Tampere University

PORT CONTENTION FOR FUN AND PROFIT

40th IEEE Symposium on Security and Privacy

May 20-22, 2019

Alejandro Cabrera Aldaya[1], Billy Bob Brumley[2],
Sohaib ul Hassan[2], Cesar Pereida García[2], Nicola Tuveri[2]

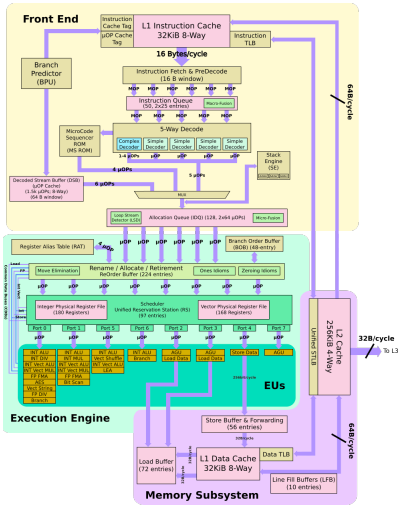
aldaya@gmail.com, {billy.brumley, sohaibulhassan, cesar.pereidagarcia, nicola.tuveri}@tuni.fi

[1]Universidad Tecnológica de la Habana (CUJAE), Habana, Cuba
[2]Network and Information Security Group (NISEC), Tampere University, Tampere, Finland

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.

Modern Microarchitecture design



Simultaneous Multithreading (SMT)

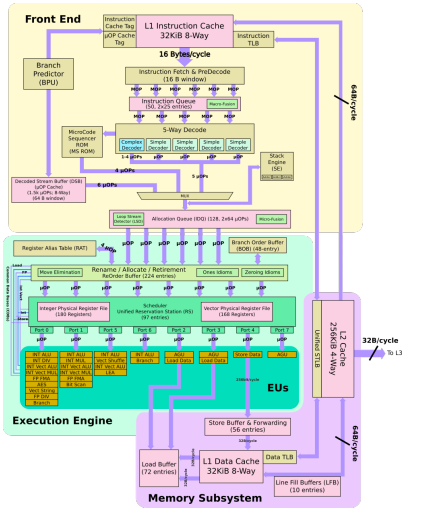
provides instruction level parallelism by supporting at least two logical cores per physical core and share components between the logical cores.

Logical Core	Logical Core	Logical Core	Logical Core	Logical Core	Logical Core	Logical Core	Logical Core
L1 and L2		L1 and L2		L1 and L2		L1 and L2	
Execution Engine		Execution Engine		Execution Engine		Execution Engine	
Last Level Cache (LLC)							

https://en.wikichip.org/wiki/intel/microarchitectures/kaby_lake

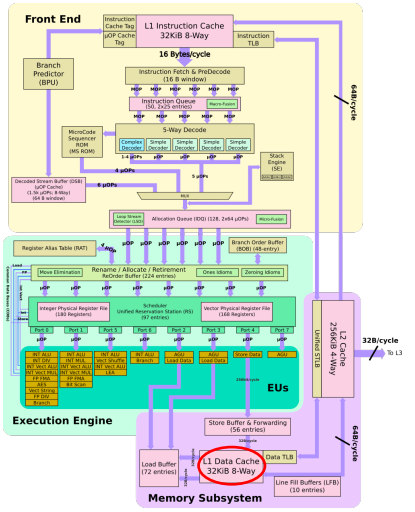
SMT Timing Attacks

- L1 data cache (Percival, 2005)
  - **OpenSSL 0.9.7c** RSA sliding window exponentiation
- Integer multiplication unit (Acıiçmez et al., 2007)
  - **OpenSSL 0.9.8e** sliding window exponentiation
- L1 instruction cache (Acıiçmez et al., 2010)
  - **OpenSSL 0.9.81** DSA secret key recovery
- CacheBleed: L1 data cache (Yarom et al., 2016)
  - **OpenSSL 1.0.2f** RSA exponentiation
- TLBleed (Gras et al., 2018)
  - **Libgcrypt** ECDSA and RSA



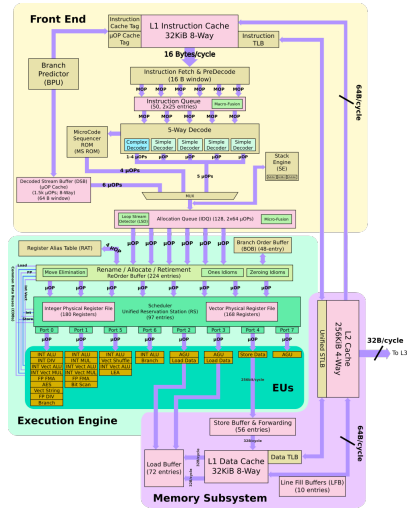
SMT Timing Attacks

- L1 data cache (Percival, 2005)
  - **OpenSSL 0.9.7c** RSA sliding window exponentiation
- Integer multiplication unit (Acıiçmez et al., 2007)
  - **OpenSSL 0.9.8e** sliding window exponentiation
- L1 instruction cache (Acıiçmez et al., 2010)
  - **OpenSSL 0.9.81** DSA secret key recovery
- CacheBleed: L1 data cache (Yarom et al., 2016)
  - **OpenSSL 1.0.2f** RSA exponentiation
- TLBleed (Gras et al., 2018)
  - **Libgcrypt** ECDSA and RSA



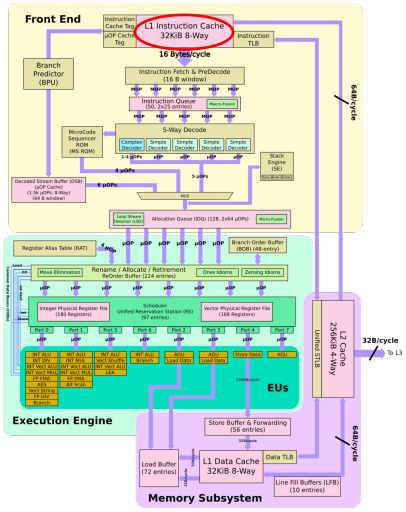
SMT Timing Attacks

- L1 data cache (Percival, 2005)
  - **OpenSSL 0.9.7c** RSA sliding window exponentiation
- Integer multiplication unit (Acıiçmez et al., 2007)
  - **OpenSSL 0.9.8e** sliding window exponentiation
- L1 instruction cache (Acıiçmez et al., 2010)
  - **OpenSSL 0.9.81** DSA secret key recovery
- CacheBleed: L1 data cache (Yarom et al., 2016)
  - **OpenSSL 1.0.2f** RSA exponentiation
- TLBleed (Gras et al., 2018)
  - **Libgcrypt** ECDSA and RSA



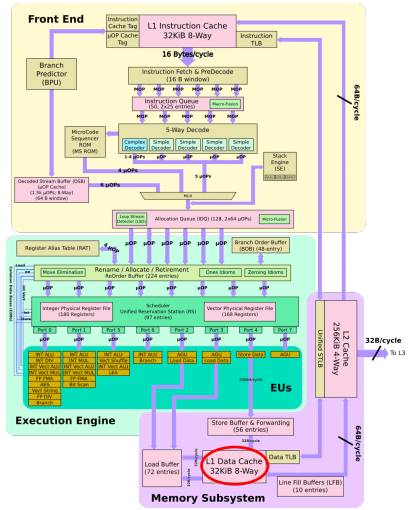
SMT Timing Attacks

- L1 data cache (Percival, 2005)
  - **OpenSSL 0.9.7c** RSA sliding window exponentiation
- Integer multiplication unit (Acıiçmez et al., 2007)
  - **OpenSSL 0.9.8e** sliding window exponentiation
- L1 instruction cache (Acıiçmez et al., 2010)
  - **OpenSSL 0.9.81** DSA secret key recovery
- CacheBleed: L1 data cache (Yarom et al., 2016)
  - **OpenSSL 1.0.2f** RSA exponentiation
- TLBleed (Gras et al., 2018)
  - **Libgcrypt** ECDSA and RSA



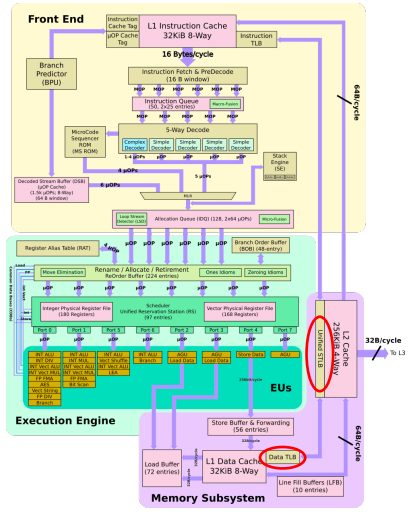
SMT Timing Attacks

- L1 data cache (Percival, 2005)
  - **OpenSSL 0.9.7c** RSA sliding window exponentiation
- Integer multiplication unit (Acıiçmez et al., 2007)
  - **OpenSSL 0.9.8e** sliding window exponentiation
- L1 instruction cache (Acıiçmez et al., 2010)
  - **OpenSSL 0.9.81** DSA secret key recovery
- CacheBleed: L1 data cache (Yarom et al., 2016)
  - **OpenSSL 1.0.2f** RSA exponentiation
- TLBleed (Gras et al., 2018)
  - **Libgcrypt** ECDSA and RSA



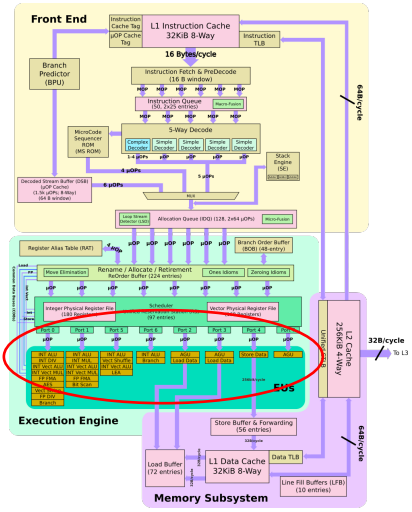
SMT Timing Attacks

- L1 data cache (Percival, 2005)
  - **OpenSSL 0.9.7c** RSA sliding window exponentiation
- Integer multiplication unit (Acıiçmez et al., 2007)
  - **OpenSSL 0.9.8e** sliding window exponentiation
- L1 instruction cache (Acıiçmez et al., 2010)
  - **OpenSSL 0.9.81** DSA secret key recovery
- CacheBleed: L1 data cache (Yarom et al., 2016)
  - **OpenSSL 1.0.2f** RSA exponentiation
- TLBLEED (Gras et al., 2018)
  - **Libgcrypt** ECDSA and RSA



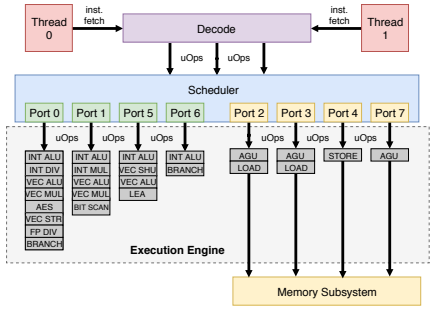
SMT Timing Attacks

- L1 data cache (Percival, 2005)
  - **OpenSSL 0.9.7c** RSA sliding window exponentiation
- Integer multiplication unit (Acıiçmez et al., 2007)
  - **OpenSSL 0.9.8e** sliding window exponentiation
- L1 instruction cache (Acıiçmez et al., 2010)
  - **OpenSSL 0.9.81** DSA secret key recovery
- CacheBleed: L1 data cache (Yarom et al., 2016)
  - **OpenSSL 1.0.2f** RSA exponentiation
- TLBleed (Gras et al., 2018)
  - **Libgcrypt** ECDSA and RSA



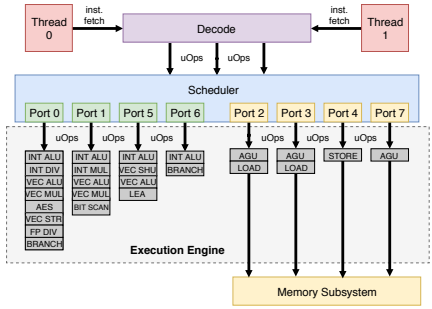
Port Contention in Execution Engine

- Execution Engine is responsible for executing instructions.
- Instructions are fetched, decoded to *uops* and scheduled to be executed.
- The scheduler issues the queued *uops* to the the execution ports.
- Ports are *channels* to a stack of execution units.



Port Contention in Execution Engine

- Covert Shotgun[2]: automated framework to find SMT covert channels (Fogh, 2016)
  - mentioned possible covert channels due to caching of decoded *uops*, port congestion, and execution unit congestion.

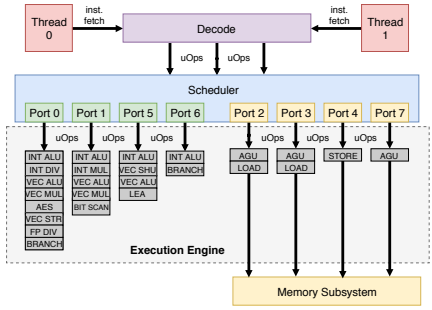


[2] https://cyber.wtf/2016/09/27/covert-shotgun/

Port Contention in Execution Engine

- Covert Shotgun[2]: automated framework to find SMT covert channels (Fogh, 2016)
  - mentioned possible covert channels due to caching of decoded *uops*, port congestion, and execution unit congestion.

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



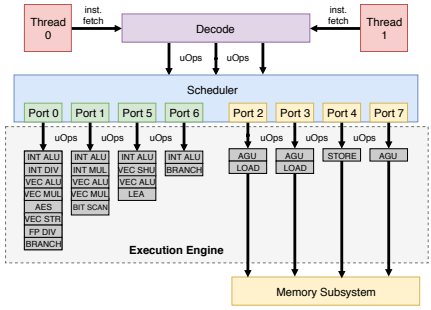
[2] https://cyber.wtf/2016/09/27/covert-shotgun/

Port Contention in Execution Engine

- Covert Shotgun[2]: automated framework to find SMT covert channels (Fogh, 2016)
  - mentioned possible covert channels due to caching of decoded *uops*, port congestion, and execution unit congestion.

"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"



[2] https://cyber.wtf/2016/09/27/covert-shotgun/

PortSmash Covert-Channel

PortSmash exploits the *shared Execution Engine* to create *timing side-channels* due to *port contention* among two or more processes running on different logical cores but the same physical cores

- We performed the experiment on Intel Core i7-7700HQ Kaby Lake.
- Alice first issues `crc32` blocks in an infinite loop, while Bob executes 2^{20} blocks of `crc32` and then `vpermd`, and vice versa.
- A block contains 3 said instruction with disjoint operands, to fill the pipeline, maximize throughput, and avoid hazards.
- Port contention is caused when both Alice and Bob issue instructions to the same port.

TABLE 1. SELECTIVE INSTRUCTIONS. ALL OPERANDS ARE REGISTERS, WITH NO MEMORY OPS.

Instruction	Ports	Latency	Reciprocal 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.

Alice	Bob	Diff. Phys. Core Cycles	Same Phys. Core Cycles
Port 1	Port 1	203331	**408322**
Port 1	Port 5	203322	203820
Port 5	Port 1	203334	203487
Port 5	Port 5	203328	**404941**

PortSmash Covert-Channel

PortSmash exploits the *shared Execution Engine* to create *timing side-channels* due to *port contention* among two or more processes running on different logical cores but the same physical cores

- We performed the experiment on Intel Core i7-7700HQ Kaby Lake.
- Alice first issues `crc32` blocks in an infinite loop, while Bob executes 2^{20} blocks of `crc32` and then `vpermd`, and vice versa.
- A block contains 3 said instruction with disjoint operands, to fill the pipeline, maximize throughput, and avoid hazards.
- Port contention is caused when both Alice and Bob issue instructions to the same port.

TABLE 1. SELECTIVE INSTRUCTIONS. ALL OPERANDS ARE REGISTERS, WITH NO MEMORY OPS.

Instruction	Ports	Latency	Reciprocal Throughput
`add`	0 1 5 6	1	0.25
Alice → `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.

Alice	Bob	Diff. Phys. Core Cycles	Same Phys. Core Cycles
Port 1	Port 1	203331	**408322**
Port 1	Port 5	203322	203820
Port 5	Port 1	203334	203487
Port 5	Port 5	203328	**404941**

PortSmash Covert-Channel

PortSmash exploits the *shared Execution Engine* to create *timing side-channels* due to *port contention* among two or more processes running on different logical cores but the same physical cores

- We performed the experiment on Intel Core i7-7700HQ Kaby Lake.
- Alice first issues `crc32` blocks in an infinite loop, while Bob executes 2^{20} blocks of `crc32` and then `vpermd`, and vice versa.
- A block contains 3 said instruction with disjoint operands, to fill the pipeline, maximize throughput, and avoid hazards.
- Port contention is caused when both Alice and Bob issue instructions to the same port.

TABLE 1. SELECTIVE INSTRUCTIONS. ALL OPERANDS ARE REGISTERS, WITH NO MEMORY OPS.

Instruction	Ports	Latency	Reciprocal 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

Alice

Bob

TABLE II
RESULTS OVER A THOUSAND TRIALS. AVERAGE CYCLES ARE IN THOUSANDS.

Alice	Bob	Diff. Phys. Core Cycles	Same Phys. Core Cycles
Port 1	Port 1	203331	**408322**
Port 1	Port 5	203322	203820
Port 5	Port 1	203334	203487
Port 5	Port 5	203328	**404941**

PortSmash Covert-Channel

PortSmash exploits the *shared Execution Engine* to create *timing side-channels* due to *port contention* among two or more processes running on different logical cores but the same physical cores

- We performed the experiment on Intel Core i7-7700HQ Kaby Lake.
- Alice first issues `crc32` blocks in an infinite loop, while Bob executes 2^{20} blocks of `crc32` and then `vpermd`, and vice versa.
- A block contains 3 said instruction with disjoint operands, to fill the pipeline, maximize throughput, and avoid hazards.
- Port contention is caused when both Alice and Bob issue instructions to the same port.

TABLE 1. SELECTIVE INSTRUCTIONS. ALL OPERANDS ARE REGISTERS, WITH NO MEMORY OPS.

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

TABLE II
RESULTS OVER A THOUSAND TRIALS. AVERAGE CYCLES ARE IN THOUSANDS.

Alice	Bob	Diff. Phys. Core Cycles	Same Phys. Core Cycles
Port 1	Port 1	203331	**408322**
Port 1	Port 5	203322	203820
Port 5	Port 1	203334	203487
Port 5	Port 5	203328	**404941**

PortSmash Side-Channel

```
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`.

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

PortSmash Side-Channel

```
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
```

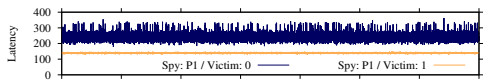
```
#elif defined(P0156)
.rept 64
add %r8, %r8
add %r9, %r9
add %r10, %r10
add %r11, %r11
.endr
#else
#error No ports defined
#endif

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.

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



Timings for the PortSmash Spy when configured with `P1`, in parallel to the Victim executing `x64_foo`.

PortSmash Side-Channel

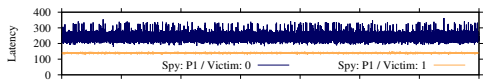
```
mov $COUNT, %rcx

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.

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



Timings for the PortSmash Spy when configured with `P1`, in parallel to the Victim executing `x64_foo`.

PortSmash Side-Channel

```
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
```

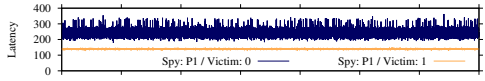
```
#elif defined(P0156)
.rept 64
add %r8, %r8
add %r9, %r9
add %r10, %r10
add %r11, %r11
.endr
#else
#error No ports defined
#endif

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.

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



Timings for the PortSmash Spy when configured with P1, in parallel to the Victim executing x64_foo.

PortSmash Spatial Resolution

```
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
```

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.

End-End Attack on a TLS Server

- **OpenSSL 1.1.0h** TLS server configured with P-384 ECDSA running on Intel Core i7-6700 Skylake 3.40GHz featuring Hyper-Threading.
- Secret dependent on *double* and *add* operations of the underlying `ec_wNAF_mul` point multiplication function during ECDSA.

```
for (k = max_len - 1; k >= 0; k--) {
        if (!r_is_at_infinity) {
            if (!EC_POINT_dbl(group, r, r, ctx))
                goto err;
        }
        for (i = 0; i < totalnum; i++) {
            if (wNAF_len[i] > (size_t)k) {
                int digit = wNAF[i][k];
                int is_neg;
                if (digit) {
                    is_neg = digit < 0;
                    if (is_neg)
                        digit = -digit;
                    if (is_neg != r_is_inverted) {
                        if (!r_is_at_infinity) {
                            if (!EC_POINT_invert(group, r, ctx))
                                goto err;
                        }
                        r_is_inverted = !r_is_inverted;
                    }
                    /* digit > 0 */
                    if (r_is_at_infinity) {
                        if (!EC_POINT_copy(r, val_sub[i][digit >> 1]))
                            goto err;
                        r_is_at_infinity = 0;
                    } else {
                        if (!EC_POINT_add(group, r, r, val_sub[i][digit >> 1], ctx))
                            goto err;
                    }
                }
            }
        }
    }
```

End-End Attack on a TLS Server

- **OpenSSL 1.1.0h** TLS server configured with P-384 ECDSA running on Intel Core i7-6700 Skylake 3.40GHz featuring Hyper-Threading.
- Secret dependent on *double* and *add* operations of the underlying `ec_wNAF_mul` point multiplication function during ECDSA.

```
for (k = max_len - 1; k >= 0; k--) {
        if (!r_is_at_infinity) {
            if (!EC_POINT_dbl(group, r, r, ctx))
                goto err;
        }
        for (i = 0; i < totalnum; i++) {
            if (wNAF_len[i] > (size_t)k) {
                int digit = wNAF[i][k];
                int is_neg;
                if (digit) {
                    is_neg = digit < 0;
                    if (is_neg)
                        digit = -digit;
                    if (is_neg != r_is_inverted) {
                        if (!r_is_at_infinity) {
                            if (!EC_POINT_invert(group, r, ctx))
                                goto err;
                        }
                        r_is_inverted = !r_is_inverted;
                    }
                    /* digit > 0 */
                    if (r_is_at_infinity) {
                        if (!EC_POINT_copy(r, val_sub[i][digit >> 1]))
                            goto err;
                        r_is_at_infinity = 0;
                    } else {
                        if (!EC_POINT_add(group, r, r, val_sub[i][digit >> 1], ctx))
                            goto err;
                    }
                }
            }
        }
    }
```

End-End Attack on a TLS Server

- **OpenSSL 1.1.0h** TLS server configured with P-384 ECDSA running on Intel Core i7-6700 Skylake 3.40GHz featuring Hyper-Threading.
- Secret dependent on *double* and *add* operations of the underlying `ec_wNAF_mul` point multiplication function during ECDSA.

```
for (k = max_len - 1; k >= 0; k--) {
        if (!r_is_at_infinity) {
            if (!EC_POINT_dbl(group, r, r, ctx))
                goto err;
        }
        for (i = 0; i < totalnum; i++) {
            if (wNAF_len[i] > (size_t)k) {
                int digit = wNAF[i][k];
                int is_neg;
                if (digit) {
                    is_neg = digit < 0;
                    if (is_neg)
                        digit = -digit;
                    if (is_neg != r_is_inverted) {
                        if (!r_is_at_infinity) {
                            if (!EC_POINT_invert(group, r, ctx))
                                goto err;
                        }
                        r_is_inverted = !r_is_inverted;
                    }
                    /* digit > 0 */
                    if (r_is_at_infinity) {
                        if (!EC_POINT_copy(r, val_sub[i][digit >> 1]))
                            goto err;
                        r_is_at_infinity = 0;
                    } else {
                        if (!EC_POINT_add(group, r, r, val_sub[i][digit >> 1], ctx))
                            goto err;
                    }
                }
            }
        }
    }
```

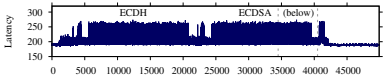
End-End Attack on a TLS Server

- **OpenSSL 1.1.0h** TLS server configured with P-384 ECDSA running on Intel Core i7-6700 Skylake 3.40GHz featuring Hyper-Threading.
- Secret dependent on *double* and *add* operations of the underlying `ec_wNAF_mul` point multiplication function during ECDSA.

```
for (k = max_len - 1; k >= 0; k--) {
        if (!r_is_at_infinity) {
            if (!EC_POINT_dbl(group, r, r, ctx))
                goto err;
        }
        for (i = 0; i < totalnum; i++) {
            if (wNAF_len[i] > (size_t)k) {
                int digit = wNAF[i][k];
                int is_neg;
                if (digit) {
                    is_neg = digit < 0;
                    if (is_neg)
                        digit = -digit;
                    if (is_neg != r_is_inverted) {
                        if (!r_is_at_infinity) {
                            if (!EC_POINT_invert(group, r, ctx))
                                goto err;
                        }
                        r_is_inverted = !r_is_inverted;
                    }
                    /* digit > 0 */
                    if (r_is_at_infinity) {
                        if (!EC_POINT_copy(r, val_sub[i][digit >> 1]))
                            goto err;
                        r_is_at_infinity = 0;
                    } else {
                        if (!EC_POINT_add(group, r, r, val_sub[i][digit >> 1], ctx))
                            goto err;
                    }
                }
            }
        }
    }
```

End-End Attack on a TLS Server

Procurement Phase



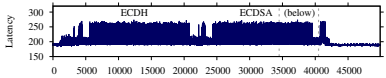
Raw TLS handshake trace showing scalar multiplications during ECDH and ECDSA.

End-End Attack on a TLS Server

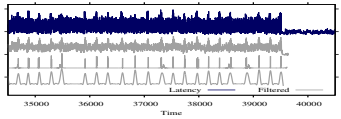
Procurement Phase

↓

Signal Processing Phase



Raw TLS handshake trace showing scalar multiplications during ECDH and ECDSA.



Zoom at the end of the previous ECDSA trace, peaks (filtered) showing repeated double and add operations.

End-End Attack on a TLS Server

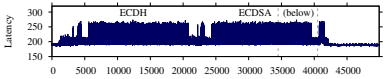
Procurement Phase

↓

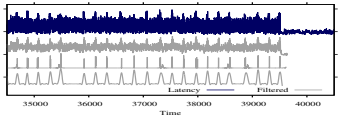
Signal Processing Phase

↓

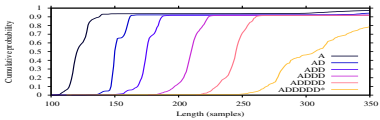
Key Recovery Phase



Raw TLS handshake trace showing scalar multiplications during ECDH and ECDSA.

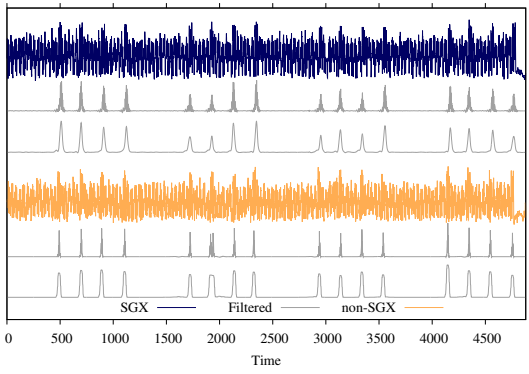


Zoom at the end of the previous ECDSA trace, peaks (filtered) showing repeated double and add operations.



Length distributions for various patterns at the end of scalar multiplication.

On Intel SGX



From top to bottom: raw trace of our SGX Victim; said trace after filtering; raw trace of our user space Victim; said trace after filtering. Both victims received the same input, i.e., a scalar that induces 16 point adds at the end of the trace, clearly identifiable by the peaks in the filtered traces.

Proposed Mitigations

- Secure code with no secret dependent branches (best solution)
  - As a result of PortSmash and CVE-2018-5407, a timing-resistant implementation from the **OpenSSL 1.1.0** branch (another work by our team) was backported to **1.0.2q** to replace the vulnerable *w*NAF implementation (PR#7593[3]).
- Disable SMT !!!
- OS support for logical core isolation from user space (difficult to implement).

[3] https://github.com/openssl/openssl/pull/7593

Conclusion

- PortSmash is a practical attack vector.
- PortSmash is configurable, portable, has very fine spatial resolution, and requires minimum prerequisites.
- Microarchitectures are becoming more complex (potential for new side-channels).
- Side-channel analysis is still a practical and powerful tool to detect vulnerabilities, even in software.

Thank You!

PortSmash POC

https://github.com/bbbrumley/portsmash

Acknowledgments

- Supported by European Research Council (ERC) (grant agreement No 804476)
- COST Action IC1403 CRYPTACUS
- Nokia Foundation
- We thank Tampere Center for Scientific Computing (TCSC) for generously granting us access to computing cluster resources

Disclosure

- **01 Oct 2018**: Notified Intel Security
- **26 Oct 2018**: Notified openssl-security
- **26 Oct 2018**: Notified CERT-FI
- **26 Oct 2018**: Notified oss-security distros list
- **01 Nov 2018**: Embargo expired

PortSmash Side-Channel Vulnerability CVE-2018-5407

PortSmash comparison with other techniques

TABLE III
COMPARISON OF MICROARCHITECTURE ATTACK TECHNIQUES (ORIGINAL VERSIONS)

Attack	Spatial Resolution	Size	Detectability	Cross-Core	Cross-VM
TLBleed	Memory Page (Very low)	4 KB	Low	No	Yes/SMT
Prime+Probe	Cache-set (Low)	512 bytes	Medium	Yes	Yes/SharedMem
Flush+Reload	Cache-line (Med)	64 bytes	High	Yes	Yes/SharedMem
Flush+Flush	Cache-line (Med)	64 bytes	Low	Yes	Yes/SharedMem
CacheBleed	Intra cache-line (High)	8 bytes	Medium	No	Yes/SMT
MemJam	Intra cache-line (High)	4 bytes	Medium	No	Yes/SMT
PortSmash	Execution port (Very High)	*uops*	Low	No	Yes/SMT