYMPHONY
Optimized Solution in LLVM with SYMPHONY

1. Build a graph
2. **Eliminate** gadgets that are mitigated by existing fences
3. **Cut** edges with the lowest possible cost to mitigate all remaining gadgets. This step uses the SYMPHONY solver
4. **Insert** new fences along the cuts, and emit the transformed program

SYMPHONY solver: [https://github.com/coin-or/SYMPHONY](https://github.com/coin-or/SYMPHONY)
LVI Software Mitigations now Available

• Optimized mitigations merged into LLVM/clang 10.0.1, 11.0
  • Enable with \texttt{-mlvi-hardening}, or \texttt{-mlvi-cfi}
  • Also available in the Rust SGX SDK
  • More info on the optimizations can be found here: https://software.intel.com/security-software-guidance/best-practices/optimized-mitigation-approach-load-value-injection

• Mitigations are also available for other compilers and frameworks:
  • https://devblogs.microsoft.com/cppblog/more-spectre-mitigations-in-msvc/
Processors Affected

Intel expects the following processors are potentially affected by the Load Value Injection issue. Refer to the disclosure for software developers and the Deep Dive: Load Value Injection for further details.

Processors marked with † may support Intel® Software Guard Extensions (Intel® SGX).

![ fiance family/processor number series | Affected]

<table>
<thead>
<tr>
<th>Family_Model</th>
<th>Step</th>
<th>Processor family/Processor number series</th>
<th>Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>06_2DH</td>
<td>All</td>
<td>Intel® Xeon® processor E5 Family based on Intel microarchitecture code name Sandy Bridge, Intel® Core™ i7-39xx Processor Extreme Edition</td>
<td>LVI-stale data</td>
</tr>
<tr>
<td>06_2AH</td>
<td>All</td>
<td>Intel® Xeon® processor E3-1200 product family; 2nd Generation Intel® Core™ i7, i5, i3 Processors 2xxx Series (Sandy Bridge)</td>
<td>LVI-stale data</td>
</tr>
<tr>
<td>06_3EH</td>
<td>All</td>
<td>Intel® Xeon® processor E7-8800/4800/2800 v2 product families based on Ivy Bridge-E microarchitecture</td>
<td>LVI-stale data</td>
</tr>
<tr>
<td>06_7DH, 06_7EH</td>
<td>All</td>
<td>10th Generation Intel® Core™ processors based on Ice Lake microarchitecture†</td>
<td>Not affected</td>
</tr>
</tbody>
</table>

Is it a Side Channel or a Covert Channel?

• An incidental channel is functioning as a **covert channel** if the adversary controls both the input and output

  ![Diagram of Covert Channel]

• An incidental channel is serving as a **side channel** if the adversary cannot exert control over the input and is only able to read the output

  ![Diagram of Side Channel]

Is it a Side Channel or a Covert Channel?

// (LVI example from before)
// victim enclave code
Store <malicious> → [0xABCD000]
Load [0x12345000] → RAX
Jump RAX (Jumps to disclosure gadget)

Cross-Domain Transient Execution Attack:
The disclosure gadget is in the victim’s domain.

// adversary non-enclave code
// Prime+Probe or Flush+Reload

How is this Terminology Helpful?

Cross-Domain Transient Execution Attack: The disclosure gadget is in the victim’s domain.

Domain-Bypass Transient Execution Attack: The disclosure gadget is in the adversary’s domain.

Load Value Injection (LVI)

Microarchitectural Data Sampling (MDS)

How is this Terminology Helpful?

Benefits software developers by helping to succinctly convey the impacts of newly discovered and existing vulnerabilities. For example:

- If a vulnerability is described as having **domain-bypass** impact, then hardware mitigation, microcode patches and/or software changes to the operating system (OS) or virtual machine monitor (VMM) are often required. Software changes to applications are usually not required.

- If a vulnerability is described as having **cross-domain** impact, then software changes to application, OS or VMM code may be desired to mitigate the vulnerability. These software changes may use processor features that prevent speculation or isolate branch predictors between modes. Developers should also take into consideration that cross-domain transient execution attacks are generally more difficult to launch than other kinds of transient execution attacks. Therefore, developers should use their own discretion when deciding how to mitigate their software (for example, whether to insert LFENCE automatically or only in places identified as vulnerable to Bounds Check Bypass).

- If a vulnerability is described as having **in-domain** impact, then affected software that uses sandboxing techniques may be vulnerable. Refer to the Managed Runtimes Deep Dive* for more information.

So... what next?
Continuous Research and Development

• New hardware designs mitigate many domain-bypass and cross-domain transient execution attacks

• Intel’s ongoing Side Channel Academic Program
  • Papers: Total (92); Tier-1 (67); Best Papers (15)

• Promising academic work on hardware-based and software-based mitigation techniques
Thank you!
(Questions are welcome)