Malware Guard Extension: Using SGX to Conceal Cache Attacks

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard
July 6, 2017

Graz University of Technology
• We show that malicious SGX enclaves are possible
• We show that malicious SGX enclaves are possible
• We present methods to circumvent certain limitations of SGX
• We show that malicious SGX enclaves are possible
• We present methods to circumvent certain limitations of SGX
• We mount a cache attack on a RSA implementation within a different enclave
• We show that malicious SGX enclaves are possible
• We present methods to circumvent certain limitations of SGX
• We mount a cache attack on a RSA implementation within a different enclave
• We abuse SGX to prevent detection of the attack
• We show that malicious SGX enclaves are possible
• We present methods to circumvent certain limitations of SGX
• We mount a cache attack on a RSA implementation within a different enclave
• We abuse SGX to prevent detection of the attack
• We show that the stealthy attack even works across Docker containers
• We show that malicious SGX enclaves are possible
• We present methods to circumvent certain limitations of SGX
• We mount a cache attack on a RSA implementation within a different enclave
• We abuse SGX to prevent detection of the attack
• We show that the stealthy attack even works across Docker containers
• We discuss countermeasures to prevent such attacks
Background
Application

Untrusted part

Operating System
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Call Gate

Trusted part

Trusted Fnc.

Operating System
Create Enclave

Call Trusted Fnc.

Call Gate

Trusted part

Application

Untrusted part

Operating System

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard — Graz University of Technology
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Call Gate

Trusted part

Trusted Fnc.

Operating System

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard — Graz University of Technology
Application

Untrusted part
- Create Enclave
- Call Trusted Fnc.

Trusted part
- Call Gate
- Trusted Fnc.
- Return

Operating System

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémantine Maurice, Stefan Mangard — Graz University of Technology
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System
Application

Untrusted part

Create Enclave

Call Trusted Fnc.

... 

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System
SGX

Application

Untrusted part

Create Enclave

Call Trusted Fnc.

... 

Trusted part

Call Gate

Trusted Fnc.

Return

Operating System

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémence Maurice, Stefan Mangard — Graz University of Technology
\[ M = C^d \mod n \]
\[ M = C^d \mod n \]
\[ M = C^d \mod n \]
M = C^d \mod n

\[
\begin{array}{ccccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 0 & \cdots
\end{array}
\]

Result = Result \times Result

\underbrace{\text{square}}
\[ M = C^d \mod n \]

Result = Result × Result

\underline{square}
\[ M = C^d \mod n \]

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 0 & \cdots \\
\end{array}
\]

\[
\text{Result} = \text{Result} \times \text{Result} \times C
\]

- square
- multiply
$$M = C^d \mod n$$

$\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 0 & \ldots
\end{array}$

$$\text{Result} = \text{Result} \times \text{Result} \times C$$

- square
- multiply
\[ M = C^d \mod n \]

\[
\begin{array}{ccccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 0 & \cdots \\
\end{array}
\]

\[
\text{Result} = \text{Result} \times \text{Result}
\]

\text{square}
Prime+Probe...
Prime+Probe...

- exploits the timing difference when accessing...
Prime+Probe...

- exploits the timing difference when accessing...
  - cached data (fast)
Prime+Probe...

- exploits the **timing difference** when accessing...
  - cached data (fast)
  - uncached data (slow)
Prime+Probe...

- exploits the timing difference when accessing...
  - cached data (fast)
  - uncached data (slow)
- is applied to one cache set
Prime+Probe...

- exploits the timing difference when accessing...
  - cached data (fast)
  - uncached data (slow)
- is applied to one cache set
- works across CPU cores as the last-level cache is shared
Step 0: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
**Step 0**: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
**Step 0**: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
Step 0: Attacker fills the cache (prime)
**Step 0**: Attacker fills the cache (prime)

**Step 1**: Victim evicts cache lines by accessing own data
**Step 0**: Attacker fills the cache (prime)

**Step 1**: Victim evicts cache lines by accessing own data
**Step 0**: Attacker fills the cache (prime)
**Step 1**: Victim evicts cache lines by accessing own data
**Step 0**: Attacker fills the cache (prime)
**Step 1**: Victim evicts cache lines by accessing own data
Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data
Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data
Step 2: Attacker probes data to determine if the set was accessed
Step 0: Attacker fills the cache (prime)
Step 1: Victim evicts cache lines by accessing own data
Step 2: Attacker probes data to determine if the set was accessed
**Step 0:** Attacker fills the cache (prime)
**Step 1:** Victim evicts cache lines by accessing own data
**Step 2:** Attacker probes data to determine if the set was accessed
Attack
Victim
Attack Settings

Victim

SGX
Attack Settings

Victim

SGX

RSA Signature + private key

Public API
Attack Settings

Attacker

Victim

SGX

RSA Signature
+ private key

Public API
Attack Settings

Attacker

SGX

Malware

Loader

Victim

SGX

RSA Signature
+ private key

Public API

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard — Graz University of Technology
Attack Settings

Attacker

SGX

Malware

Loader

Victim

SGX

RSA Signature
+ private key

Public API
Attack Settings

Attacker

SGX

Malware

Loader

L1/L2 Cache

Victim

SGX

RSA Signature + private key

Public API

L1/L2 Cache

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard — Graz University of Technology
Attack Settings

Attacker

SGX

Malware

(Prime+Probe)

Loader

L1/L2 Cache

Shared LLC

Victim

SGX

RSA Signature

+ private key

Public API

L1/L2 Cache

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémantine Maurice, Stefan Mangard — Graz University of Technology
• No access to high-precision timer (rdtsc)
- No access to high-precision timer (rdtsc)
- No syscalls
SGX Limitations

- No access to high-precision timer (rdtsc)
- No syscalls
- No shared memory
SGX Limitations

- No access to high-precision timer (rdtsc)
- No syscalls
- No shared memory
- No physical addresses
SGX Limitations

- No access to high-precision timer (rdtsc)
- No syscalls
- No shared memory
- No physical addresses
- No 2 MB large pages
We have to build our own timer
• We have to build our own timer
• Timer resolution must be in the order of cycles
• We have to build our own timer
• Timer resolution must be in the order of cycles
• Start a thread that continuously increments a global variable
• We have to build our own timer
• Timer resolution must be in the order of cycles
• Start a thread that continuously increments a global variable
• The global variable is our timestamp
CPU cycles one increment takes

```
rdtsc  # 1
```

```
timestamp = rdtsc();
```
CPU cycles one increment takes

```
rdtsc  1

C  4.7

while (1) {
  timestamp++;
}
```
CPU cycles one increment takes

\[
\text{rdtsc} \quad 1
\]

C

\begin{verbatim}
  mov &timestamp, %rcx
  jmp 1b
\end{verbatim}

Assembly

\[
4.7
\]

\[
4.67
\]
CPU cycles one increment takes

rdtsc  1  

C  4.7

Assembly  4.67

Optimized  0.87

```
1  mov &timestamp, %rcx
2   inc %rax
3  mov %rax, (%rcx)
4   jmp 1b
```
• Cache set is determined by part of physical address
- Cache set is determined by part of physical address
- We have no knowledge of physical addresses
• **Cache set** is determined by part of physical address
• We have no knowledge of physical addresses
• Use the DRAM mapping reverse engineered by Pessl et al.
• Cache set is determined by part of physical address
• We have no knowledge of physical addresses
• Use the DRAM mapping reverse engineered by Pessl et al.
• Exploit timing differences to find DRAM row borders
• **Cache set** is determined by part of physical address
• We have no knowledge of **physical addresses**
• Use the **DRAM mapping** reverse engineered by Pessl et al.
• Exploit timing differences to find DRAM **row borders**
• The 18 LSBs are ‘0’ at a row border
Physical Addresses

8 kB row x in BG0 (1) and channel (1)

8 kB row x in BG0 (0) and channel (1)

8 kB row x in BG0 (1) and channel (0)

8 kB row x in BG0 (0) and channel (0)
Physical Addresses

0
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
BG0 (0), Channel (0)
BG0 (1), Channel (0)
BG0 (0), Channel (1)
BG0 (1), Channel (0)
Physical Addresses

8 kB row × in BG0 (1) and channel (1)

8 kB row × in BG0 (0) and channel (1)

8 kB row × in BG0 (1) and channel (0)

8 kB row × in BG0 (0) and channel (0)
Physical Addresses

row $n$
row $n+1$
row $n+2$
row $n+3$
row $n+4$
row $n+5$
Physical Addresses

row \( n \)  row \( n + 1 \)  row \( n + 2 \)  row \( n + 3 \)  row \( n + 4 \)  row \( n + 5 \)
Physical Addresses

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard — Graz University of Technology
Physical Addresses

row $n$
row $n + 1$
row $n + 2$
row $n + 3$
row $n + 4$
row $n + 5$
Physical Addresses
Physical Addresses

row $n$
row $n + 1$
row $n + 2$
row $n + 3$
row $n + 4$
row $n + 5$
Physical Addresses

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard — Graz University of Technology
Physical Addresses
Physical Addresses

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard — Graz University of Technology
Physical Addresses

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard — Graz University of Technology
Physical Addresses
Physical Addresses

row $n$
row $n+1$
row $n+2$
row $n+3$
row $n+4$
row $n+5$
Physical Addresses

row $n$
row $n + 1$
row $n + 2$
row $n + 3$
row $n + 4$
row $n + 5$
Physical Addresses

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard — Graz University of Technology
Physical Addresses

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard — Graz University of Technology
Physical Addresses

www.tugraz.at
Physical Addresses
Result on an Intel i5-6200U

Latency [cycles] vs. Array index [kB]
1. Use the counting primitive to measure DRAM accesses
1. Use the counting primitive to measure DRAM accesses
2. Through the DRAM side channel, determine the row borders
1. Use the **counting primitive** to measure DRAM accesses
2. Through the DRAM side channel, determine the **row borders**
3. Row borders have the 18 LSBs set to ‘0’ → maps to **cache set ‘0’**
1. Use the **counting primitive** to measure DRAM accesses
2. Through the DRAM side channel, determine the **row borders**
3. Row borders have the 18 LSBs set to ‘0’ → maps to **cache set ‘0’**
4. Build the **eviction set** for the Prime+Probe attack
1. Use the counting primitive to measure DRAM accesses
2. Through the DRAM side channel, determine the row borders
3. Row borders have the 18 LSBs set to ‘0’ → maps to cache set ‘0’
4. Build the eviction set for the Prime+Probe attack
5. Mount Prime+Probe on the buffer containing the multiplier
Results
Raw Prime+Probe trace...
...processed with a simple moving average...
...allows to clearly see the bits of the exponent
Error probability depends on which cache set of the key we attack
Error probability depends on which cache set of the key we attack.
Full recovery of a 4096-bit RSA key in approximately 5 minutes
Performance Counters

L1 Hits
L1 Misses
L3 Hits
L3 Misses

-10^9

L1 Hits
L1 Misses
L3 Hits
L3 Misses

Performance counter value

Native
Performance Counters

L1 Hits: 0
L1 Misses: 5
L3 Hits: 0.5
L3 Misses: 0.5

Performance counter value

L1 Hits
L1 Misses
L3 Hits
L3 Misses

Native
SGX

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémantine Maurice, Stefan Mangard — Graz University of Technology
Bonus: Docker

SGX
Malware
(*Prime+Probe*)

SGX
RSA
(+ private key)

Loader

API
Bonus: Docker

Attacker container

Loader

SGX

Malware

(Prime+Probe)

Victim container

RSA

(+ private key)

Docker engine

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard — Graz University of Technology
Malware
(Prime+Probe)

RSA
(+ private key)

Docker engine

SGX

SGX

Attacker
container

Loader

Victim
container

API

SGX driver

Docker engine

www.tugraz.at

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard — Graz University of Technology
Countermeasures
Cache attacks can be prevented on source level
• Cache attacks can be prevented on source level
• Use side-channel resistant crypto implementations
Source Level

- Cache attacks can be prevented on source level
- Use side-channel resistant crypto implementations
- Exponent blinding for RSA prevents multi-trace attacks
Cache attacks can be prevented on source level
Use side-channel resistant crypto implementations
Exponent blinding for RSA prevents multi-trace attacks
Bit-sliced implementations are not vulnerable to cache attacks
• Trusting the operating system weakens SGX threat model
• Trusting the operating system weakens SGX threat model
• Method for the operating system to inspect enclave code
• Trusting the operating system weakens SGX threat model
• Method for the operating system to inspect enclave code
• Re-enable certain performance counters, such as L3 hits/misses
• Trusting the operating system weakens SGX threat model
• Method for the operating system to inspect enclave code
• Re-enable certain performance counters, such as L3 hits/misses
• Enclave coloring to prevent cross-enclave attacks
• Trusting the operating system weakens SGX threat model
• Method for the operating system to inspect enclave code
• Re-enable certain performance counters, such as L3 hits/misses
• Enclave coloring to prevent cross-enclave attacks
• Heap randomization to randomize cache sets
• Intel could prevent attacks by changing the hardware
• Intel could prevent attacks by changing the hardware
• Combine Cache Allocation Technology (CAT) with SGX
- Intel could prevent attacks by changing the hardware
- Combine Cache Allocation Technology (CAT) with SGX
  - Instead of controlling CAT from the OS, combine it with eenter
• Intel could prevent attacks by changing the hardware

• Combine Cache Allocation Technology (CAT) with SGX
  • Instead of controlling CAT from the OS, combine it with eenter
  • Entering an enclave would automatically activate CAT for this core
Intel could prevent attacks by changing the hardware

Combine Cache Allocation Technology (CAT) with SGX
  - Instead of controlling CAT from the OS, combine it with eenter
  - Entering an enclave would automatically activate CAT for this core
  - L3 is then isolated from all other enclaves and applications
• Intel could prevent attacks by changing the hardware
• Combine Cache Allocation Technology (CAT) with SGX
  • Instead of controlling CAT from the OS, combine it with eenter
  • Entering an enclave would automatically activate CAT for this core
  • L3 is then isolated from all other enclaves and applications
• Provide a non-shared secure memory element which is not cached
Conclusion
• We showed that attacks can be mounted from within SGX enclaves
• We showed that attacks can be mounted from within SGX enclaves
• We presented new techniques for side-channel attacks, including a timing measurement with the currently highest resolution
• We showed that attacks can be mounted from within SGX enclaves
• We presented new techniques for side-channel attacks, including a timing measurement with the currently highest resolution
• Our end-to-end attack recovered 96% of a 4096-bit RSA key from a single trace, and the full key using only 11 traces
• We showed that attacks can be mounted from within SGX enclaves
• We presented new techniques for side-channel attacks, including a timing measurement with the currently highest resolution
• Our end-to-end attack recovered 96% of a 4096-bit RSA key from a single trace, and the full key using only 11 traces
• SGX allows to completely hide an attack from state-of-the-art detection techniques
• We showed that attacks can be mounted from within SGX enclaves
• We presented new techniques for side-channel attacks, including a timing measurement with the currently highest resolution
• Our end-to-end attack recovered 96% of a 4096-bit RSA key from a single trace, and the full key using only 11 traces
• SGX allows to completely hide an attack from state-of-the-art detection techniques
• The attack showed that SGX is not a magic solution to make software safe
Thank you!
Malware Guard Extension: Using SGX to Conceal Cache Attacks

Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard

July 6, 2017

Graz University of Technology