Page Table-Based Side-Channel Attacks against Intel SGX: Attacks and Defenses

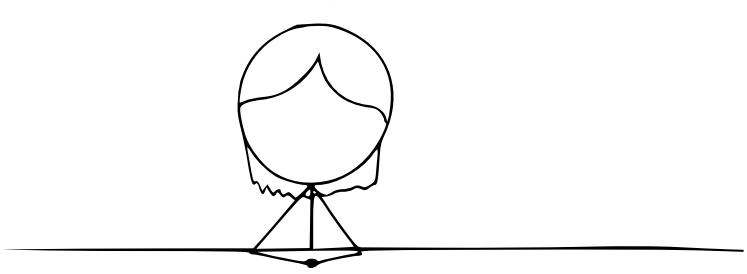
Raoul Strackx
raoul.strackx@cs.kuleuven.be
imec-DistriNet, KU Leuven, Celestijnenlaan 200A, B-3001 Belgium

Security of Software/Hardware Interfaces, July 8th 2019

DistriNet

The Key Problem

“How do we share a single hardware platform with multiple users/processes in an easy, fast and secure way?”



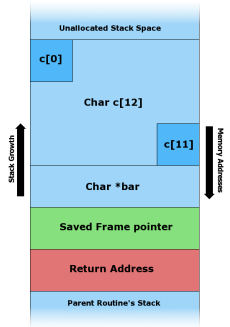
Programmers' Mistakes

- **Arithmetic bugs** (e.g., div by zero, integer overflow, . . .)
- **Logical bugs** (e.g., Infinite loops, . . .)
- **Syntax bugs** (e.g., assignment instead of comparison, . . .)
- **Multi-threaded bugs** (e.g., deadlocks, race conditions, . . .)
- **Interfacing bugs** (e.g., incorrect API use, . . .)
- **Resource bugs** (e.g., uninitialized variables, buffer overflows, . . .)

```
1 #include <string.h>
2
3 void foo (char *bar)
4 {
5     char c[12];
6
7     strcpy(c, bar); // no bounds checking
8 }
9
10 int main (int argc, char **argv)
11 {
12     foo(argv[1]);
13 }
```

Programmers' Mistakes

- **Arithmetic bugs** (e.g., div by zero, integer overflow, . . .)
- **Logical bugs** (e.g., Infinite loops, . . .)
- **Syntax bugs** (e.g., assignment instead of comparison, . . .)
- **Multi-threaded bugs** (e.g., deadlocks, race conditions, . . .)
- **Interfacing bugs** (e.g., incorrect API use, . . .)
- **Resource bugs** (e.g., uninitialized variables, buffer overflows, . . .)



Security Measures: Hardening Legacy Software Automatically

- Automatically add security measures during compilation
+ Low effort
+ Can be applied to existing software
- Cannot provide strong security measures
- Example: StackGuard, ASLR, ...

Security Measures: More Secure Libraries

- Use libraries that force programmers to think about buffer boundaries
- \+ Fast
- \- Does not protect against other vulnerabilities
- \- Can only be applied to existing software

Replace:

```
1 strcpy( char *dest, char const *src );
```

With:

```
1 strncpy( char *dest, char const *src, size_t n );
```

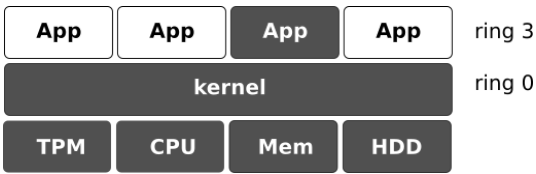
Security Measures: Use Memory-Safe Languages

- Combine static and run-time checks to avoid vulnerabilities
- \+ Low overhead
- \- Cannot be applied to existing software
- \- Many memory-safe languages rely on a large code base written in an unsafe language

Security Measures: Verify Software

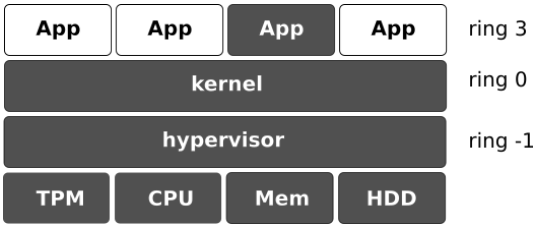
- Prove mathematically that software behaves as required
+ Very strong security guarantees
- Very labor intensive

Security Measures: Privilege Separation



- Firefox: 6 million LoC
- Windows 7: 40 million LoC
- Linux kernel: 12 million LoC

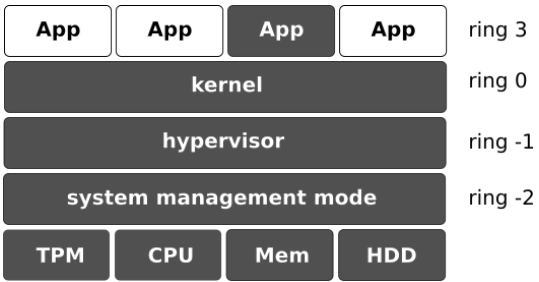
Security Measures: Privilege Separation



Hypervisor

- Security layer for virtual machine monitor (a.k.a. hypervisor)
- Optional
- Ring -1 is a representation, it's more complicated in practice

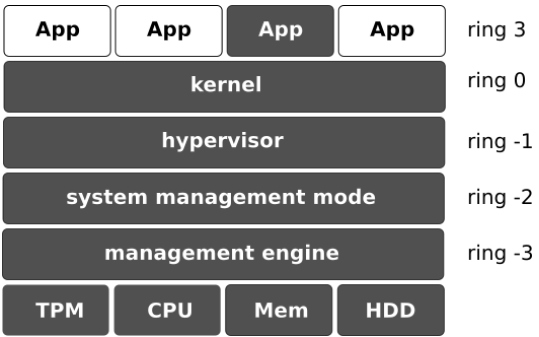
Security Measures: Privilege Separation



System Management Mode (SMM) handles:

- Temperature fluctuations (e.g., turning on fans)
- Memory errors
- . . .

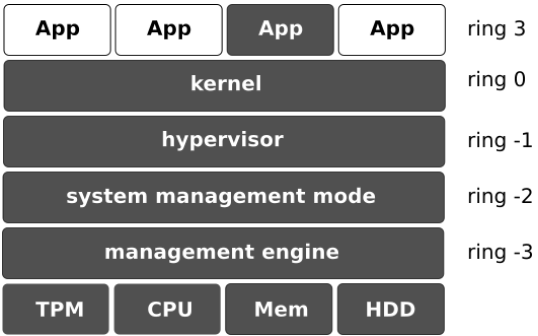
Security Measures: Privilege Separation



(Intel) Management Engine

- Separate processor on motherboard
- Always on
- Becoming standard
- Previously used for remote access (e.g., remotely fix broken OS)
- . . .

Security Measures: Privilege Separation



Is that all of it!?

- Microcode in CPU
- Firmware on peripherals
- . . .
- But let's stop here

Security Measures: Privilege Separation

+ Easy separation
- (Monolithic) OS and applications are too big to provide strong security
- Microkernels are more difficult to implement, and are not compatible with (many) existing software
- Inflexible: difficult to use by programmers

Security Measures: Hardware Security Modules

+ Strong, physical separation of sensitive code/data
+ Present in most commodity systems: TPM
- Cannot execute arbitrary code
- Slow

So Many Options, We Need a New Idea

- Hardened Libraries
- Memory-Safe Languages
- Software Verifications
- Privilege Separation
- Hardware Security Modules

We need something else

Core Idea:

- Protect small, security sensitive parts of an application
- Use, but don't trust the underlying (software) layers (e.g., OS)

We need something else

Example:

Network-level attacker:

- May observe network packets
- . . . re-order them
- . . . drop some

But! TLS . . .

- will prevent an attacker reading network packets
- or modify them
- or re-order the plaintext content

We need something else

Example: Now imagine: TLS-handling code in a small container

- The Operating System:
  - Schedules the process containing the container
  - Reads/write (TLS) packets from/to the network card
  - . . .
- A kernel-level attacker. . .
  - will prevent an attacker reading network packets
  - or modify them
  - or re-order the plaintext content

We need something else

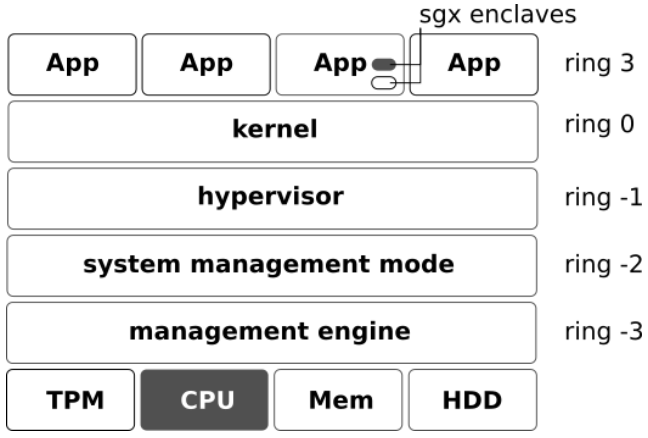
Attack model:

- Assume an attacker infiltrated in lower-layers

Security guarantees:

- Complete isolation of protected modules
- We do **not** (aim to provide) availability guarantees

Protected-Module Architectures



Protected-Module Architectures

Side-note

Many PMAs exists:

- Flicker (2008, CMU)
- SPMs (2010, KULeuven)
- TrustVisor (2010, CMU)
- Fides (2012, KULeuven)
- Sancus (2013, KULeuven)
- Oasis (2013, CMU)
- **Software Guard eXtensions (2013, Intel)**
- TrustLite (2014, Intel Labs/TU Darmstadt)
- TyTan (2015, Intel Labs/TU Darmstadt)
- . . .

Protected-Module Architectures

Side-note

Many PMAs exists:

- Flicker (2008, CMU)
- SPMs (2010, KULeuven)
- TrustVisor (2010, CMU)
- Fides (2012, KULeuven)
- Sancus (2013, KULeuven)
- Oasis (2013, CMU)
- **Software Guard eXtensions (2013, Intel)**
- TrustLite (2014, Intel Labs/TU Darmstadt)
- TyTan (2015, Intel Labs/TU Darmstadt)
- . . .

SGX vs PMAs

We'll use "protected module" as the generic term, "enclaves" to mean Intel SGX protected modules specifically

Protected-Module Architectures

Key primitives:

- Isolation
- Key derivation:
  - Sealing
  - Attestation

Isolation primitive

“Don’t Miss a Sec” art installation by Monica Bonvicini (2004)

Isolation primitive

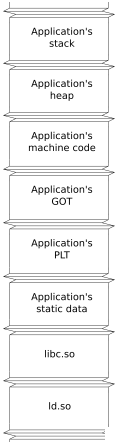
“Don’t Miss a Sec” art installation by Monica Bonvicini (2004)

Isolation primitive

“Don’t Miss a Sec” art installation by Monica Bonvicini (2004)

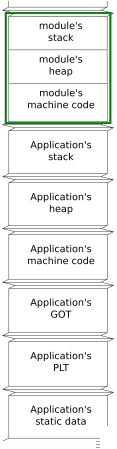
An Application's Memory Layout

```
27 //secret.c
28 #include "secret.h"
29
30 static int tries_left = 3;
31 static int PIN = 1234;
32 static int secret = 666;
33
34 int ENTRYPOINT get_secret(int provided_pin) {
35   if (tries_left > 0) {
36     if (PIN == provided_pin) {
37       tries_left = 3;
38       return secret;
39     } else {
40       tries_left--;
41       return 0;
42     }
43   } else
44     return 0;
45 }
```



An Application's Memory Layout

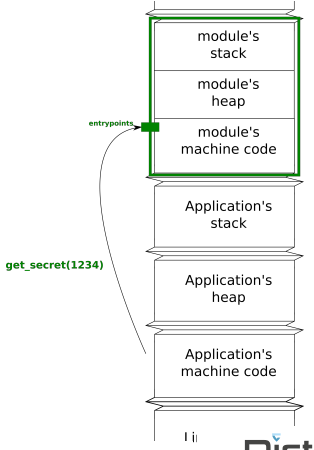
```
46 //secret.c
47 #include "secret.h"
48
49 static int tries_left = 3;
50 static int PIN = 1234;
51 static int secret = 666;
52
53 int ENTRYPOINT get_secret(int provided_pin) {
54   if (tries_left > 0) {
55     if (PIN == provided_pin) {
56       tries_left = 3;
57       return secret;
58     } else {
59       tries_left--;
60       return 0;
61     }
62   } else
63     return 0;
```



An Application's Memory Layout

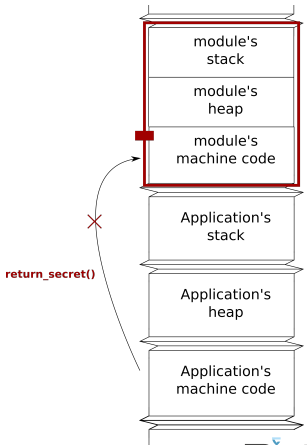
```
65 static int tries_left = 3;
66 static int PIN = 1234;
67 static int secret = 666;
68
69 int ENTRYPOINT get_secret(int provided_pin) {
70   if (tries_left > 0) {
71     if (PIN == provided_pin) {
72       tries_left = 3;
73       return secret;
74     } else {
75       tries_left--;
76       return 0;
77     }
78   } else
79     return 0;
80 }
```

```
81 void main() {
82   get_secret(1234);
83 }
```



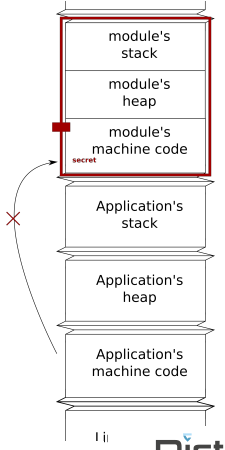
An Application's Memory Layout

```
84 static int tries_left = 3;
85 static int PIN = 1234;
86 static int secret = 666;
87
88 int ENTRYPOINT get_secret(int provided_pin) {
89   if (tries_left > 0) {
90     if (PIN == provided_pin) {
91       tries_left = 3;
92 return_secret:
93       return secret;
94     } else {
95       tries_left--;
96       return 0;
97     }
98   } else
99     return 0;
100 }
101
102 typedef int (*Func)();
103 void main() {
104   Func get = &return_secret;
105   (*get)(); // Not allowed!
106 }
```



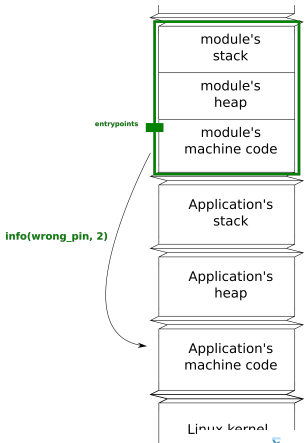
An Application's Memory Layout

```
107 static int tries_left = 3;
108 static int PIN = 1234;
109 static int secret = 666;
110
111 int ENTRYPOINT get_secret(int provided_pin) {
112   if (tries_left > 0) {
113     if (PIN == provided_pin) {
114       tries_left = 3;
115       return secret;
116     } else {
117       tries_left--;
118       return 0;
119     }
120   } else
121     return 0;
122 }
123
124 void main() {
125   printf( "secret = %i\n", secret ); // Not allowed!
126 }
```



An Application's Memory Layout

```
127 static int tries_left = 3;
128 static int PIN = 1234;
129 static int secret = 666;
130
131 int ENTRYPOINT get_secret(int provided_pin) {
132   if (tries_left > 0) {
133     if (PIN == provided_pin) {
134       tries_left = 3;
135       return secret;
136     } else {
137       tries_left--;
138       info( "wrong pin", tries_left );
139       return 0;
140     }
141   } else
142     return 0;
143 }
144
145 void info( char const*str, int tries_left ) {
146   printf( "%s (tries left: %i\n)", tries_left );
147 }
```



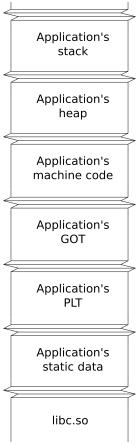
Isolation primitive

- The memory of a protected module can **only** be accessed by the module itself.
- Module can only be entered using an entry point.

from \ to	*Protected*		*Unprotected*
	Entry point	*Code/Data*	
Protected		r w x	r w x
Unprotected / other module	x		r w x

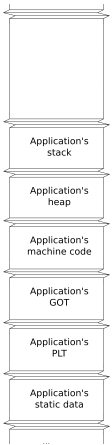
Table: The enforced memory access control model.

How to create a protected module



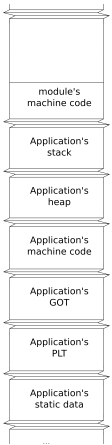
How to create a protected module

- Ask OS for some space
-
-



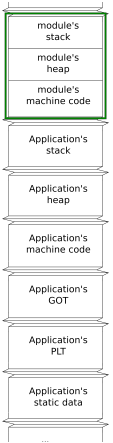
How to create a protected module

- Ask OS for some space
- Load in the module's content
-



How to create a protected module

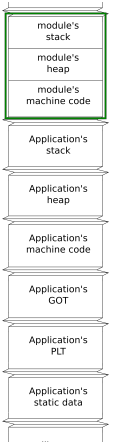
- Ask OS for some space
- Load in the module's content
- Enable the access control mechanism



How to create a protected module

- Ask OS for some space
- Load in the module's content
- Enable the access control mechanism

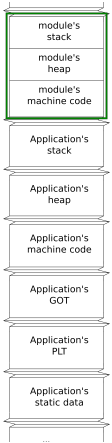
What if an attacker modifies any of these steps?



How to create a protected module

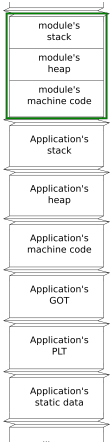
- Ask OS for some space
- Load in the module's content
- Enable the access control mechanism

What if an attacker modifies any of these steps?
See key derivation



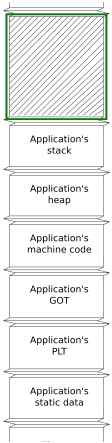
How to destroy a protected module

- Write sensitive data out for future use
-
-



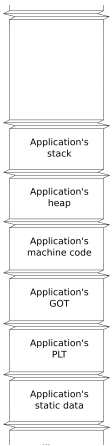
How to destroy a protected module

- Write sensitive data out for future use
- Overwrite sensitive data (e.g., keys, stack!, . . .)
-



How to destroy a protected module

- Write sensitive data out for future use
- Overwrite sensitive data (e.g., keys, stack!, . . .)
- Disable special memory access control mechanism



Key Derivation

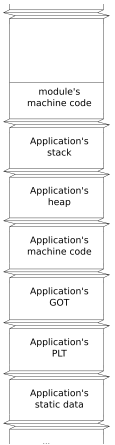
Problem 1: How to provide each module with its secrets while keeping the setup public?
Problem 2: How can we guarantee that code executed correctly on a remote platform?

Key Derivation

Problem 1: How to provide each module with its secrets while keeping the setup public?
Problem 2: How can we guarantee that code executed correctly on a remote platform?
Solution: Key derivation functions providing unique keys to unique modules.

Key Derivation

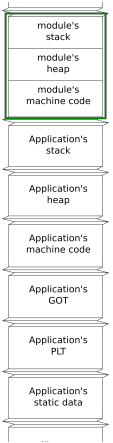
$$k = kdf(K_{platform}, hash(module_{initial\_state}))$$



Key Derivation

$$k = kdf(K_{platform}, hash(module_{initial\_state}))$$

- unique key per enclave per platform!
- (Any) change in enclave $\rightarrow$ different key



Key Derivation

Was the module modified at creation-time?

- (Un)sealing: Try to access old data
- Attestation: Proof the state of the (remote) module

Sealing

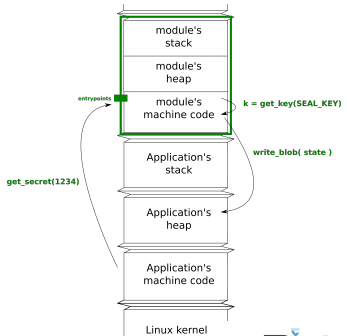
Sealing/Unsealing:

- Used to store/retrieve sensitive data for protected modules
- Derive protected module-specific key
- Encrypt and MAC stored data

An Application's Memory Layout

```
148 static int tries_left = 3;
149 static int PIN = 1234;
150 static int secret = 666;
151
152 void store_state( void ) {
153   EncKey k = get_key( SEAL_KEY );
154   write_blob( seal( k, tries_left || PIN || secret ) );
155 }
156
157 int ENTRYPOINT get_secret(int provided_pin) {
158   if (tries_left > 0) {
159     if (PIN == provided_pin) {
160       tries_left = 3;
161       store_state();
162       return secret;
163     } else {
164       tries_left−−;
165       store_state();
166       return 0;
167     }
168   } else
169     return 0;
170 }
```

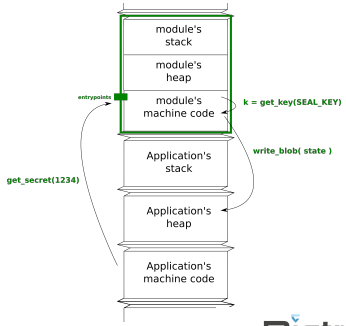
```
171 void main() {
172   get_secret(1234);
173 }
```



An Application's Memory Layout

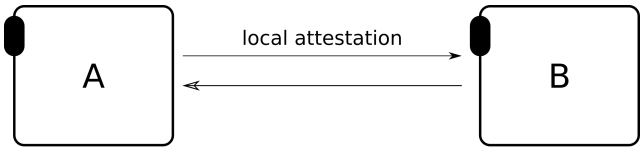
```
174 static int tries_left = 3;
175 static int PIN = 1234;
176 static int secret = 666;
177
178 void store_state( void ) {
179   EncKey k = get_key( SEAL_KEY );
180   write_blob( seal( k, tries_left || PIN || secret ) );
181 }
182
183 int ENTRYPOINT get_secret(int provided_pin) {
184   if (tries_left > 0) {
185     if (PIN == provided_pin) {
186       tries_left = 3;
187       store_state();
188       return secret;
189     } else {
190       tries_left--;
191       store_state();
192       return 0;
193     }
194   } else
195     return 0;
196 }
```

```
197 void main() {
198   get_secret(1234);
199 }
```



Attestation

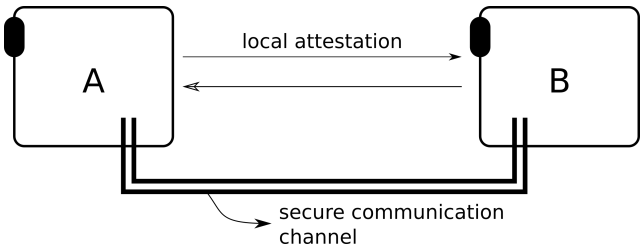
- Local attestation



- Remote attestation

Attestation

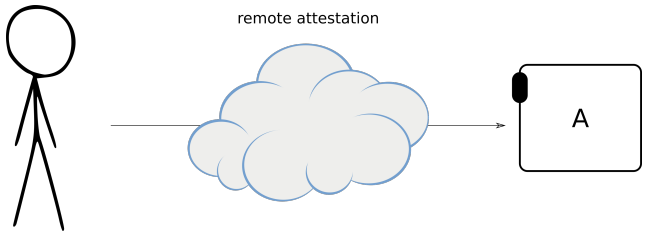
- Local attestation



- Remote attestation

Attestation

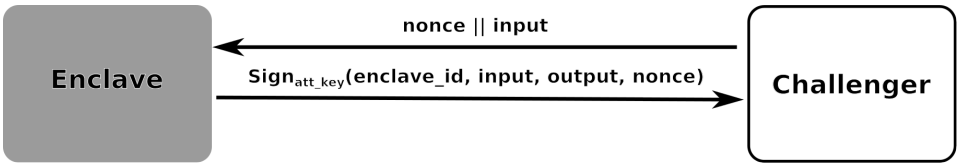
- Local attestation
- Remote attestation



Remote Attestation

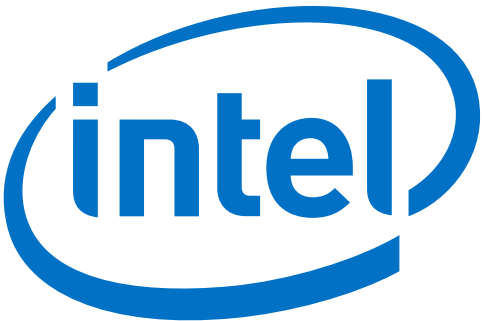
There is a need for remote attestation:

- Processing sensitive data in the cloud
- Simple key derivation does not suffice here
- Newly developed protocols (e.g., Intel EPID)



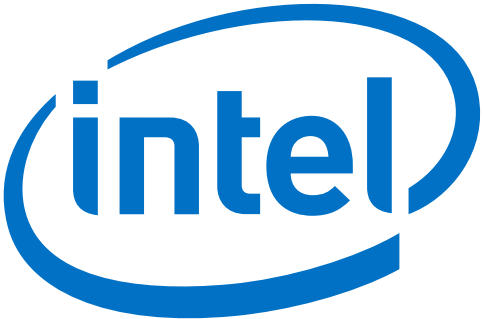
Introduction

- Announced in 2013
- Available since Skylake processor generation (2015)
- Use cases:
  - Fingerprint readers
  - BlueRay disc
  - Netflix DRM used to require SGX for 4K content
  - Fortanix:
    - Running complete applications in an enclave
    - Enclaved Key management



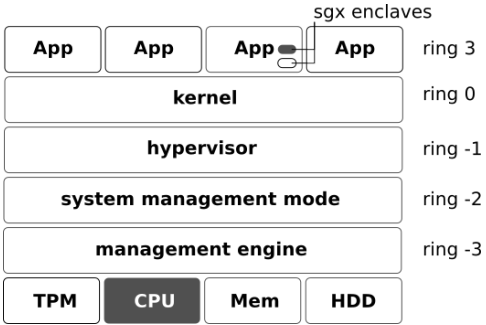
Design Goals/Requirements

- Strong isolation of SGX enclaves
- Defend against HW attacker
- Interruptible
- Sealed storage
- Attestation
- Architecture **must** remain backwards compatible!



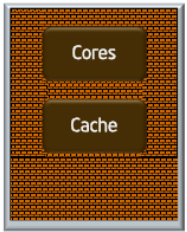
Design Goals/Requirements

- Strong isolation of SGX enclaves
  - Live in userspace
  - Isolated from OS/untrusted part of process
  - No `syscalls`!
- Defend against HW attacker
- Interruptible
- Sealed storage
- Attestation
- Architecture **must** remain backwards compatible!



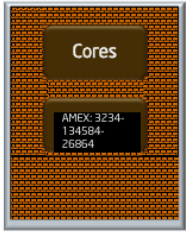
Design Goals/Requirements

- Strong isolation of SGX enclaves
- Defend against HW attacker
  - Defend against cold-boot attacks
  - Enclaved memory is stored confidentially, integrity and version protected in main memory
  - Current limit: 128 MiB (96 MiB of EPC memory)
- Interruptible
- Sealed storage
- Attestation
- Architecture **must** remain



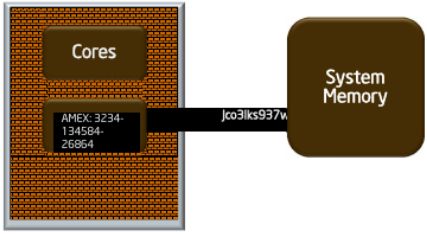
Design Goals/Requirements

- Strong isolation of SGX enclaves
- Defend against HW attacker
  - Defend against cold-boot attacks
  - Enclaved memory is stored confidentially, integrity and version protected in main memory
  - Current limit: 128 MiB (96 MiB of EPC memory)
- Interruptible
- Sealed storage
- Attestation
- Architecture **must** remain



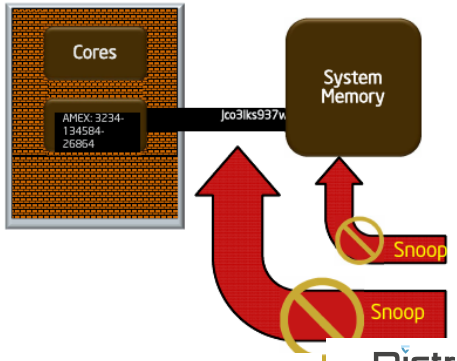
Design Goals/Requirements

- Strong isolation of SGX enclaves
- Defend against HW attacker
  - Defend against cold-boot attacks
  - Enclaved memory is stored confidentially, integrity and version protected in main memory
  - Current limit: 128 MiB (96 MiB of EPC memory)
- Interruptible
- Sealed storage
- Attestation
- Architecture **must** remain



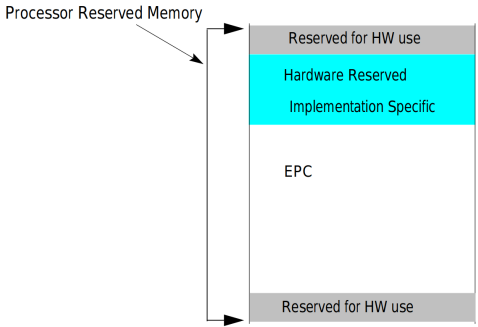
Design Goals/Requirements

- Strong isolation of SGX enclaves
- Defend against HW attacker
  - Defend against cold-boot attacks
  - Enclaved memory is stored confidentially, integrity and version protected in main memory
  - Current limit: 128 MiB (96 MiB of EPC memory)
- Interruptible
- Sealed storage
- Attestation
- Architecture **must** remain



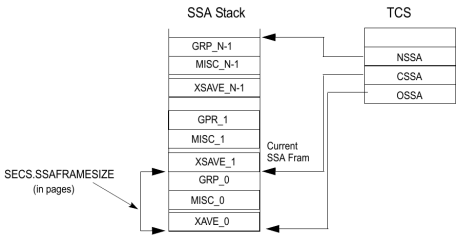
Design Goals/Requirements

- Strong isolation of SGX enclaves
- Defend against HW attacker
  - Defend against cold-boot attacks
  - Enclaved memory is stored confidentially, integrity and version protected in main memory
  - Current limit: 128 MiB (96 MiB of EPC memory)
- Interruptible
- Sealed storage
- Attestation
- Architecture **must** remain



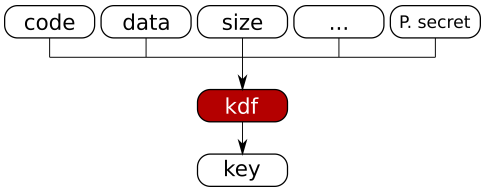
Design Goals/Requirements

- Strong isolation of SGX enclaves
- Defend against HW attacker
- Interruptible
  - Upon fault (e.g., page fault)
  - Upon external interrupt
- Sealed storage
- Attestation
- Architecture **must** remain backwards compatible!



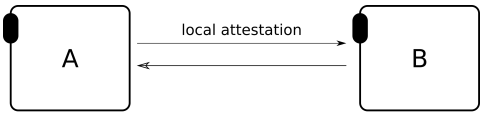
Design Goals/Requirements

- Strong isolation of SGX enclaves
- Defend against HW attacker
- Interruptible
- Sealed storage
  - `MRENCLAVE`: Seal on enclave measurement
  - `MRSIGNER`: Seal on signer of enclave
- Attestation
- Architecture **must** remain backwards compatible!



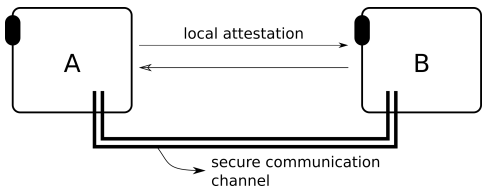
Design Goals/Requirements

- Strong isolation of SGX enclaves
- Defend against HW attacker
- Interruptible
- Sealed storage
- Attestation
  - Both local as remote
  - EPID attestation request can ensure that attestation responses cannot be linked
- Architecture **must** remain backwards compatible!



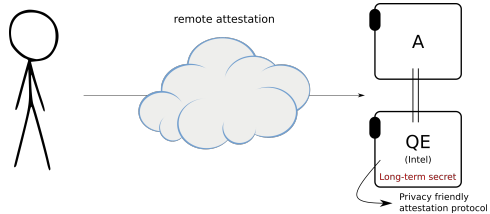
Design Goals/Requirements

- Strong isolation of SGX enclaves
- Defend against HW attacker
- Interruptible
- Sealed storage
- Attestation
  - Both local as remote
  - EPID attestation request can ensure that attestation responses cannot be linked
- Architecture **must** remain backwards compatible!



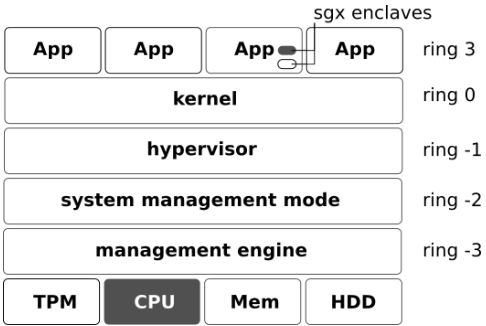
Design Goals/Requirements

- Strong isolation of SGX enclaves
- Defend against HW attacker
- Interruptible
- Sealed storage
- Attestation
  - Both local as remote
  - EPID attestation request can ensure that attestation responses cannot be linked
- Architecture **must** remain backwards compatible!



Design Goals/Requirements

- Strong isolation of SGX enclaves
- Defend against HW attacker
- Interruptible
- Sealed storage
- Attestation
- Architecture **must** remain backwards compatible!
  - OS/VMM must remain in charge of SGX processor reserved memory
  - OS/VMM must be able to interrupt enclave
  - Upon fault, control returned to OS



Enclaves Pages: Checks and Managements

Intel® SGX and Side-Channels

By Simon Johnson, published on March 16, 2017, updated February 27, 2018 Translate

Since launching Intel® Software Guard Extensions (Intel® SGX) on 6th Generation Intel® Core™ processors in 2015, there have been a number of academic articles looking at various usage models and the security of Intel SGX. Some of these papers focus on a class of attack known as a side-channel attack, where the attacker relies on the use of a shared resource to discover information about processing occurring in some other privileged domain that it does not have direct access to.

In general, these research papers do not demonstrate anything new or unexpected about the Intel SGX architecture. Preventing side channel attacks is a matter for the enclave developer. Intel makes this clear In the security objectives for Intel SGX, which we published as part of our workshop tutorial at the International Symposium on Computer Architecture in 2015, the slides for which can be found here [slides 109-121], and in the Intel® SGX SDK Developer's Manual.

Enclaves Pages: Checks and Managements

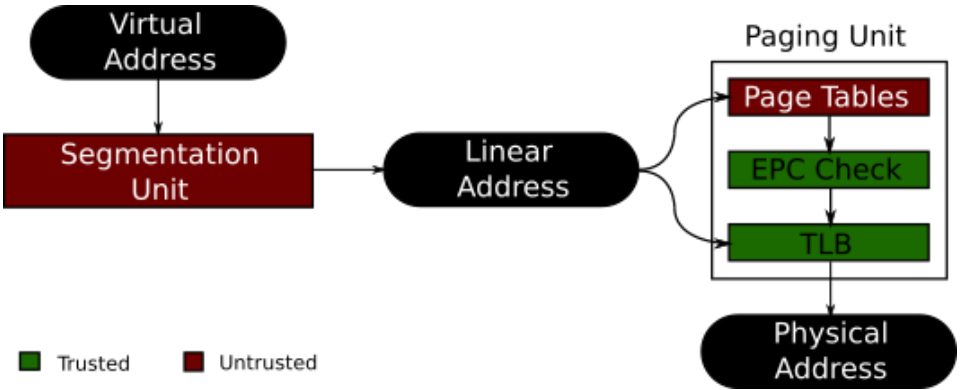
OS/VMM is in control over:

- Swapping SGX pages in/out SGX PRM memory
- Enclave memory creation/destruction
- Calling/Resuming SGX enclaves

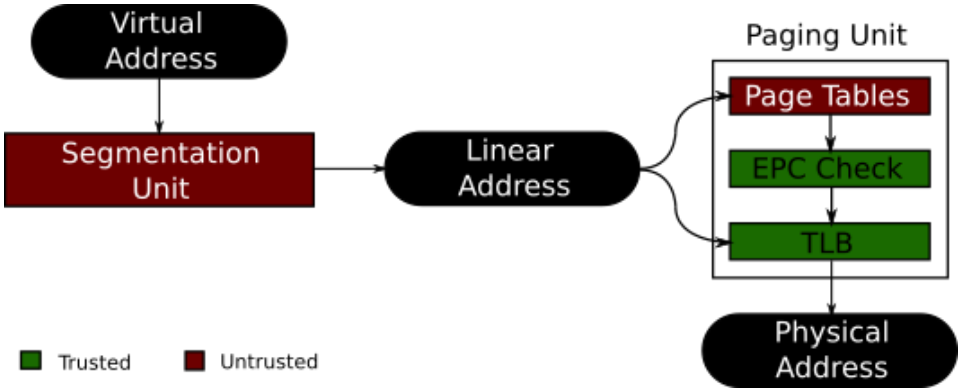
HW checks:

- Are enclaves pages loaded correctly at virtual address
- Are unprotected pages *not* loaded at enclave addresses
- TLB entries are always correctly loaded

Enclaves Pages: Checks and Managements



Enclaves Pages: Checks and Managements



→ What happens when an enclave page is not present in memory?

A Simple Attack [XCP15]

- Assume `WelcomeMessageForFemale` and `WelcomeMessageForMale` are located on different pages
- An attacker marks page table entry for both as not-present
- Execution of `WelcomeMessage`:
  - Will result in a Page Fault (#PF)
  - Control is handed back to the OS/Attacker
  - `CR2` records faulting page (12 LSB are cleared)

input-dependent control flow

```
200 char *WelcomeMessage( GENDER s ) {
201   char *mesg; // GENDER is an enum of MALE and FEMALE
202
203   if( s == MALE )
204     mesg = WelcomeMessageForMale();
205   else
206     // FEMALE
207     mesg = WelcomeMessageForFemale();
208
209   return mesg;
210 }
```

A Simple Attack [XCP15]

- Assume `WelcomeMessageForFemale` and `WelcomeMessageForMale` are located on different pages
- An attacker marks page table entry for both as not-present
- Execution of `WelcomeMessage`:
  - Will result in a Page Fault (#PF)
  - Control is handed back to the OS/Attacker
  - `CR2` records faulting page (12 LSB are cleared)

input-dependent data access

```
228 void CountLogin( GENDER s ) {
229   if ( s == MALE )
230     gMaleCount++;
231   else
232     gFemaleCount++;
233 }
```

A Simple Attack [XCP15]

- Assume `WelcomeMessageForFemale` and `WelcomeMessageForMale` are located on different pages
- An attacker marks page table entry for both as not-present
- Execution of `WelcomeMessage`:
  - Will result in a Page Fault (#PF)
  - Control is handed back to the OS/Attacker
  - `CR2` records faulting page (12 LSB are cleared)

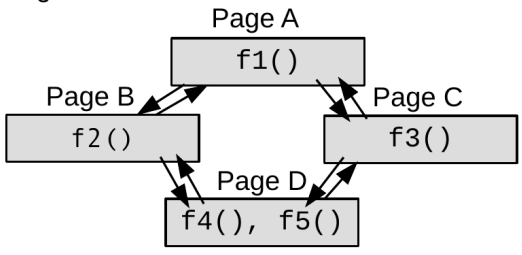
input-dependent data access

```
245 void CountLogin( GENDER s ) {
246   if ( s == MALE )
247     gMaleCount++;
248   else
249     gFemaleCount++;
250 }
```

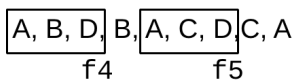
What if interesting functions/data resides on the *same* page?

Page Fault Sequences [XCP15]

Page-level control transfers



Code page fault sequence:



Source code

```
f1() {
  …
  f2();
  …
  f3();
  …
}
```

```
f2() {
  …
  f4();
  …
}
```

```
f3() {
  …
  f5();
  …
}
```

Page Fault Sequences [XCP15]

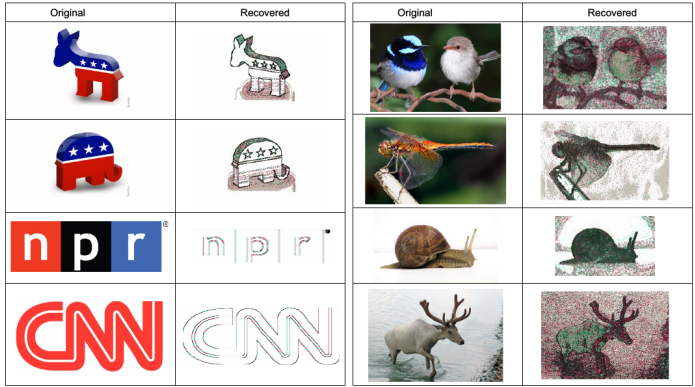
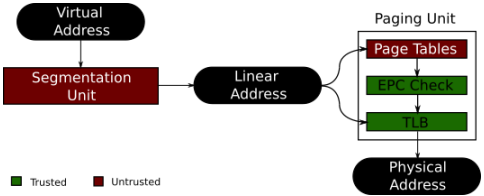


Figure: Extracting data from an enclaved libjpeg library

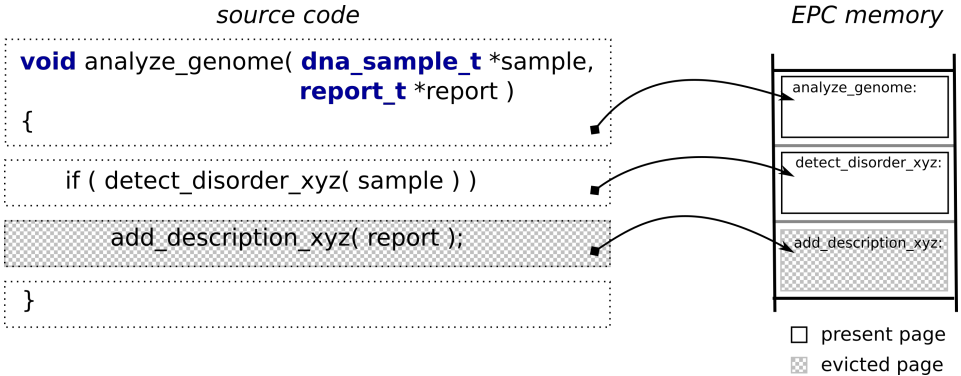
Closely Related Attacks

Access restrictions in page tables are still enforced during enclave execution!

- Reading not-present page → #PF
- Writing not-writable page → #PF
- Executing not-executable page → #PF
- Malformed PTE entry → #PF



Attacking SGX Enclaves Without Page Faults [SP17]



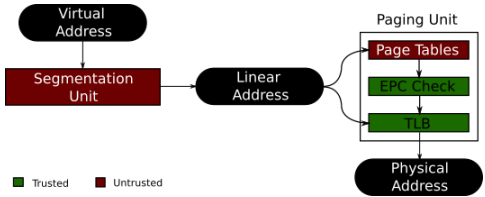
Even absence of #PF leaks information!

Attacking SGX Enclaves Without Page Faults [BWK+17]

Other side-effects of page table walks still apply as well

- Accessed bits are still set
- Dirty bits are still set

→ Information leaks even without page faults



Attacking SGX Enclaves Without Page Faults [BWK+17]

Other side-effects of page table walks still apply as well

- Accessed bits are still set
- Dirty bits are still set

→ Information leaks even without page faults

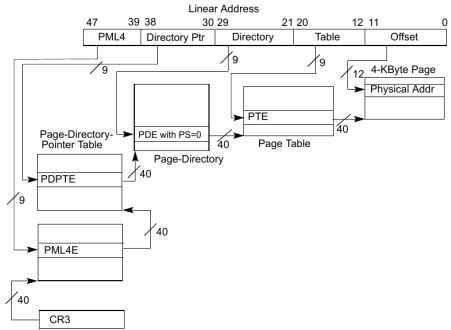
Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
(M-1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Attacking SGX Enclaves Without Page Faults [BWK+17]

Other side-effects of page table walks still apply as well

- PT entries still end up in the cache
  - `sizeof(PTE) = 8 bytes`
  - `sizeof(cache line) = 64 bytes`
  - → coarser access granularity
  - → ever growing set of accessed pages
  - → IPI can be fired from another logical core when a trigger page is accessed

→ Information leaks even without page faults

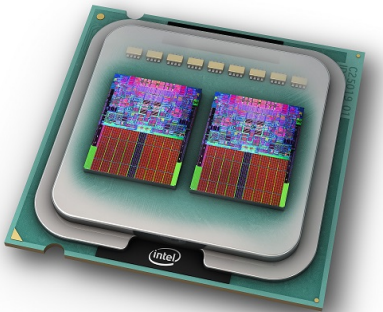


Attacking SGX Enclaves Without Page Faults [BWK+17]

Other side-effects of page table walks still apply as well

- PT entries still end up in the cache
  - `sizeof(PTE) = 8 bytes`
  - `sizeof(cache line) = 64 bytes`
  - → coarser access granularity
  - → ever growing set of accessed pages
  - → IPI can be fired from another logical core when a trigger page is accessed

→ Information leaks even without page faults



Requirements

Security Guarantees

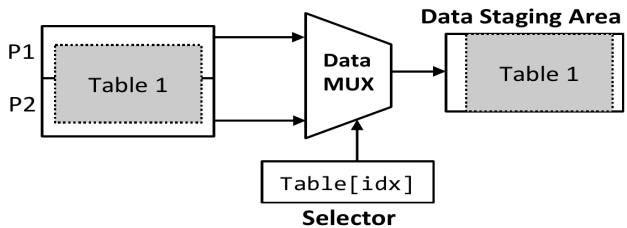
- Not all attacks rely on #PFs
- Absence of a #PF also leaks information

→ There is no silver bullet!

Operational Guarantees for System Software and Enclaves

- OS/VMM *must* remain in full control over *all* system resources (incl. SGX PRM memory)
  - Do not lock (parts of) enclaves in memory
  - HW should aid in multiplexing SGX PRM memory
  - Do not disable interrupts
- Correctly-written enclaves when not under attack, should never end up in a state where they cannot advance

Page Obliviousness [SCNS16]

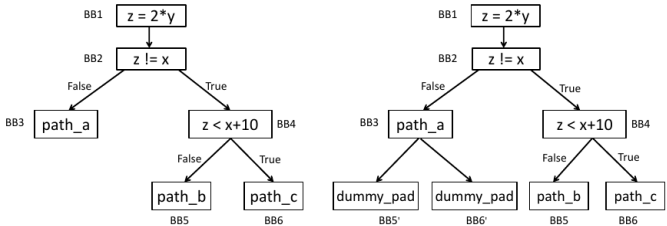


Key Idea:

- Access all *potentially* required pages to a staging area.
- Only code/data actually required is copied to staging area.
- Execute/Compute on staging area

Page Obliviousness [SCNS16]

```
1  foo (int x, int y)
2  {
3    z = 2 * y
4    if (z != x)
5    {
6      if (z < x + 10)
7        path_c()
8      else
9        path_b()
10   }
11   else
12     path_a()
13 }
```



Key Idea:

- Access all *potentially* required pages to a staging area.
- Only code/data actually required is copied to staging area.
- Execute/Compute on staging area

Page Obliviousness [SCNS16]

Library	Cases	Vanilla		Unoptimized Deterministic Multiplexing						Optimized Deterministic Multiplexing			
		PF	T (ms)	PF	Tc (ms)	Te (ms)	T (ms)	Tc / T (%)	Ovh (%)	Opt	PF	T (ms)	Ovh (%)
Libgcrypt (v1.6.3)	AES	4 - 5	4.711	4	7.357	4.013	11.370	64.70	**141.35**	O1,O2	4	4.566	**-3.08**
	CAST5	2	3.435	2	8.050	2.578	10.629	75.74	**209.47**	O1,O2	1	3.086	**-10.15**
	EdDSA	0	10498.674	0	—		>10 hrs	—	**>300000**	O5	0	13566.122	**29.22**
	powm	0	5318.501	0					**>400000**	O3	0	399614.244	7413.66
										O4	0	5513.712	**3.67**
	SEED	2	1.377	2	4.559	1.057	5.615	81.18	**307.79**	O1, O2	1	1.311	**-4.80**
	Stribog	5	27.397	5	329.743	10.836	340.579	96.82	**1143.13**	O1, O2	4	28.563	**4.26**
	Tiger	3	2.020	3	64.482	0.546	65.029	99.16	**3119.69**	O1, O2	2	1.840	**-8.89**
	Whirlpool	5	27.052	5	141.829	10.174	151.490	93.28	**459.99**	O1, O2	4	23.744	**-12.23**
OpenSSL (v1.0.2)	CAST5	2	11.249	2	17.083	8.295	25.378	67.31	**125.60**	O1, O2	1	10.623	**-3.41**
	SEED	2	3.684	2	8.998	3.737	12.734	70.66	**245.69**	O1, O2	1	3.558	**-5.57**
Average Performance Overhead									**70575.27**				**-1.10**

Assumptions

- Pages can only be unloaded after a page fault
- Cannot be applied to all applications

Page Obliviousness [SCNS16]

execution tree. We checked the programs FreeType, Hunspell, and libjpeg discussed in [52], they exhibit unbalanced execution tree. Transforming these programs to exhibit balanced execution tree causes an unacceptable loss in the performance, even without our defense [49]. Hence, we limit our evaluation to cryptographic implementations.

Assumptions

- Pages can only be unloaded after a page fault
- Cannot be applied to all applications

T-SGX [SLKP17]

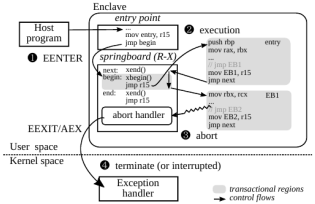
```
unsigned status;

// begin a transaction
if ((status = _xbegin()) == _XBEGIN_STARTED) {
  // execute a transaction
  [code]
  // atomic commit
  _xend();
} else {
  // abort
}
```

Key Idea:

- Wrap all code in a TSX transaction
- Transactions always start/end at the springboard page
- Each transaction aborts on interrupt/#PF, restart it
- Destruct enclave after 10 aborts of the same transaction

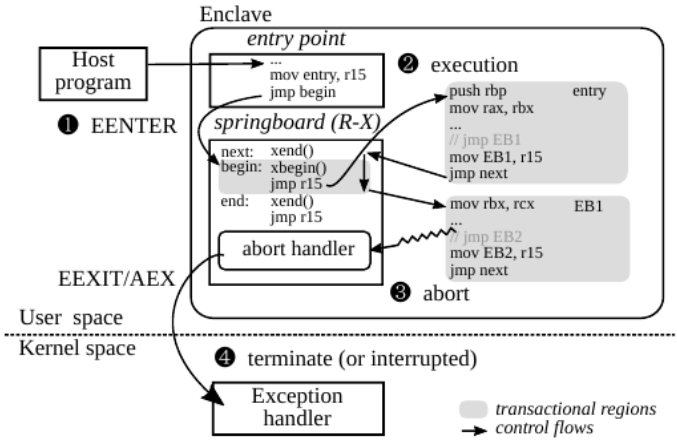
T-SGX [SLKP17]



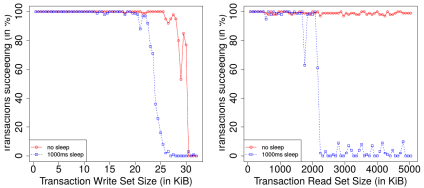
Key Idea:

- Wrap all code in a TSX transaction
- Transactions always start/end at the springboard page
- Each transaction aborts on interrupt/#PF, restart it
- Destruct enclave after 10 aborts of the same transaction

T-SGX [SLKP17]


Enclave
entry point
Host program
...
mov entry, r15
jmp begin
❷ execution
push rbp
mov rax, rbx
...
// jmp EB1
mov EB1, r15
jmp next
entry
❶ EENTER
springboard (R-X)
next: xend()
begin: xbegin()
jmp r15
end: xend()
jmp r15
mov rbx, rcx
...
// jmp EB2
mov EB2, r15
jmp next
EB1
abort handler
EEXIT/AEX
❸ abort
User space
Kernel space
❹ terminate (or interrupted)
Exception handler
transactional regions
control flows

T-SGX [SLKP17]



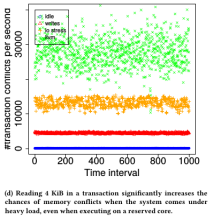
(a) Write set size (b) Read set size

Figure: Transaction sizes are limited [SP17]

Limitations

- Does not detect #PF-less side channels!
- TSX transaction sizes are limited
- Unclear how enclaves can be restarted securely

T-SGX [SLKP17]



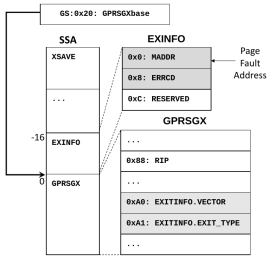
(d) Reading 4 KiB in a transaction significantly increases the chances of memory conflicts when the system comes under heavy load, even when executing on a reserved core.

Figure: Transaction aborts more repeatedly when system comes under heavy load[SP17]

Limitations

- Does not detect #PF-less side channels!
- TSX transaction sizes are limited
- Unclear how enclaves can be restarted securely

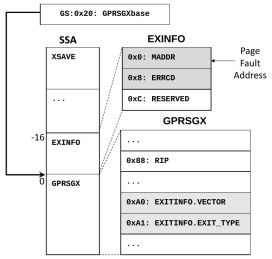
SGX-LAPD [FBQL17]



Key Idea:

- SGX enclaves can record the faulting page after a #PF in `EXINFO` structure
- Upon enclave re-entry: check if #PF occurred
- Only 2 MB boundary crosses are considered harmful

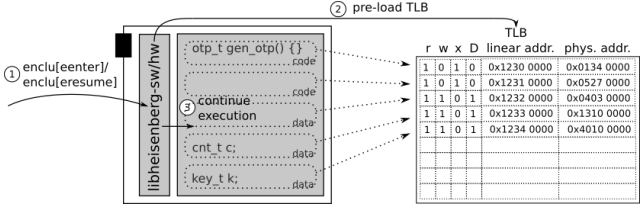
SGX-LAPD [FBQL17]



Limitations:

- Data accesses are considered future work
- `EXINFO` is overwritten for each malformed APIC timer interrupt [SLKP17]

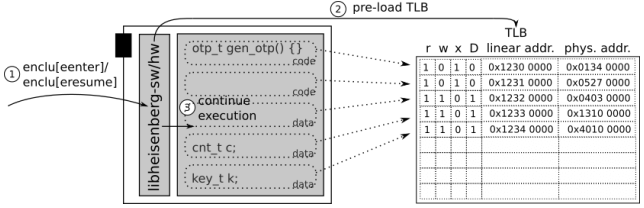
Heisenberg [SP17]



Key Idea:

- Hook enclave entry/re-entry → requires TSX or HW new features
- Preload all required enclave pages in TLB

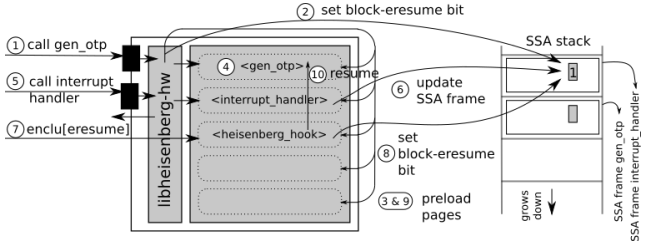
Heisenberg [SP17]



Heisenberg-SW

- Suffers from the same problems as T-SGX w.r.t. TSX
- Does prevent all known attacks

Heisenberg [SP17]



Heisenberg-HW

- New hardware: Hook code pointer called upon enclave `ERESUME`
- Problem: Maximum SSA stack required

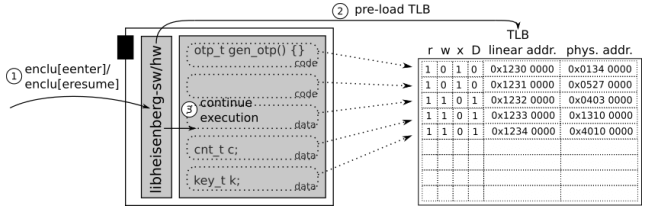
Heisenberg [SP17]

	unprot.	Heisenberg-HW		Heisenberg-SW		
Benchmark	time	overhead	# int.	overhead	# aborts	# commits
Fibo	714.052ms	-4.57%	173	22.54%	1,544,	2,019,273
SHA512	10.087μs	1.56%	0	-34.43%	0	5

Performance

- TSX aborts cause significant performance hit, especially when system is under heavy load

Heisenberg [SP17]



Limitations

- Requires knowledge of TLB implementation
- Only part of a solution when enclaves are larger than SGX PRM

Conclusion

- #PF and #PF-less side channels are still an open problem
- What happens when an enclave is larger than SGX PRM?
- We probably need:
  - New hardware features
  - Language support

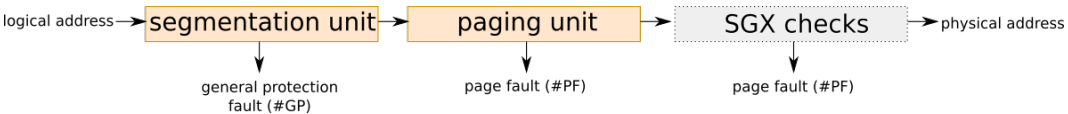
Thank you!

Thank you! Questions?

@raoul_strackx

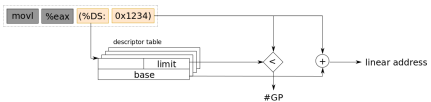
Research Question

“Can we leverage the segmentation unit to extract sensitive enclave data?”



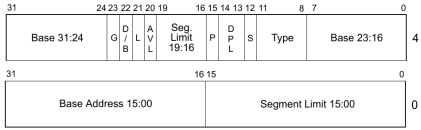
Segmentation in 32-bit mode

- Maps variable-length segments logical address space
- Many different segments
  - %CS (code)
  - %DS (data)
  - %SS (stack)
  - %ES
  - %FS
  - %GS
- G = 1, size = 1 Byte to 1 MB, 1 Byte incr.
- G = 0, size = 4 KB to 4 GB, 4 KB incr.



Segmentation in 32-bit mode

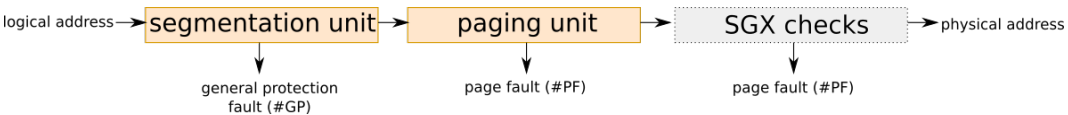
- Maps variable-length segments logical address space
- Many different segments:
  - %CS (code)
  - %DS (data)
  - %SS (stack)
  - %ES
  - %FS
  - %GS
- G = 1, size = 1 Byte to 1 MB, 1 Byte incr.
- G = 0, size = 4 KB to 4 GB, 4 KB incr.



L — 64-bit code segment (IA-32e mode only)
AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type

Interaction Between Segmentation and Intel SGX

- “enclaves abide by all segmentation policies set up by the OS” [Int18]
- but additional security measures:
  - Segment base of %CS, %DS, %SS and %ES must be `0x00000000`
  - Segment selectors/descriptors of %FS and %GS are save/restored on enclave boundaries
  - **Segment limits of %CS, %DS, %SS and %ES can still be set**



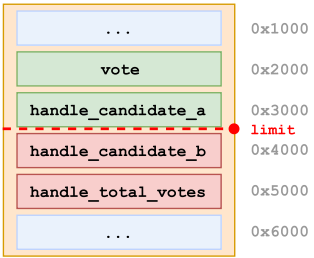
Attack Model

Let's assume:

- 32-bit enclave
- microcode version 0xba (April 9th, 2017) or older
- kernel-level attacker
- (focus on 4 KiB granular accesses now)

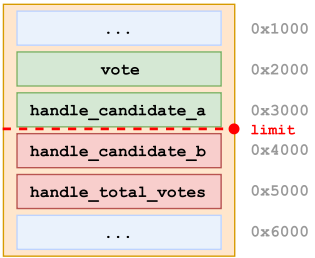
Proof-of-Concept

```
1 void vote(enum candidate c) {
2   if (c == candidate_a)
3     handle_candidate_a();
4   else
5     handle_candidate_b();
6   handle_total_votes();
7   return;
8 }
9
10 void handle_candidate_a() {...}
11
12 // limit!
13 void handle_candidate_b() {...}
14
15 void handle_total_votes() {...}
```



Proof-of-Concept

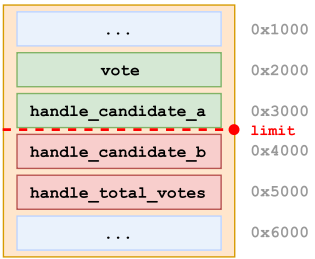
```
1 void vote(enum candidate c) {
2   if (c == candidate_a)
3     handle_candidate_a();
4   else
5     handle_candidate_b();
6   handle_total_votes();
7   return;
8 }
9
10 void handle_candidate_a() {...}
11
12 // limit!
13 void handle_candidate_b() {...}
14
15 void handle_total_votes() {...}
```



- vote == B →
- vote == A →

Proof-of-Concept

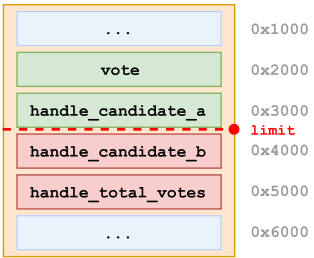
```
1 void vote(enum candidate c) {
2   if (c == candidate_a)
3     handle_candidate_a();
4   else
5     handle_candidate_b();
6   handle_total_votes();
7   return;
8 }
9
10 void handle_candidate_a() {...}
11
12 // limit!
13 void handle_candidate_b() {...}
14
15 void handle_total_votes() {...}
```



- vote == B → General Protection (#GP) fault
- vote == A →

Proof-of-Concept

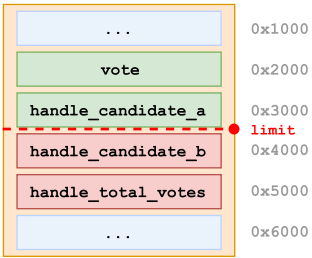
```
1 void vote(enum candidate c) {
2   if (c == candidate_a)
3     handle_candidate_a();
4   else
5     handle_candidate_b();
6   handle_total_votes();
7   return;
8 }
9
10 void handle_candidate_a() {...}
11
12 // limit!
13 void handle_candidate_b() {...}
14
15 void handle_total_votes() {...}
```



- vote == B → General Protection (#GP) fault
- vote == A → General Protection (#GP) fault!

Proof-of-Concept

```
1 void vote(enum candidate c) {
2   if (c == candidate_a)
3     handle_candidate_a();
4   else
5     handle_candidate_b();
6   handle_total_votes();
7   return;
8 }
9
10 void handle_candidate_a() {...}
11
12 // limit!
13 void handle_candidate_b() {...}
14
15 void handle_total_votes() {...}
```

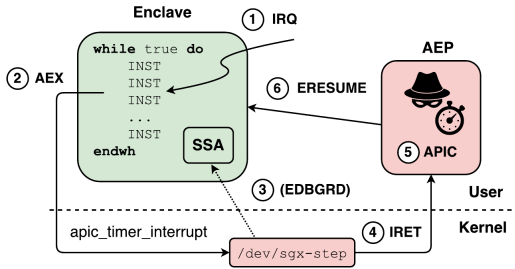


We need a second information channel!

SGX-Step

Single-stepping through an enclave: [VBPS17]

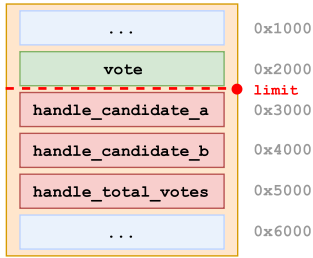
- Precisely configures APIC timer
- Starts the enclave
- Enclave exits immediately after the first instruction



Proof-of-Concept

```
1 void vote(enum candidate c) {
2   if (c == candidate_a)
3     handle_candidate_a();
4   else
5     handle_candidate_b();
6   handle_total_votes();
7   return;
8 }
9
10 void handle_candidate_a() {...}
11 void handle_candidate_b() {...}
12 void handle_total_votes() {...}
```

- Track `vote` function
- Schedule APIC interrupt + extend limit
- Observe execution path

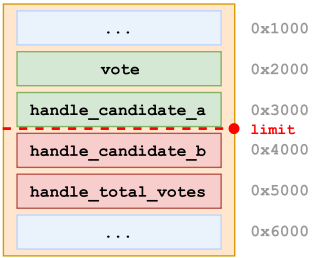


- vote == B →
- vote == A →

Proof-of-Concept

```
1 void vote(enum candidate c) {
2   if (c == candidate_a)
3     handle_candidate_a();
4   else
5     handle_candidate_b();
6   handle_total_votes();
7   return;
8 }
9
10 void handle_candidate_a() {...}
11 void handle_candidate_b() {...}
12 void handle_total_votes() {...}
```

- Track `vote` function
- Schedule APIC interrupt + extend limit
- Observe execution path

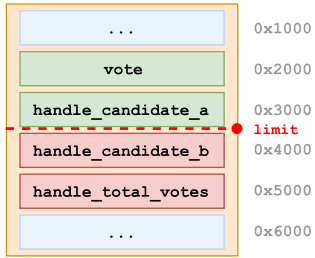


- vote == B $\rightarrow$
- vote == A $\rightarrow$

Proof-of-Concept

```
1 void vote(enum candidate c) {
2   if (c == candidate_a)
3     handle_candidate_a();
4   else
5     handle_candidate_b();
6   handle_total_votes();
7   return;
8 }
9
10 void handle_candidate_a() {...}
11 void handle_candidate_b() {...}
12 void handle_total_votes() {...}
```

- Track `vote` function
- Schedule APIC interrupt + extend limit
- Observe execution path

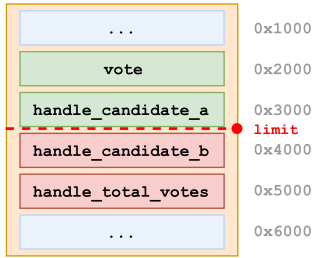


- vote == B $\rightarrow$ #GP fault
- vote == A $\rightarrow$

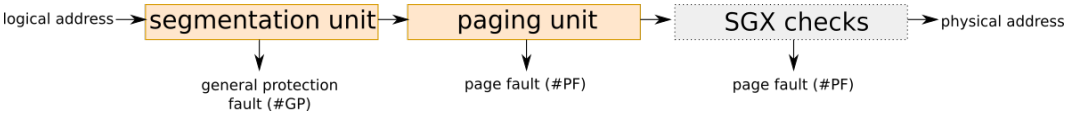
Proof-of-Concept

```
1 void vote(enum candidate c) {
2   if (c == candidate_a)
3     handle_candidate_a();
4   else
5     handle_candidate_b();
6   handle_total_votes();
7   return;
8 }
9
10 void handle_candidate_a() {...}
11 void handle_candidate_b() {...}
12 void handle_total_votes() {...}
```

- Track `vote` function
- Schedule APIC interrupt + extend limit
- Observe execution path



- vote == B $\rightarrow$ #GP fault
- vote == A $\rightarrow$ AEX!



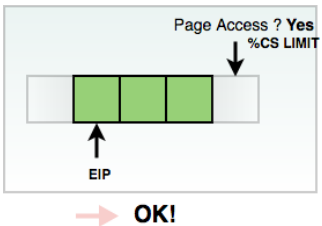
What happens when we combine Paging/Segmentation attacks?

Attack Model

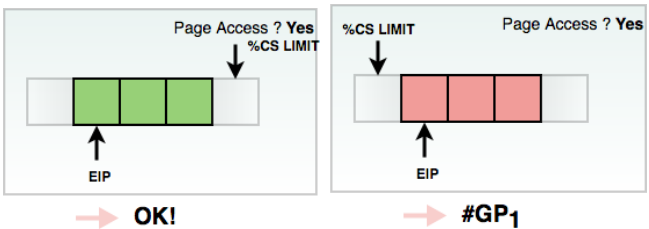
Let's assume:

- Enclave is relocatable
- Code is within first 1 MiB of enclave
- microcode version 0xba (April 9th, 2017) or older
- user-level attacker

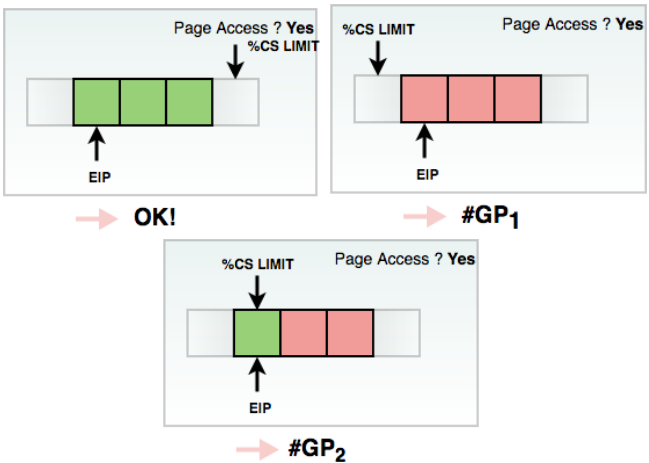
Combining Segmentation/Paging units



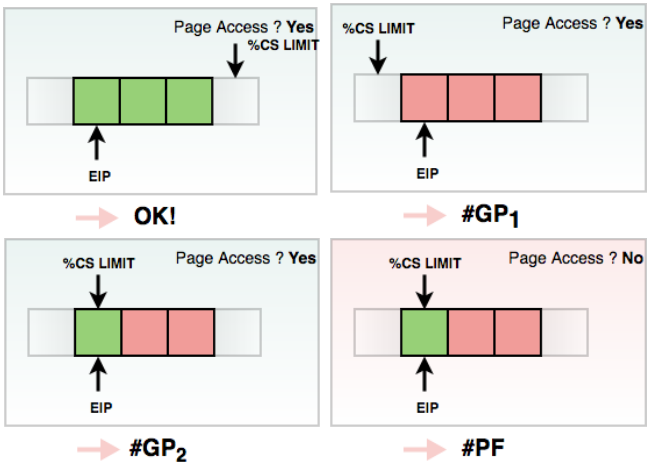
Combining Segmentation/Paging units



Combining Segmentation/Paging units



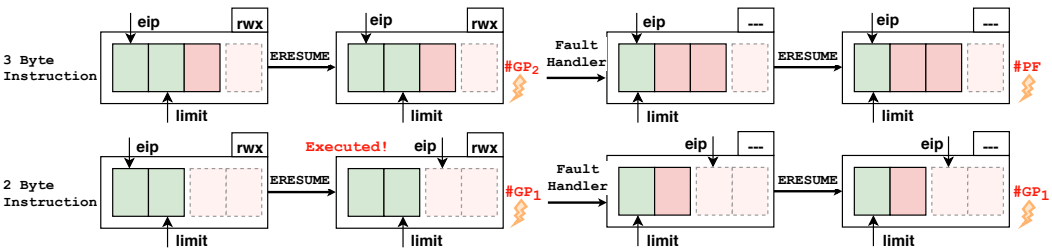
Combining Segmentation/Paging units



Combining Segmentation/Paging units

eip $\leq$ limit	page access rights	(eip + inst size) $\leq$ limit	Fault type
✗	-	-	#GP$_1$
✓	✓	✗	#GP$_2$
✓	✗	-	#PF

Can we Extract Instruction Sizes?



Mitigations

We observed something interesting:

version	release date	CPUSVN	vulnerable
0x1E	unknown	020202ffffff00000000000000000000	Yes
0x2E	unknown	020202ffffff00000000000000000000	Yes
0x9E	unknown	020202ffffff00000000000000000000	Yes
0x4A	unknown	020202ffffff00000000000000000000	Yes
0x8A	unknown	020202ffffff00000000000000000000	Yes
0xBA	April 9th, 2017	020202ffffff00000000000000000000	No
0xC2	November 16th, 2017	02**07**02ffffff00000000000000000000	No

→ Intel silently patched this vulnerability

What changed!?

- Not yet incorporated in the manual → based on observations!
- Placing *any* segment limit within enclave → #GP
- Placing limit below enclave base:
  - %CS: #GP (used during enclave (re-)entry)
  - %DS: #GP (used during enclave (re-)entry)
  - **%ES: #GP when used!** [Gys18]
  - **%SS: #GP when used!** [Gys18]
  - %FS: OK (overwritten during enclave (re-)entry)
  - %GS: OK (overwritten during enclave (re-)entry)

References I


J. V. Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx.
Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution.
In *26th USENIX Security Symposium (USENIX Security 17)*, pp. 1041–1056, Vancouver, BC, 2017. USENIX Association.


Y. Fu, E. Bauman, R. Quinonez, and Z. Lin.
S gx-I apd: Thwarting controlled side channel attacks via enclave verifiable page faults.
In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 357–380. Springer, 2017.


J. Gyselinck.
Segmentation-based side-channel attacks on enclaved execution.
Master's thesis, KU Leuven, 2018.


Intel.
Intel 64 and IA-32 Architectures Software Developer's Manual – Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C and 3D, May 2018.


S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena.
Preventing page faults from telling your secrets.
In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIACCS'16, pp. 317–328, New York, NY, USA, 2016. ACM.

References II


M.-W. Shih, S. Lee, T. Kim, and M. Peinado.
T-SGX: Eradicating controlled-channel attacks against enclave programs.
In *Proceedings of the 2017 Network and Distributed System Security Symposium (NDSS 2017)*, February 2017.


R. Strackx and F. Piessens.
The heisenberg defense: Proactively defending sgx enclaves against page-table-based side-channel attacks, 2017.


J. Van Bulck, F. Piessens, and R. Strackx.
Sgx-step: A practical attack framework for precise enclave execution control.
2017.


Y. Xu, W. Cui, and M. Peinado.
Controlled-channel attacks: Deterministic side channels for untrusted operating systems.
In *36th IEEE Symposium on Security and Privacy*. IEEE, May 2015.