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.
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

<table>
<thead>
<tr>
<th>Logical Core</th>
<th>Logical Core</th>
<th>Logical Core</th>
<th>Logical Core</th>
<th>Logical Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 and L2</td>
<td>L1 and L2</td>
<td>L1 and L2</td>
<td>L1 and L2</td>
<td>L1 and L2</td>
</tr>
<tr>
<td>Execution Engine</td>
<td>Execution Engine</td>
<td>Execution Engine</td>
<td>Execution Engine</td>
<td>Execution Engine</td>
</tr>
</tbody>
</table>

Last Level Cache (LLC)
“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.
Cheap Hardware Parallelism Implies Cheap Security

Covert Shotgun
Anders Fogh / September 27, 2016 / meta

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”
### TABLE I
Selective instructions. All operands are registers, with no memory ops. Throughput is reciprocal.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Ports</th>
<th>Latency</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>1, 5</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>crc32</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>popcnt</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>vpermd</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>vpbroadcast</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

### TABLE II
Results over a thousand trials. Average cycles are in thousands, relative standard deviation in percentage.

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
<th>Diff. Phys. Core</th>
<th>Same Phys. Core</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Cycles</td>
<td>Rel. SD</td>
</tr>
<tr>
<td>Port 1</td>
<td>Port 1</td>
<td>203331</td>
<td>0.32%</td>
</tr>
<tr>
<td>Port 1</td>
<td>Port 5</td>
<td>203322</td>
<td>0.25%</td>
</tr>
<tr>
<td>Port 5</td>
<td>Port 1</td>
<td>203334</td>
<td>0.31%</td>
</tr>
<tr>
<td>Port 5</td>
<td>Port 5</td>
<td>203328</td>
<td>0.26%</td>
</tr>
</tbody>
</table>
mov $COUNT, %rcx

1:
lfence
rdtsc
lfence
mov %rax, %rsi

#ifdef P1
.rept 48
crc32 %r8, %r8
crc32 %r9, %r9
crc32 %r10, %r10
.endr
#elifdef P5
.rept 48
vpermd %ymm0, %ymm1, %ymm0
vpermd %ymm2, %ymm3, %ymm2
vpermd %ymm4, %ymm5, %ymm4
.endr
#endif

#elwi defined(P0156)
.rept 64
add %r8, %r8
add %r9, %r9
add %r10, %r10
add %r11, %r11
.endr

#error No ports defined
#endif

add
crc32
popcnt
vpermd
vpbroadcastd

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>0 1 5 6</td>
</tr>
<tr>
<td>crc32</td>
<td>1</td>
</tr>
<tr>
<td>popcnt</td>
<td>1</td>
</tr>
<tr>
<td>vpermd</td>
<td>5</td>
</tr>
<tr>
<td>vpbroadcastd</td>
<td>5</td>
</tr>
</tbody>
</table>

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

Victim with port footprint at port 1 and port 5.
The PORTSMASH technique with multiple build-time port configurations P1, P5, and P0156.

Victim with port footprint at port 1 and port 5.
The PORTSMASH technique with multiple build-time port configurations P1, P5, and P0156.

Victim with port footprint at port 1 and port 5.
The PORTSMASH technique with multiple build-time port configurations P1, P5, and P0156.

Victim with port footprint at port 1 and port 5.
The PORTSMASH technique with multiple build-time port configurations P1, P5, and P0156.

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)
Mitigations?

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.
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)
Mitigations?

CVE-2018-5407 fix: ECC ladder #7593

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

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**

**Similarity:**
How does instruction block X affect 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 changes 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, $j = i \mod 2$, 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.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>0 1 5 6</td>
</tr>
<tr>
<td>crc32</td>
<td>1</td>
</tr>
<tr>
<td>popcnt</td>
<td>1</td>
</tr>
<tr>
<td>vpermd</td>
<td>5</td>
</tr>
<tr>
<td>vpbroadcastb</td>
<td>5</td>
</tr>
</tbody>
</table>
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
Citations


Citations


Citations


