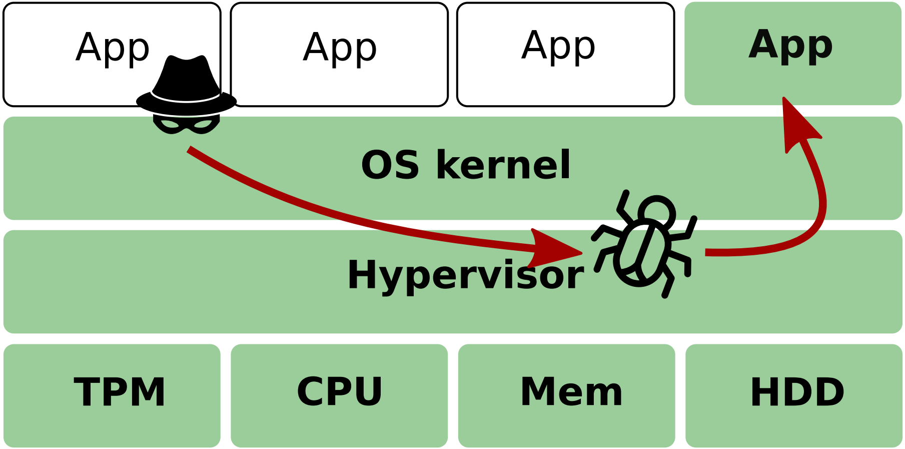
The Hitchhiker's Guide to Subverting Intel SGX Enclaves

Jo Van Bulck

Project Circuit Breaker Intel SGX Bootcamp (online), March 27, 2022

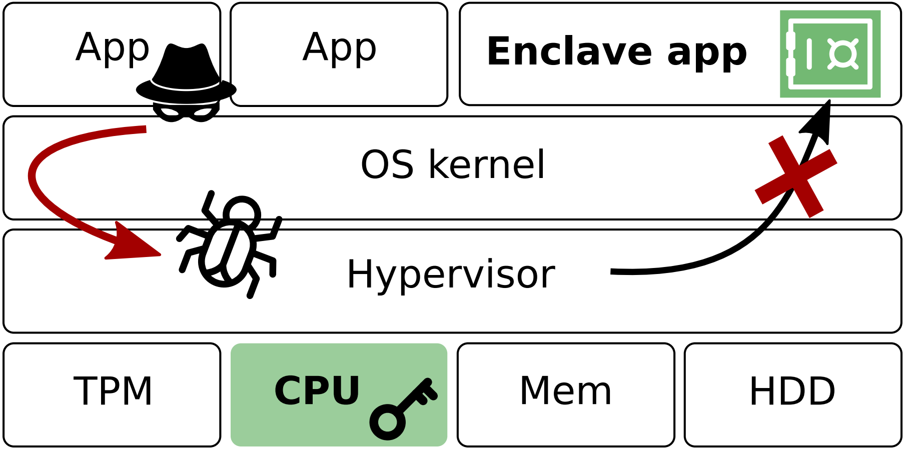
🏠 imec-DistriNet, KU Leuven, Belgium ✉ jo.vanbulck@cs.kuleuven.be 🐦 jovanbulck

Enclaved execution: Reducting attack surface



Traditional **layered designs:** Large trusted computing base

Enclaved execution: Reducting attack surface



App

App

Enclave app

OS kernel

Hypervisor

TPM

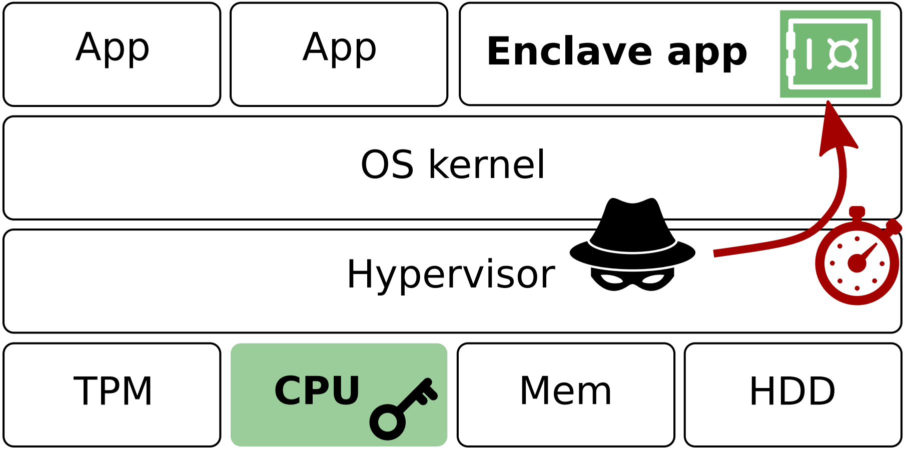
CPU

Mem

HDD

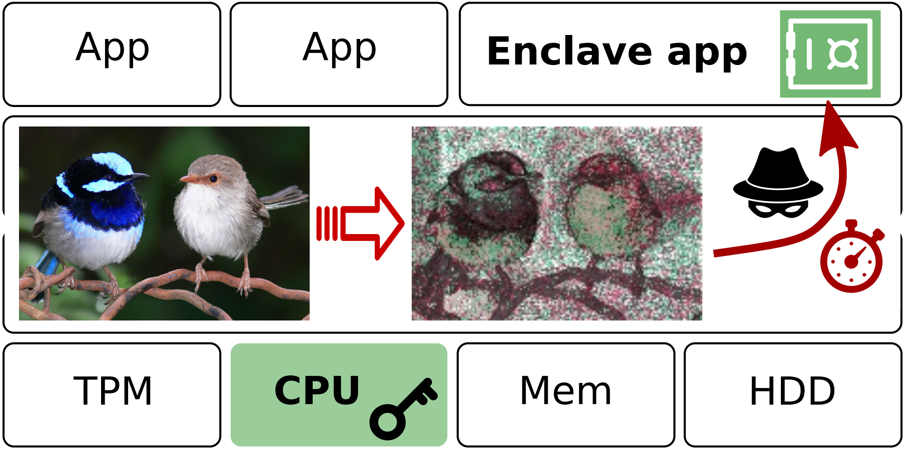
Intel SGX promise: Hardware-level **isolation and attestation**

Enclaved execution: Reducting attack surface



Game changer: Untrusted OS → new class of powerful side-channel attacks!

Enclaved execution: Reducting attack surface



App

App

Enclave app

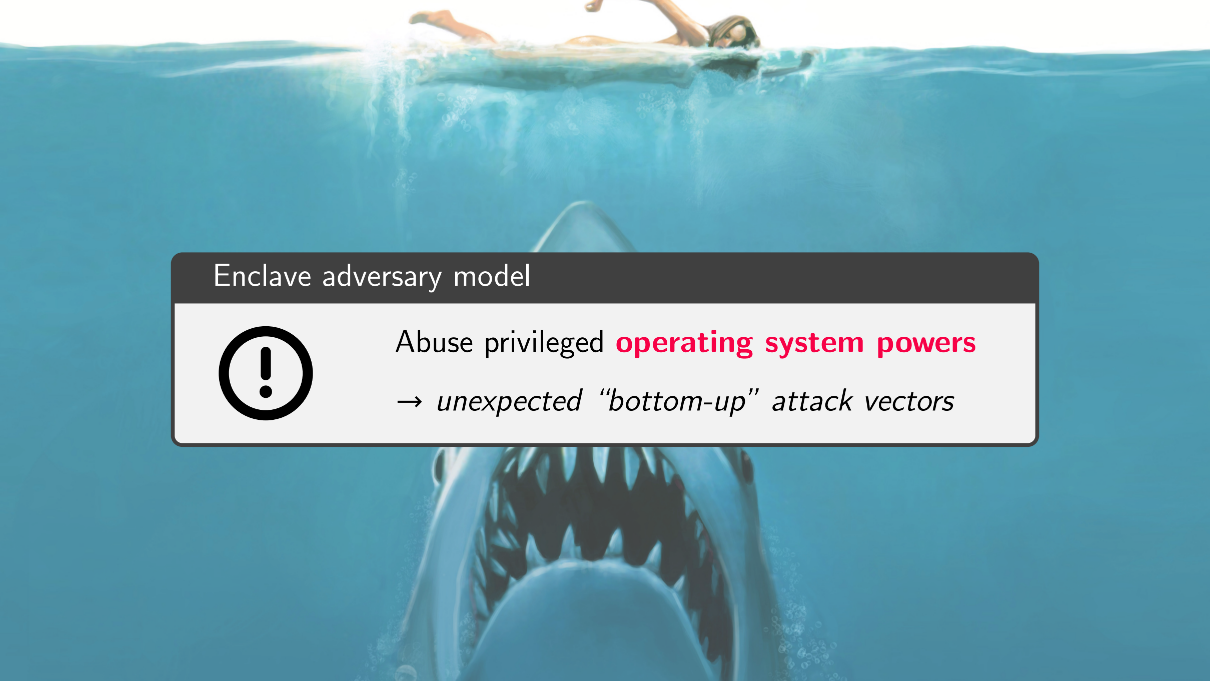
TPM

CPU

Mem

HDD

Xu et al. "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", IEEE S&P 2015

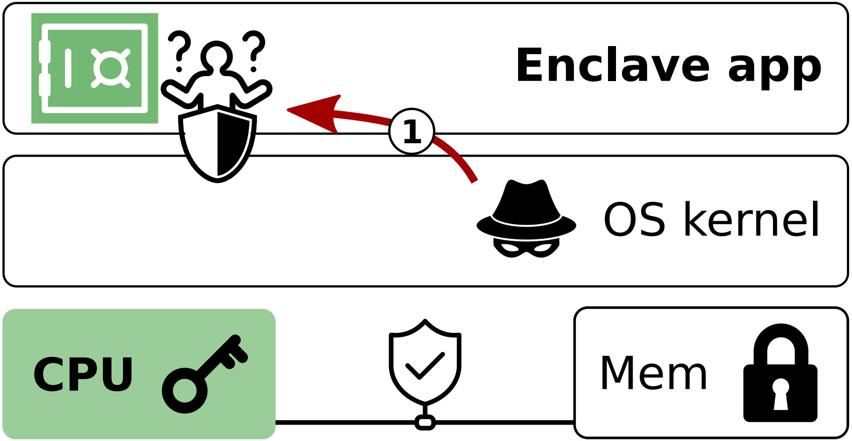


Enclave adversary model

Abuse privileged **operating system powers**

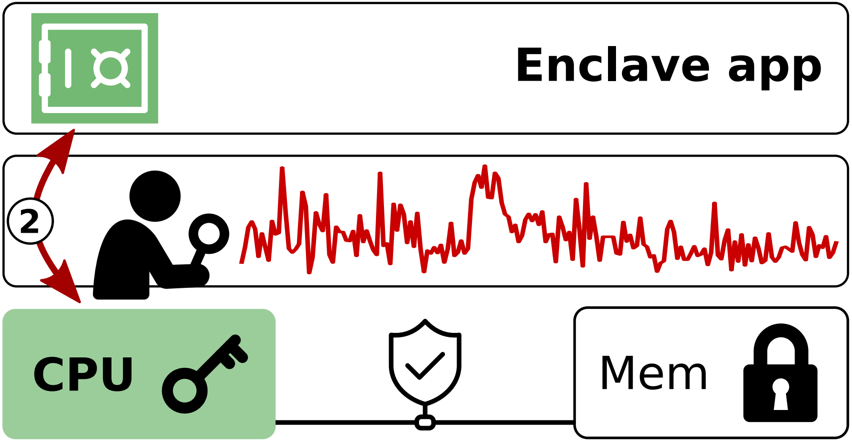
→ *unexpected “bottom-up” attack vectors*

Lecture overview: Software interface attacks (part 1)



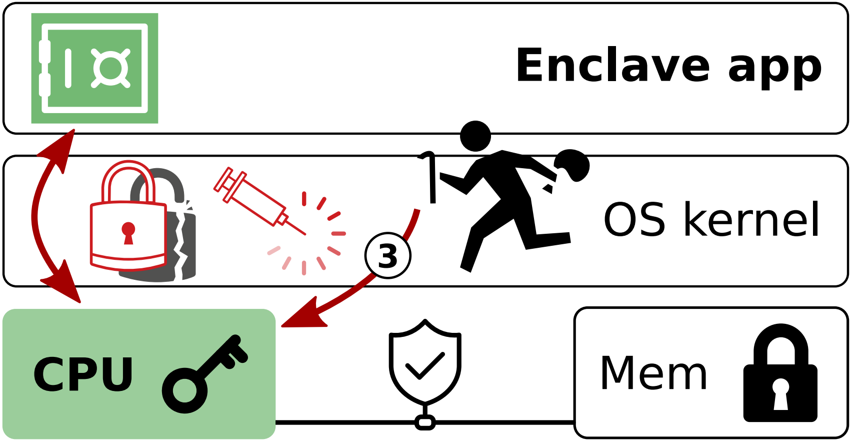
SGX not immune to **interface sanitization** oversights in enclave software

Lecture overview: Privileged side-channel attacks (part 2)



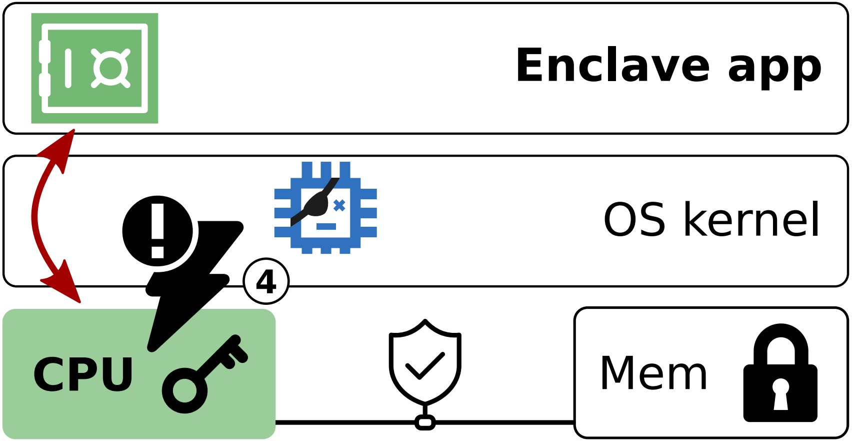
Privileged side channels to spy on enclave-CPU interaction metadata

Lecture overview: Transient-execution attacks (part 3)



Transient-execution data extraction from CPU to break enclave confidentiality

Lecture overview: Physical CPU interface attacks (part 4)



Unexpected (physical) **CPU interface** manipulations to break enclave integrity

Attack idea 1: Enclave interface

Outline: How to besiege a fortress?

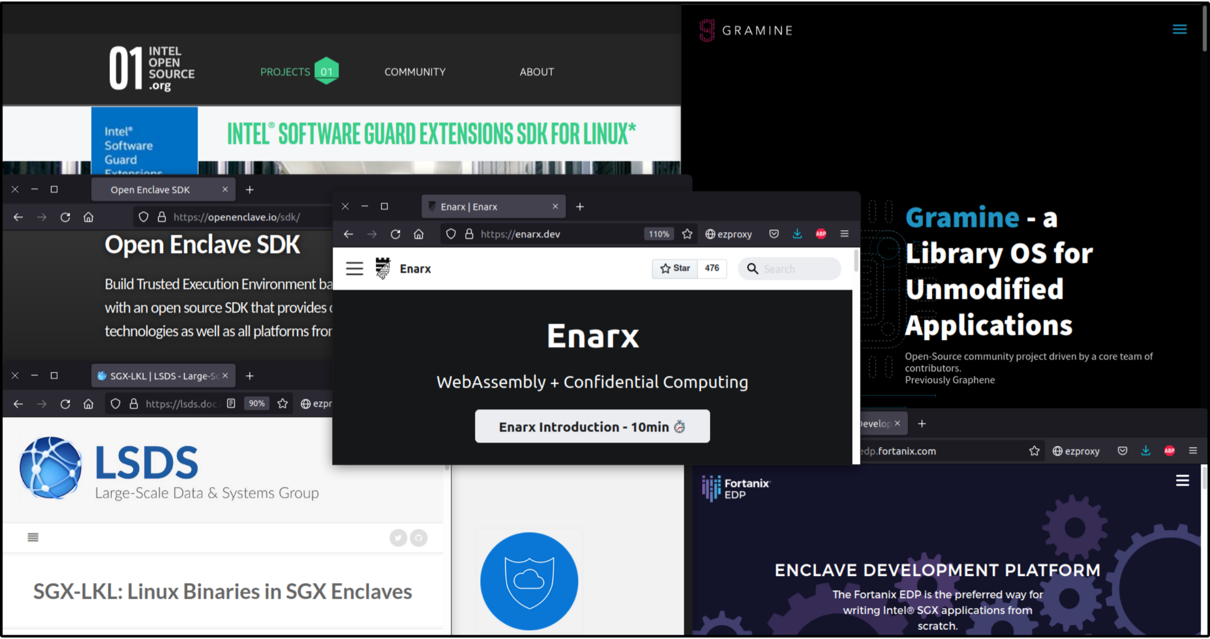
Idea: security is weakest at the input/output interface(!)

Outline: How to besiege an enclave?

github.com/jovanbulck/0xbadc0de/

	Runtime Vulnerability	SGX-SDK	OpenEnclave	Graphene	SGX-LKL	Rust-EDP	Asylo	Keystone	Sancus
Tier1 (ABI)	#1 Entry status flags sanitization	★	★	◐	●	◐	●	○	○
	#2 Floating-point register sanitization	★	★	○	★	★	●	○	○
	#3 Entry stack pointer restore	○	○	★	●	○	○	○	★
	#4 Exit register leakage	○	○	○	★	○	○	○	○
Tier2 (API)	#5 Missing pointer range check	○	★	★	★	○	●	○	★
	#6 Null-terminated string handling	⯪	★	○	○	○	○	○	○
	#7 Integer overflow in range check	○	○	●	○	●	○	●	●
	#8 Incorrect pointer range check	○	○	●	○	○	●	○	●
	#9 Double fetch untrusted pointer	○	○	●	○	○	○	○	○
	#10 Ocall return value not checked	○	★	★	★	○	●	★	○
	#11 Uninitialized padding leakage	[Lee17]	★	○	●	○	●	★	★

Summary: 7 CVEs, > 40 vulnerabilities across > 8 projects


01 INTEL OPEN SOURCE .org
PROJECTS 01
COMMUNITY
ABOUT
Intel® Software Guard Extensions
INTEL® SOFTWARE GUARD EXTENSIONS SDK FOR LINUX*
Open Enclave SDK
https://openenclave.io/sdk/
Enarx | Enarx
https://enarx.dev
110%
ezproxy
Enarx
Star
476
Search
Enarx Introduction - 10min
SGX-LKL | LSDS - Large-Sc
https://lsds.doc
90%
ezpr
LSDS
Large-Scale Data & Systems Group
GRAMINE
Develop
edp.fortanix.com
ezproxy
Fortanix EDP

Open Enclave SDK

Build Trusted Execution Environment ba with an open source SDK that provides technologies as well as all platforms fror

Enarx

WebAssembly + Confidential Computing

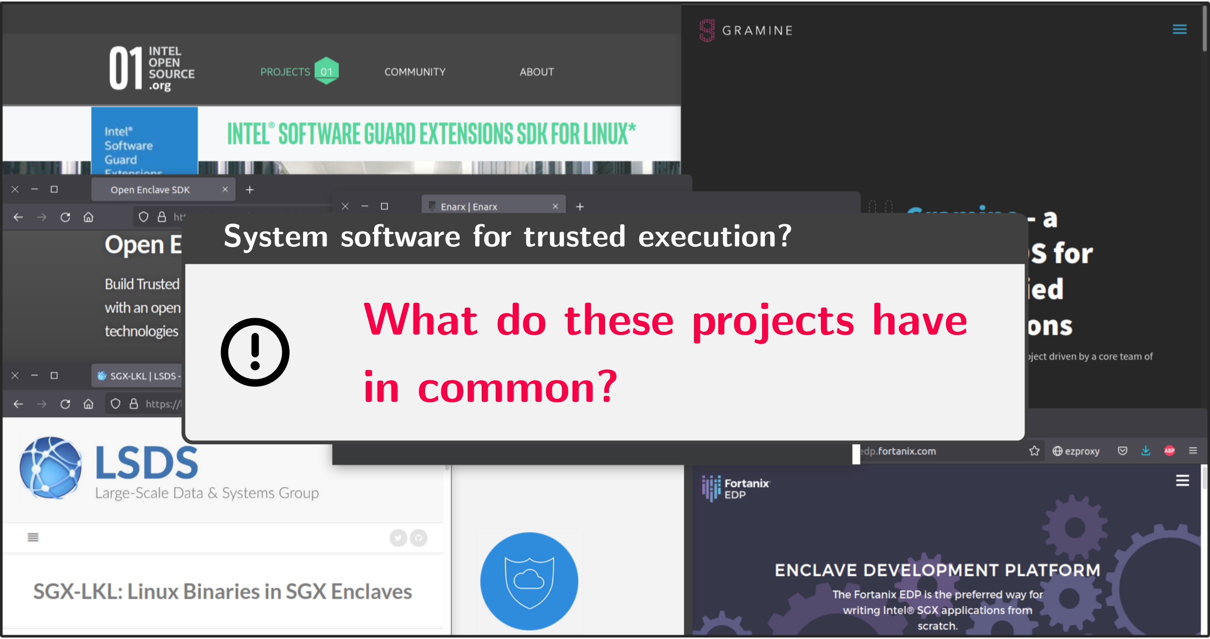
SGX-LKL: Linux Binaries in SGX Enclaves

Gramine - a Library OS for Unmodified Applications

Open-Source community project driven by a core team of contributors.
Previously Graphene

ENCLAVE DEVELOPMENT PLATFORM

The Fortanix EDP is the preferred way for writing Intel® SGX applications from scratch.


01 INTEL OPEN SOURCE .org
PROJECTS 01
COMMUNITY
ABOUT
Intel® Software Guard Extensions
INTEL® SOFTWARE GUARD EXTENSIONS SDK FOR LINUX*
GRAMINE
Open Enclave SDK
Open E
Enarx | Enarx
SGX-LKL | LSDS -
https://
edp.fortanix.com
ezproxy
LSDS
Large-Scale Data & Systems Group
Fortanix EDP

Build Trusted with an open technologies

ject driven by a core team of

SGX-LKL: Linux Binaries in SGX Enclaves

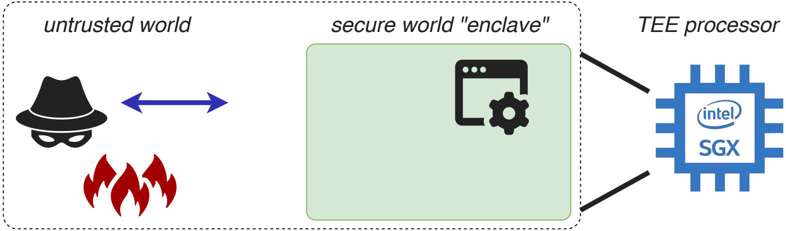
ENCLAVE DEVELOPMENT PLATFORM

The Fortanix EDP is the preferred way for writing Intel® SGX applications from scratch.

System software for trusted execution?

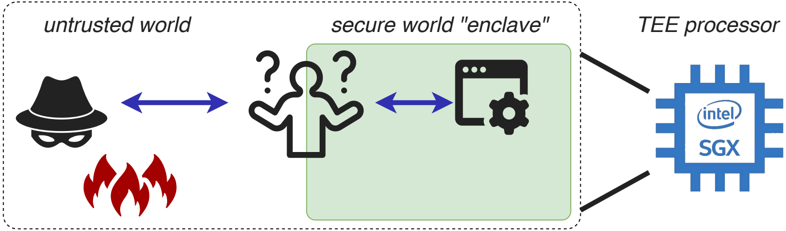
What do these projects have in common?

Why isolation is not enough: Enclave shielding runtimes



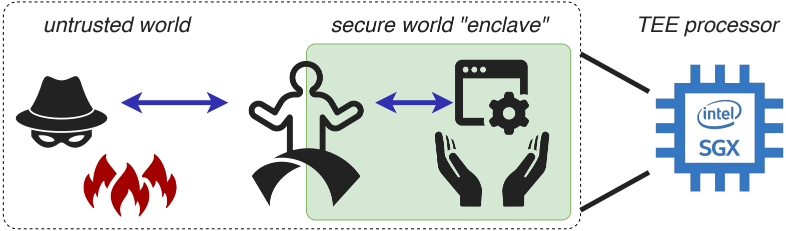
- TEE promise: enclave == “secure oasis” in a **hostile environment**

Why isolation is not enough: Enclave shielding runtimes



- TEE promise: enclave == "secure oasis" in a **hostile environment**
- . . . but application and compilers largely unaware of **isolation boundaries**

Why isolation is not enough: Enclave shielding runtimes



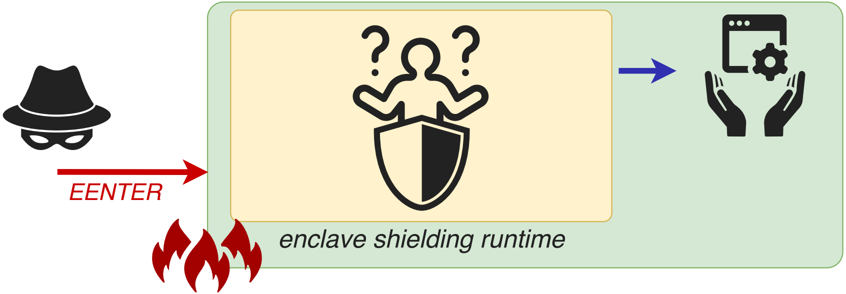
- TEE promise: enclave == "secure oasis" in a **hostile environment**
- . . . but application and compilers largely unaware of **isolation boundaries**

Shielding runtime == secure bridge on enclave entry/exit

. . . but what if the bridge itself is flawed?

The big picture: Enclave shielding responsibilities

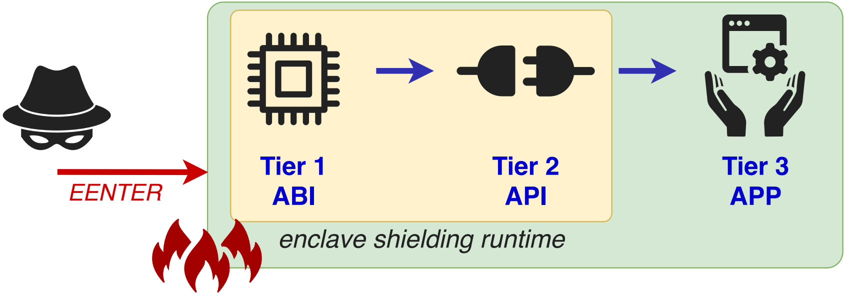
🔔 **Key questions:** how to securely bootstrap from the untrusted world to the enclaved application binary (and back)? Which sanitizations to apply?



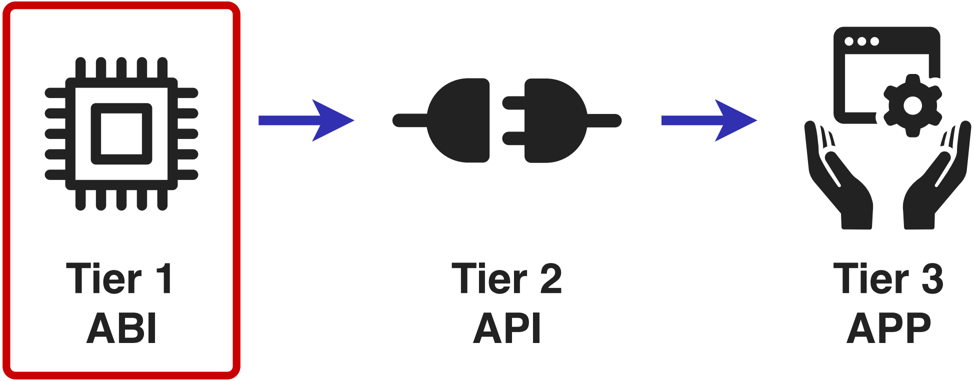
Van Bulck et al. "A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes", CCS 2019.

The big picture: Enclave shielding responsibilities

🔔 **Key insight:** split sanitization responsibilities across the ABI and API tiers: *machine state* vs. higher-level *programming language interface*



Van Bulck et al. “A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes”, CCS 2019.



Tier 1
ABI

Tier 2
API

Tier 3
APP

Tier1: Establishing a trustworthy enclave ABI

⇝ Attacker controls CPU register contents on enclave entry/exit

↔ Compiler expects well-behaved **calling convention** (e.g., stack)

Tier1: Establishing a trustworthy enclave ABI

⇝ Attacker controls CPU register contents on enclave entry/exit

⟷ Compiler expects well-behaved **calling convention** (e.g., stack)

⟹ Need to **initialize CPU registers** on entry and **scrub** before exit!

Tier1: Establishing a trustworthy enclave ABI

⇝ Attacker controls CPU register contents on enclave entry/exit

↔ Compiler expects well-behaved **calling convention** (e.g., stack)

⇒ Need to **initialize CPU registers** on entry and **scrub** before exit!

Non-trivial for x86 ISA!

SHELDON COOPER
presents
FUN WITH FLAGS
REC
GENETICS
COSMOLOGY
COMETS

x86 string instructions: Direction Flag (DF) operation

- x86 rep string instructions to speed up streamed memory operations

```c
/* memset(buf, 0x0, 100) */
for (int i=0; i < 100; i++)
  buf[i] = 0x0;
```

→

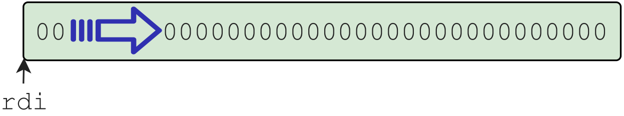
```asm
lea rdi, buf
mov al, 0x0
mov ecx, 100
rep stos [rdi], al
```

x86 string instructions: Direction Flag (DF) operation

- x86 rep string instructions to speed up streamed memory operations
- Default operate **left-to-right**

```c
/* memset(buf, 0x0, 100) */
for (int i=0; i < 100; i++)
  buf[i] = 0x0;
```

```asm
lea rdi, buf
mov al, 0x0
mov ecx, 100
rep stos [rdi], al
```



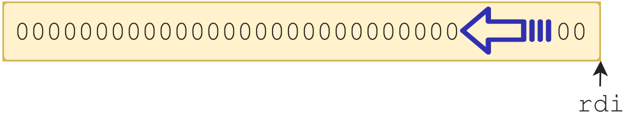
x86 string instructions: Direction Flag (DF) operation

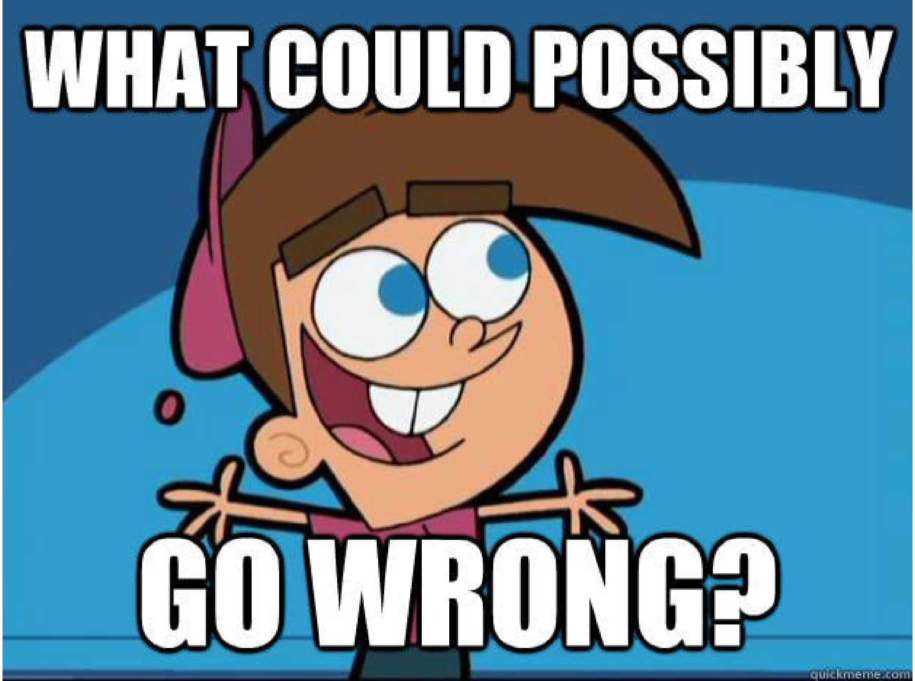
- x86 rep string instructions to speed up streamed memory operations
- Default operate **left-to-right**, unless software sets *RFLAGS.DF=1*

```c
/* memset(buf, 0x0, 100) */
for (int i=0; i < 100; i++)
  buf[i] = 0x0;
```

→

```asm
lea rdi, buf+100
mov al, 0x0
mov ecx, 100
std ; set direction flag
rep stos [rdi], al
```



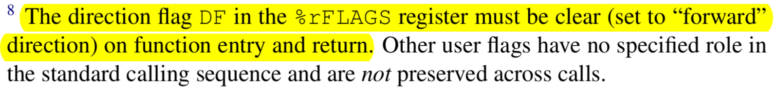

WHAT COULD POSSIBLY
GO WRONG?
quickmeme.com

Inverting enclaved string memory operations

CVE-2019-14566

x86 System-V ABI

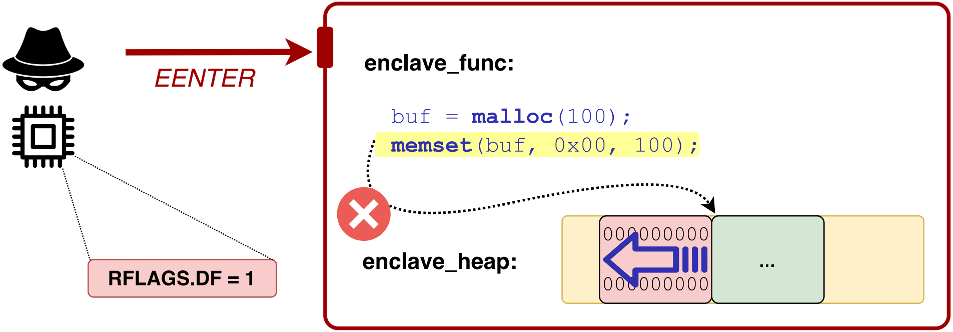
😮



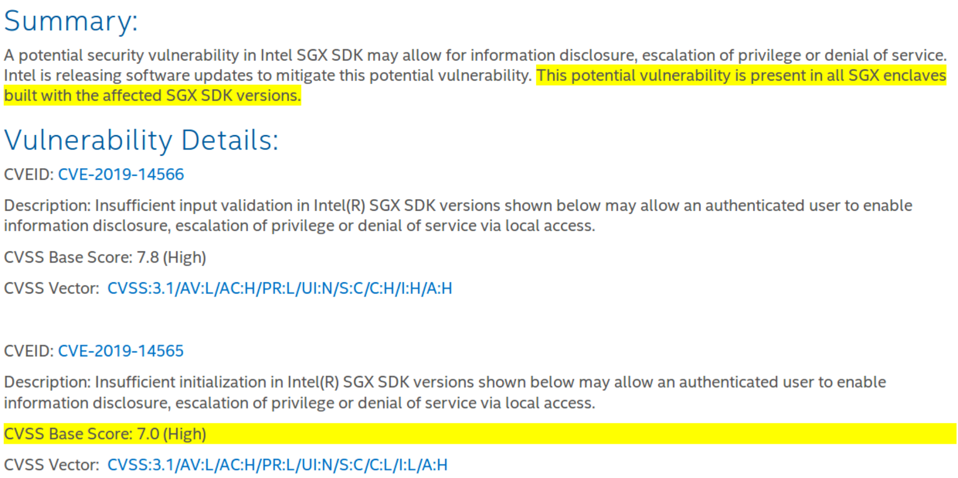
[8] The direction flag DF in the %rFLAGS register must be clear (set to "forward" direction) on function entry and return. Other user flags have no specified role in the standard calling sequence and are *not* preserved across calls.

Inverting enclaved string memory operations CVE-2019-14566

⚠ Enclave heap **memory corruption:** right-to-left. . .



```
buf = malloc(100);
memset(buf, 0x00, 100);
```



Summary:

A potential security vulnerability in Intel SGX SDK may allow for information disclosure, escalation of privilege or denial of service. Intel is releasing software updates to mitigate this potential vulnerability. This potential vulnerability is present in all SGX enclaves built with the affected SGX SDK versions.

Vulnerability Details:

CVEID: CVE-2019-14566

Description: Insufficient input validation in Intel(R) SGX SDK versions shown below may allow an authenticated user to enable information disclosure, escalation of privilege or denial of service via local access.

CVSS Base Score: 7.8 (High)

CVSS Vector: CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:C/C:H/I:H/A:H

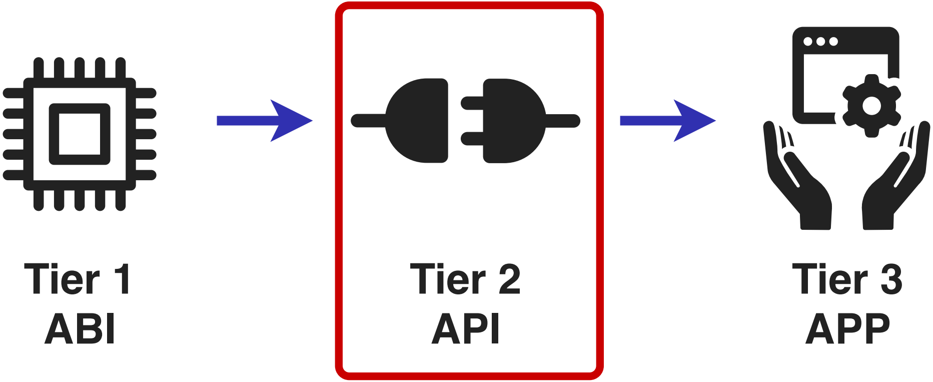
CVEID: CVE-2019-14565

Description: Insufficient initialization in Intel(R) SGX SDK versions shown below may allow an authenticated user to enable information disclosure, escalation of privilege or denial of service via local access.

CVSS Base Score: 7.0 (High)

CVSS Vector: CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:C/C:L/I:L/A:H

https://github.com/jovanbulck/0xbadc0de/ https://github.com/fritzalder/faulty-point-unit/

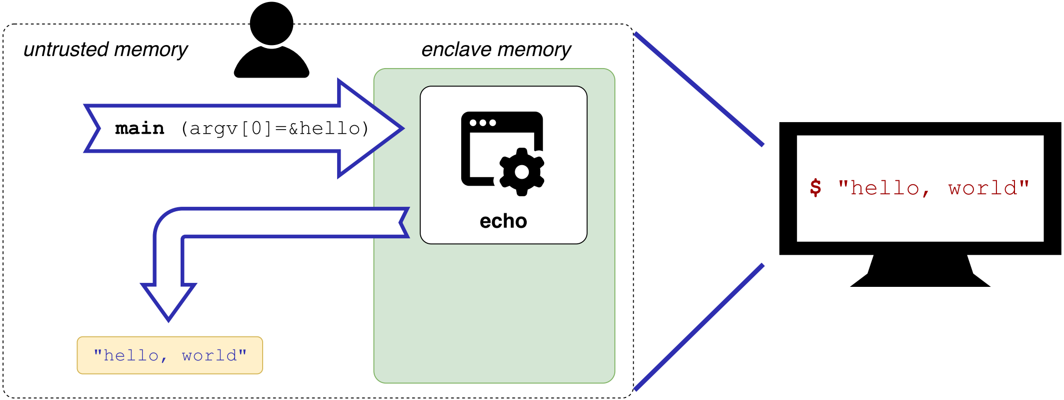


Tier 1
ABI

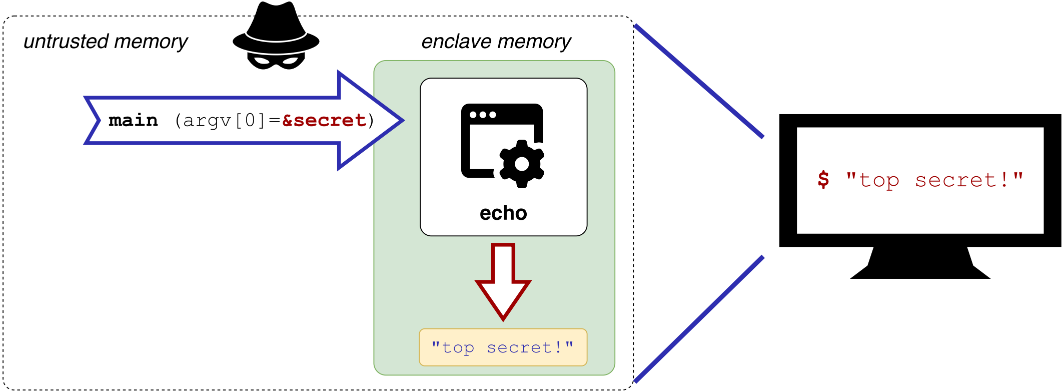
Tier 2
API

Tier 3
APP

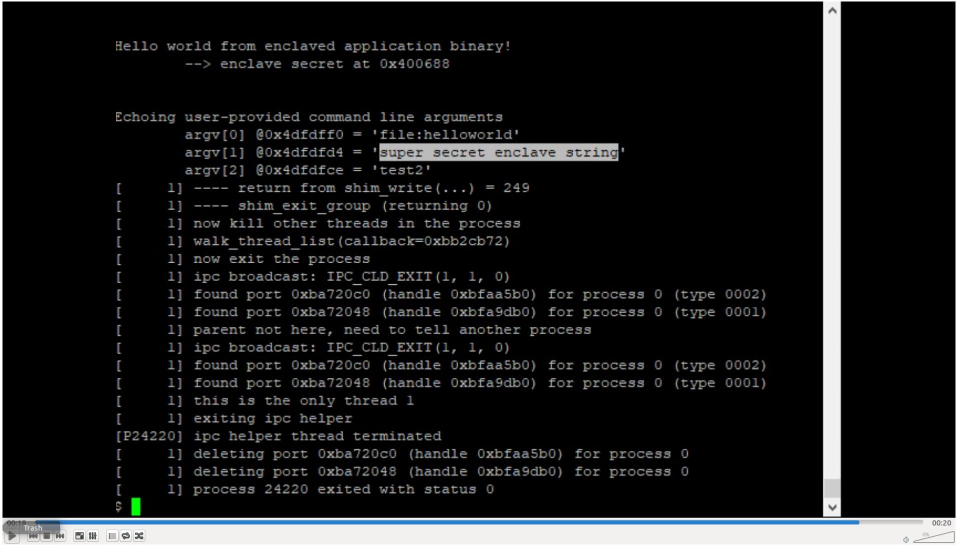
Validating pointer arguments: Confused-deputy attacks



Validating pointer arguments: Confused-deputy attacks



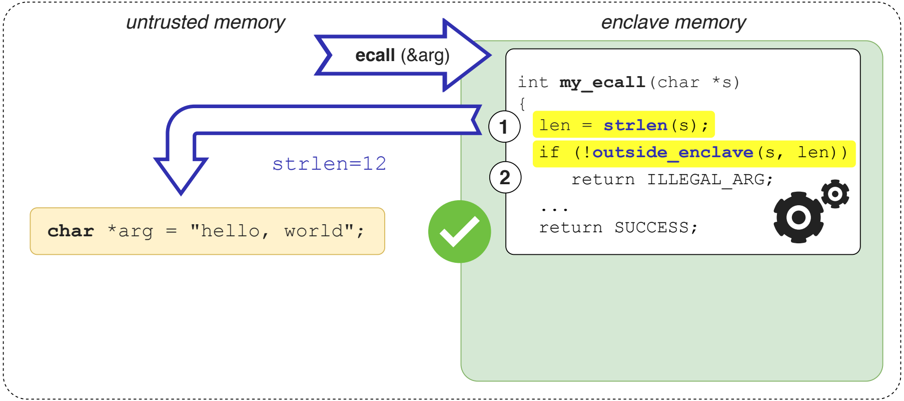
Validating pointer arguments: Confused-deputy attacks



```
Hello world from enclaved application binary!
        --> enclave secret at 0x400688


Echoing user-provided command line arguments
        argv[0] @0x4dfdff0 = 'file:helloworld'
        argv[1] @0x4dfdfd4 = 'super secret enclave string'
        argv[2] @0x4dfdfce = 'test2'
[      1] ---- return from shim_write(...) = 249
[      1] ---- shim_exit_group (returning 0)
[      1] now kill other threads in the process
[      1] walk_thread_list(callback=0xbb2cb72)
[      1] now exit the process
[      1] ipc broadcast: IPC_CLD_EXIT(1, 1, 0)
[      1] found port 0xba720c0 (handle 0xbfaa5b0) for process 0 (type 0002)
[      1] found port 0xba72048 (handle 0xbfa9db0) for process 0 (type 0001)
[      1] parent not here, need to tell another process
[      1] ipc broadcast: IPC_CLD_EXIT(1, 1, 0)
[      1] found port 0xba720c0 (handle 0xbfaa5b0) for process 0 (type 0002)
[      1] found port 0xba72048 (handle 0xbfa9db0) for process 0 (type 0001)
[      1] this is the only thread 1
[      1] exiting ipc helper
[P24220] ipc helper thread terminated
[      1] deleting port 0xba720c0 (handle 0xbfaa5b0) for process 0
[      1] deleting port 0xba72048 (handle 0xbfa9db0) for process 0
[      1] process 24220 exited with status 0
$ 
```

Intel SGX-SDK: Null-terminated strings are hard. . . CVE-2018-3626

🔔 **Idea:** 2-stage approach ensures string arguments fall *entirely* outside enclave

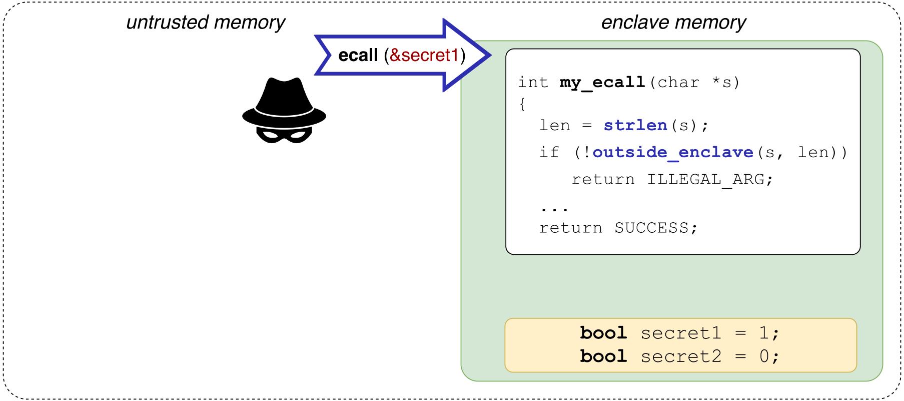


```
char *arg = "hello, world";
```

```
int my_ecall(char *s)
{
  len = strlen(s);
  if (!outside_enclave(s, len))
     return ILLEGAL_ARG;
  ...
  return SUCCESS;
```

Intel SGX-SDK: Null-terminated strings are hard. . . CVE-2018-3626

✗ . . . **but** what if we try passing an illegal, in-enclave pointer anyway?

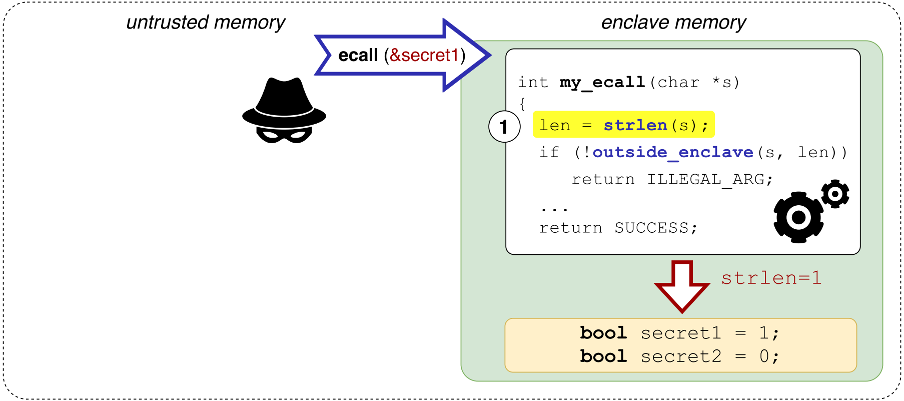


```
int my_ecall(char *s)
{
  len = strlen(s);
  if (!outside_enclave(s, len))
     return ILLEGAL_ARG;
  ...
  return SUCCESS;
```

```
bool secret1 = 1;
bool secret2 = 0;
```

Intel SGX-SDK: Null-terminated strings are hard. . . CVE-2018-3626

Enclave **first** computes length of secret, in-enclave buffer!

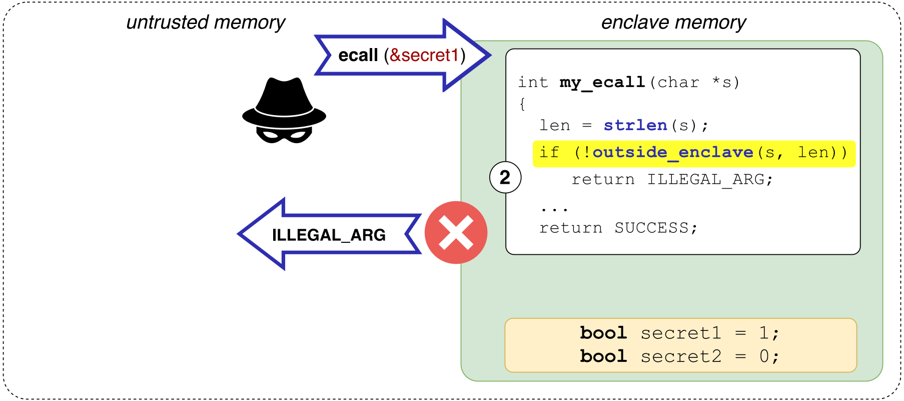


```
int my_ecall(char *s)
{
  len = strlen(s);
  if (!outside_enclave(s, len))
     return ILLEGAL_ARG;
  ...
  return SUCCESS;
```

```
bool secret1 = 1;
bool secret2 = 0;
```

Intel SGX-SDK: Null-terminated strings are hard. . . CVE-2018-3626

. . . and only **afterwards verifies** whether *entire string* falls outside enclave

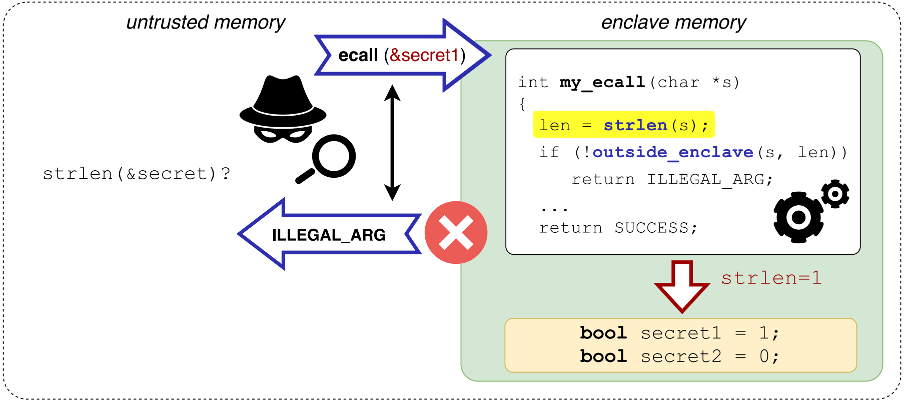


```
int my_ecall(char *s)
{
  len = strlen(s);
  if (!outside_enclave(s, len))
     return ILLEGAL_ARG;
  ...
  return SUCCESS;
```

```
bool secret1 = 1;
bool secret2 = 0;
```

Intel SGX-SDK: Null-terminated strings are hard. . . CVE-2018-3626

Idea: `strlen()` timing as a side-channel oracle for in-enclave null bytes ☺

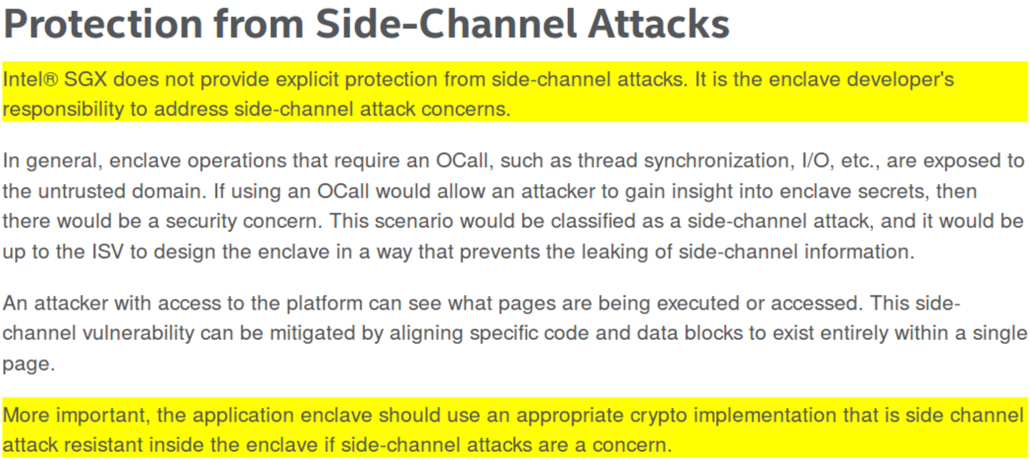


```
int my_ecall(char *s)
{
  len = strlen(s);
  if (!outside_enclave(s, len))
     return ILLEGAL_ARG;
  ...
  return SUCCESS;
```

```
bool secret1 = 1;
bool secret2 = 0;
```

Attack idea 2: Side-channel analysis

A note on side-channel attacks (Intel)



Protection from Side-Channel Attacks

Intel® SGX does not provide explicit protection from side-channel attacks. It is the enclave developer's responsibility to address side-channel attack concerns.

In general, enclave operations that require an OCall, such as thread synchronization, I/O, etc., are exposed to the untrusted domain. If using an OCall would allow an attacker to gain insight into enclave secrets, then there would be a security concern. This scenario would be classified as a side-channel attack, and it would be up to the ISV to design the enclave in a way that prevents the leaking of side-channel information.

An attacker with access to the platform can see what pages are being executed or accessed. This side-channel vulnerability can be mitigated by aligning specific code and data blocks to exist entirely within a single page.

More important, the application enclave should use an appropriate crypto implementation that is side channel attack resistant inside the enclave if side-channel attacks are a concern.

`https://software.intel.com/en-us/node/703016`

Vulnerable patterns: Secret-dependent code/data accesses

```
void secret_vote(char candidate)
{
    if (candidate == 'a')
        vote_candidate_a();
    else
        vote_candidate_b();
}
```

```
int secret_lookup(int s)
{
    if (s > 0 && s < ARRAY_LEN)
        return array[s];
    return -1;

}
```

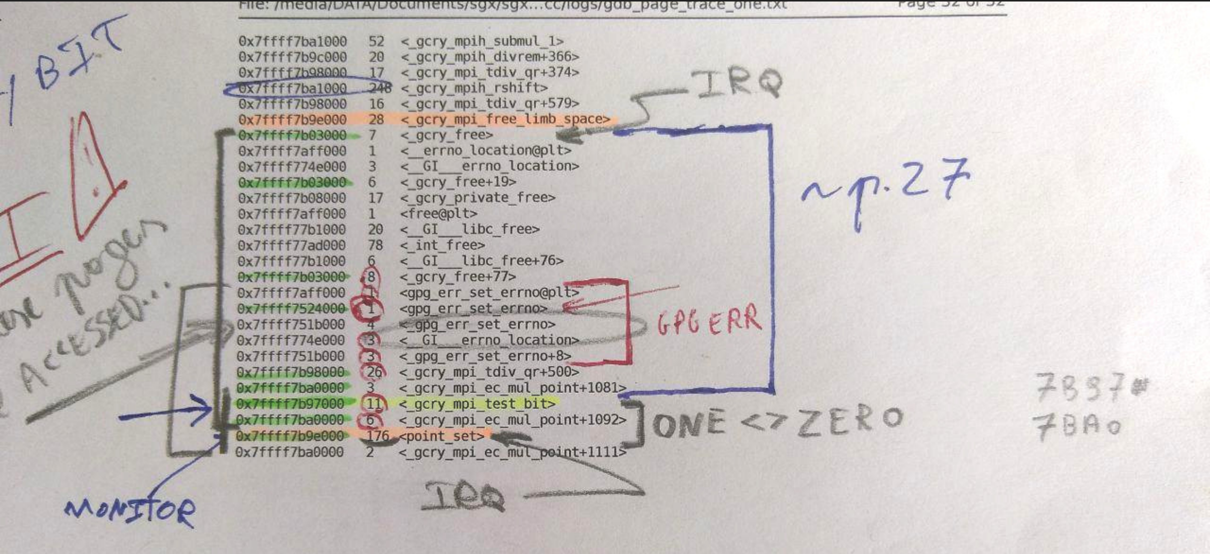
Vulnerable patterns: Secret-dependent code/data accesses

```c
void secret_vote(char candidate)
{
    if (candidate == 'a')
        vote_candidate_a();
    else
        vote_candidate_b();
}
```

```c
int secret_lookup(int s)
{
    if (s > 0 && s < ARRAY_LEN)
        return array[s];
    return -1;

}
```

What are new ways for privileged adversaries to create an “oracle” for enclave code+data memory accesses?

Side-channel analysis: From metadata patterns to secrets

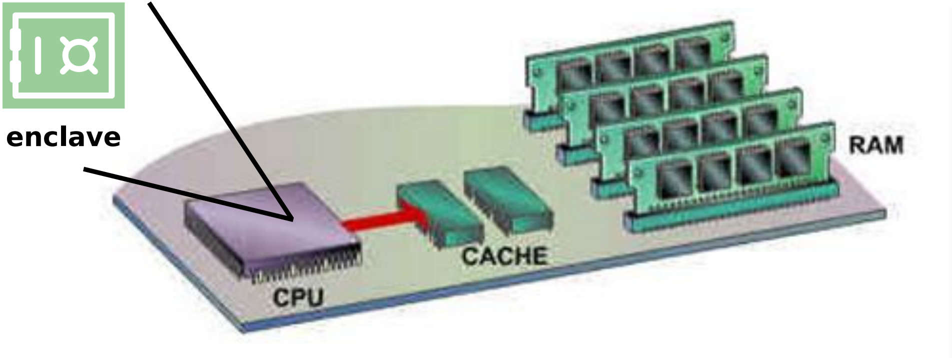


File: /media/DATA/Documents/sgx/sgx...cc/logs/gdb_page_trace_one.txt   Page 32 of 32

```
0x7ffff7ba1000   52   <_gcry_mpih_submul_1>
0x7ffff7b9c000   20   <_gcry_mpih_divrem+366>
0x7ffff7b98000   17   <_gcry_mpi_tdiv_qr+374>
0x7ffff7ba1000   248  <_gcry_mpih_rshift>
0x7ffff7b98000   16   <_gcry_mpi_tdiv_qr+579>
0x7ffff7b9e000   28   <_gcry_mpi_free_limb_space>
0x7ffff7b03000   7    <_gcry_free>
0x7ffff7aff000   1    <__errno_location@plt>
0x7ffff774e000   3    <__GI___errno_location>
0x7ffff7b03000   6    <_gcry_free+19>
0x7ffff7b08000   17   <_gcry_private_free>
0x7ffff7aff000   1    <free@plt>
0x7ffff77b1000   20   <__GI___libc_free>
0x7ffff77ad000   78   <_int_free>
0x7ffff77b1000   6    <__GI___libc_free+76>
0x7ffff7b03000   8    <_gcry_free+77>
0x7ffff7aff000   1    <gpg_err_set_errno@plt>
0x7ffff7524000   1    <gpg_err_set_errno>
0x7ffff751b000   4    <_gpg_err_set_errno>
0x7ffff774e000   3    <__GI___errno_location>
0x7ffff751b000   3    <_gpg_err_set_errno+8>
0x7ffff7b98000   26   <_gcry_mpi_tdiv_qr+500>
0x7ffff7ba0000   3    <_gcry_mpi_ec_mul_point+1081>
0x7ffff7b97000   11   <_gcry_mpi_test_bit>
0x7ffff7ba0000   6    <_gcry_mpi_ec_mul_point+1092>
0x7ffff7b9e000   176  <point_set>
0x7ffff7ba0000   2    <_gcry_mpi_ec_mul_point+1111>
```

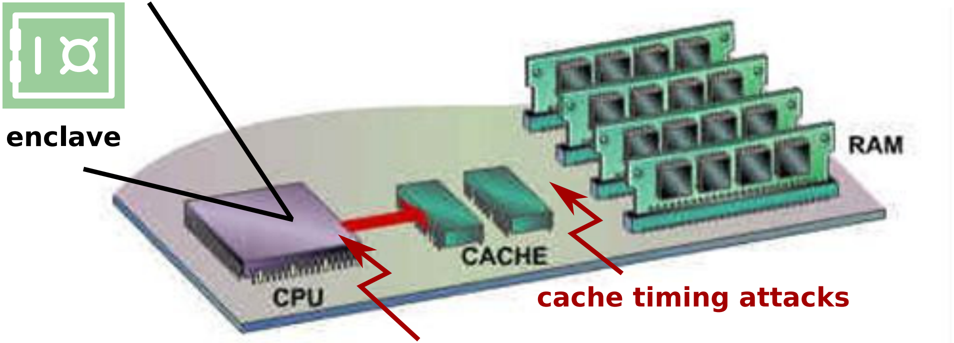
Van Bulck et al. “Telling Your Secrets Without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution”, USENIX 2017.

Overview: Spying on enclave memory accesses



enclave

Overview: Spying on enclave memory accesses

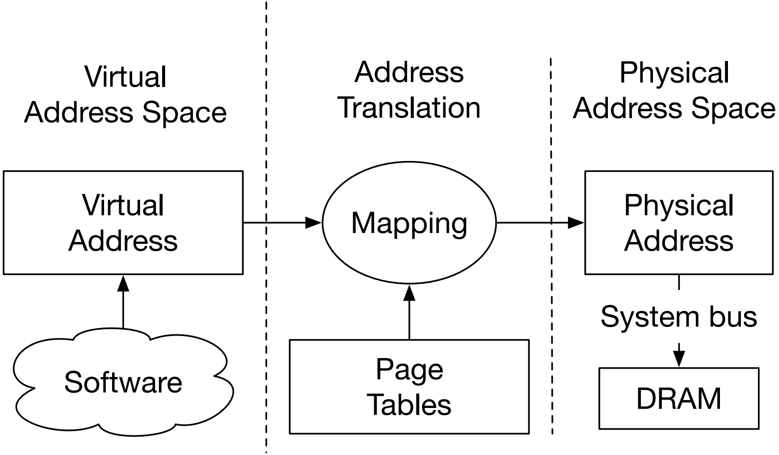


enclave

cache timing attacks

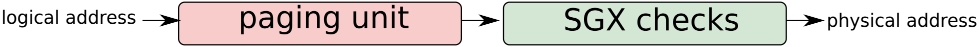
address translation attacks

The virtual memory abstraction



Costan et al. "Intel SGX explained", IACR 2016

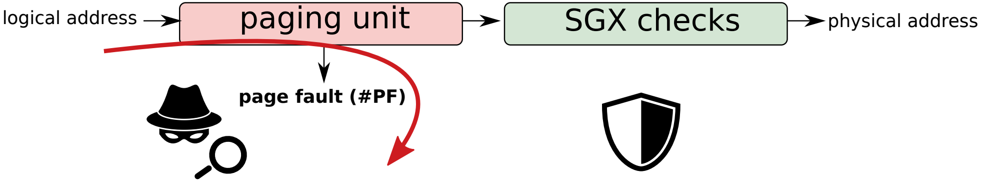
Intel SGX: Page faults as a side channel



logical address → paging unit → SGX checks → physical address

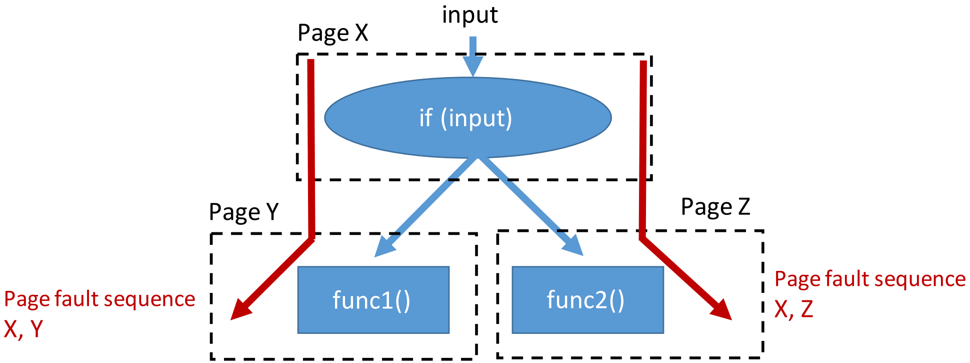
SGX machinery protects against direct address remapping attacks

Intel SGX: Page faults as a side channel



. . . but untrusted address translation may **fault(!)**

Intel SGX: Page faults as a side channel

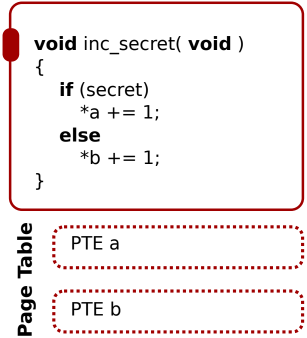


Xu et al.: “Controlled-channel attacks: Deterministic side channels for untrusted operating systems”, Oakland 2015

⇒ Page fault traces leak **private control data/flow**

An end-to-end example

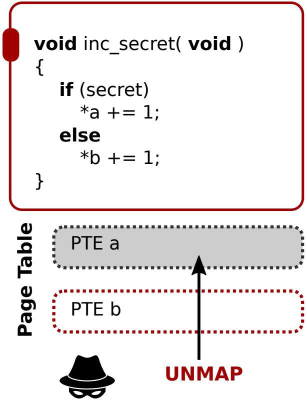
https://github.com/jovanbulck/sgx-tutorial-space18



```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

An end-to-end example

1. Revoke access rights on *unprotected* enclave page table entry

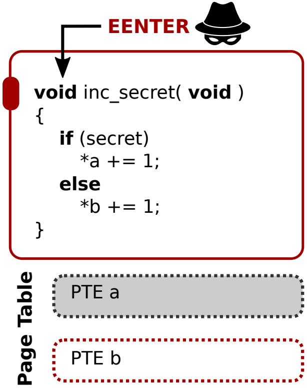


```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

An end-to-end example

1. Revoke access rights on *unprotected* enclave page table entry

2. Enter victim enclave

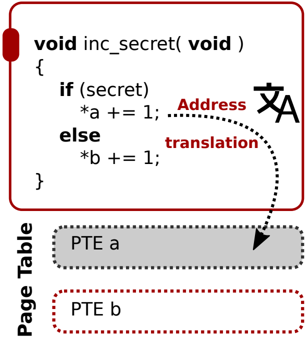


```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

An end-to-end example

https://github.com/jovanbulck/sgx-tutorial-space18

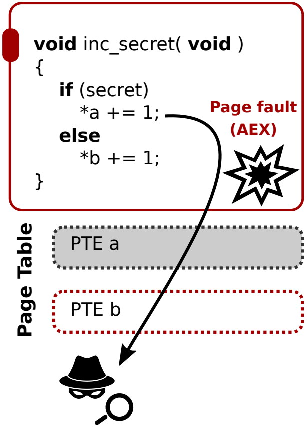
1. Revoke access rights on *unprotected* enclave page table entry
2. Enter victim enclave
3. Secret-dependent data memory access
   - ⇝ CPU performs virt-to-phys address translation!
   - ⇝ By reading page table entry setup by *untrusted* OS



```
void inc_secret( void )
{
    if (secret)
      *a += 1;
    else
      *b += 1;
}
```

An end-to-end example

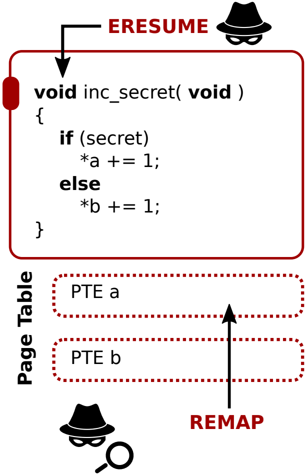
1. Revoke access rights on *unprotected* enclave page table entry
2. Enter victim enclave
3. Secret-dependent data memory access
   - ⇝ CPU performs virt-to-phys address translation!
   - ⇝ By reading page table entry setup by *untrusted* OS
4. Virtual address not present → raise page fault
   - ⇝ CPU exits enclave and vectors to untrusted OS



```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

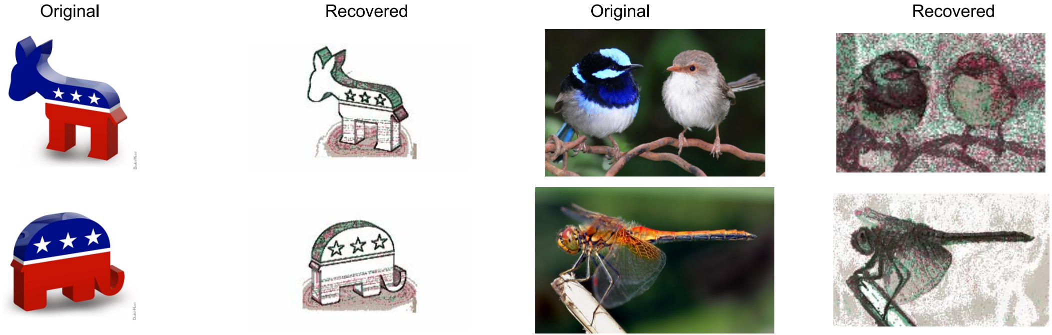
An end-to-end example

1. Revoke access rights on *unprotected* enclave page table entry
2. Enter victim enclave
3. Secret-dependent data memory access
   - ⤳ CPU performs virt-to-phys address translation!
   - ⤳ By reading page table entry setup by *untrusted* OS
4. Virtual address not present → raise page fault
   - ⤳ CPU exits enclave and vectors to untrusted OS
5. Restore access rights and resume victim enclave



```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

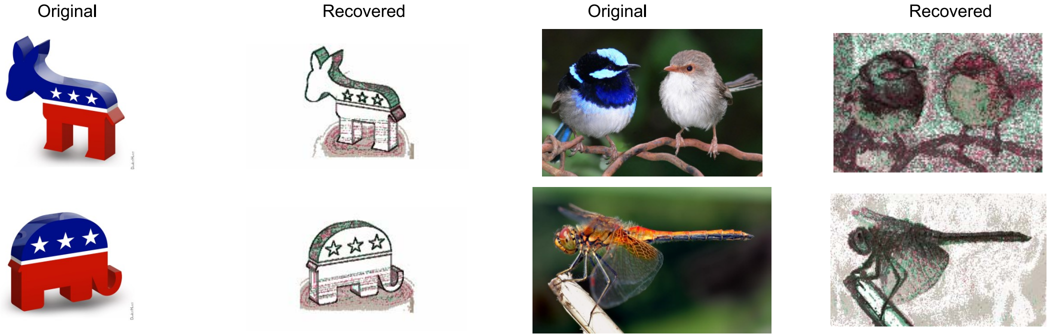
Page table-based attacks in practice



Xu et al.: “Controlled-channel attacks: Deterministic side channels for untrusted operating systems”, Oakland 2015

⟹ **Low-noise, single-run** exploitation of legacy applications

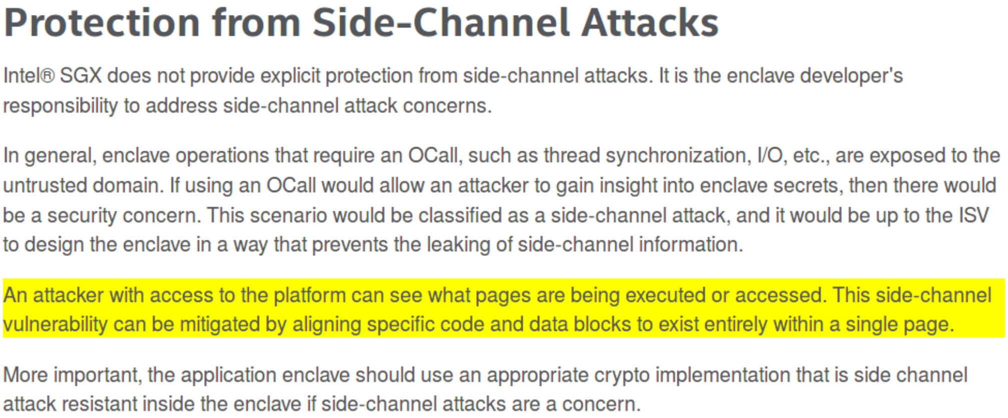
Page table-based attacks in practice



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015

. . . but at a relative coarse-grained **4 KiB granularity**

Intel’s note on side-channel attacks (revisited)



Protection from Side-Channel Attacks

Intel® SGX does not provide explicit protection from side-channel attacks. It is the enclave developer's responsibility to address side-channel attack concerns.

In general, enclave operations that require an OCall, such as thread synchronization, I/O, etc., are exposed to the untrusted domain. If using an OCall would allow an attacker to gain insight into enclave secrets, then there would be a security concern. This scenario would be classified as a side-channel attack, and it would be up to the ISV to design the enclave in a way that prevents the leaking of side-channel information.

An attacker with access to the platform can see what pages are being executed or accessed. This side-channel vulnerability can be mitigated by aligning specific code and data blocks to exist entirely within a single page.

More important, the application enclave should use an appropriate crypto implementation that is side channel attack resistant inside the enclave if side-channel attacks are a concern.

https://software.intel.com/en-us/node/703016

Temporal resolution limitations for the page fault oracle

```c
size_t strlen (char *str)
{
  char *s;

  for (s = str; *s; ++s);
  return (s - str);
}
```

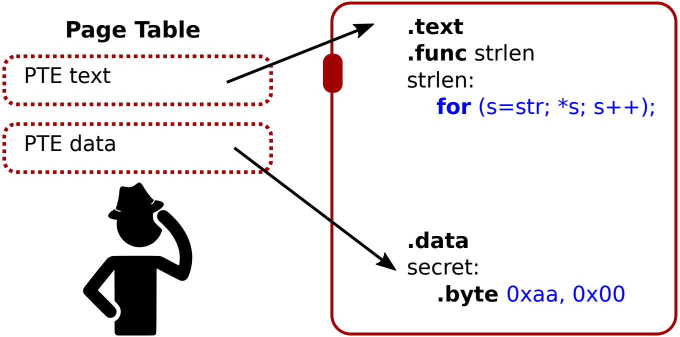
```
     mov   %rdi,%rax
1:   cmpb  $0x0,(%rax)
     je    2f
     inc   %rax
     jmp   1b
2:   sub   %rdi,%rax
     retq
```

⇒ tight loop: 4 instructions, single memory operand, single code + data page

Counting `strlen` loop iterations?

Note: page-fault attacks cannot make progress for 1 code + data page

Temporal resolution limitations for the page fault oracle

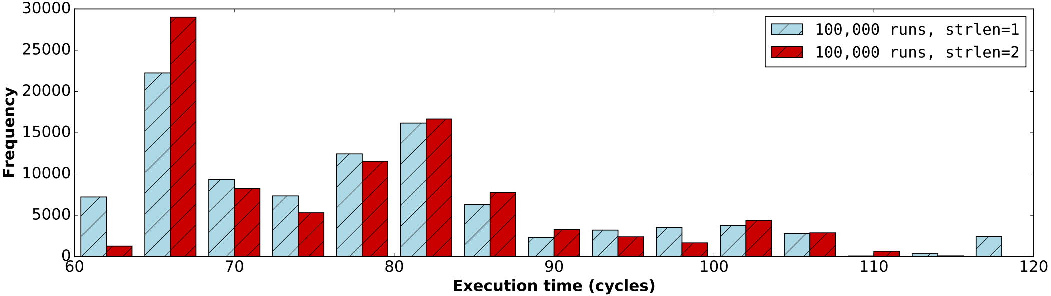


```
.text
.func strlen
strlen:
    for (s=str; *s; s++);

.data
secret:
    .byte 0xaa, 0x00
```

Counting `strlen` loop iterations?

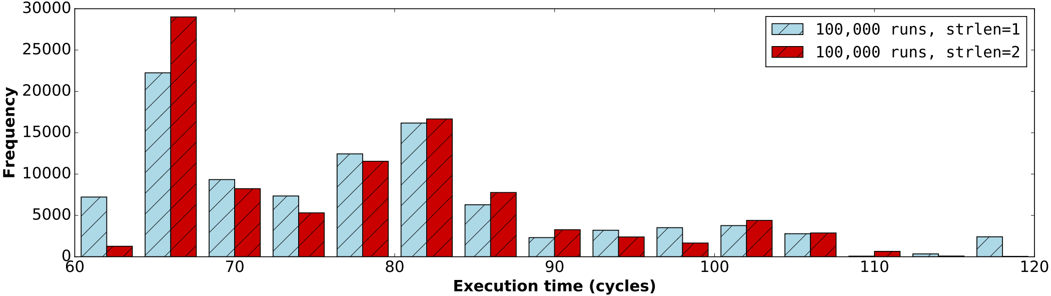
Progress requires both pages present (non-faulting) ↔ page fault oracle

Building the `strlen()` side-channel oracle with execution timing?

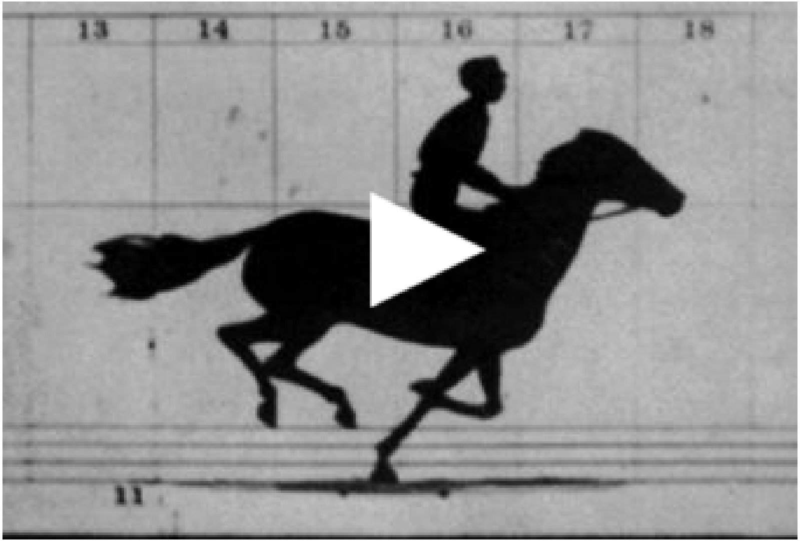


Building the `strlen()` side-channel oracle with execution timing?

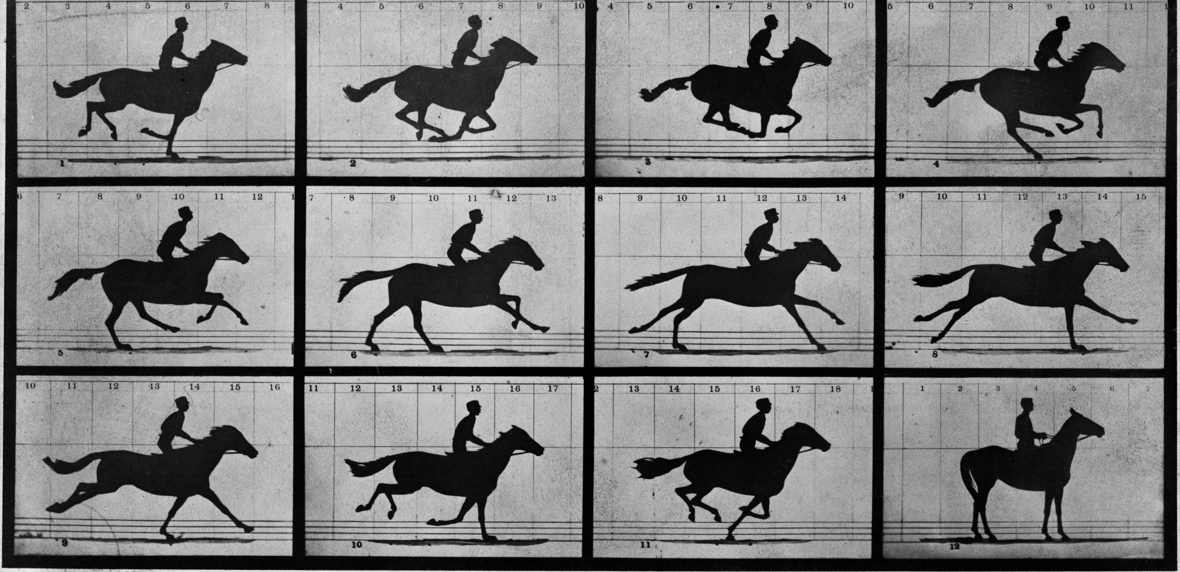
Too noisy: modern x86 processors are lightning fast. . .



Analogy: Studying galloping horse dynamics



https://en.wikipedia.org/wiki/Sallie_Gardner_at_a_Gallop



MORSE'S Gallery, 417 Montgomery St., San Francisco.

2432 Kw

THE HORSE IN MOTION.

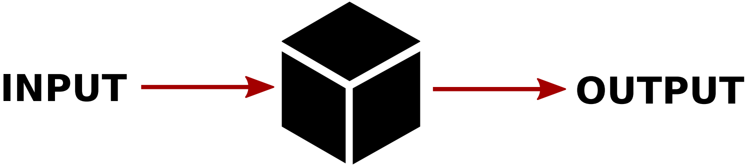
Illustrated by
MUYBRIDGE.

AUTOMATIC ELECTRO-PHOTOGRAPH.

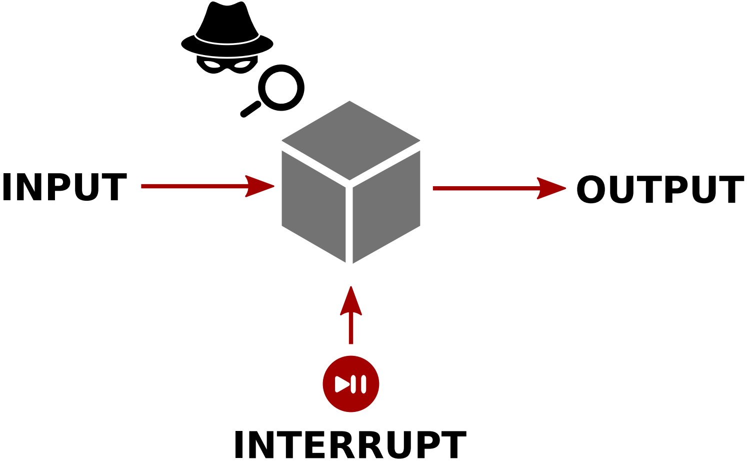
"SALLIE GARDNER," owned by LELAND STANFORD; running at a 1.40 gait over the Palo Alto track, 19th June, 1878.

The negatives of these photographs were made at intervals of twenty-seven inches of distance, and about the twenty-fifth part of a second of time; they illustrate consecutive positions

SGX-Step: Executing enclaves one instruction at a time



SGX-Step: Executing enclaves one instruction at a time

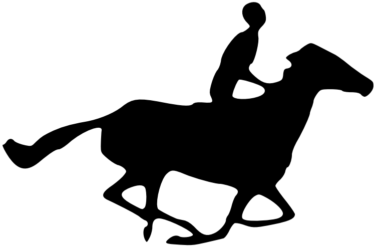


INPUT

OUTPUT

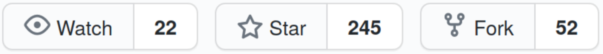
INTERRUPT

SGX-Step: Executing enclaves one instruction at a time



SGX-Step

https://github.com/jovanbulck/sgx-step



Building a precise single-stepping primitive

🔔 **SGX-Step goal:** Executing enclaves one instruction at a time

Challenge: we need a very precise timer interrupt:

- ☹ x86 hardware *debug features* disabled in enclave mode
- ☺ . . . but we have *root access!*

Building a precise single-stepping primitive

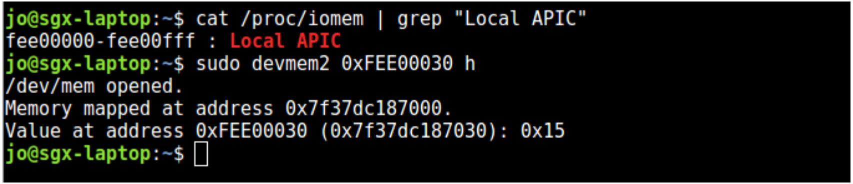
🔔 **SGX-Step goal:** Executing enclaves one instruction at a time

Challenge: we need a very precise timer interrupt:

☹ x86 hardware *debug features* disabled in enclave mode

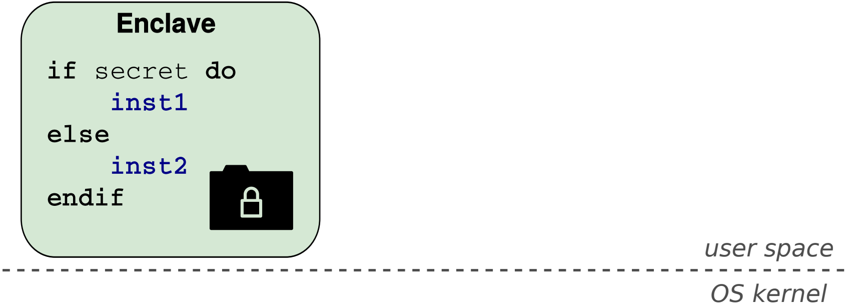
☺ . . . but we have *root access!*

⇒ Setup user-space virtual **memory mappings** for x86 APIC (+ PTEs)



```
jo@sgx-laptop:~$ cat /proc/iomem | grep "Local APIC"
fee00000-fee00fff : Local APIC
jo@sgx-laptop:~$ sudo devmem2 0xFEE00030 h
/dev/mem opened.
Memory mapped at address 0x7f37dc187000.
Value at address 0xFEE00030 (0x7f37dc187030): 0x15
jo@sgx-laptop:~$ 
```

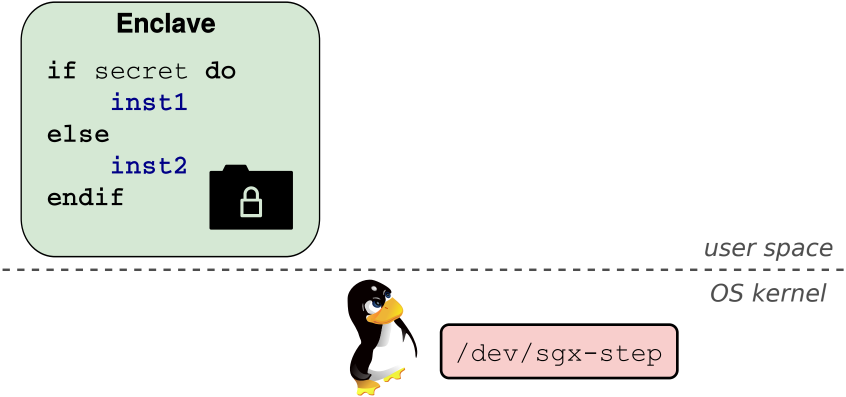
SGX-Step: Executing enclaves one instruction at a time



Enclave

```
if secret do
     inst1
else
     inst2
endif
```

SGX-Step: Executing enclaves one instruction at a time



Enclave

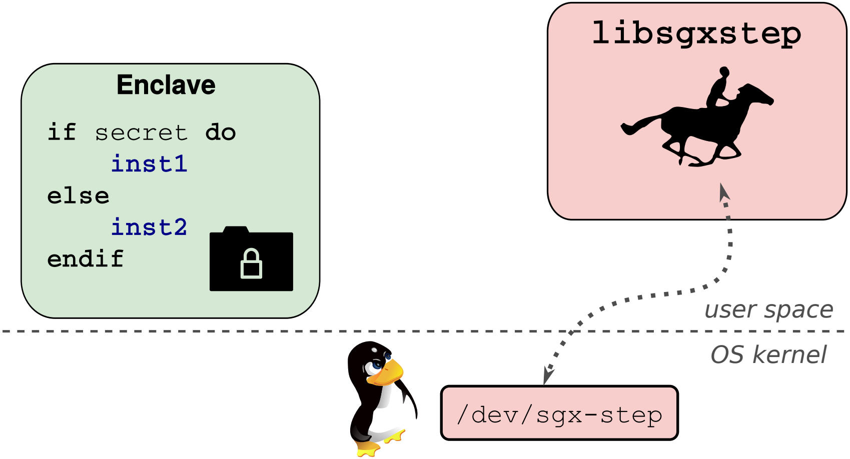
```
if secret do
     inst1
else
     inst2
endif
```

user space

OS kernel

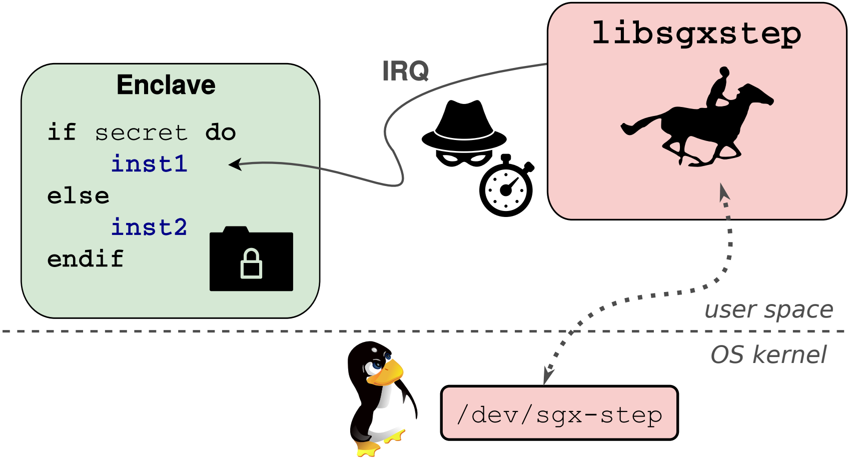
`/dev/sgx-step`

SGX-Step: Executing enclaves one instruction at a time



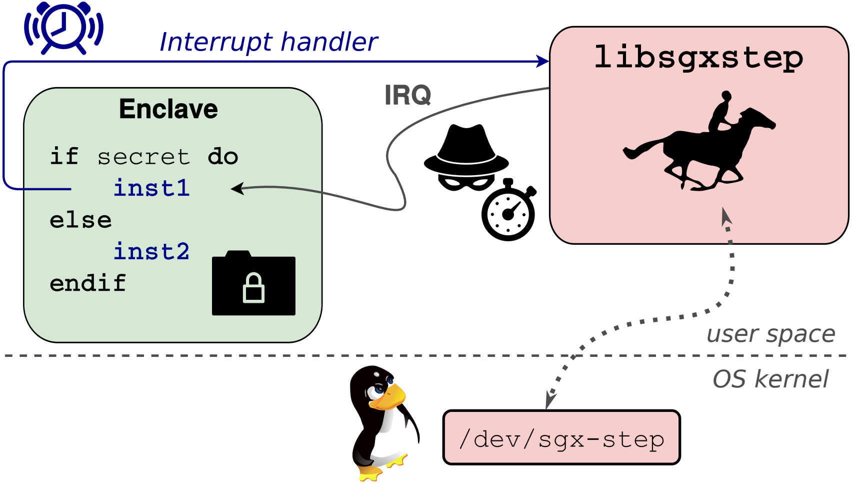
```
if secret do
      inst1
else
      inst2
endif
```

SGX-Step: Executing enclaves one instruction at a time



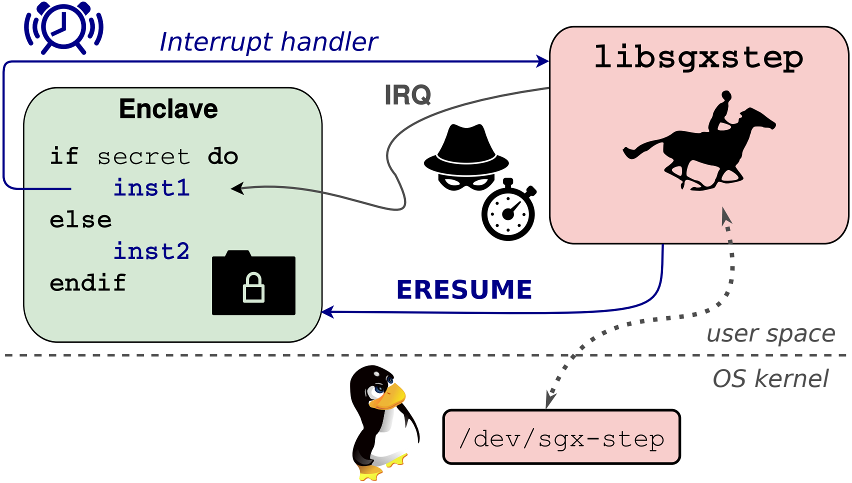
```
if secret do
      inst1
else
      inst2
endif
```

SGX-Step: Executing enclaves one instruction at a time



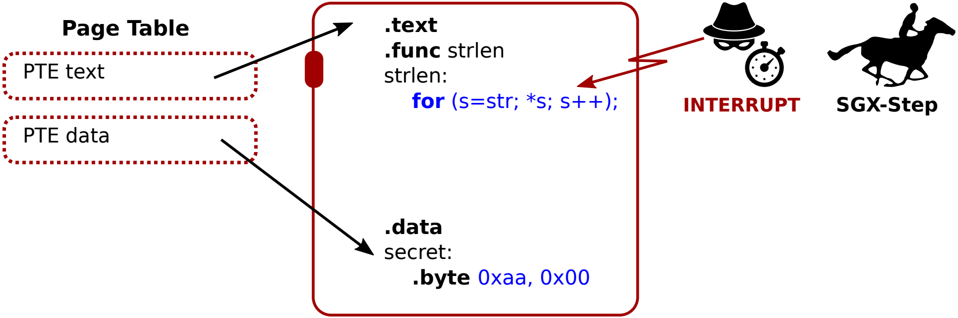
```
if secret do
      inst1
else
      inst2
endif
```

SGX-Step: Executing enclaves one instruction at a time



Building a deterministic `strlen()` null byte oracle with SGX-Step

Execute *exactly* one enclave instruction → **timer interrupt**



Page Table

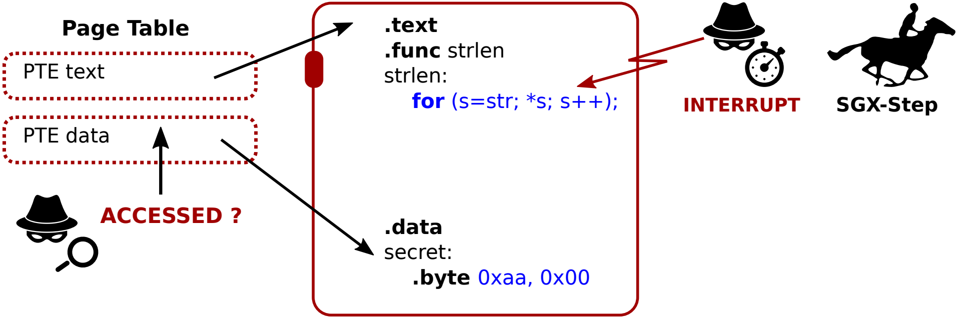
PTE text

PTE data

```
.text
.func strlen
strlen:
    for (s=str; *s; s++);

.data
secret:
    .byte 0xaa, 0x00
```

Building a deterministic `strlen()` null byte oracle with SGX-Step

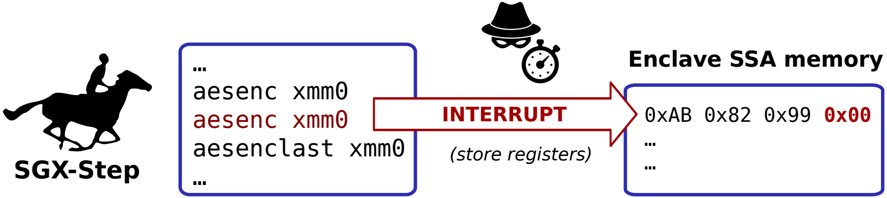
🔔 Page table accessed bit set? → **strlen++** → resume



```
.text
.func strlen
strlen:
    for (s=str; *s; s++);

.data
secret:
    .byte 0xaa, 0x00
```

CVE-2018-3626: ALL YOUR ZERO BYTES
ARE BELONG TO US
imgflip.com

SGX-Step: Breaking AES-NI with the strlen() null byte oracle



SGX-Step

```
…
aesenc xmm0
aesenc xmm0
aesenclast xmm0
…
```

INTERRUPT

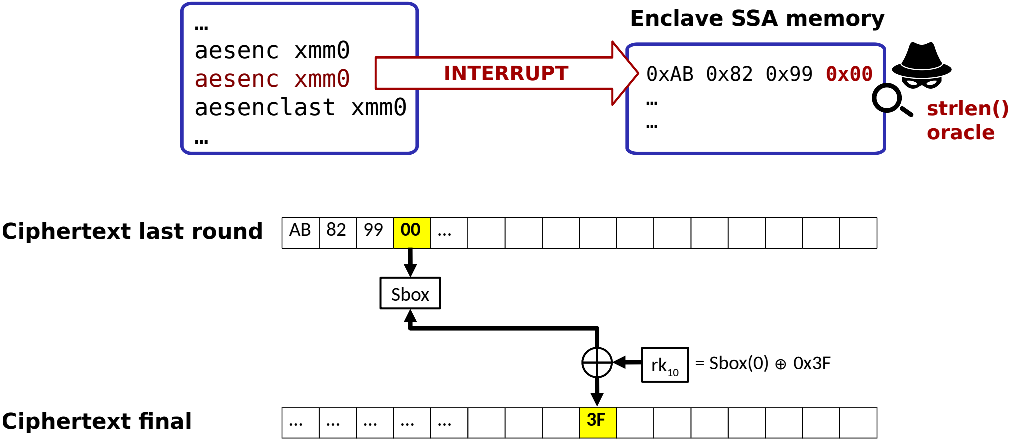
(store registers)

Enclave SSA memory

```
0xAB 0x82 0x99 0x00
…
…
```

Van Bulck et al. “A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes”, CCS 2019.

SGX-Step: Breaking AES-NI with the `strlen()` null byte oracle



```
…
aesenc xmm0
aesenc xmm0
aesenclast xmm0
…
```

Ciphertext last round

AB	82	99	00	...											

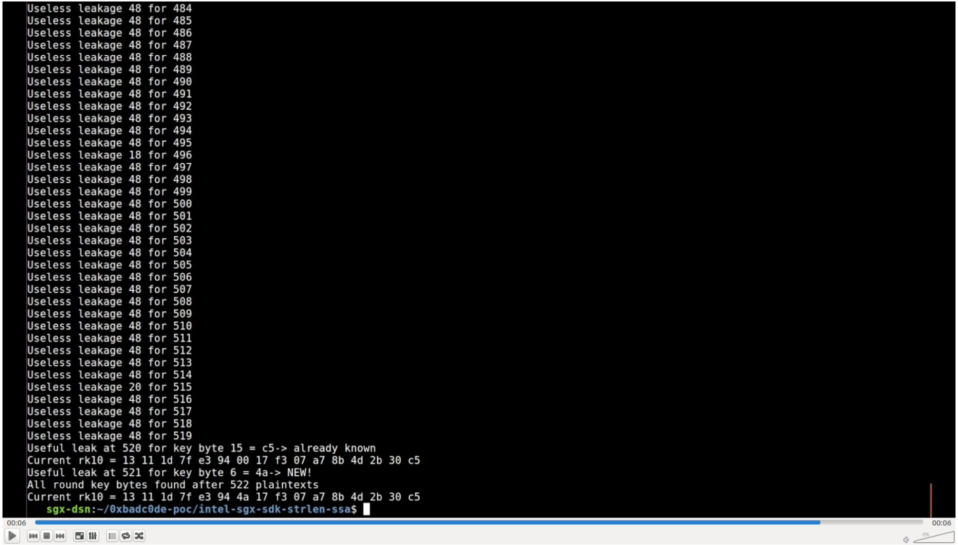
rk_{10} = Sbox(0) ⊕ 0x3F

Ciphertext final

...	...	...	...	...				3F							

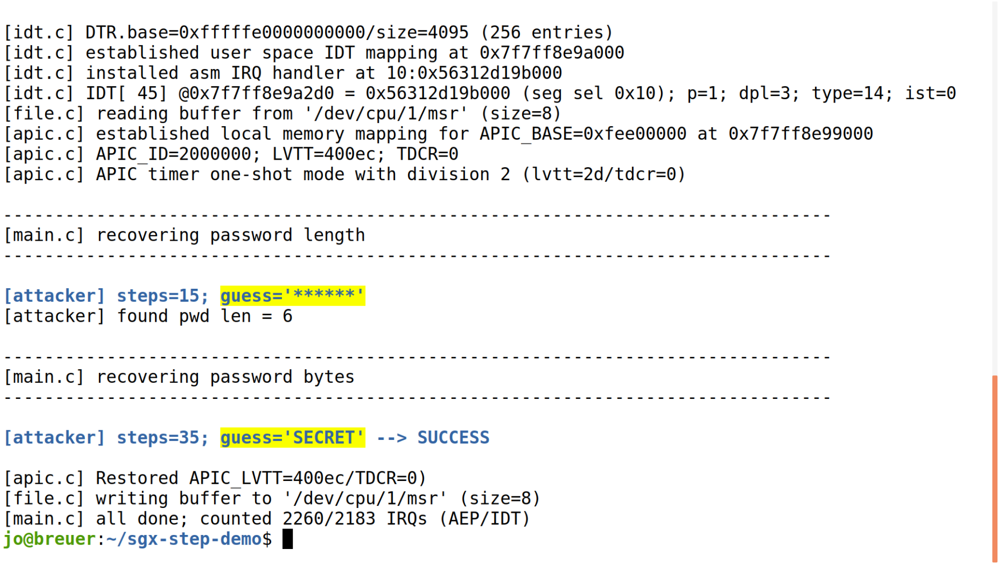
Van Bulck et al. “A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes”, CCS 2019.

SGX-Step: Breaking AES-NI with the `strlen()` null byte oracle



```
Useless leakage 48 for 484
Useless leakage 48 for 485
Useless leakage 48 for 486
Useless leakage 48 for 487
Useless leakage 48 for 488
Useless leakage 48 for 489
Useless leakage 48 for 490
Useless leakage 48 for 491
Useless leakage 48 for 492
Useless leakage 48 for 493
Useless leakage 48 for 494
Useless leakage 48 for 495
Useless leakage 18 for 496
Useless leakage 48 for 497
Useless leakage 48 for 498
Useless leakage 48 for 499
Useless leakage 48 for 500
Useless leakage 48 for 501
Useless leakage 48 for 502
Useless leakage 48 for 503
Useless leakage 48 for 504
Useless leakage 48 for 505
Useless leakage 48 for 506
Useless leakage 48 for 507
Useless leakage 48 for 508
Useless leakage 48 for 509
Useless leakage 48 for 510
Useless leakage 48 for 511
Useless leakage 48 for 512
Useless leakage 48 for 513
Useless leakage 48 for 514
Useless leakage 20 for 515
Useless leakage 48 for 516
Useless leakage 48 for 517
Useless leakage 48 for 518
Useless leakage 48 for 519
Useful leak at 520 for key byte 15 = c5-> already known
Current rk10 = 13 11 1d 7f e3 94 00 17 f3 07 a7 8b 4d 2b 30 c5
Useful leak at 521 for key byte 6 = 4a-> NEW!
All round key bytes found after 522 plaintexts
Current rk10 = 13 11 1d 7f e3 94 4a 17 f3 07 a7 8b 4d 2b 30 c5
  sgx-dsn:~/0xbadc0de-poc/intel-sgx-sdk-strlen-ssa$
```

SGX-Step: Building a deterministic `memcmp()` password oracle



```
[idt.c] DTR.base=0xfffffe0000000000/size=4095 (256 entries)
[idt.c] established user space IDT mapping at 0x7f7ff8e9a000
[idt.c] installed asm IRQ handler at 10:0x56312d19b000
[idt.c] IDT[ 45] @0x7f7ff8e9a2d0 = 0x56312d19b000 (seg sel 0x10); p=1; dpl=3; type=14; ist=0
[file.c] reading buffer from '/dev/cpu/1/msr' (size=8)
[apic.c] established local memory mapping for APIC_BASE=0xfee00000 at 0x7f7ff8e99000
[apic.c] APIC_ID=2000000; LVTT=400ec; TDCR=0
[apic.c] APIC timer one-shot mode with division 2 (lvtt=2d/tdcr=0)

--------------------------------------------------------------------------------
[main.c] recovering password length
--------------------------------------------------------------------------------

[attacker] steps=15; guess='******'
[attacker] found pwd len = 6

--------------------------------------------------------------------------------
[main.c] recovering password bytes
--------------------------------------------------------------------------------

[attacker] steps=35; guess='SECRET' --> SUCCESS

[apic.c] Restored APIC_LVTT=400ec/TDCR=0)
[file.c] writing buffer to '/dev/cpu/1/msr' (size=8)
[main.c] all done; counted 2260/2183 IRQs (AEP/IDT)
jo@breuer:~/sgx-step-demo$ 
```

SGX-Step: Enabling a new line of high-precision enclave attacks

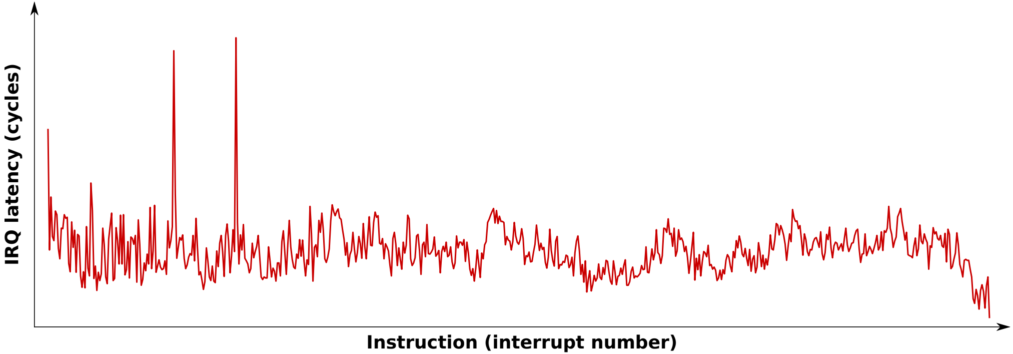
			APIC		PTE			Desc		
Yr	Attack	Temporal resolution	IRQ	IPI	#PF	A/D	PPN	GDT	IDT	Drv
'15	Ctrl channel	~ Page	○	○	●	○	○	○	●	✓ ⊞
'16	AsyncShock	~ Page	○	○	●	○	○	○	○	– 🐧
'17	CacheZoom	✗ > 1	●	○	○	○	○	○	○	✓ 🐧
'17	Hahnel et al.	✗ 0 - > 1	●	○	○	○	○	○	●	✓ ⊞
'17	BranchShadow	✗ 5 - 50	●	○	○	○	○	○	○	✗ 🐧
'17	Stealthy PTE	~ Page	○	●	○	●	○	○	●	✓ 🐧
'17	DarkROP	~ Page	○	○	●	○	○	○	○	✓ 🐧
'17	SGX-Step	✓ 0 - 1	●	○	●	●	○	○	○	✓ 🏇
'18	Off-limits	✓ 0 - 1	●	○	●	○	○	●	○	✓ 🏇
'18	Single-trace RSA	~ Page	○	○	●	○	○	○	○	✓ 🏇
'18	Foreshadow	✓ 0 - 1	●	○	●	○	●	○	○	✓ 🏇
'18	SgxPectre	~ Page	○	○	●	○	○	○	○	✓ 🐧
'18	CacheQuote	✗ > 1	●	○	○	○	○	○	○	✓ 🐧
'18	SGXlinger	✗ > 1	●	○	○	○	○	○	○	✗ 🐧
'18	Nemesis	✓ 1	●	○	●	●	○	○	●	✓ 🏇

			APIC		PTE			Desc		
Yr	Attack	Temporal resolution	IRQ	IPI	#PF	A/D	PPN	GDT	IDT	Drv
'19	Spoiler	✓ 1	●	○	○	●	○	○	●	✓ 🏇
'19	ZombieLoad	✓ 0 - 1	●	○	●	●	○	○	●	✓ 🏇
'19	Tale of 2 worlds	✓ 1	●	○	●	●	○	○	●	✓ 🏇
'19	MicroScope	~ 0 - Page	○	○	●	○	○	○	○	✗ 🐧
'20	Bluethunder	✓ 1	●	○	○	○	○	○	●	✓ 🏇
'20	Big troubles	~ Page	○	○	●	○	○	○	○	✓ 🏇
'20	Viral primitive	✓ 1	●	○	●	●	○	○	●	✓ 🏇
'20	CopyCat	✓ 1	●	○	●	●	○	○	●	✓ 🏇
'20	LVI	✓ 1	●	○	●	●	●	○	●	✓ 🏇
'20	A to Z	~ Page	○	○	●	○	○	○	○	✓ 🏇
'20	Frontal	✓ 1	●	○	●	●	○	○	●	✓ 🏇
'20	CrossTalk	✓ 1	●	○	●	○	○	○	●	✓ 🏇
'20	Online template	~ Page	○	○	●	○	○	○	○	✓ 🏇
'20	Déjà Vu NSS	~ Page	○	○	●	○	○	○	○	✓ 🏇

https://jovanbulck.github.io/files/phd-thesis.pdf

Nemesis: Extracting IRQ latency traces with SGX-Step

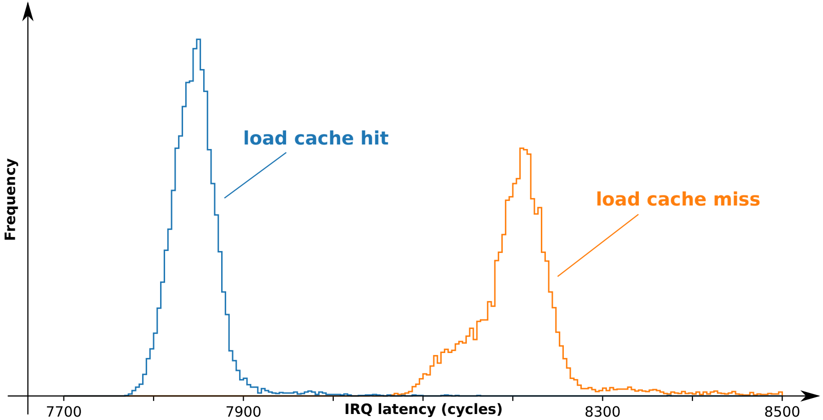
☢ **Enclave x-ray:** IRQ latency leaks instruction-level μ-arch timing!



Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018.

Nemesis microbenchmarks: Measuring x86 cache misses

☝ **Instruction timing leak:** Reconstruct *microarchitectural state*



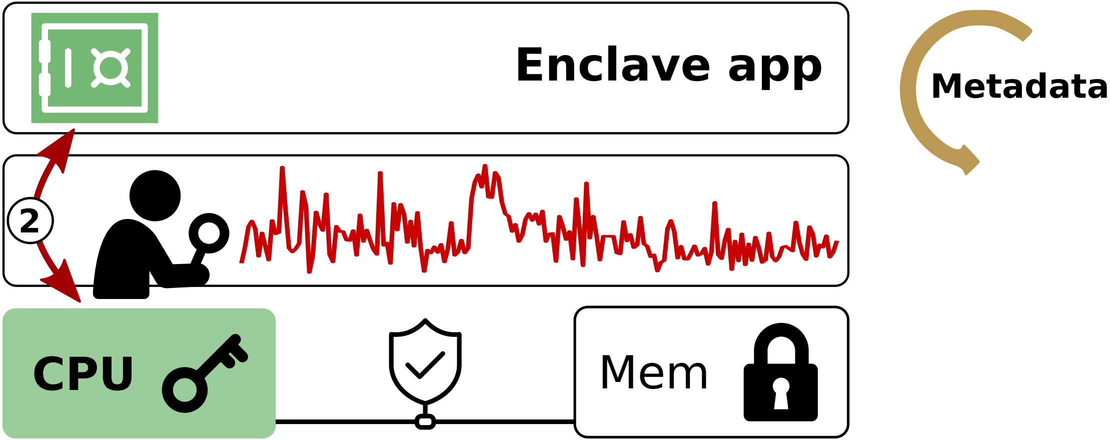
Revisited: Intel’s note on side-channel attacks (2017)

“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.”*

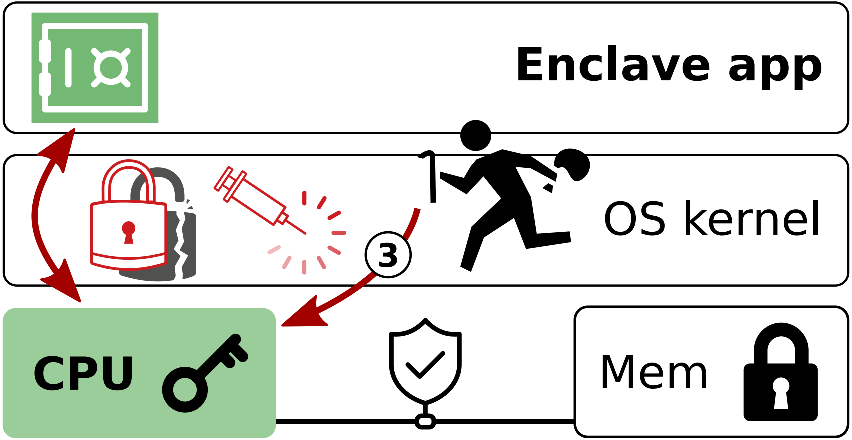
`https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sgx-and-side-channels.html`

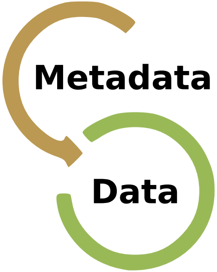
Attack idea 3: Transient execution

Outline: Privileged side-channel attacks



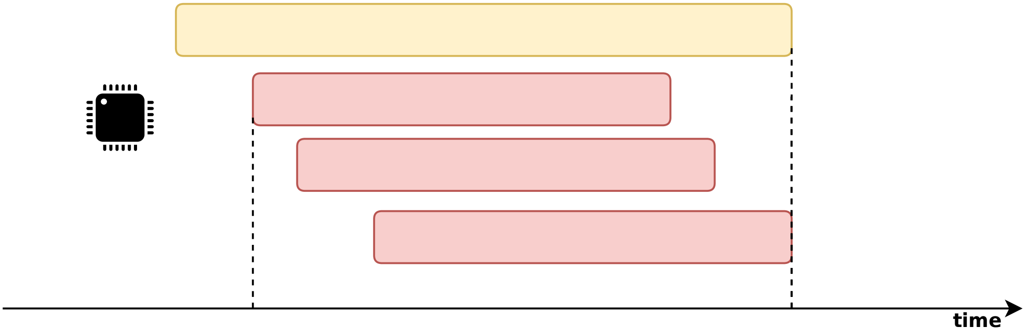
Outline: Transient-execution attacks



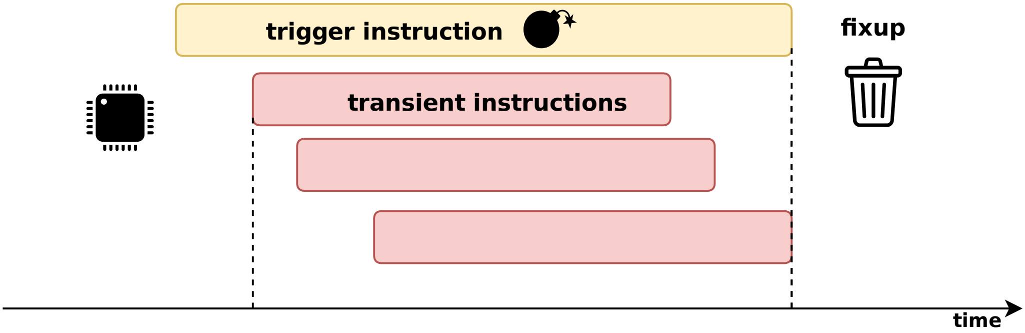


WHAT IF I TOLD YOU
YOU CAN CHANGE RULES MID-GAME

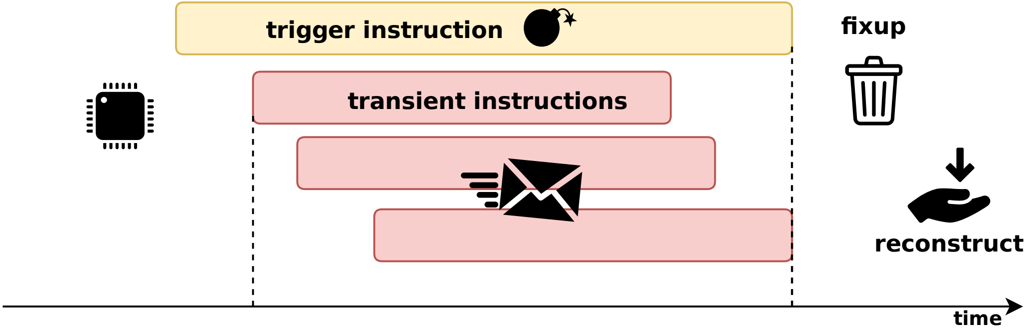
Abusing out-of-order and speculative execution



Abusing out-of-order and speculative execution



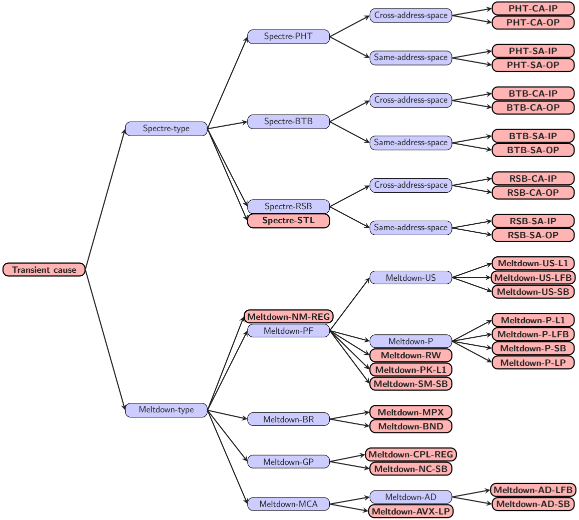
Abusing out-of-order and speculative execution



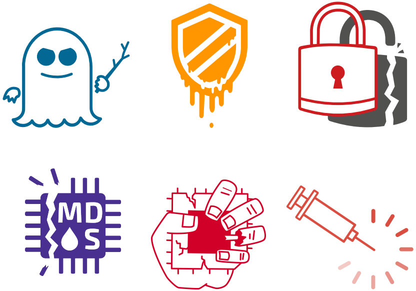
Transient-execution attacks: Welcome to the world of fun!

The transient-execution zoo

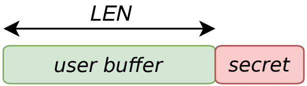
https://transient.fail



- Transient cause
  - Spectre-type
    - Spectre-PHT
      - Cross-address-space
        - PHT-CA-IP
        - PHT-CA-OP
      - Same-address-space
        - PHT-SA-IP
        - PHT-SA-OP
    - Spectre-BTB
      - Cross-address-space
        - BTB-CA-IP
        - BTB-CA-OP
      - Same-address-space
        - BTB-SA-IP
        - BTB-SA-OP
    - Spectre-RSB
      - Cross-address-space
        - RSB-CA-IP
        - RSB-CA-OP
      - Same-address-space
        - RSB-SA-IP
        - RSB-SA-OP
    - Spectre-STL
  - Meltdown-type
    - Meltdown-NM-REG
    - Meltdown-PF
      - Meltdown-US
        - Meltdown-US-L1
        - Meltdown-US-LFB
        - Meltdown-US-SB
      - Meltdown-P
        - Meltdown-P-L1
        - Meltdown-P-LFB
        - Meltdown-P-SB
        - Meltdown-P-LP
      - Meltdown-RW
      - Meltdown-PK-L1
      - Meltdown-SM-SB
    - Meltdown-BR
      - Meltdown-MPX
      - Meltdown-BND
    - Meltdown-GP
      - Meltdown-CPL-REG
      - Meltdown-NC-SB
    - Meltdown-MCA
      - Meltdown-AD
        - Meltdown-AD-LFB
        - Meltdown-AD-SB
      - Meltdown-AVX-LP



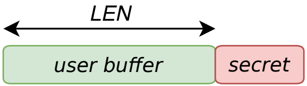
Spectre v1: Speculative buffer over-read



```
if (idx < LEN)
{
  s = buffer[idx];
  t = lookup[s];
  ...
}
```

- Programmer *intention:* no out-of-bounds accesses

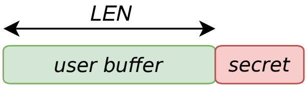
Spectre v1: Speculative buffer over-read



```
if (idx < LEN)
{
  s = buffer[idx];
  t = lookup[s];
  ...
}
```

- Programmer *intention:* no out-of-bounds accesses
- **Mistrain gadget** to speculatively "ahead of time" execute with $idx \geq LEN$ in the transient world

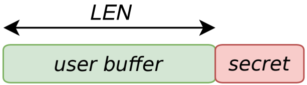
Spectre v1: Speculative buffer over-read



```
if (idx < LEN)
{
  s = buffer[idx];
  t = lookup[s];
  ...
}
```

- Programmer *intention:* no out-of-bounds accesses
- **Mistrain gadget** to speculatively “ahead of time” execute with *idx* ≥ *LEN* in the transient world
- **Side channels** may leave traces after roll-back!

Spectre v1: Speculative buffer over-read



```
if (idx < LEN)
{
  asm("lfence\n\t");
  s = buffer[idx];
  t = lookup[s];
  ...
}
```

- Programmer *intention:* no out-of-bounds accesses
- **Mistrain gadget** to speculatively "ahead of time" execute with *idx* ≥ *LEN* in the transient world
- **Side channels** may leave traces after roll-back!
- Insert explicit **speculation barriers** to tell the CPU to halt the transient world...

Spectre take-away

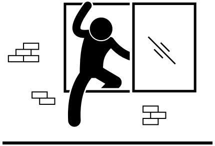
- CPU **transiently** executes wrong code paths
- **Confused-deputy gadgets** encode secrets via side channels

inside™

inside™

inside™

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

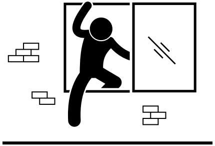
Listing 1: x86 assembly

```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  movb (%rsi), %al
  shl $0xc, %rax
  movq (%rdi, %rax), %rdi
  retq
```

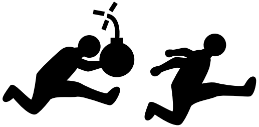
Listing 2: C code.

```
void meltdown(
        uint8_t *oracle,
        uint8_t *secret_ptr)
{
  uint8_t v = *secret_ptr;
  v = v * 0x1000;
  uint64_t o = oracle[v];
}
```

Meltdown: Transiently encoding unauthorized memory



Unauthorized access



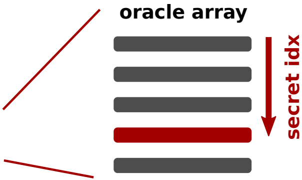
Transient out-of-order window

Listing 1: x86 assembly.

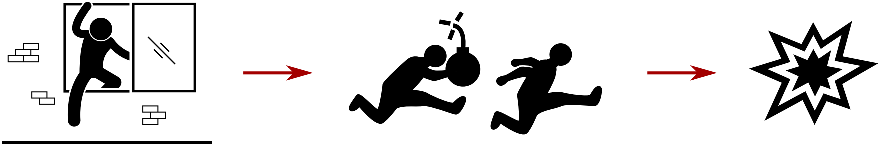
```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  movb (%rsi), %al
  shl $0xc, %rax
  movq (%rdi, %rax), %rdi
  retq
```

Listing 2: C code.

```
void meltdown(
      uint8_t *oracle,
      uint8_t *secret_ptr)
{
  uint8_t v = *secret_ptr;
  v = v * 0x1000;
  uint64_t o = oracle[v];
}
```



Meltdown: Transiently encoding unauthorized memory



Unauthorized access

Transient out-of-order window

Exception

(discard architectural state)

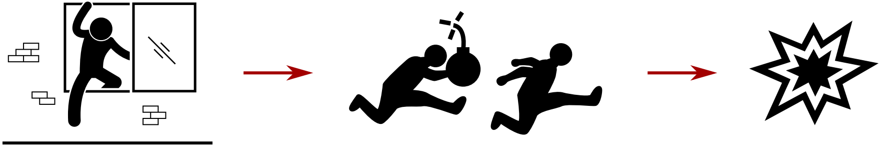
Listing 1: x86 assembly.

```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  movb (%rsi), %al
  shl $0xc, %rax
  movq (%rdi, %rax), %rdi
  retq
```

Listing 2: C code.

```
void meltdown(
      uint8_t *oracle,
      uint8_t *secret_ptr)
{
  uint8_t v = *secret_ptr;
  v = v * 0x1000;
  uint64_t o = oracle[v];
}
```

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

Transient out-of-order window

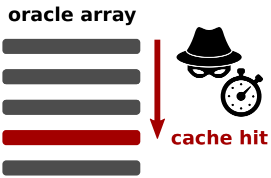
Exception handler

Listing 1: x86 assembly.

```
meltdown:
  // %rdi: oracle
  // %rsi: secret_ptr

  movb (%rsi), %al
  shl $0xc, %rax
  movq (%rdi, %rax), %rdi
  retq
```

Listing 2: C code.

```
void meltdown(
      uint8_t *oracle,
      uint8_t *secret_ptr)
{
  uint8_t v = *secret_ptr;
  v = v * 0x1000;
  uint64_t o = oracle[v];
}
```

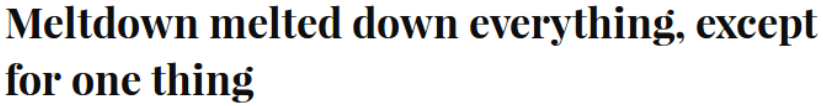


oracle array

cache hit

https://foreshadowattack.eu/

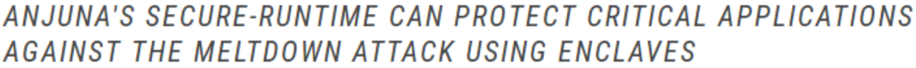
Rumors: Meltdown immunity for SGX enclaves?



Meltdown melted down everything, except for one thing

“[enclaves] remain protected and completely secure”

— *International Business Times, February 2018*

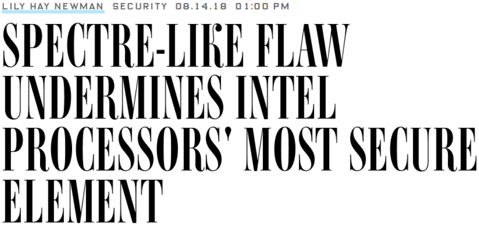


ANJUNA'S SECURE-RUNTIME CAN PROTECT CRITICAL APPLICATIONS AGAINST THE MELTDOWN ATTACK USING ENCLAVES

“[enclave memory accesses] redirected to an abort page, which has no value”

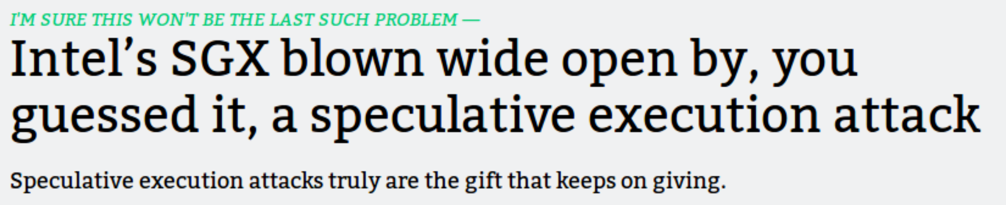
— *Anjuna Security, Inc., March 2018*

~~Rumors: Meltdown immunity for SGX enclaves?~~



LILY HAY NEWMAN SECURITY 08.14.18 01:00 PM

SPECTRE-LIKE FLAW UNDERMINES INTEL PROCESSORS' MOST SECURE ELEMENT

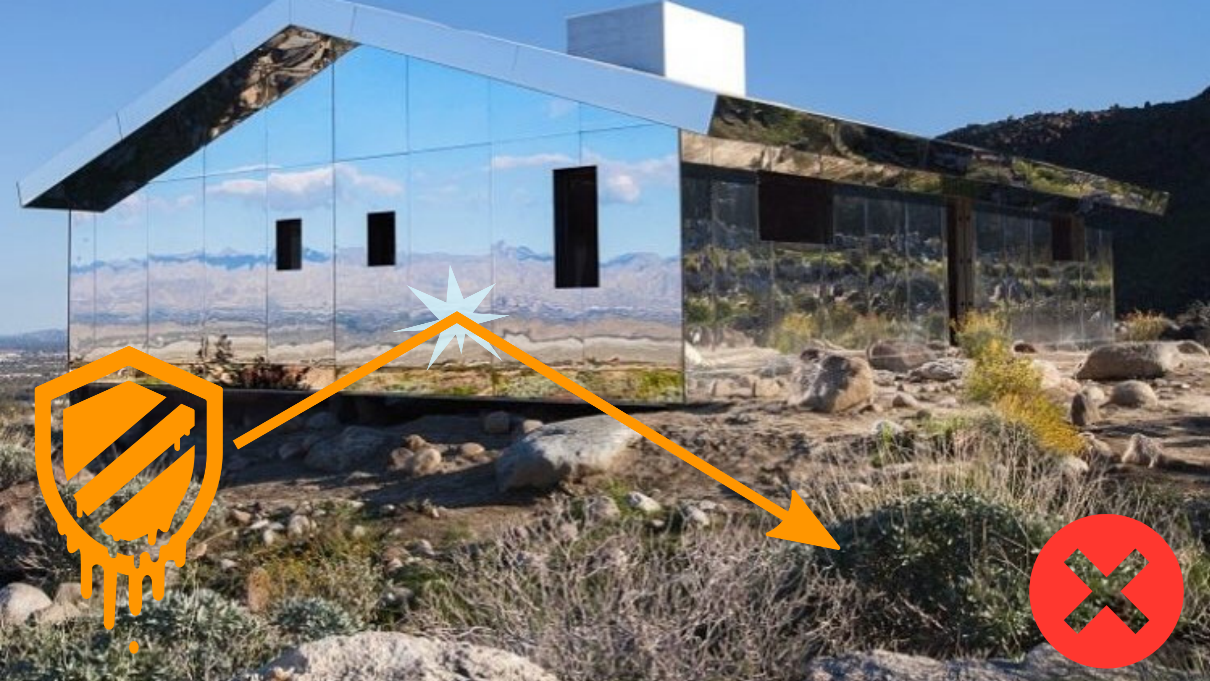


I'M SURE THIS WON'T BE THE LAST SUCH PROBLEM —

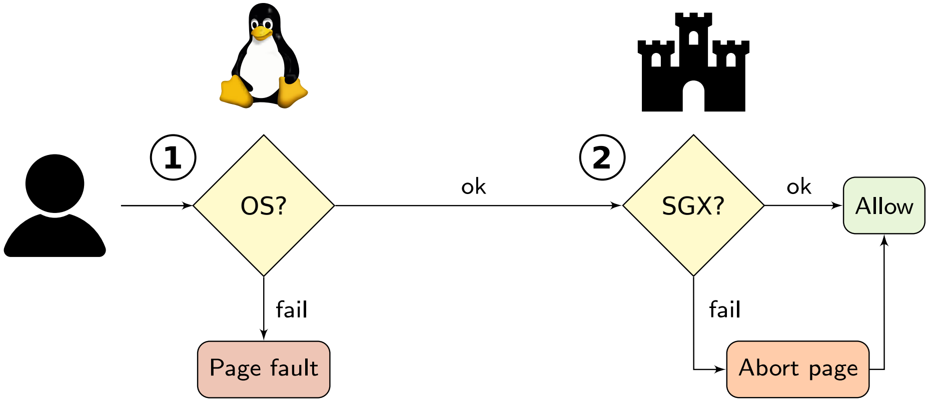
Intel's SGX blown wide open by, you guessed it, a speculative execution attack

Speculative execution attacks truly are the gift that keeps on giving.

https://wired.com and https://arstechnica.com

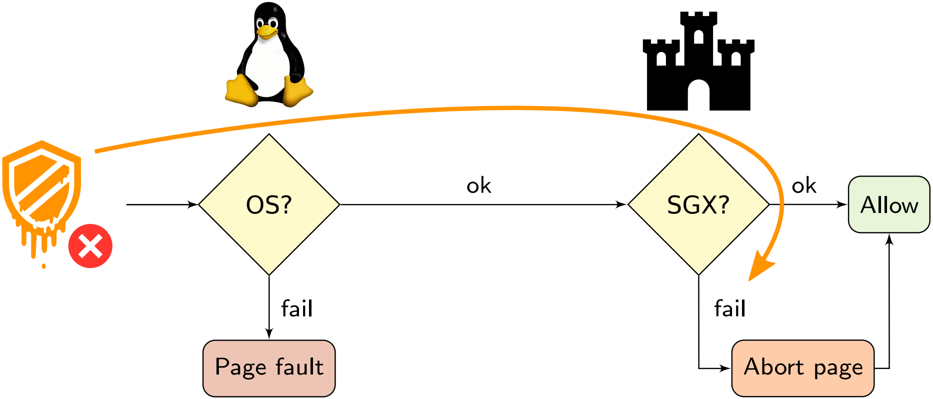


Building Foreshadow: Evade SGX abort page semantics



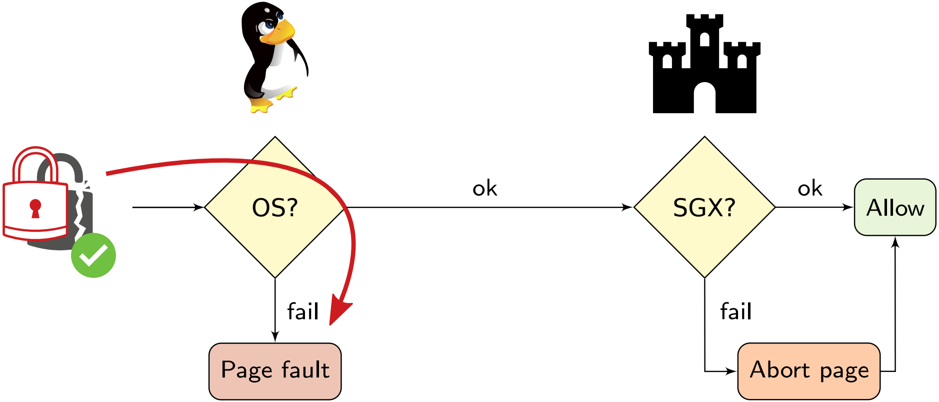
SGX checks prohibit unauthorized access

Building Foreshadow: Evade SGX abort page semantics



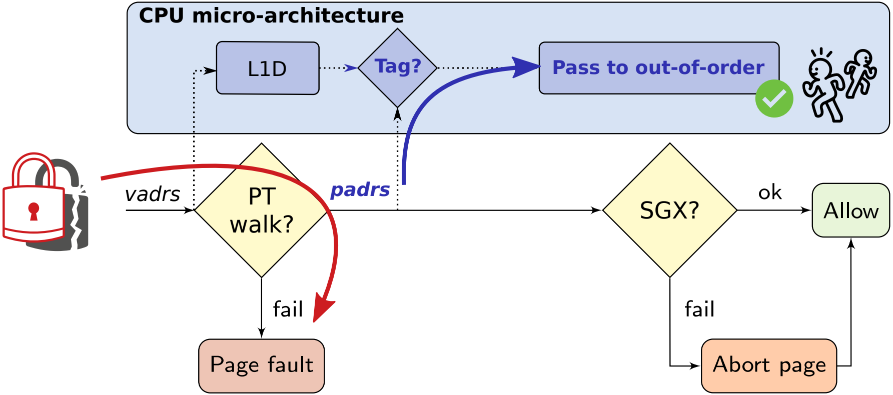
SGX checks prohibit unauthorized access

Building Foreshadow: Evade SGX abort page semantics



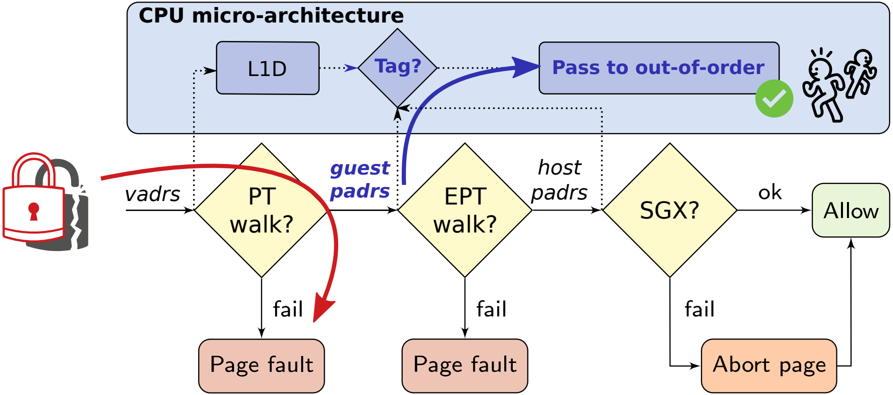
. . . but attackers can **unmap** enclave pages!

The microarchitecture behind Foreshadow

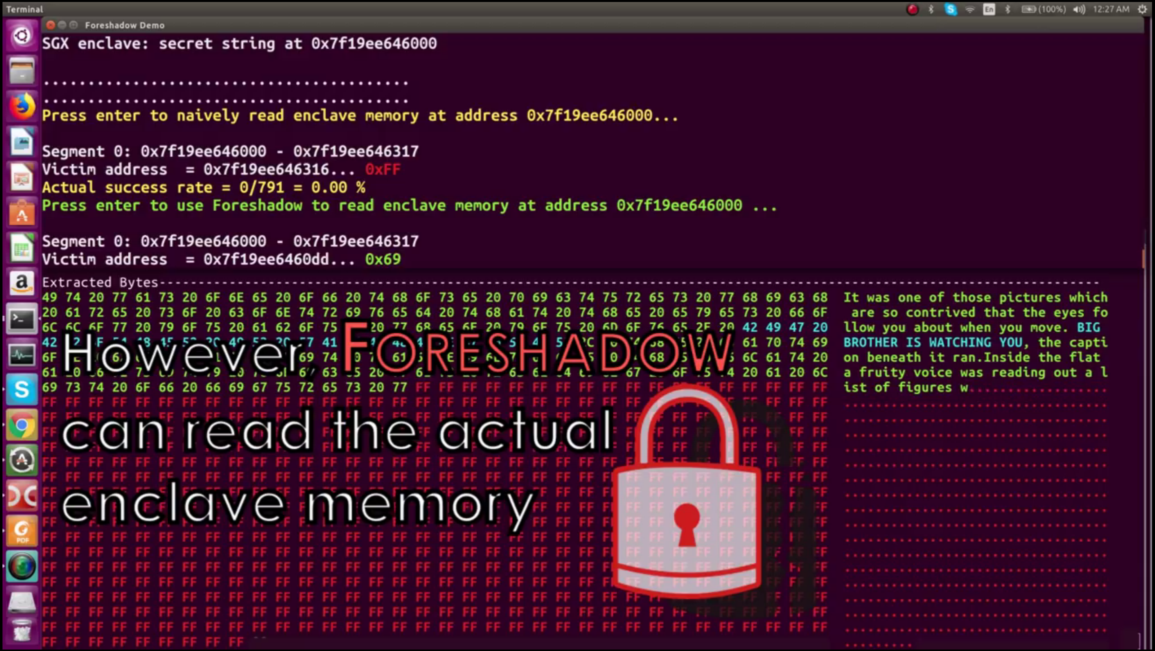


Foreshadow-SGX: Bypass enclave isolation

The microarchitecture behind Foreshadow



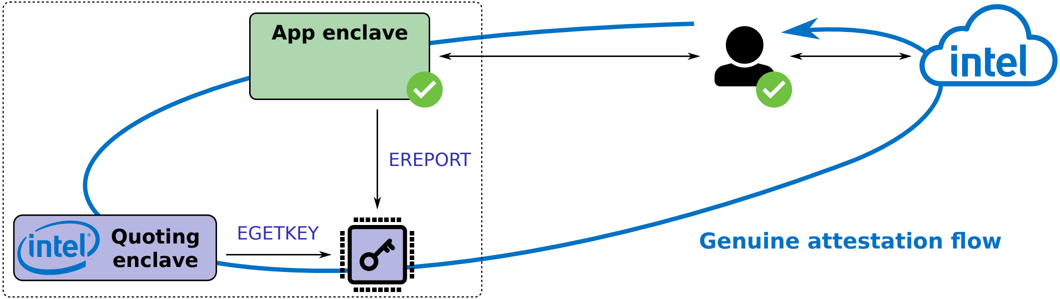
Foreshadow-NG: Bypass virtual machine isolation



```
SGX enclave: secret string at 0x7f19ee646000

........................................
........................................
Press enter to naively read enclave memory at address 0x7f19ee646000...

Segment 0: 0x7f19ee646000 - 0x7f19ee646317
Victim address  = 0x7f19ee646316... 0xFF
Actual success rate = 0/791 = 0.00 %
Press enter to use Foreshadow to read enclave memory at address 0x7f19ee646000 ...

Segment 0: 0x7f19ee646000 - 0x7f19ee646317
Victim address  = 0x7f19ee6460dd... 0x69
Extracted Bytes----------------------------------------------------------------------------------------------------
49 74 20 77 61 73 20 6F 6E 65 20 6F 66 20 74 68 6F 73 65 20 70 69 63 74 75 72 65 73 20 77 68 69 63 68   It was one of those pictures which
20 61 72 65 20 73 6F 20 63 6F 6E 74 72 69 76 65 64 20 74 68 61 74 20 74 68 65 20 65 79 65 73 20 66 6F    are so contrived that the eyes fo
6C 6C 6F 77 20 79 6F 75 20 61 62 6F 75 74 20 77 68 65 6E 20 79 6F 75 20 6D 6F 76 65 2E 20 42 49 47 20   llow you about when you move. BIG 
42 52 4F 54 48 45 52 20 49 53 20 57 41 54 43 48 49 4E 47 20 59 4F 55 2C 20 74 68 65 20 63 61 70 74 69   BROTHER IS WATCHING YOU, the capti
6F 6E 20 62 65 6E 65 61 74 68 20 69 74 20 72 61 6E 2E 49 6E 73 69 64 65 20 74 68 65 20 66 6C 61 74 20   on beneath it ran.Inside the flat 
61 20 66 72 75 69 74 79 20 76 6F 69 63 65 20 77 61 73 20 72 65 61 64 69 6E 67 20 6F 75 74 20 61 20 6C   a fruity voice was reading out a l
69 73 74 20 6F 66 20 66 69 67 75 72 65 73 20 77 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ist of figures w..................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ..................................
FF FF FF FF FF FF FF FF FF                                                                             .........
```

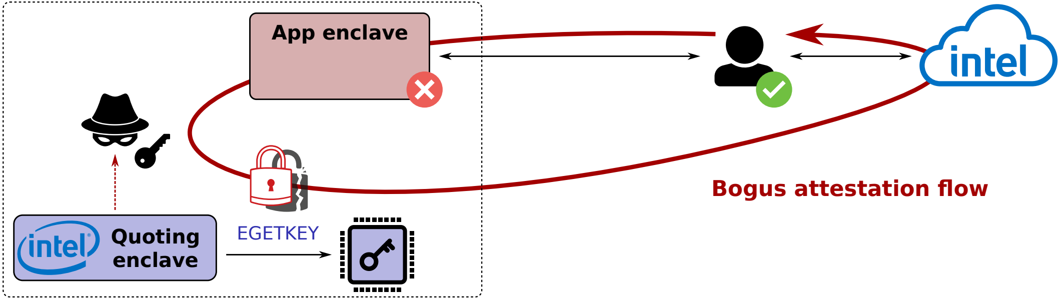
However FORESHADOW can read the actual enclave memory

Foreshadow: Extracting the keys to the Intel SGX kingdom



Intel == trusted 3th party (shared **CPU master secret**)

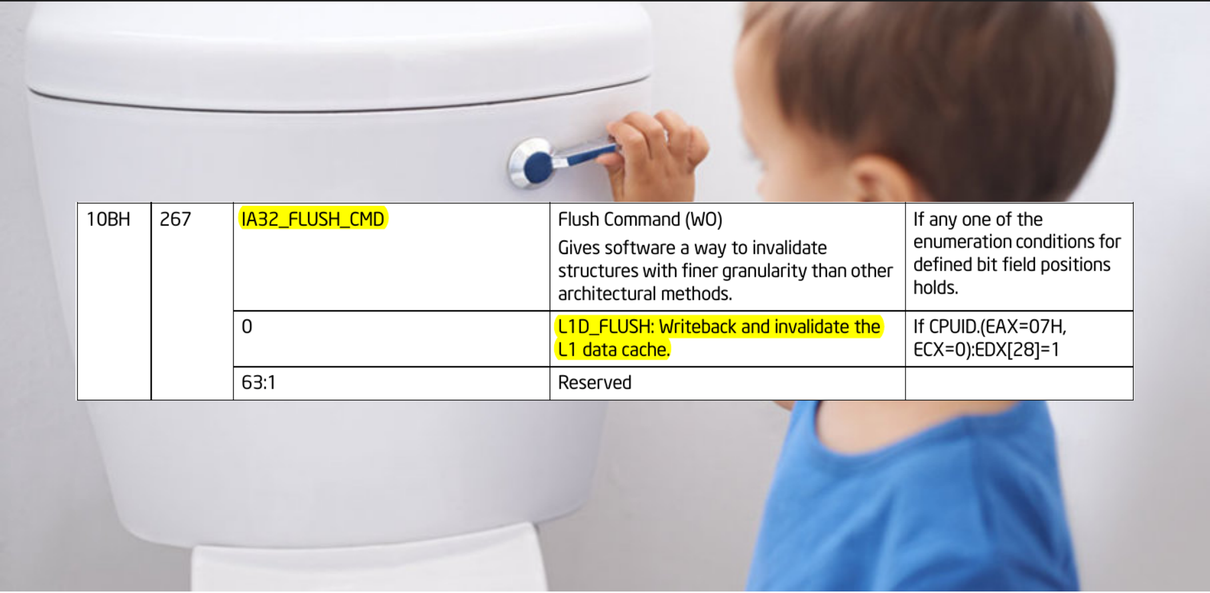
Foreshadow: Extracting the keys to the Intel SGX kingdom



ⓘ Extract long-term platform **attestation key** → forge Intel signatures

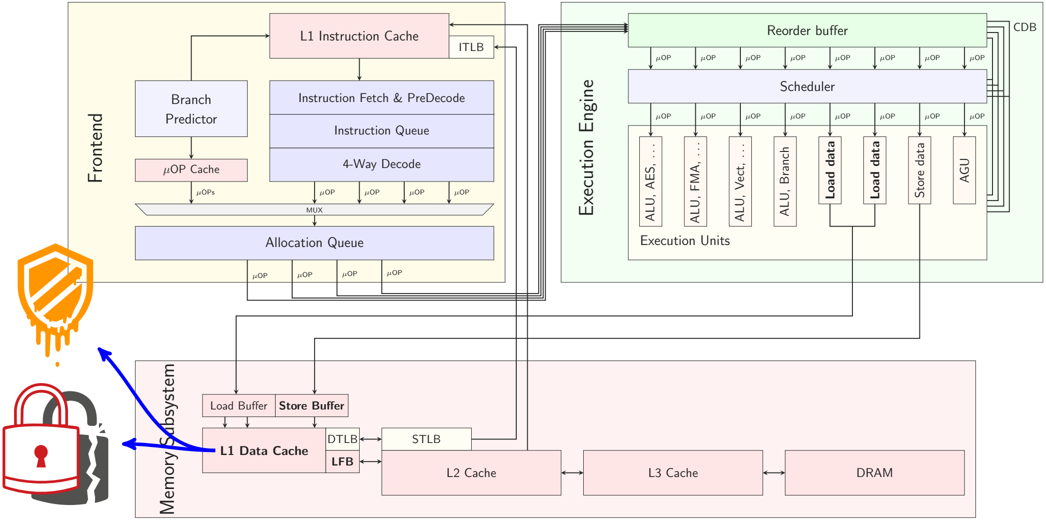
Mitigating Foreshadow: Flush CPU microarchitecture

Mitigating Foreshadow: Flush CPU microarchitecture

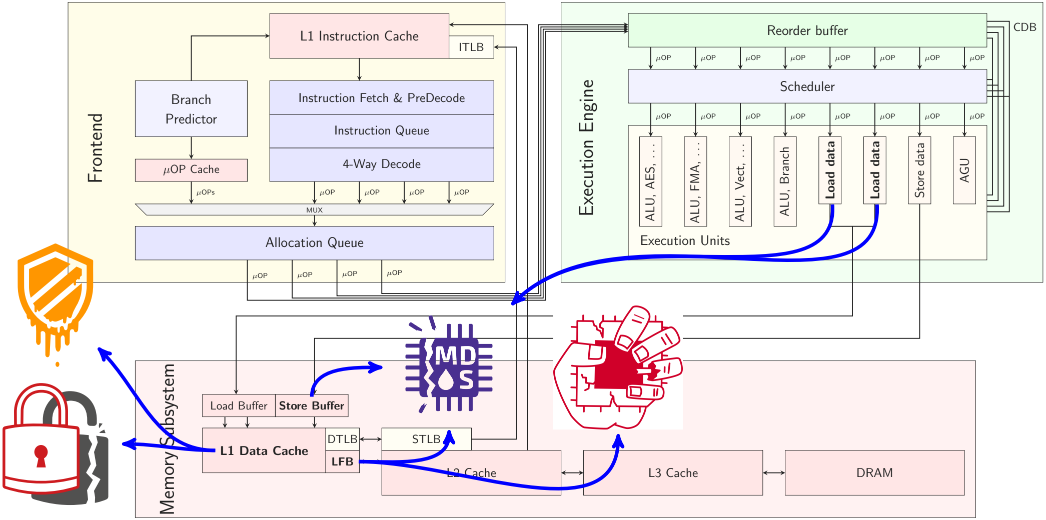


10BH	267	IA32_FLUSH_CMD	Flush Command (WO) Gives software a way to invalidate structures with finer granularity than other architectural methods.	If any one of the enumeration conditions for defined bit field positions holds.
		0	L1D_FLUSH: Writeback and invalidate the L1 data cache.	If CPUID.(EAX=07H, ECX=0):EDX[28]=1
		63:1	Reserved	

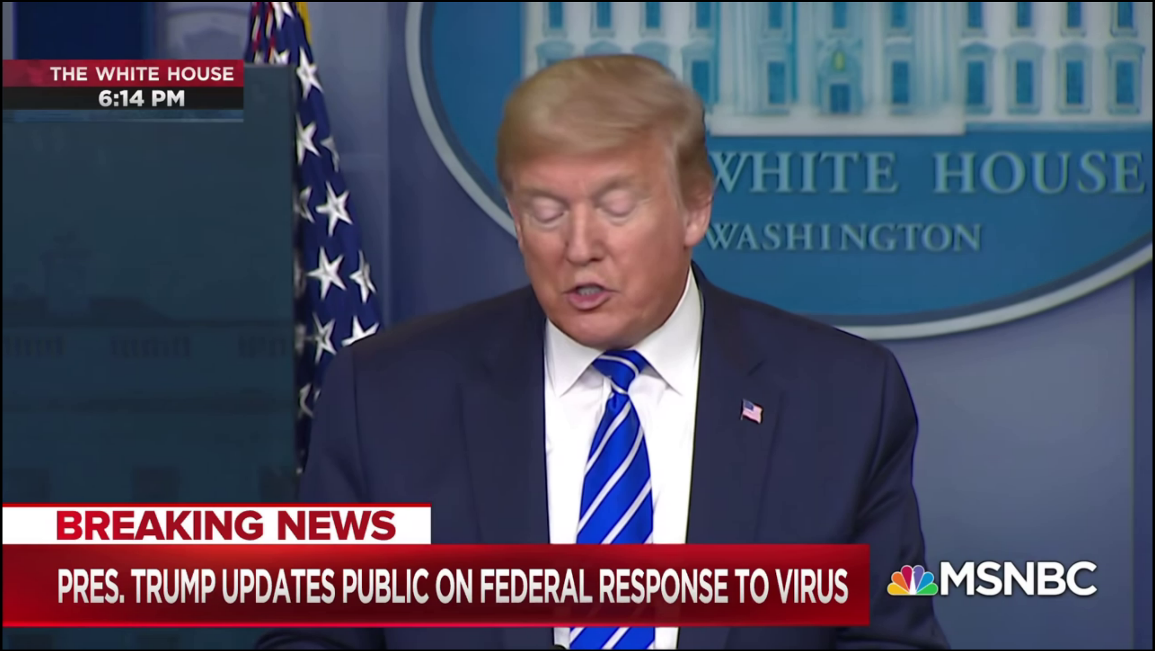
MDS variants: Flushing additional microarchitectural buffers



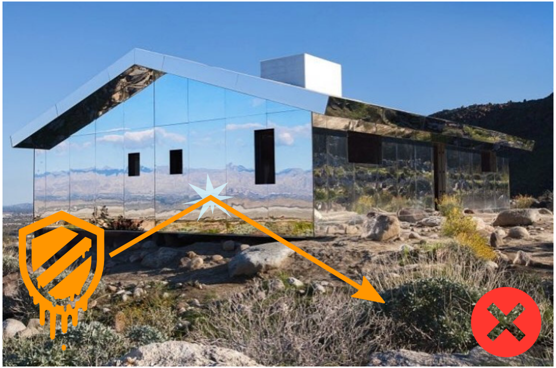
MDS variants: Flushing additional microarchitectural buffers



https://lviattack.eu/


THE WHITE HOUSE
6:14 PM
WHITE HOUSE
WASHINGTON
BREAKING NEWS
PRES. TRUMP UPDATES PUBLIC ON FEDERAL RESPONSE TO VIRUS
MSNBC

Idea: Can we turn Foreshadow around?



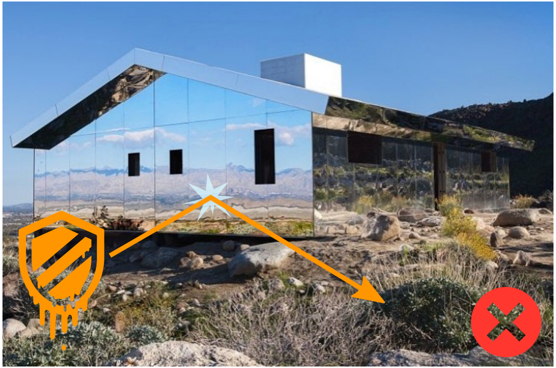
Outside view

- Meltdown: out-of-reach
- Foreshadow: cache emptied

Intra-enclave view

- Access enclave + outside memory

Idea: Can we turn Foreshadow around?



Outside view

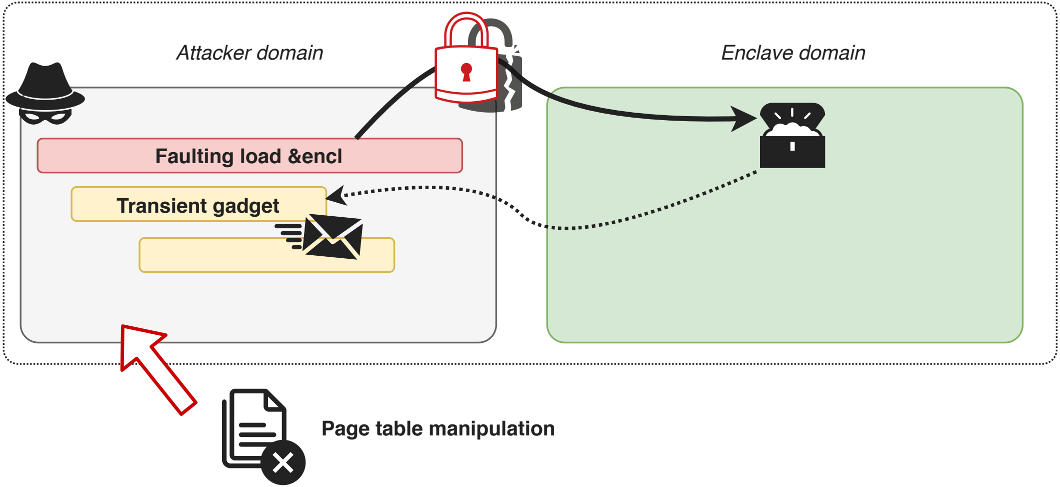
- Meltdown: out-of-reach
- Foreshadow: cache emptied

Intra-enclave view

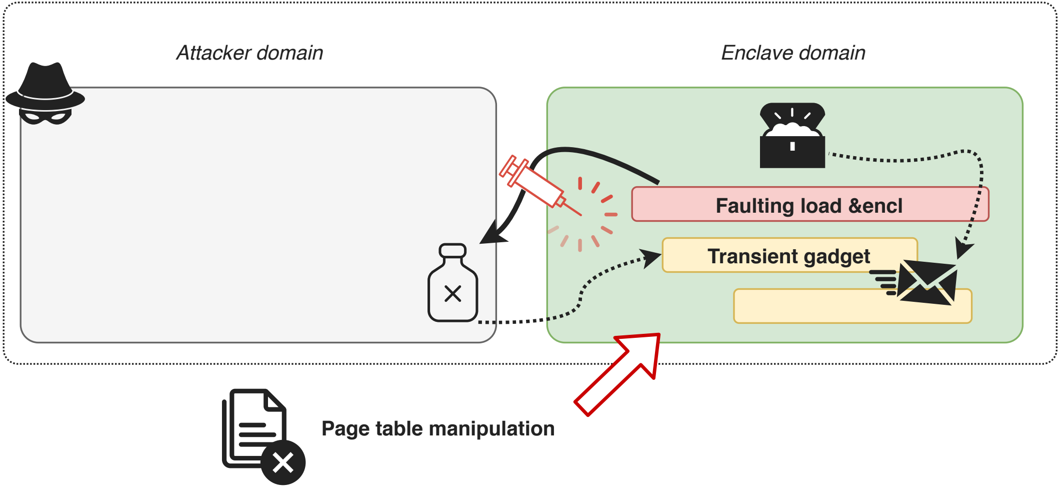
- Access enclave + outside memory

→ Abuse **in-enclave code gadgets!**

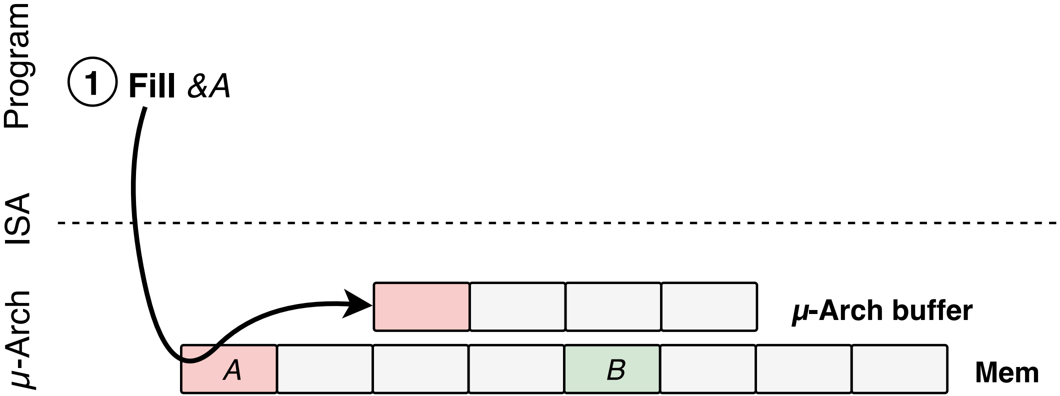
Reviving Foreshadow with Load Value Injection (LVI)



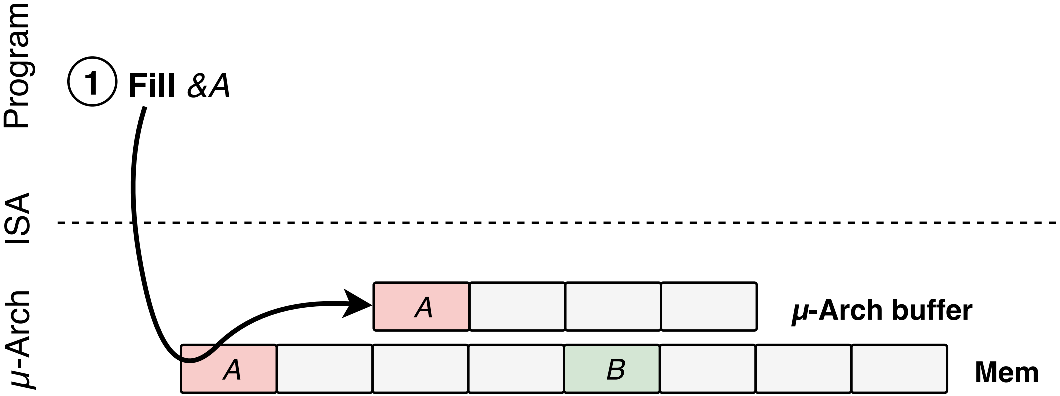
Reviving Foreshadow with Load Value Injection (LVI)



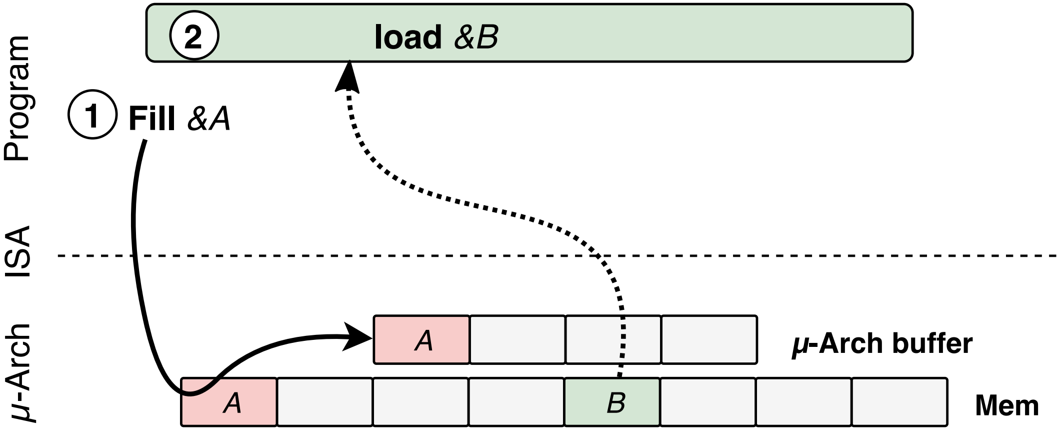
LVI: The basic idea



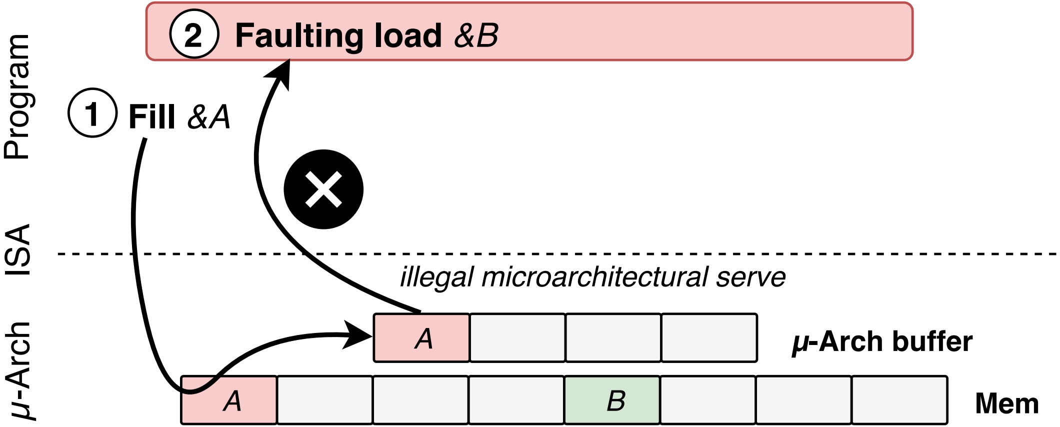
LVI: The basic idea



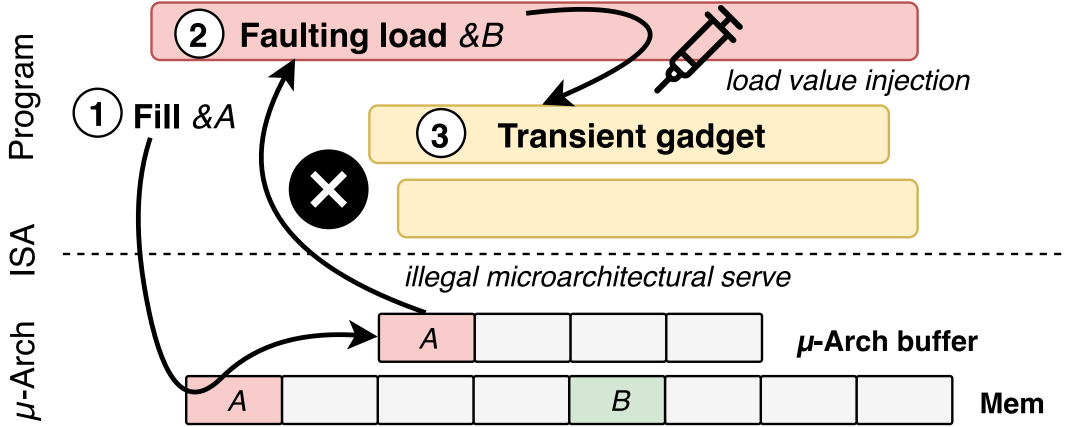
LVI: The basic idea



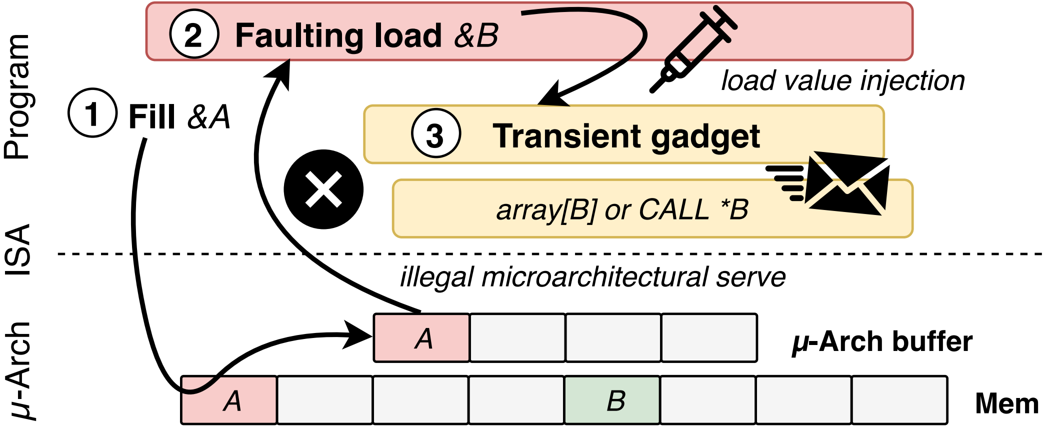
LVI: The basic idea



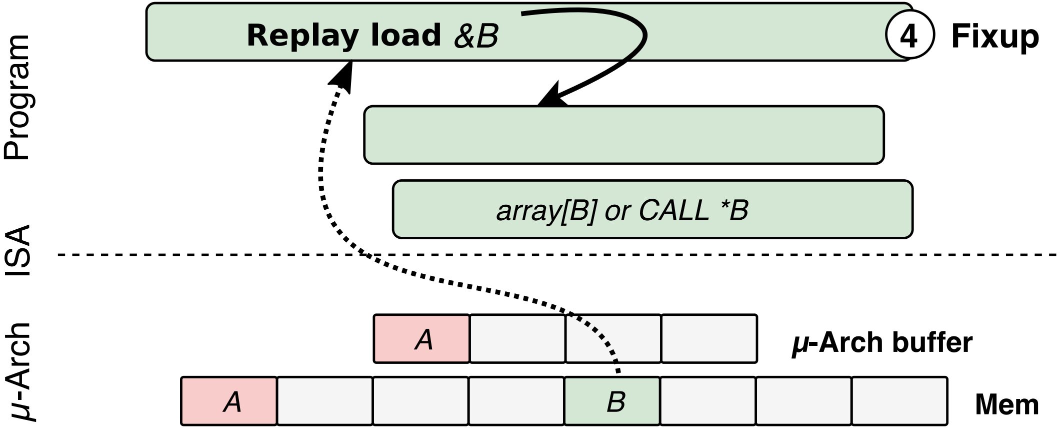
LVI: The basic idea



LVI: The basic idea



LVI: The basic idea



FOOD POISONING

Overdue products

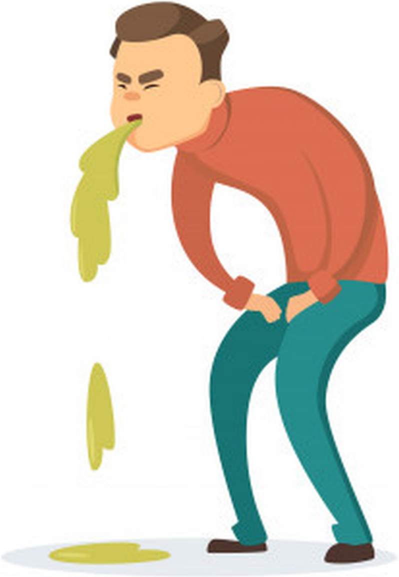
Medicine

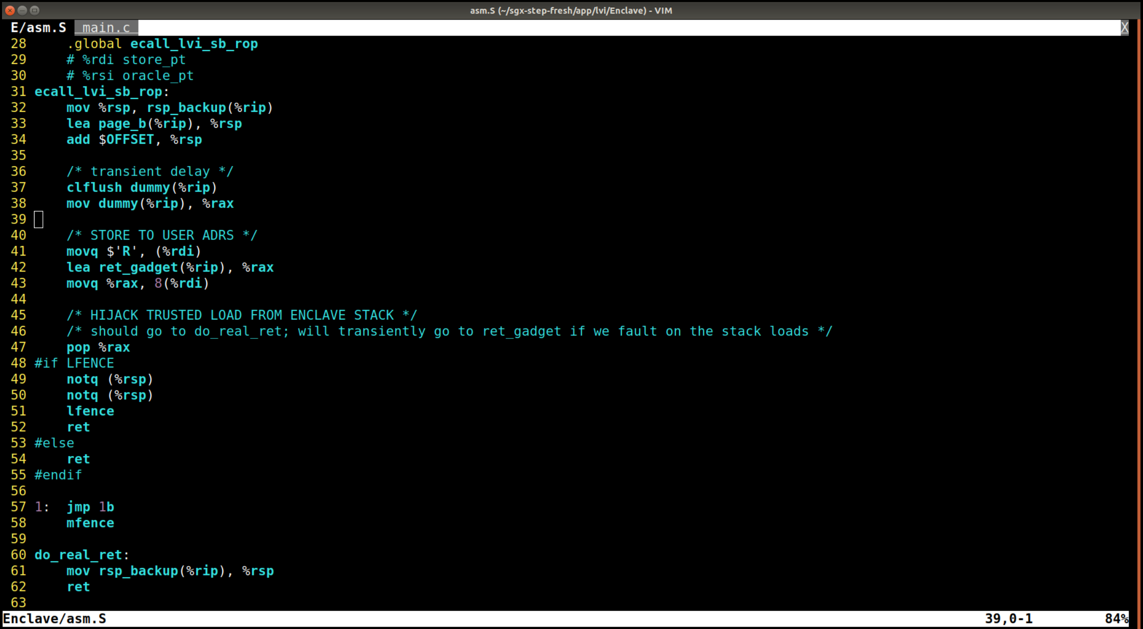
Dizziness

Intestinal colic

Diarrhea

Headache





```
    .global ecall_lvi_sb_rop
    # %rdi store_pt
    # %rsi oracle_pt
ecall_lvi_sb_rop:
    mov %rsp, rsp_backup(%rip)
    lea page_b(%rip), %rsp
    add $OFFSET, %rsp

    /* transient delay */
    clflush dummy(%rip)
    mov dummy(%rip), %rax

    /* STORE TO USER ADRS */
    movq $'R', (%rdi)
    lea ret_gadget(%rip), %rax
    movq %rax, 8(%rdi)

    /* HIJACK TRUSTED LOAD FROM ENCLAVE STACK */
    /* should go to do_real_ret; will transiently go to ret_gadget if we fault on the stack loads */
    pop %rax
#if LFENCE
    notq (%rsp)
    notq (%rsp)
    lfence
    ret
#else
    ret
#endif

1:  jmp 1b
    mfence

do_real_ret:
    mov rsp_backup(%rip), %rsp
    ret

```

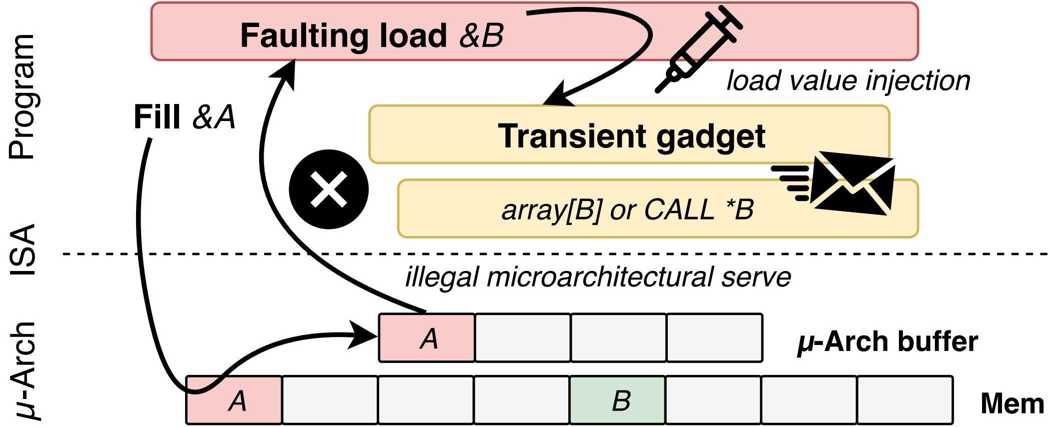
Mitigating LVI: Fencing vulnerable load instructions

Mitigating LVI: Fencing vulnerable load instructions

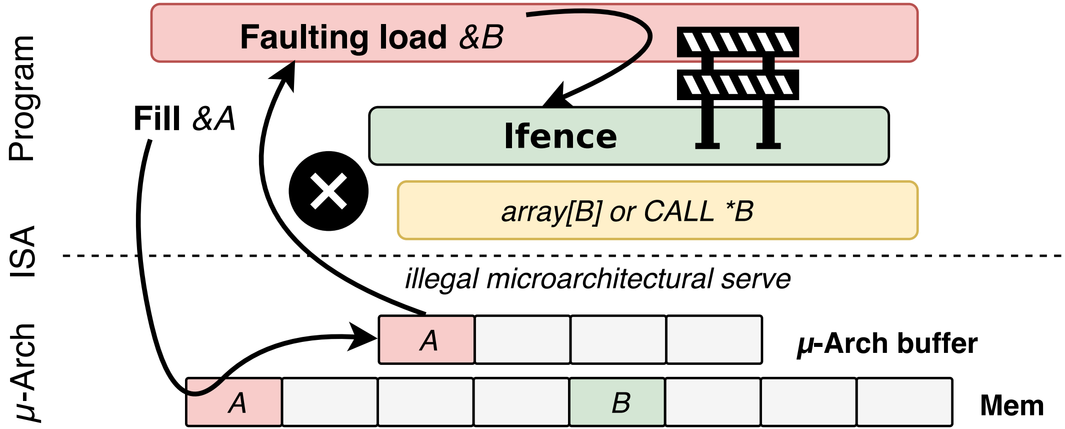
LFENCE—Load Fence

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
NP 0F AE E8	LFENCE	ZO	Valid	Valid	Serializes load operations.

Mitigation idea: Fencing vulnerable load instructions

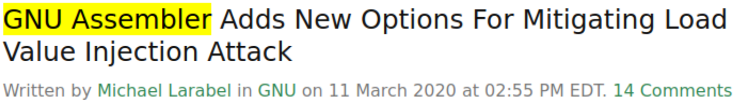


Mitigation idea: Fencing vulnerable load instructions



Mitigating LVI: Compiler and assembler support

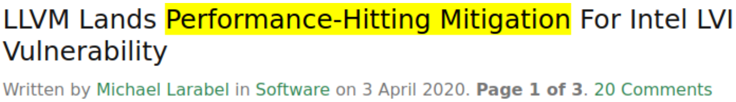
`-mlfence-after-load`



GNU Assembler Adds New Options For Mitigating Load Value Injection Attack

Written by Michael Larabel in GNU on 11 March 2020 at 02:55 PM EDT. 14 Comments

`-mlvi-hardening`

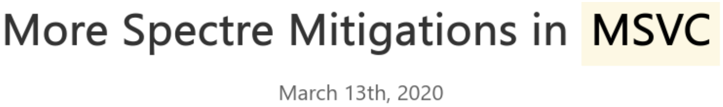


LLVM Lands Performance-Hitting Mitigation For Intel LVI Vulnerability

Written by Michael Larabel in Software on 3 April 2020. **Page 1 of 3**. 20 Comments

Visual C++

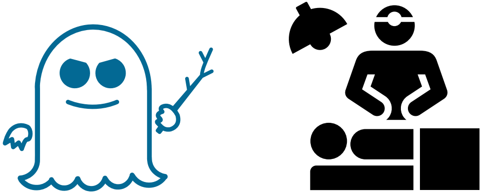
`-Qspectre-load`



More Spectre Mitigations in MSVC

March 13th, 2020

Intel architectural enclaves: `lfence` counts `libsgx_qe.signed.so`



23 fences

October 2019—“surgical precision”

Intel architectural enclaves: `lfence` counts

`libsgx_qe.signed.so`

23 fences

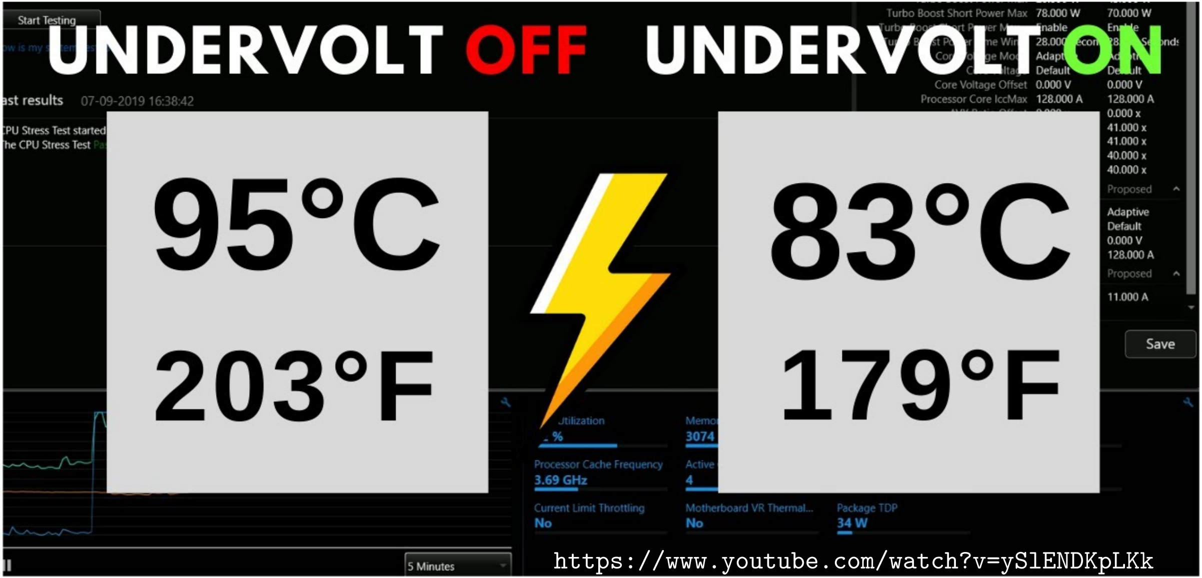
October 2019—“surgical precision”

49,315 fences

March 2020—“big hammer”

Attack idea 4: CPU interfaces

TEMPERATURES I5 9300H



UNDERVOLT OFF UNDERVOLT ON

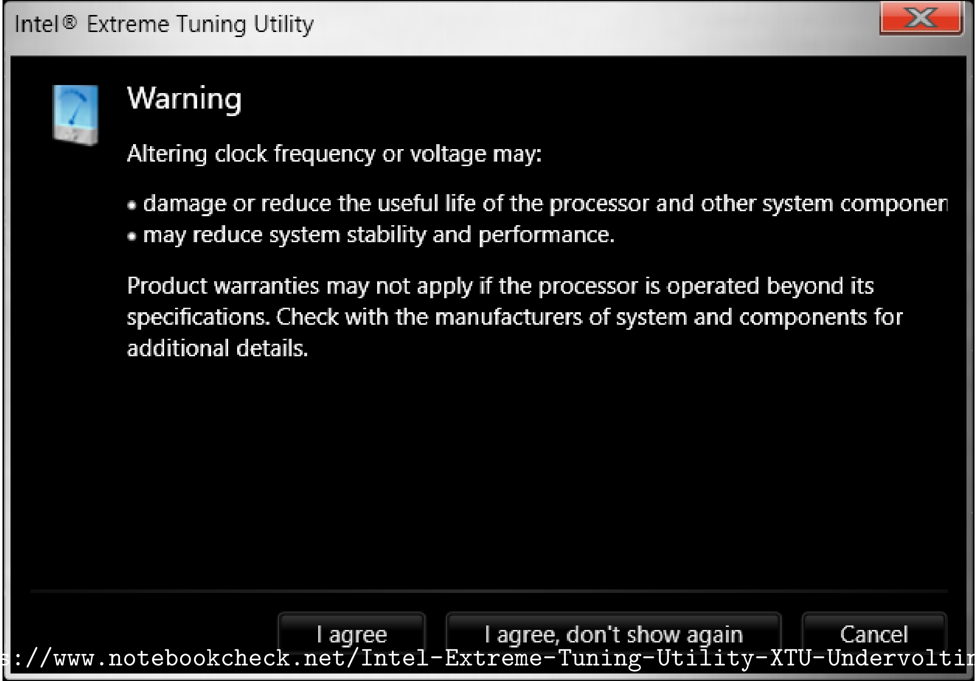
95°C

203°F

83°C

179°F

https://www.youtube.com/watch?v=ySlENDKpLKk



Warning

Altering clock frequency or voltage may:

- damage or reduce the useful life of the processor and other system componen
- may reduce system stability and performance.

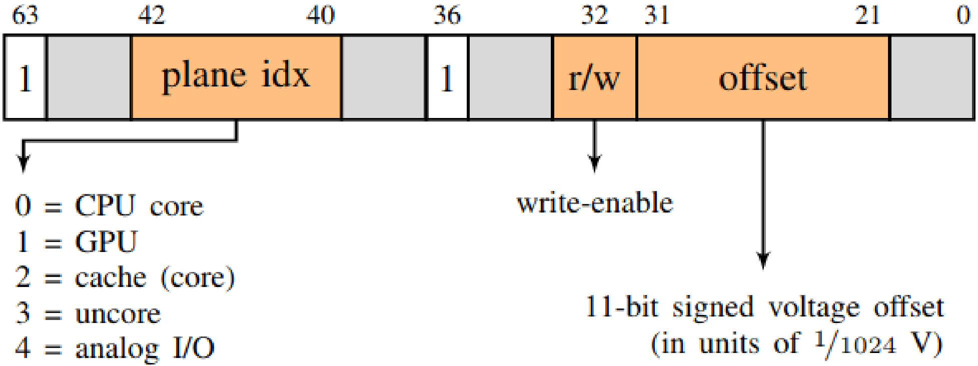
Product warranties may not apply if the processor is operated beyond its specifications. Check with the manufacturers of system and components for additional details.

s://www.notebookcheck.net/Intel-Extreme-Tuning-Utility-XTU-Undervoltir

How a little bit of undervolting can cause a lot of problems

https://plundervolt.com/

Intel Power Management



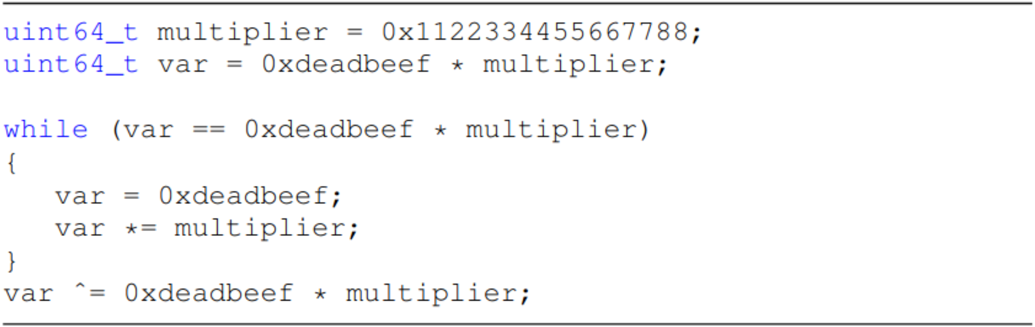
0 = CPU core
1 = GPU
2 = cache (core)
3 = uncore
4 = analog I/O

write-enable

11-bit signed voltage offset
(in units of $^1/_{1024}$ V)

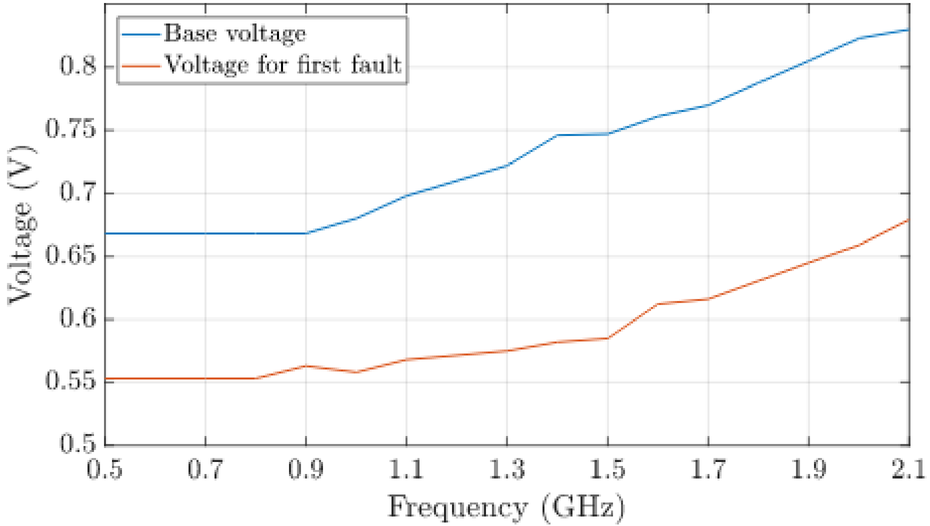
Fig. 1. Layout of the undocumented undervolting MSR with address `0x150`.

Plundervolt: Will it fault?



```
uint64_t multiplier = 0x1122334455667788;
uint64_t var = 0xdeadbeef * multiplier;

while (var == 0xdeadbeef * multiplier)
{
   var = 0xdeadbeef;
   var *= multiplier;
}
var ^= 0xdeadbeef * multiplier;
```

Plundervolt: Will it fault?



Plundervolt: Will it fault?

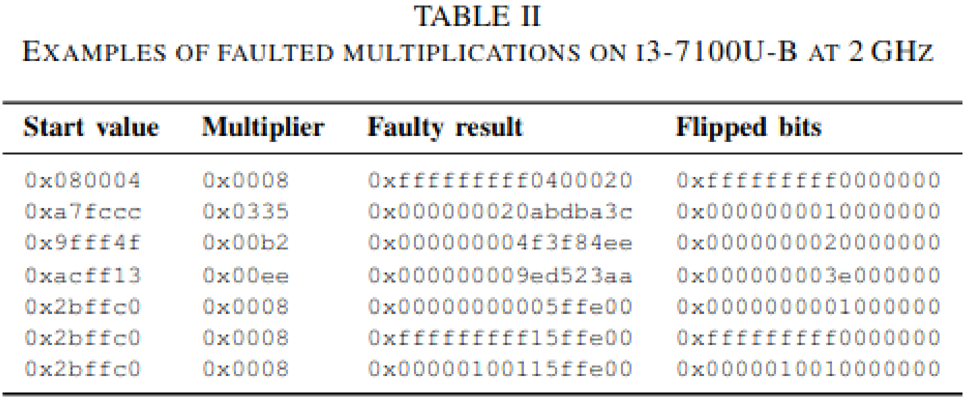
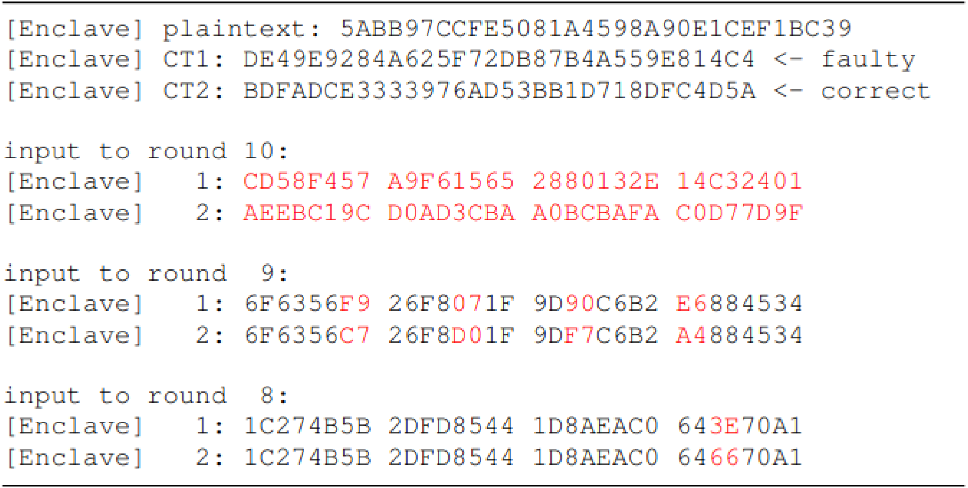


TABLE II
EXAMPLES OF FAULTED MULTIPLICATIONS ON I3-7100U-B AT 2 GHZ

Start value	Multiplier	Faulty result	Flipped bits
0x080004	0x0008	0xfffffffff0400020	0xfffffffff0000000
0xa7fccc	0x0335	0x000000020abdba3c	0x0000000010000000
0x9fff4f	0x00b2	0x000000004f3f84ee	0x0000000020000000
0xacff13	0x00ee	0x000000009ed523aa	0x000000003e000000
0x2bffc0	0x0008	0x00000000005ffe00	0x0000000001000000
0x2bffc0	0x0008	0xfffffffff15ffe00	0xfffffffff0000000
0x2bffc0	0x0008	0x00000100115ffe00	0x0000010010000000

Plundervolt: Differential Fault Analysis of AES-NI in SGX



```
[Enclave] plaintext: 5ABB97CCFE5081A4598A90E1CEF1BC39
[Enclave] CT1: DE49E9284A625F72DB87B4A559E814C4 <- faulty
[Enclave] CT2: BDFADCE3333976AD53BB1D718DFC4D5A <- correct

input to round 10:
[Enclave]   1: CD58F457 A9F61565 2880132E 14C32401
[Enclave]   2: AEEBC19C D0AD3CBA A0BCBAFA C0D77D9F

input to round  9:
[Enclave]   1: 6F6356F9 26F8071F 9D90C6B2 E6884534
[Enclave]   2: 6F6356C7 26F8D01F 9DF7C6B2 A4884534

input to round  8:
[Enclave]   1: 1C274B5B 2DFD8544 1D8AEAC0 643E70A1
[Enclave]   2: 1C274B5B 2DFD8544 1D8AEAC0 646670A1
```

Plundervolt: Beyond crypto—inducing memory-safety faults

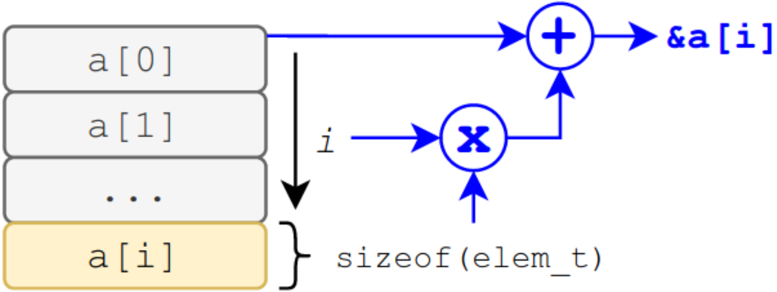
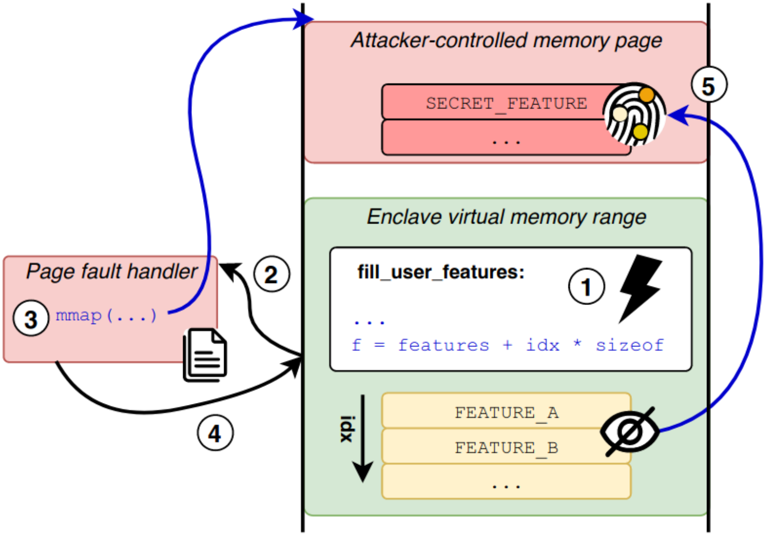
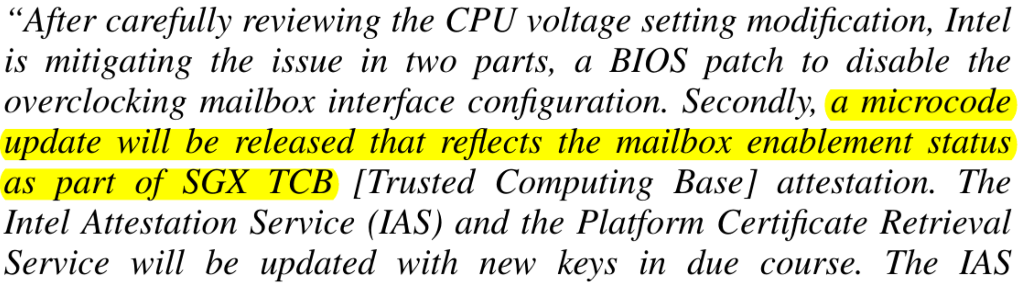


Figure 4. The address of element a[i] in an array is computed as &a[0]+i*sizeof(elem_t).

Plundervolt: Beyond crypto—inducing memory-safety faults

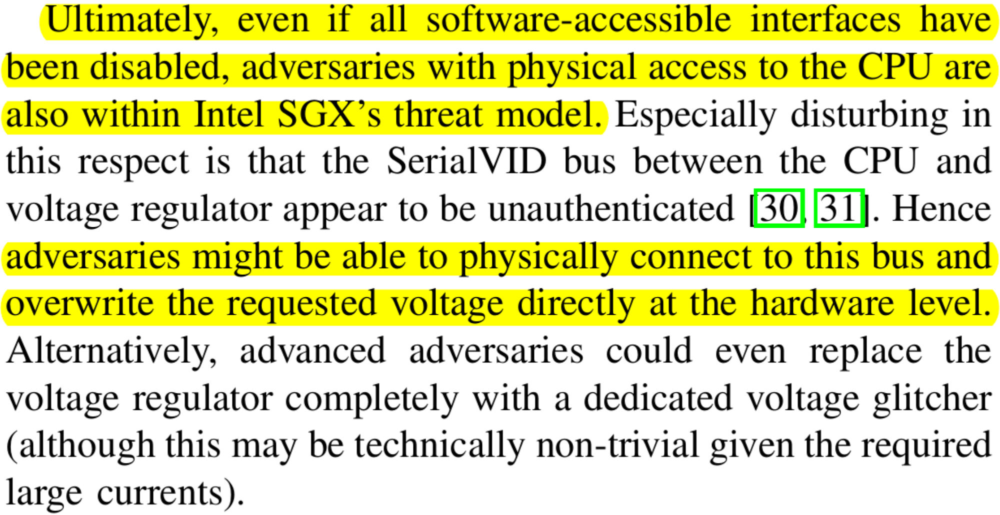


Plundervolt: Software mitigations

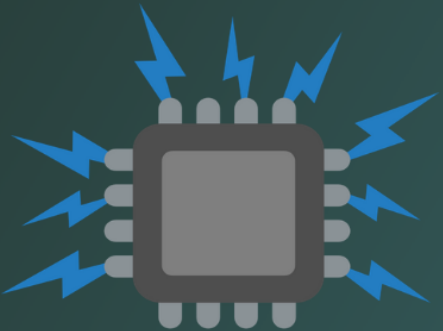


“After carefully reviewing the CPU voltage setting modification, Intel is mitigating the issue in two parts, a BIOS patch to disable the overclocking mailbox interface configuration. Secondly, a microcode update will be released that reflects the mailbox enablement status as part of SGX TCB [Trusted Computing Base] attestation. The Intel Attestation Service (IAS) and the Platform Certificate Retrieval Service will be updated with new keys in due course. The IAS

Plundervolt: Software mitigations



Ultimately, even if all software-accessible interfaces have been disabled, adversaries with physical access to the CPU are also within Intel SGX's threat model. Especially disturbing in this respect is that the SerialVID bus between the CPU and voltage regulator appear to be unauthenticated [30], [31]. Hence adversaries might be able to physically connect to this bus and overwrite the requested voltage directly at the hardware level. Alternatively, advanced adversaries could even replace the voltage regulator completely with a dedicated voltage glitcher (although this may be technically non-trivial given the required large currents).

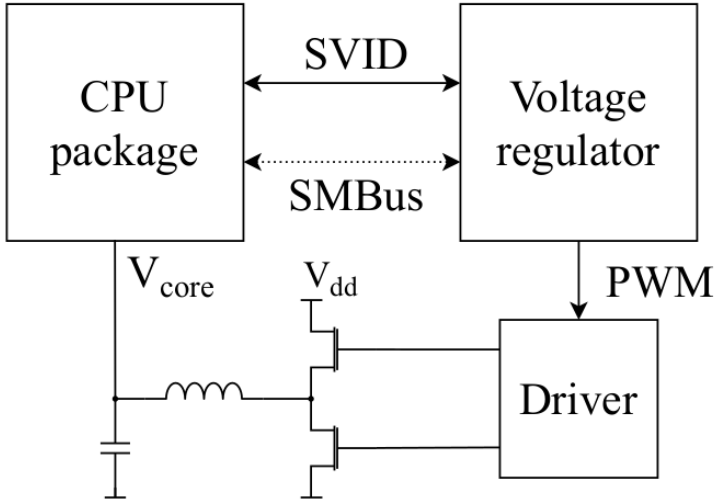


VOLT PILLAGER

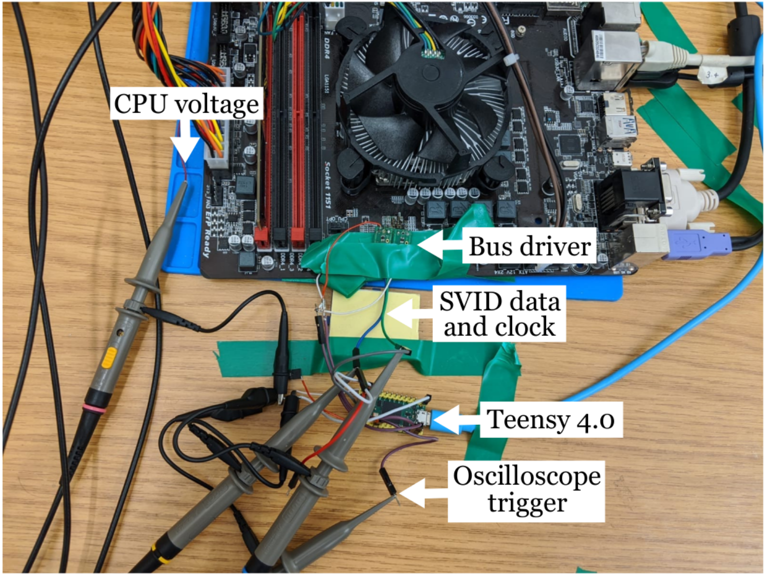
CPU undervolting with low-cost tools

https://zt-chen.github.io/voltpillager/

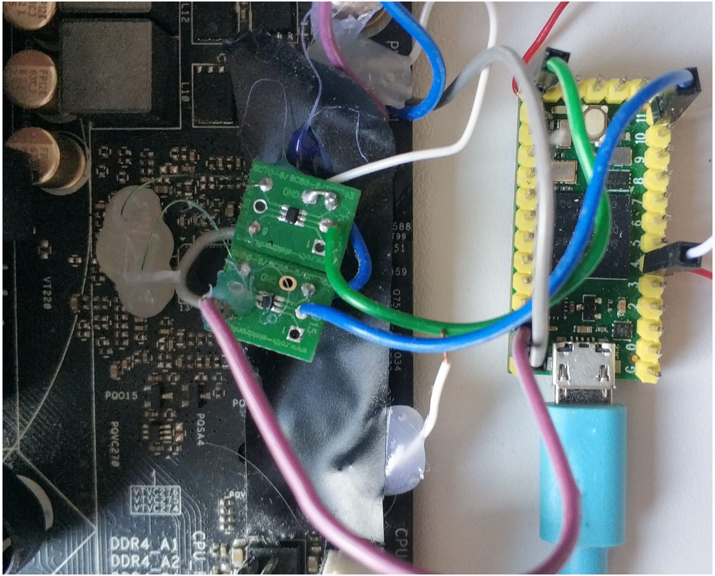
VoltPillager: Physical SVID command injection



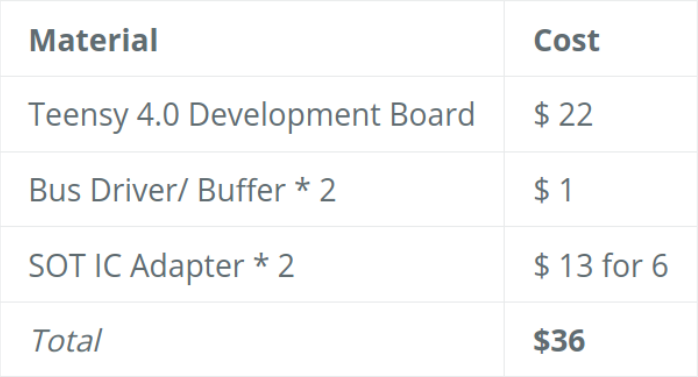
VoltPillager: Physical SVID command injection



VoltPillager: Physical SVID command injection



VoltPillager: Physical SVID command injection



Material	Cost
Teensy 4.0 Development Board	$ 22
Bus Driver/ Buffer * 2	$ 1
SOT IC Adapter * 2	$ 13 for 6
Total	**$36**

ATTACK...
DEPENDS WHO'S ASKING
memegenerator.net

Takeaways

Conclusions and takeaway

⇒ **Trusted execution** environments (Intel SGX) ≠ perfect(!)

⇒ Subtle **side channels** can go a long way...

⇒ **Privileged adversary model** = game changer

Conclusions and takeaway

⇒ **Trusted execution** environments (Intel SGX) ≠ perfect(!)

⇒ Subtle **side channels** can go a long way. . .

⇒ **Privileged adversary model** = game changer

Thank you! Questions?