Port Contention for Fun and Profit

2019 IEEE Symposium on Security and Privacy

Natalie Muradyan

6.s983 – Spring 2023

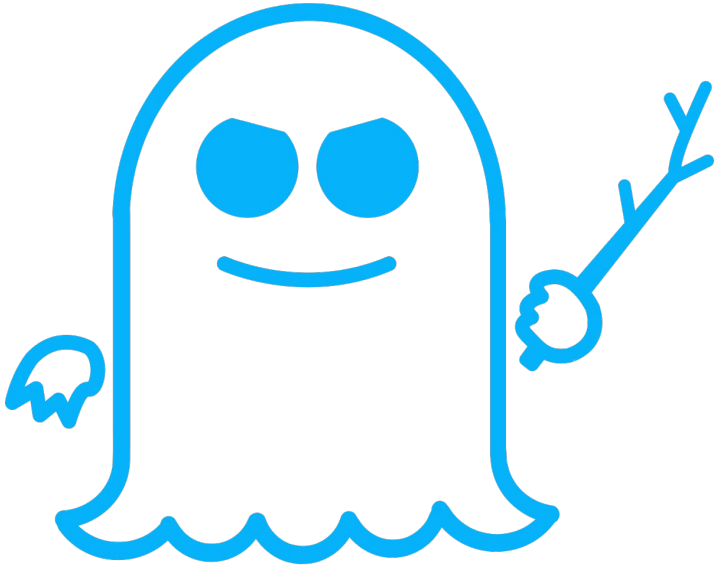
PortSmash

is a novel **side-channel** analysis technique that targets the **shared execution units** in **Simultaneous Multithreading (SMT)** architectures by monitoring the **port usage footprint** of the secret data dependent execution flows.

Side-Channel Attacks

Side-Channel attacks attempt measuring or exploiting indirect effects of the system or its hardware.

MELTDOWN



SPECTRE

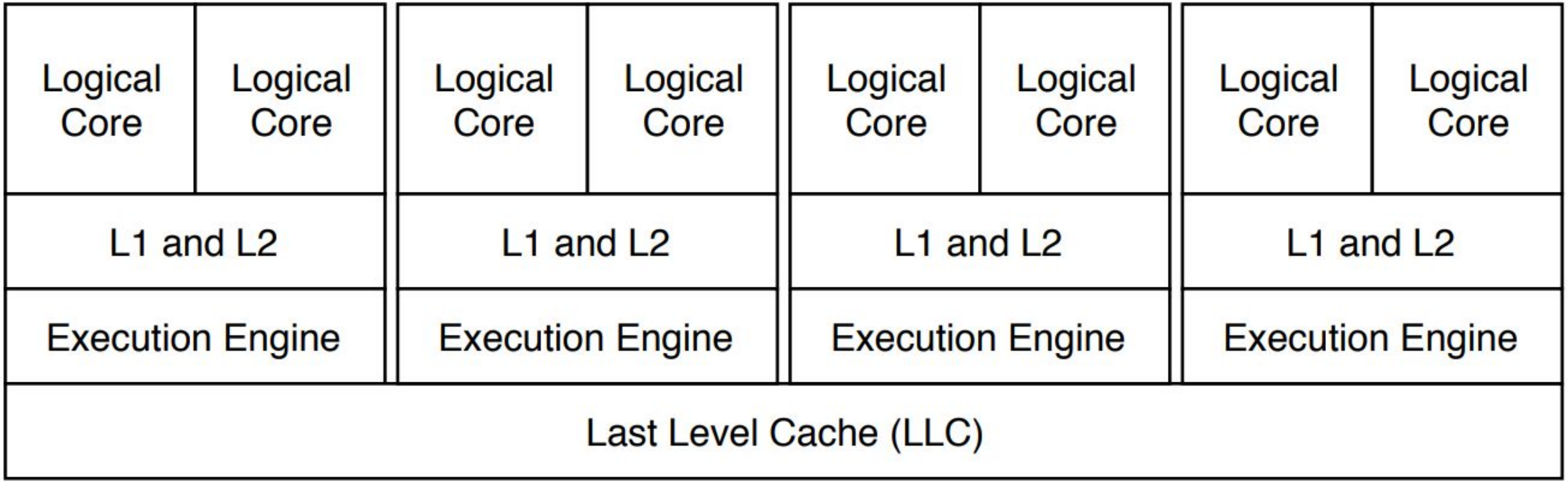
Simultaneous Multithreading (SMT)

- Each physical core is divided into multiple logical cores, allowing multiple threads to execute simultaneously on the same physical core.
- Logical cores share various hardware resources, including ports to the execution units.

Hyper-Threading (HT)

- Intel's proprietary simultaneous multithreading (SMT) implementation used to improve parallelization of computations performed on x86 microprocessors.

Hyper-Threading



“Researchers knew that **resource sharing leads to resource contention**, and it took *a remarkably short time* to notice that contention introduces timing variations during execution, which can be used as a covert channel, and as a side-channel.”

- A. C. Aldaya et al.

CACHE MISSING FOR FUN AND PROFIT

Cheap Hardware Parallelism Implies Cheap Security

r Architecture*

Covert Shotgun

Anders Fogh / September 27, 2016 / meta

New I cks

CacheBleed: A Timing Attack on OpenSSL Constant Time RSA

Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks

Covert Shotgun

An automated framework to find SMT covert channels.

1. Enumerate all instruction pairs in the ISA.
2. Duplicate each instruction a few times.
3. Run each instruction block **in parallel on the same physical core but separate logical cores**.
4. Measure the **clock-cycle performance**.
5. Analyze the resulting table for timing discrepancies.
6. Identify potential **covert channels** based on timing discrepancies.

Covert Shotgun Open Questions

“Another interesting project would be **identifying [subsystems]** which are being congested by specific instructions”

“it would be interesting to investigate to what extent these **covert channels extend to spying**”

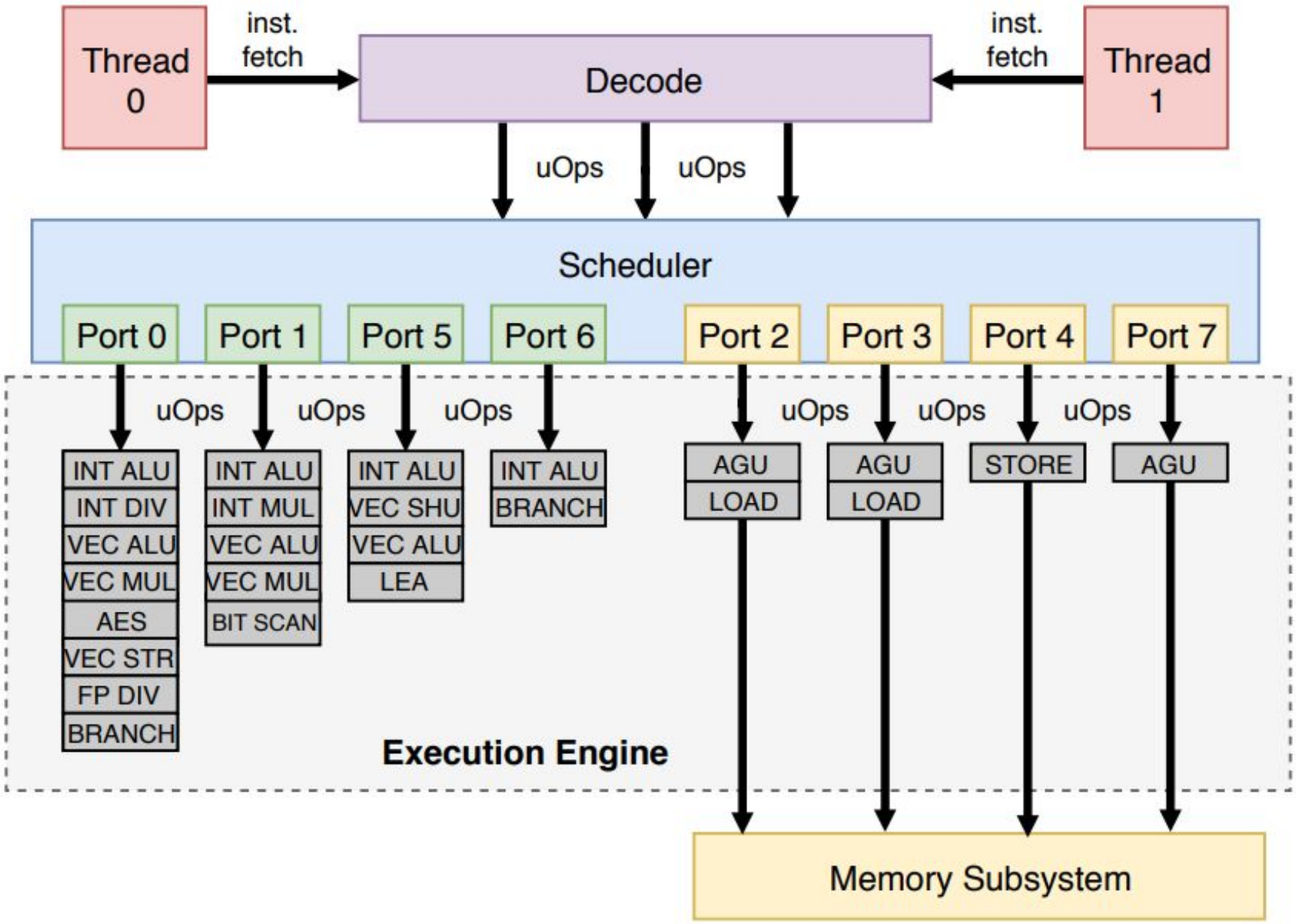

Thread 0
inst. fetch
Decode
inst. fetch
Thread 1
uOps
uOps
Scheduler
Port 0
Port 1
Port 5
Port 6
Port 2
Port 3
Port 4
Port 7
uOps
uOps
uOps
uOps
uOps
uOps
INT ALU
INT DIV
VEC ALU
VEC MUL
AES
VEC STR
FP DIV
BRANCH
INT ALU
INT MUL
VEC ALU
VEC MUL
BIT SCAN
INT ALU
VEC SHU
VEC ALU
LEA
INT ALU
BRANCH
AGU
LOAD
AGU
LOAD
STORE
AGU
Execution Engine
Memory Subsystem

TABLE I

SELECTIVE INSTRUCTIONS. ALL OPERANDS ARE REGISTERS, WITH NO MEMORY OPS. THROUGHPUT IS RECIPROCAL.

Instruction	Ports	Latency	Throughput
`add`	0 1 5 6	1	0.25
`crc32`	1	3	1
`popcnt`	1	3	1
`vpermd`	5	3	1
`vpbroadcastd`	5	3	1

TABLE II

RESULTS OVER A THOUSAND TRIALS. AVERAGE CYCLES ARE IN THOUSANDS, RELATIVE STANDARD DEVIATION IN PERCENTAGE.

		Diff. Phys. Core		Same Phys. Core	
Alice	Bob	Cycles	Rel. SD	Cycles	Rel. SD
Port 1	Port 1	203331	0.32%	**408322**	0.05%
Port 1	Port 5	203322	0.25%	203820	0.07%
Port 5	Port 1	203334	0.31%	203487	0.07%
Port 5	Port 5	203328	0.26%	**404941**	0.05%

```
mov $COUNT, %rcx

1:
lfence
rdtsc
lfence
mov %rax, %rsi

#ifdef P1
.rept 48
crc32 %r8, %r8
crc32 %r9, %r9
crc32 %r10, %r10
.endr
#elif defined(P5)
.rept 48
vpermd %ymm0, %ymm1, %ymm0
vpermd %ymm2, %ymm3, %ymm2
vpermd %ymm4, %ymm5, %ymm4
.endr
#elif defined(P0156)
.rept 64
add %r8, %r8
add %r9, %r9
add %r10, %r10
add %r11, %r11
.endr
#else
#error No ports defined
#endif

lfence
rdtsc
shl $32, %rax
or %rsi, %rax
mov %rax, (%rdi)
add $8, %rdi
dec %rcx
jnz 1b
```

Instruction	Ports
add	0 1 5 6
crc32	1
popcnt	1
vpermd	5
vpbroadcastd	5

Fig. 3. The PORTSMASH technique with multiple build-time port configurations P1, P5, and P0156.

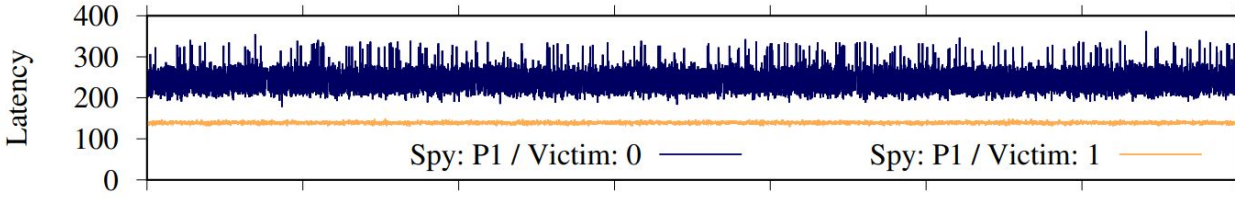
```
mov $COUNT, %rcx

1:
lfence
rdtsc
lfence
mov %rax, %rsi

#ifdef P1
.rept 48
crc32 %r8, %r8
crc32 %r9, %r9
crc32 %r10, %r10
.endr
#elif defined(P5)
.rept 48
vpermd %ymm0, %ymm1, %ymm0
vpermd %ymm2, %ymm3, %ymm2
vpermd %ymm4, %ymm5, %ymm4
.endr
```

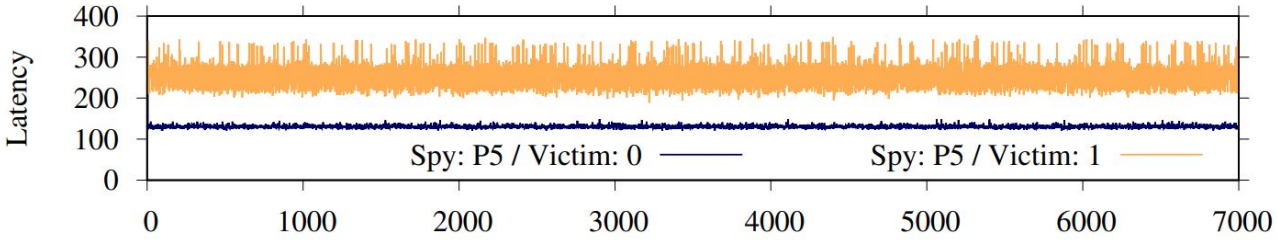
```
#elif defined(P0156)
.rept 64
add %r8, %r8
add %r9, %r9
add %r10, %r10
add %r11, %r11
.endr
#else
#error No ports defined
#endif

lfence
rdtsc
shl $32, %rax
or %rsi, %rax
mov %rax, (%rdi)
add $8, %rdi
dec %rcx
jnz 1b
```

The PORTSMASH technique with multiple build-time port configurations P1, P5, and P0156.

```
30f0  <x64_foo>:
30f0  test    %rdi,%rdi
30f3  je      4100 <x64_foo+0x1010>
30f9  jmpq    4120 <x64_foo+0x1030>
....
4100  popcnt %r8,%r8
4105  popcnt %r9,%r9
410a  popcnt %r10,%r10
410f  popcnt %r8,%r8
4114  popcnt %r9,%r9
4119  popcnt %r10,%r10
411e  jmp     4100 <x64_foo+0x1010>
4120  vpbroadcastd %xmm0,%ymm0
4125  vpbroadcastd %xmm1,%ymm1
412a  vpbroadcastd %xmm2,%ymm2
412f  vpbroadcastd %xmm0,%ymm0
4134  vpbroadcastd %xmm1,%ymm1
4139  vpbroadcastd %xmm2,%ymm2
413e  jmp     4120 <x64_foo+0x1030>
4140  retq
```

Victim with port footprint at port 1 and port 5.





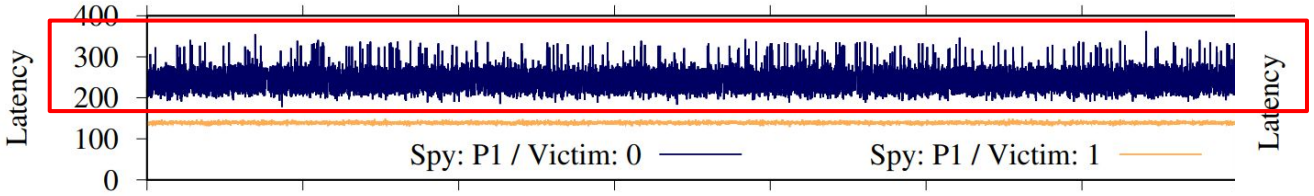
```
mov $COUNT, %rcx

1:
lfence
rdtsc
lfence
mov %rax, %rsi

#ifdef P1
.rept 48
crc32 %r8, %r8
crc32 %r9, %r9
crc32 %r10, %r10
.endr
#elif defined(P5)
.rept 48
vpermd %ymm0, %ymm1, %ymm0
vpermd %ymm2, %ymm3, %ymm2
vpermd %ymm4, %ymm5, %ymm4
.endr
```

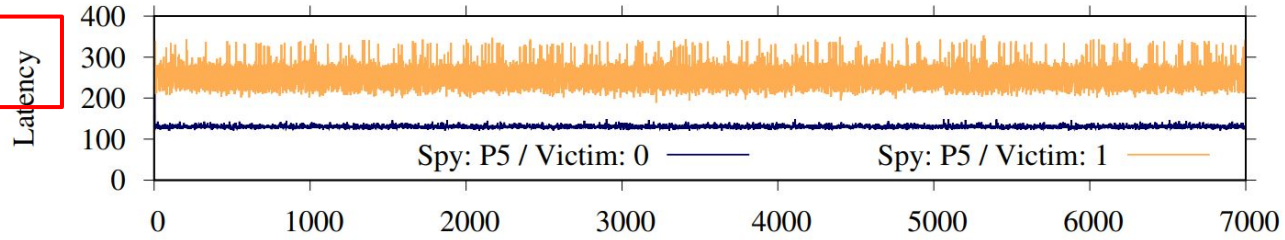
```
#elif defined(P0156)
.rept 64
add %r8, %r8
add %r9, %r9
add %r10, %r10
add %r11, %r11
.endr
#else
#error No ports defined
#endif

lfence
rdtsc
shl $32, %rax
or %rsi, %rax
mov %rax, (%rdi)
add $8, %rdi
dec %rcx
jnz 1b
```

The PORTSMASH technique with multiple build-time port configurations P1, P5, and P0156.

```
30f0  <x64_foo>:
30f0  test    %rdi,%rdi
30f3  je      4100 <x64_foo+0x1010>
30f9  jmpq    4120 <x64_foo+0x1030>
....
4100  popcnt %r8,%r8
4105  popcnt %r9,%r9
410a  popcnt %r10,%r10
410f  popcnt %r8,%r8
4114  popcnt %r9,%r9
4119  popcnt %r10,%r10
411e  jmp     4100 <x64_foo+0x1010>
4120  vpbroadcastd %xmm0,%ymm0
4125  vpbroadcastd %xmm1,%ymm1
412a  vpbroadcastd %xmm2,%ymm2
412f  vpbroadcastd %xmm0,%ymm0
4134  vpbroadcastd %xmm1,%ymm1
4139  vpbroadcastd %xmm2,%ymm2
413e  jmp     4120 <x64_foo+0x1030>
4140  retq
```

Victim with port footprint at port 1 and port 5.





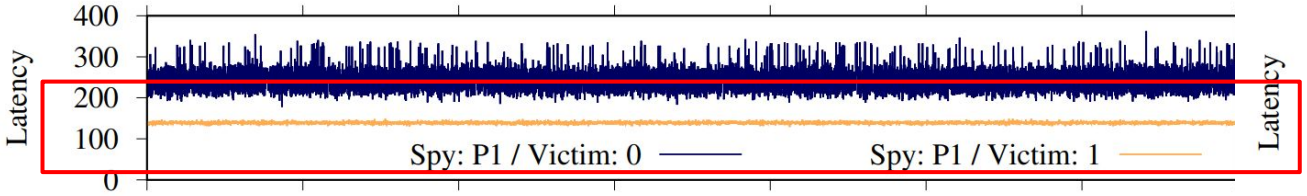
```
mov $COUNT, %rcx

1:
lfence
rdtsc
lfence
mov %rax, %rsi

#ifdef P1
.rept 48
crc32 %r8, %r8
crc32 %r9, %r9
crc32 %r10, %r10
.endr
#elif defined(P5)
.rept 48
vpermd %ymm0, %ymm1, %ymm0
vpermd %ymm2, %ymm3, %ymm2
vpermd %ymm4, %ymm5, %ymm4
.endr
```

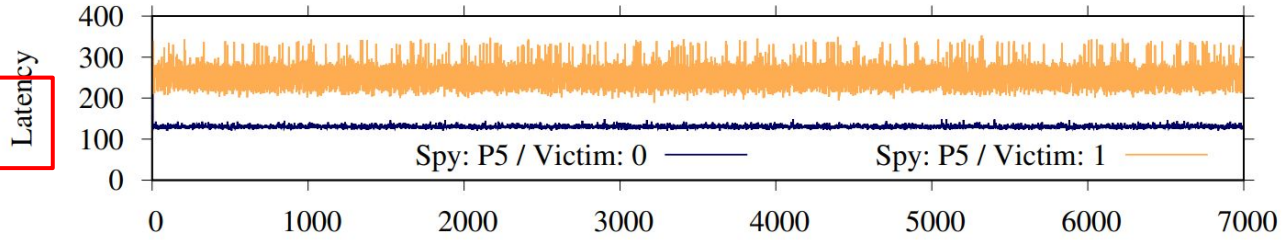
```
#elif defined(P0156)
.rept 64
add %r8, %r8
add %r9, %r9
add %r10, %r10
add %r11, %r11
.endr
#else
#error No ports defined
#endif

lfence
rdtsc
shl $32, %rax
or %rsi, %rax
mov %rax, (%rdi)
add $8, %rdi
dec %rcx
jnz 1b
```

The PORTSMASH technique with multiple build-time port configurations P1, P5, and P0156.

```
30f0  <x64_foo>:
30f0  test    %rdi,%rdi
30f3  je      4100 <x64_foo+0x1010>
30f9  jmpq    4120 <x64_foo+0x1030>
....
4100  popcnt %r8,%r8
4105  popcnt %r9,%r9
410a  popcnt %r10,%r10
410f  popcnt %r8,%r8
4114  popcnt %r9,%r9
4119  popcnt %r10,%r10
411e  jmp     4100 <x64_foo+0x1010>
4120  vpbroadcastd %xmm0,%ymm0
4125  vpbroadcastd %xmm1,%ymm1
412a  vpbroadcastd %xmm2,%ymm2
412f  vpbroadcastd %xmm0,%ymm0
4134  vpbroadcastd %xmm1,%ymm1
4139  vpbroadcastd %xmm2,%ymm2
413e  jmp     4120 <x64_foo+0x1030>
4140  retq
```

Victim with port footprint at port 1 and port 5.





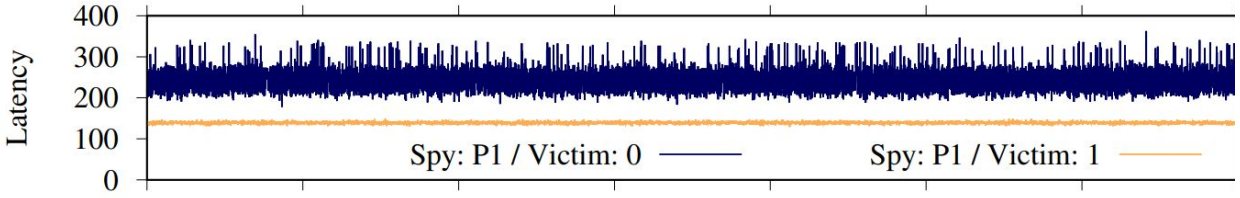
```
mov $COUNT, %rcx

1:
lfence
rdtsc
lfence
mov %rax, %rsi

#ifdef P1
.rept 48
crc32 %r8, %r8
crc32 %r9, %r9
crc32 %r10, %r10
.endr
#elif defined(P5)
.rept 48
vpermd %ymm0, %ymm1, %ymm0
vpermd %ymm2, %ymm3, %ymm2
vpermd %ymm4, %ymm5, %ymm4
.endr
```

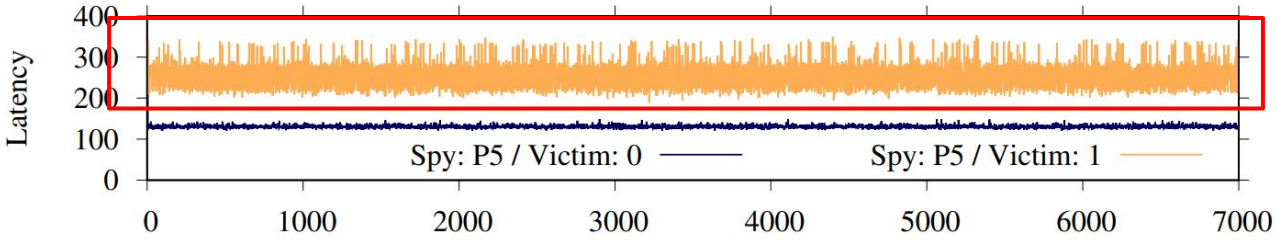
```
#elif defined(P0156)
.rept 64
add %r8, %r8
add %r9, %r9
add %r10, %r10
add %r11, %r11
.endr
#else
#error No ports defined
#endif

lfence
rdtsc
shl $32, %rax
or %rsi, %rax
mov %rax, (%rdi)
add $8, %rdi
dec %rcx
jnz 1b
```

The PORTSMASH technique with multiple build-time port configurations P1, P5, and P0156.

```
30f0  <x64_foo>:
30f0  test    %rdi,%rdi
30f3  je      4100 <x64_foo+0x1010>
30f9  jmpq    4120 <x64_foo+0x1030>
....
4100  popcnt  %r8,%r8
4105  popcnt  %r9,%r9
410a  popcnt  %r10,%r10
410f  popcnt  %r8,%r8
4114  popcnt  %r9,%r9
4119  popcnt  %r10,%r10
411e  jmp     4100 <x64_foo+0x1010>
4120  vpbroadcastd %xmm0,%ymm0
4125  vpbroadcastd %xmm1,%ymm1
412a  vpbroadcastd %xmm2,%ymm2
412f  vpbroadcastd %xmm0,%ymm0
4134  vpbroadcastd %xmm1,%ymm1
4139  vpbroadcastd %xmm2,%ymm2
413e  jmp     4120 <x64_foo+0x1030>
4140  retq
```

Victim with port footprint at port 1 and port 5.





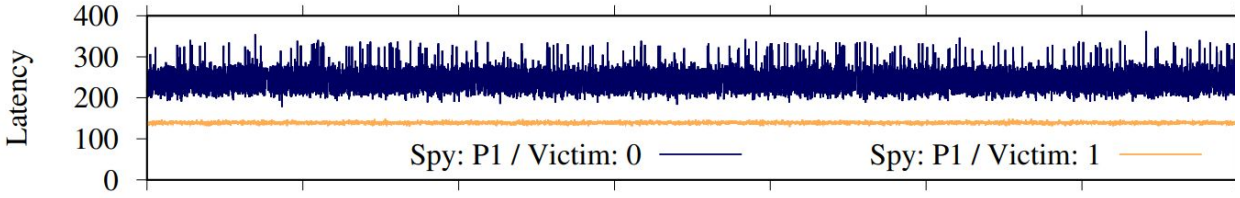
```
mov $COUNT, %rcx

1:
lfence
rdtsc
lfence
mov %rax, %rsi

#ifdef P1
.rept 48
crc32 %r8, %r8
crc32 %r9, %r9
crc32 %r10, %r10
.endr
#elif defined(P5)
.rept 48
vpermd %ymm0, %ymm1, %ymm0
vpermd %ymm2, %ymm3, %ymm2
vpermd %ymm4, %ymm5, %ymm4
.endr
```

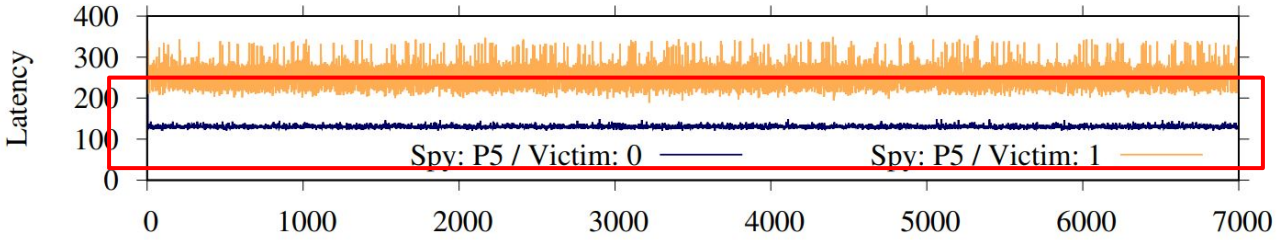
```
#elif defined(P0156)
.rept 64
add %r8, %r8
add %r9, %r9
add %r10, %r10
add %r11, %r11
.endr
#else
#error No ports defined
#endif

lfence
rdtsc
shl $32, %rax
or %rsi, %rax
mov %rax, (%rdi)
add $8, %rdi
dec %rcx
jnz 1b
```

The PORTSMASH technique with multiple build-time port configurations P1, P5, and P0156.

```
30f0  <x64_foo>:
30f0  test    %rdi,%rdi
30f3  je      4100 <x64_foo+0x1010>
30f9  jmpq    4120 <x64_foo+0x1030>
....
4100  popcnt %r8,%r8
4105  popcnt %r9,%r9
410a  popcnt %r10,%r10
410f  popcnt %r8,%r8
4114  popcnt %r9,%r9
4119  popcnt %r10,%r10
411e  jmp     4100 <x64_foo+0x1010>
4120  vpbroadcastd %xmm0,%ymm0
4125  vpbroadcastd %xmm1,%ymm1
412a  vpbroadcastd %xmm2,%ymm2
412f  vpbroadcastd %xmm0,%ymm0
4134  vpbroadcastd %xmm1,%ymm1
4139  vpbroadcastd %xmm2,%ymm2
413e  jmp     4120 <x64_foo+0x1030>
4140  retq
```

Victim with port footprint at port 1 and port 5.





P-384

is a type of **elliptic curve cryptography** that uses a prime field of size 384 bits. At the time of writing the paper, P-384 was **the only compliant ECC option** for **Secret and Top Secret levels** as approved by the NSA.

During **OpenSSL P-384 ECDSA signature generation**, PortSmash can measure the **timing variations** due to **port contention**.

Real World Example

PortSmash allows to implement an **end-to-end P-384 private key recovery attack**. The attack has three phases:

1. **Procurement phase**: the attack targets a stunnel TLS server with a P-384 certificate, measuring port contention with a Spy while the server produces ECDSA signatures.
2. **Signal processing phase**: the collected traces are filtered to obtain partial ECDSA nonce information for each digital signature.
3. **Key recovery phase**: the partial nonce information is used in a lattice attack to fully recover the server's P-384 private key.

Mitigations?

CVE-2018-5407: new side-channel vulnerability on SMT/Hyper-Threading architectures

---

From: Billy Brumley <bbrumley () gmail com>
Date: Fri, 2 Nov 2018 00:12:27 +0200

```
## Fix

Disable SMT/Hyper-Threading in the bios

Upgrade to OpenSSL 1.1.1 (or >= 1.1.0i if you are looking for patches)
```

seclists.org/oss-sec/2018/q4/123

Mitigations?

Published:
11/02/2018

After careful assessment, Intel determined that this method was similar to previously disclosed execution timing side channels and not a variation of speculative execution side channels such as Spectre, Meltdown, and L1TF. Existing programming best practices, such as employing constant execution timing and/or avoiding control flows that vary depending on secret data, can mitigate against PortSmash.

Intel does not recommend turning off Intel HT Technology as a mitigation technique because other programming methods are effective and higher-performing.

intel.com/content/www/us/en/developer/articles/news/more-information-portsmash.html

Mitigations?

CVE-2018-5407: new side-channel vulnerability on SMT/Hyper-Threading architectures

---

From: Billy Brumley <bbrumley () gmail com>
Date: Fri, 2 Nov 2018 00:12:27 +0200

```
## Fix

Disable SMT/Hyper-Threading in the bios

Upgrade to OpenSSL 1.1.1 (or >= 1.1.0i if you are looking for patches)
```

seclists.org/oss-sec/2018/q4/123

Mitigations?

CVE-2018-5407 fix: ECC ladder #7593

Closed bbbrumley wants to merge 3 commits into `openssl:OpenSSL_1_0_2-stable` from `bbbrumley:bbb_102_ladder`

Conversation 23 | Commits 3 | Checks 0 | Files changed 3

bbbrumley commented on Nov 8, 2018 — Contributor

This is a backport of #6009 #6535 #6648 to 1.0.2. In the context of CVE-2018-5407, this changes the code path for generic curves over prime fields from wNAF to ladder scalar multiplication.

Read the technical details.

`CVE-2018-5407 fix: ECC ladder` 93acd59

romen closed this on Nov 12, 2018

github.com/openssl/openssl/pull/7593

Conclusion

- **SMT architectures** create vulnerabilities via **port contention**, allowing attackers to extract sensitive information from victims.
- The PortSmash technique features properties like **high adaptability** through **various configurations**, very **fine spatial granularity**, **high portability**, and **minimal prerequisites**.
- It is a **practical attack vector** with a **real-world end-to-end attack** against a TLS server, successfully recovering an ECDSA P-384 secret key.

Discussion

Labs / Cache Attacks

Cache Side Channel Lab

Similarity:

How does instruction block X affects the latency of instruction block Y.

Difference:

Operation X and the access to line Y do not need to happen sequentially.

Covert Shotgun

Anders Fogh / September 27, 2016 / meta

Similarity:

How does some eviction operation that change s the cache state, X, affect the cache line Y.

Difference:

Instruction block X and Y should happen in parallel.

Discussion

Assume cores C_0 and C_1 are two logical cores of the same physical core. To make efficient and fair use of the shared EE, a simple strategy for port allocation is as follows. Denote i the clock cycle number, $j = i \bmod 2$, and $\mathcal{P}$ the set of ports.

1) C_j is allotted $\mathcal{P}_j \subseteq \mathcal{P}$ such that $|\mathcal{P} \setminus \mathcal{P}_j|$ is minimal.
2) C_{1-j} is allotted $\mathcal{P}_{1-j} = \mathcal{P} \setminus \mathcal{P}_j$.

There are two extremes in this strategy. For instance, if C_0 and C_1 are executing fully pipelined code with no hazards, yet make use of disjoint ports, then both C_0 and C_1 can issue in every clock cycle since there is no port contention. On the other hand, if C_0 and C_1 are utilizing the same ports, then C_0 and C_1 alternate, issuing every other clock cycle, realizing only half the throughput performance-wise.

Discussion

- **crc32**: Performs a cyclic redundancy check (CRC) on a specified data stream. Useful in error detection and correction, and data verification applications.
- **popcnt**: Counts the number of 1 bits in a data stream. Used in algorithms involving bit manipulation or searching, in optimization of programs that require counting or accumulation of data.
- **vpermd**: Performs a vector permute operation on the source and destination operands. Useful in applications that require reordering of data, such as multimedia processing or data compression.
- **vpbroadcastb**: Broadcasts a byte-sized value to all elements of a vector. Used in applications that require initialization of vector data or constant propagation.

Instruction	Ports
`add`	0 1 5 6
`crc32`	1
`popcnt`	1
`vpermd`	5
`vpbroadcastd`	5

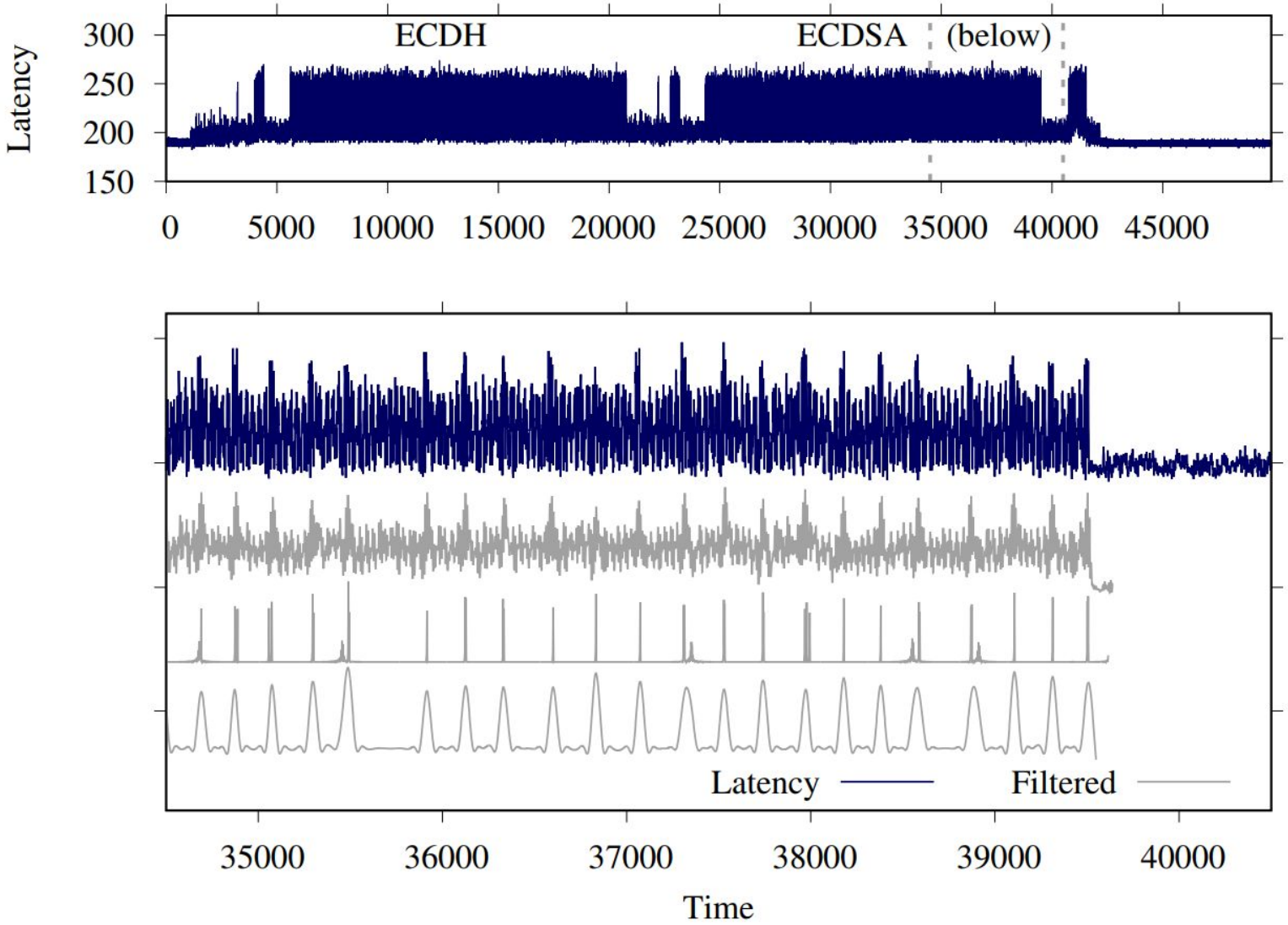
Discussion

```
30f0  <x64_foo>:
30f0  test    %rdi,%rdi
30f3  je      4100 <x64_foo+0x1010>
30f9  jmpq    4120 <x64_foo+0x1030>
....
4100  popcnt %r8,%r8
4105  popcnt %r9,%r9
410a  popcnt %r10,%r10
410f  popcnt %r8,%r8
4114  popcnt %r9,%r9
4119  popcnt %r10,%r10
411e  jmp     4100 <x64_foo+0x1010>
4120  vpbroadcastd %xmm0,%ymm0
4125  vpbroadcastd %xmm1,%ymm1
412a  vpbroadcastd %xmm2,%ymm2
412f  vpbroadcastd %xmm0,%ymm0
4134  vpbroadcastd %xmm1,%ymm1
4139  vpbroadcastd %xmm2,%ymm2
413e  jmp     4120 <x64_foo+0x1030>
4140  retq
```

```
4150  <x64_bar>:
4150  test    %rdi,%rdi
4153  je      5100 <x64_bar+0xfb0>
4159  jmpq    5140 <x64_bar+0xff0>
....
5100  popcnt %r8,%r8
5105  popcnt %r9,%r9
510a  popcnt %r10,%r10
510f  popcnt %r8,%r8
5114  popcnt %r9,%r9
5119  popcnt %r10,%r10
511e  popcnt %r8,%r8
5123  popcnt %r9,%r9
5128  popcnt %r10,%r10
512d  popcnt %r8,%r8
5132  popcnt %r9,%r9
5137  popcnt %r10,%r10
513c  jmp     5100 <x64_bar+0xfb0>
513e  xchg    %ax,%ax
5140  vpbroadcastd %xmm0,%ymm0
5145  vpbroadcastd %xmm1,%ymm1
514a  vpbroadcastd %xmm2,%ymm2
514f  vpbroadcastd %xmm0,%ymm0
5154  vpbroadcastd %xmm1,%ymm1
5159  vpbroadcastd %xmm2,%ymm2
515e  vpbroadcastd %xmm0,%ymm0
5163  vpbroadcastd %xmm1,%ymm1
5168  vpbroadcastd %xmm2,%ymm2
516d  vpbroadcastd %xmm0,%ymm0
5172  vpbroadcastd %xmm1,%ymm1
5177  vpbroadcastd %xmm2,%ymm2
517c  jmp     5140 <x64_bar+0xff0>
517e  retq
```

Fig. 4. Two Victims with similar port footprint, i.e., port 1 and port 5, but different cache footprint. Left: Instructions span a single cache-line. Right: Instructions span multiple cache-lines.

Discussion


Latency
300
250
200
150
ECDH
ECDSA
(below)
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
Latency
Filtered
35000
36000
37000
38000
39000
40000
Time

Citations

[1] A. C. Aldaya, B. B. Brumley, S. ul Hassan, C. Pereida García and N. Tuveri, "Port Contention for Fun and Profit," 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2019, pp. 870-887, doi: 10.1109/SP.2019.00066.

[2] C. Percival, “Cache missing for fun and profit,” in BSDCan 2005, Ottawa, Canada, May 13-14, 2005, Proceedings, 2005. [Online]. Available: http://www.daemonology.net/papers/cachemissing.pdf

[3] O. Acıiçmez and J. Seifert, “Cheap hardware parallelism implies cheap security,” in Fourth International Workshop on Fault Diagnosis and Tolerance in Cryptography, 2007, FDTC 2007: Vienna, Austria, 10 September 2007, L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J. Seifert, Eds. IEEE Computer Society, 2007, pp. 80–91. [Online]. Available: https://doi.org/10.1109/FDTC.2007.4318988

[4] Z. Wang and R. B. Lee, “Covert and side channels due to processor architecture,” in Proceedings of the 22nd Annual Conference on Computer Security Applications, ACSAC 2006, Miami Beach, FL, USA, December 11-15, 2006. IEEE Computer Society, 2006, pp. 473–482. [Online]. Available: https://doi.org/10.1109/ACSAC.2006.20

Citations

[5] O. Aciiçmez, B. B. Brumley, and P. Grabher, “New results on instruction cache attacks,” in Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings, ser. Lecture Notes in Computer Science, S. Mangard and F. Standaert, Eds., vol. 6225. Springer, 2010, pp. 110–124. [Online]. Available: https://doi.org/10.1007/978-3-642-15031-9_8

[6] Y. Yarom, D. Genkin, and N. Heninger, “CacheBleed: A timing attack on OpenSSL constant time RSA,” in Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings, ser. Lecture Notes in Computer Science, B. Gierlichs and A. Y. Poschmann, Eds., vol. 9813. Springer, 2016, pp. 346–367. [Online]. Available: https://doi.org/10.1007/978-3-662-53140-2_17

[7] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, “Translation leakaside buffer: Defeating cache side-channel protections with TLB attacks,” in 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018, W. Enck and A. P. Felt, Eds. USENIX Association, 2018, pp. 955–972. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/gras

Citations

[8] Fogh, Author Anders. “Covert Shotgun.” cyber.wtf, 27 Sept. 2016, cyber.wtf/2016/09/27/covert-shotgun.

[9] “More Information on PortSmash.” Intel, www.intel.com/content/www/us/en/developer/articles/news/more-information-portsmash.html.

[10] Openssl. “CVE-2018-5407 Fix: ECC Ladder by Bbbrumley · Pull Request #7593 · Openssl/Openssl.” GitHub, github.com/openssl/openssl/pull/7593.

[11] Oss-sec: CVE-2018-5407: New Side-channel Vulnerability on SMT/Hyper-Threading Architectures. seclists.org/oss-sec/2018/q4/123.

[12] IEEE Symposium on Security and Privacy. “Port Contention for Fun and Profit.” YouTube, 28 May 2019, www.youtube.com/watch?v=ELs8U8zTk5o.

[13] Computerphile. “What’s Behind Port Smash? - Computerphile.” YouTube, 13 Nov. 2018, www.youtube.com/watch?v=k6PzjGwyKuY.