Black Box CPU Reverse Engineering

Brad Morgan

Supervisors

Defence Science and Technology Group — Mr Nick Manser

The University of Adelaide — Dr Chitchanok Chuengsatiansup | Dr Yuval Yarom

Australian Government

Department of Defence

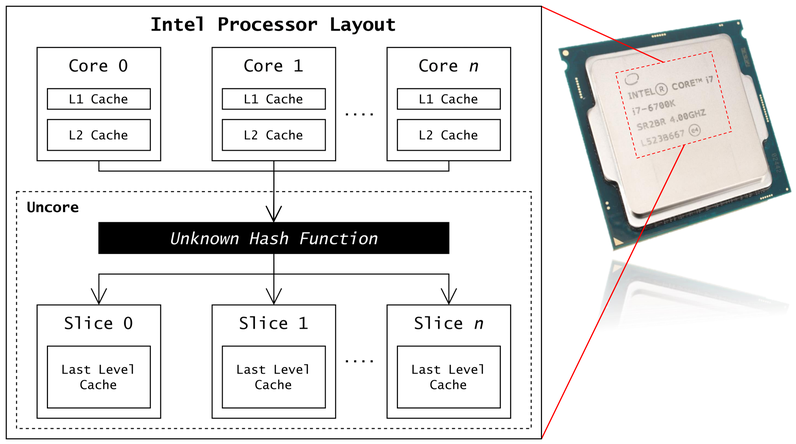
Defence Science and Technology Group

Introduction

Motivation

Intel and AMD do not disclose important design details of their processors

- Some vulnerabilities exist for years before discovery (e.g. Spectre, Meltdown)
- We seek to reveal underlying hardware decisions made by Intel
- These are crucial for furthering research into micro-architectural security



Objectives

- ✓ Reverse engineer an undisclosed **hash function** within Intel processors
  - Researchers cannot control memory in an Intel CPU without this
- ✓ Create tool to **automatically retrieve this hash function**
- ✓ Use the tool on a variety of Intel CPUs
- ✓ Report results in a simple format for use in future research

Background

...on an entire field of research in 7 dot points or less

CPU Architecture Crash Course

Current processors use caches to speed up memory accesses

- Each processor core has L1, L2 and Last Level Caches (LLC)
- On access, an address is stored in the CPU's cache for future retrieval
- The LLC is split into several shared slices; **One core, one slice**

Previous Discoveries

- Hash function takes memory address and computes LLC slice
- Evenly distributes memory accesses between each shared slice
- Memory addresses differing by 1 bit reveal how the slice mapping changes
- Machines with a non-2^n number of slices use a **second reduction step**

Disclaimer:
No CPUs were harmed in the process of conducting this project

Observations

Intel's hash function behaves as an XOR operation on the computed slice

- We use pairs of 'adjacent' address **differing on one bit** to find a piece of the hash function
- If an address bit is 1, slice mapping changes in a determinable way

Note: Below experiment conducted on a quad-core Intel i7-6700K
This CPU has a slice range of 0-3 (00-11 in binary)

Adjacent Address (Hex)	Adjacent Address Bits (33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00)	Observed Slice (S) [Binary]
0x267a00**0**40	1 1 0 0 1 1 0 0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 **0** 0 0 0 0 0 1 0 0 0 0 0	S_a = 0 [00]
0x267a00**8**40	1 1 0 0 1 1 0 0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 1 0 0 0 0 0	S_b = 2 [10]

Methodology

for frying your computer

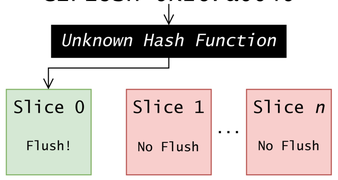
1) Find adjacent address pairs in available memory

Bit 11 Pair

0x267a00**0**40
0x267a00**8**40

2) Flush each address from CPU cache, observe slice access

CLFLUSH 0x267a0040



3) In failure, fall-back to access time

- Each core has an associated LLC slice
- Measure address access time from each processor core; find minimum

S_a = 0 [00]
S_b = 2 [10]

4) Compute XOR operation for bit

- Compare slices with XOR
- Store result as slice mapping

$S_a \oplus S_b = 2$

Therefore, Bit 11 XOR operation is 2 [10]

Hey give me a slice!

Results

Collating each XOR reveals the slice function

- XOR operations are visualised as binary
- If an address bit is 1, an XOR is applied to the result
- Bit 11's XOR is [10] as found previously
- We have a live USB with the tool installed to try it!

Slice Hash Function	Address Bit XOR Operations – Intel i7-6700K (33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00)
Bit 0	■ ■
Bit 1	■ ■

Non-2^n Core Processors: Two Stage Slice Mapping

Intel's above method only works for CPUs with 2^n slices

- 6, 10 & 12 core CPUs **reduce a large 2^n value into a smaller range**
- Our tool adapts and finds *sequences* of slice mappings using the found pairs of adjacent addresses
- A sequence is made by finding the slices for a series of sequential addresses
- The XOR function in 6 & 10 core CPUs generates 128 unique sequences
- We assign each sequence an ID (0-127) to recover the initial XOR function

Sequence ID Hash Function	Address Bit XOR Operations – i9-10900K (33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00)
Bit 0	■ ■
Bit 1	■ ■
Bit 2	■ ■
Bit 3	■ ■
Bit 4	■ ■
Bit 5	■ ■
Bit 6	■ ■

Each ID always maps to the same slice and creates a master sequence

- A logical circuit in the CPU reduces the initial output into slice values 0-9
- To find an address's slice, we initially determine its sequence ID
- Indexing this ID into the master sequence returns the address's slice value

ID Value	Master Sequence Slice Mapping – i9-10900K							
00	0505	3636	1414	2727	1414	2727	0589	3698
32	4141	7272	5098	6389	5050	6363	4189	7298
64	4141	7272	5050	6363	5050	6363	8941	9872
96	0505	3636	9814	8927	1414	2727	8905	9836