Hardware-Based Trusted Computing Architectures

From an Attack and Defense Perspective

Jo Van Bulck

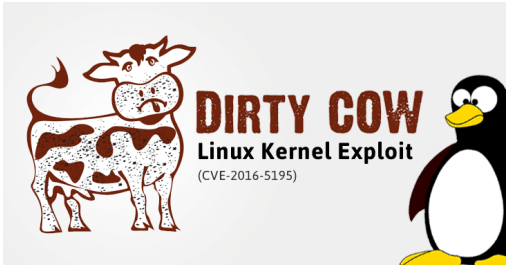
imec-DistriNet, KU Leuven – jo.vanbulck@cs.kuleuven.be

Newline 0x08 @Hackerspace.gent, April 14, 2018

Road Map

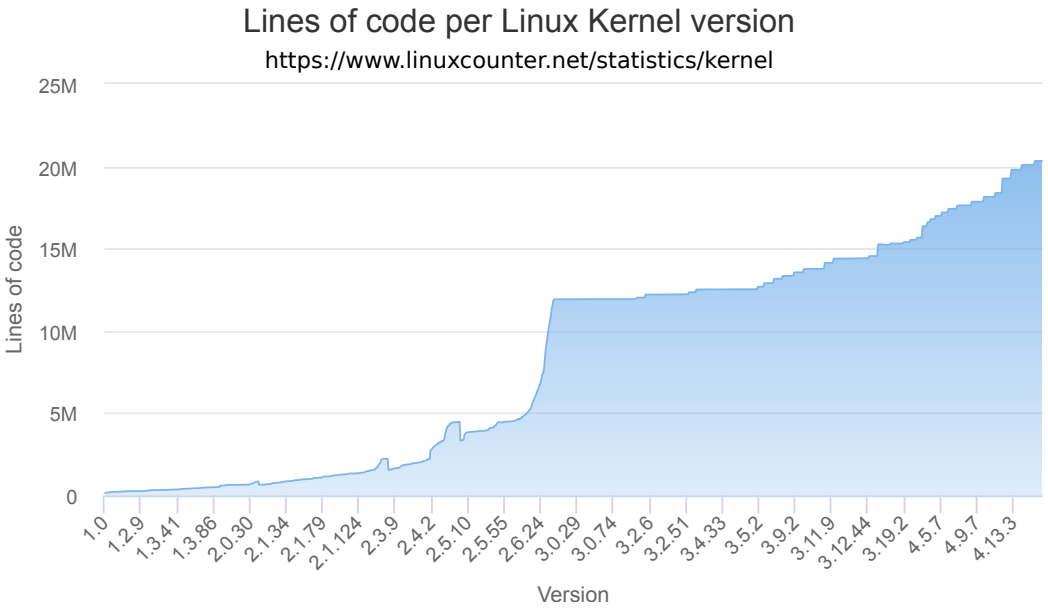
thehackernews.com/2015/10/windows-patch-update.html

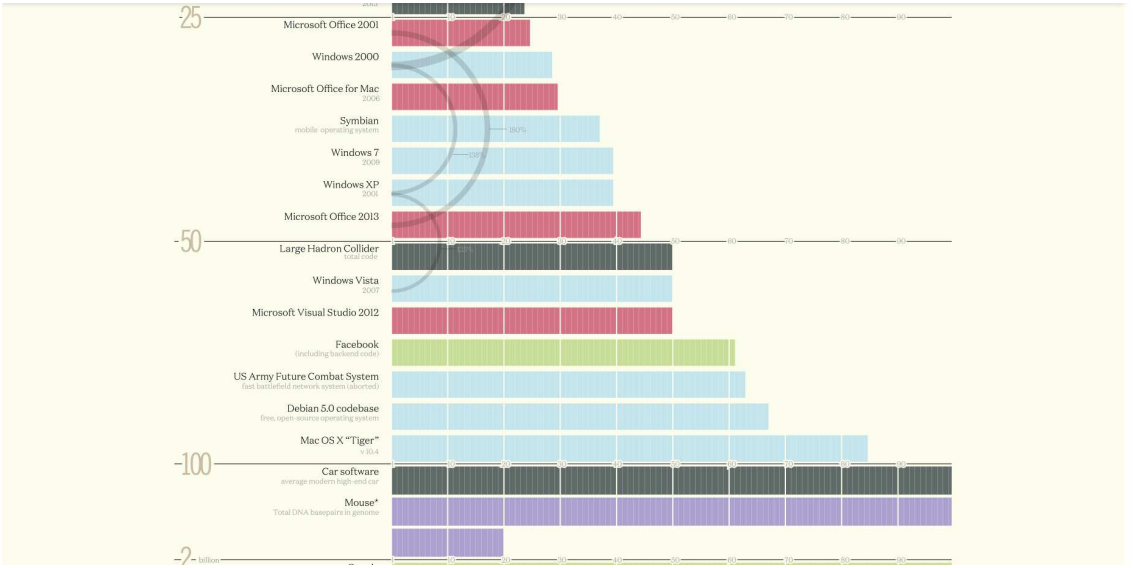
thehackernews.com/2017/06/cia-linux-hacking-tool-malware.html



thehackernews.com/2016/10/linux-kernel-exploit.html

thehackernews.com/2015/04/rootpipe-mac-os-x-vulnerability.html

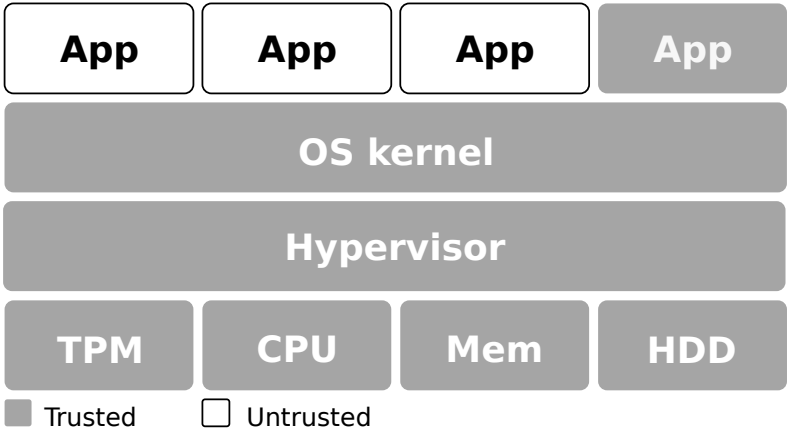

Lines of code per Linux Kernel version
https://www.linuxcounter.net/statistics/kernel
Lines of code
25M
20M
15M
10M
5M
0
1.0
1.2.9
1.3.41
1.3.86
2.0.30
2.1.34
2.1.79
2.1.124
2.3.9
2.4.2
2.5.10
2.5.55
2.6.24
3.0.29
3.0.74
3.2.6
3.2.51
3.4.33
3.5.2
3.9.2
3.11.9
3.12.44
3.19.2
4.5.7
4.9.7
4.13.3
Version



Source: `https://informationisbeautiful.net/visualizations/million-lines-of-code/`

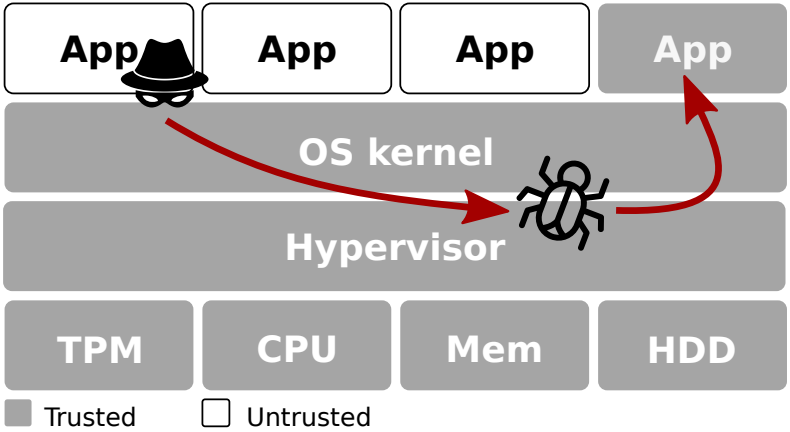
Source: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

Motivation: Application Attack Surface



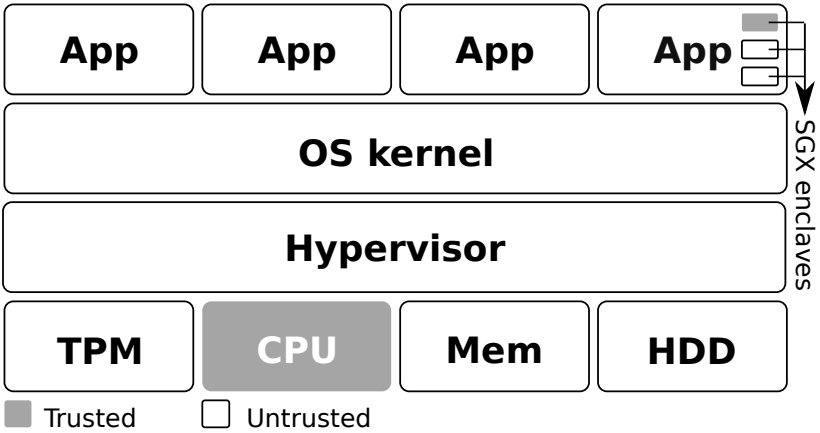
Layered architecture → large **trusted computing base**

Motivation: Application Attack Surface



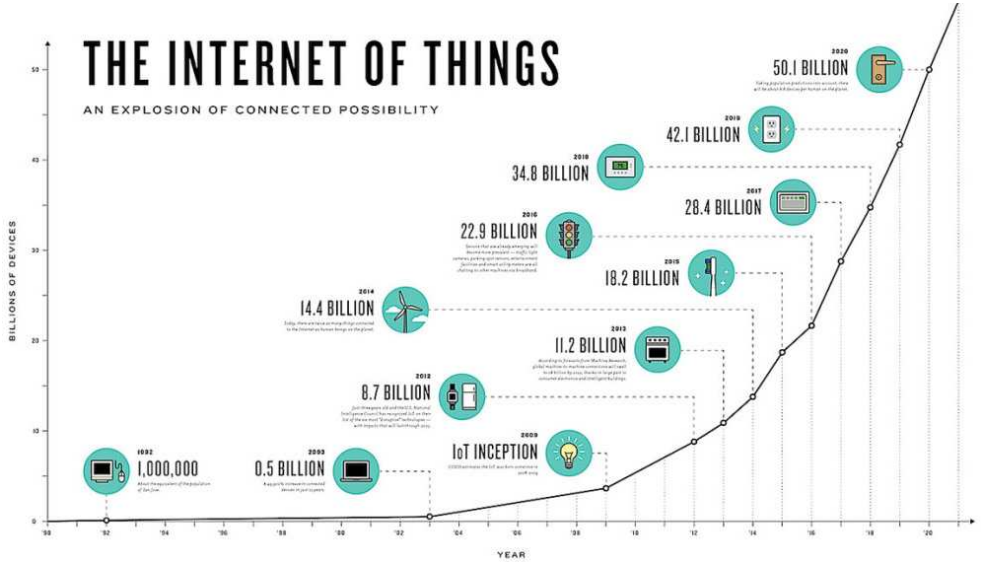
Layered architecture → large **trusted computing base**

Motivation: Application Attack Surface



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

Road Map



Source: https://www.ncta.com/platform/industry-news/infographic-the-growth-of-the-internet-of-things/

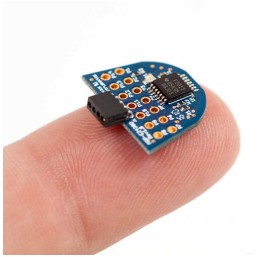
"Embedded-systems security is for lack of a better word, a mess."

— John Viega & Hugh Thompson
(IEEE Security and Privacy, September 2012).

Motivation: Embedded Device Security

TI MSP430: low-cost, low-power computing

- Runs ˜13 years on a single AA battery [Sea08]
- *Single-address-space* without memory protection
- Attacker can modify all code and data + forge sensor readings

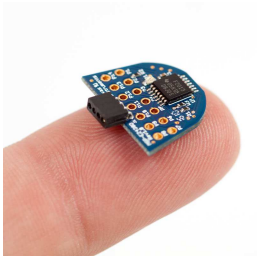


http://martybugs.net/electronics/msp430/

Motivation: Embedded Device Security

TI MSP430: low-cost, low-power computing

- Runs ˜13 years on a single AA battery [Sea08]
- *Single-address-space* without memory protection
- Attacker can modify all code and data + forge sensor readings



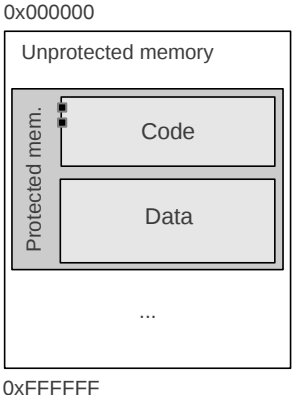
http://martybugs.net/electronics/msp430/

Protected Module Architectures: isolation and attestation

- Minimal (hardware-only) Trusted Computing Base
- Server/desktop: Intel SGX, ARM TrustZone
- Low-end embedded: SMART, TrustLite, TyTAN, Sancus

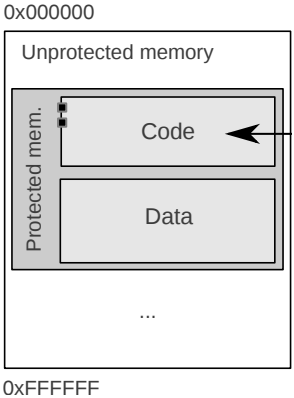
Maene et al.: "Hardware-Based Trusted Computing Architectures for Isolation and Attestation", 2017 [MGdC+17].

Background: Protected Module Architectures



- **Isolated execution** in a single-address-space

Background: Protected Module Architectures

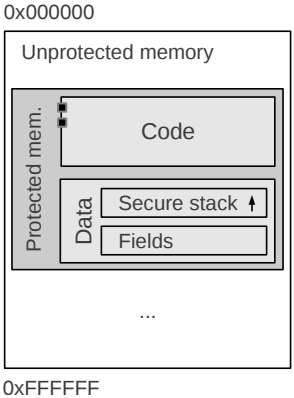


- **Isolated execution** in a single-address-space
- **Program counter** based access control

From \ to	Protected			Unprotected
	Entry	Code	Data	
Protected	r-x	r-x	rw-	rwx
Unprotected / other SM	r-x	r--	---	rwx

Strackx et al.: "Efficient Isolation of Trusted Subsystems in Embedded Systems", 2010 [SPP10].

Background: Protected Module Architectures

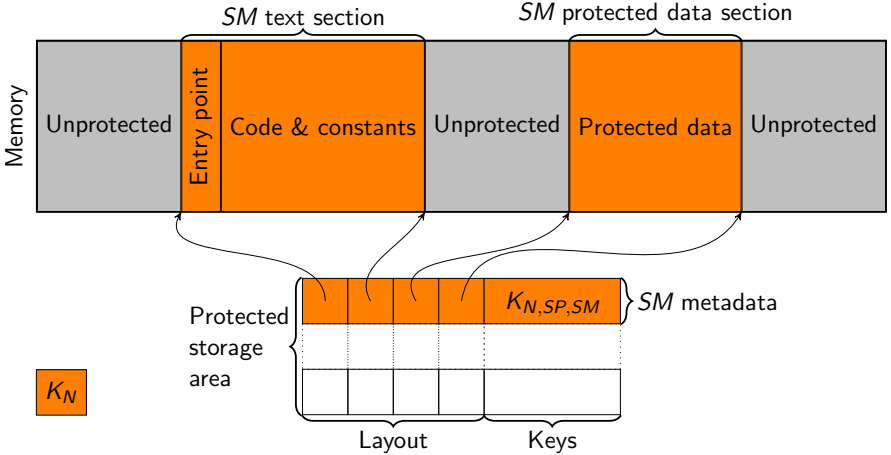


- **Isolated execution** in a single-address-space
- **Program counter** based access control
- Secure fully abstract **compilation**

From \ to	Protected			Unprotected
	Entry	Code	Data	
Protected	`r-x`	`r-x`	`rw-`	`rwx`
Unprotected / other SM	`r-x`	`r--`	`---`	`rwx`

Strackx et al.: "Efficient Isolation of Trusted Subsystems in Embedded Systems", 2010 [SPP10].
Agten et al.: "Secure Compilation to Modern Processors", 2012 [ASJP12].

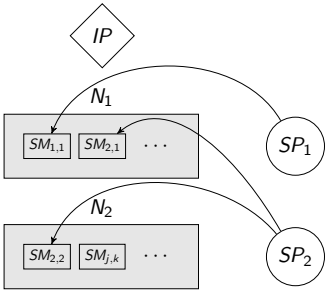
Sancus Enclaved Execution Processor [NAD+13, NVBM+17]



Noorman et al.: "Sancus 2.0: A Low-Cost Security Architecture for IoT Devices", 2017 [NVBM+17].

Establishing Trust with Sancus

System model: remote software providers (SP) deploy sofware modules (SM) on nodes (N) maintained by a (trusted) infrastructure provider (IP)

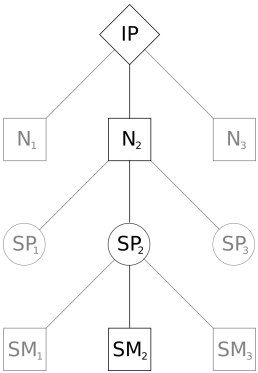


Establishing Trust with Sancus

System model: remote software providers (*SP*) deploy sofware modules (*SM*) on nodes (*N*) maintained by a (trusted) infrastructure provider (*IP*)

Key hierachy: Sancus processor derives module-specific key $K_{N,SP,SM}$ from node-master key K_N:

- based on *hash(text section) + layout SM + SP id*
- stored in CPU: exclusively accessible from *within SM*
- *shared secret* with *SP*: symmetric derivation from $K_{N,SP}$

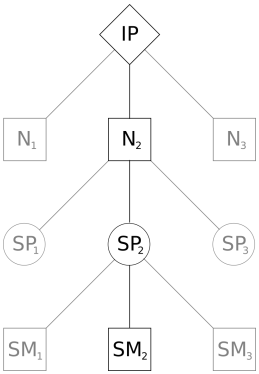


Establishing Trust with Sancus

System model: remote software providers (SP) deploy sofware modules (SM) on nodes (N) maintained by a (trusted) infrastructure provider (IP)

Key hierachy: Sancus processor derives module-specific key $K_{N,SP,SM}$ from node-master key K_N:

- based on *hash(text section) + layout SM + SP id*
- stored in CPU: exclusively accessible from *within SM*
- *shared secret* with SP: symmetric derivation from $K_{N,SP}$



Attestation Principle

$\Rightarrow$ *Use of* $K_{N,SP,SM}$ *proves integrity and isolation of SM deployed by SP on N*

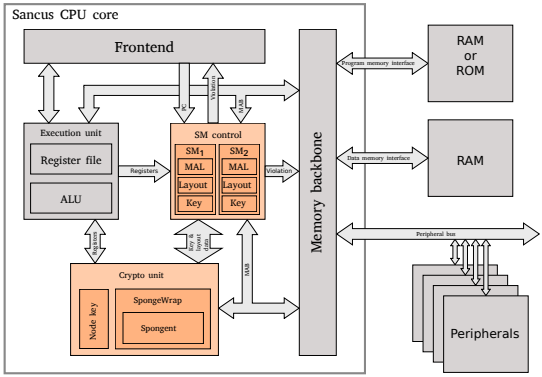
Sancus Enclaved Execution Platform [NAD+13, NVBM+17]

Strong security primitives for openMSP430

- Software component isolation
- Cryptography & attestation
- Secure memory-mapped I/O

Efficient

- Area: $\leq$ 2 kLUTs
- Real-Time: Authentication in μs
- Low-Power: + 6% increase



Sancus is fully **free and open-source**:

- `https://distrinet.cs.kuleuven.be/software/sancus/`
- `https://github.com/sancus-pma`

Sancus: Lightweight and Open-Source Trusted Computing for the IoT

View on GitHub | Watch a demo | Explore Research

> *A project based on open-source building blocks and free-software ethos that attempts to provide a layer of integrity and deterministic behavior to microcontrollers should be lauded and considered by anyone building hardware applications where security and reliability are strong requirements.*
>
> — *Mischa Spiegelmock* ***LWN.net***

SOFTWARE ISOLATION

Outside software cannot read or write a protected module's runtime state. A module can only be called through one of its designated entry points.

LIGHTWEIGHT CRYPTOGRAPHY

A minimalist cryptographic hardware unit enables low-overhead symmetric key derivation, authenticated encryption, and hashing.

SOFTWARE ATTESTATION

Remote or local parties can verify at runtime that a particular software module has been isolated on a specific node without having been tampered with.

SECURE COMMUNICATION

Sancus safeguards the authenticity, integrity, and freshness of all traffic between a protected module and its remote provider.

SECURE I/O

Secure driver modules have exclusive ownership over memory-mapped I/O peripheral devices, and can implement software-defined access control policies.

BACKWARDS COMPATIBILITY

Legacy applications continue to function as expected; critical components can be migrated gradually into Sancus-protected modules.

Road Map

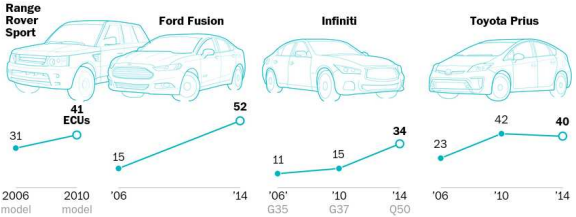
The State of Automotive Security

Recent Incidents

- PoC remote takeover of a Jeep on the highway [MV15]
- CIA revelations [Wik]

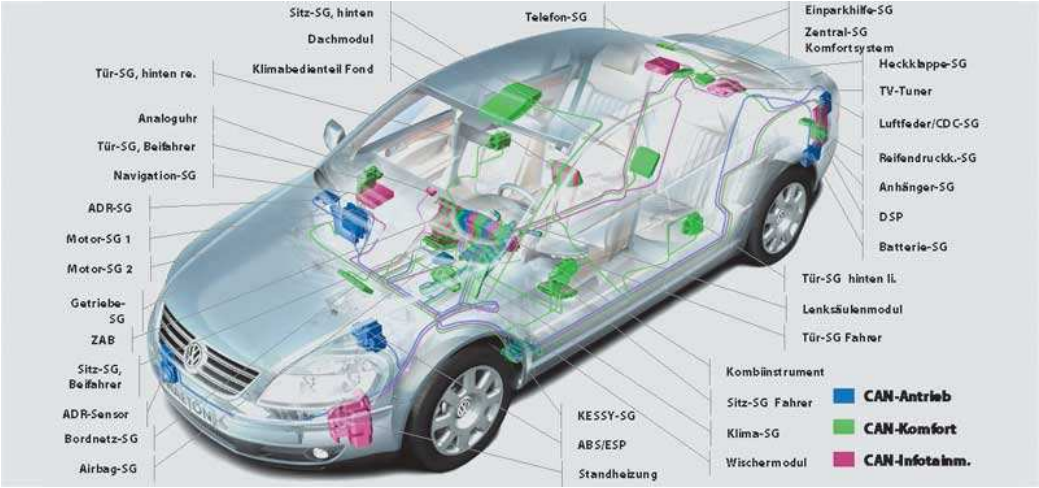
Why?

- CAN communication protocol (°'86): all devices considered trusted
- Remote connectivity, infotainment systems, self-driving cars
- Increasing ECU usage: over 50 interconnected microcontrollers



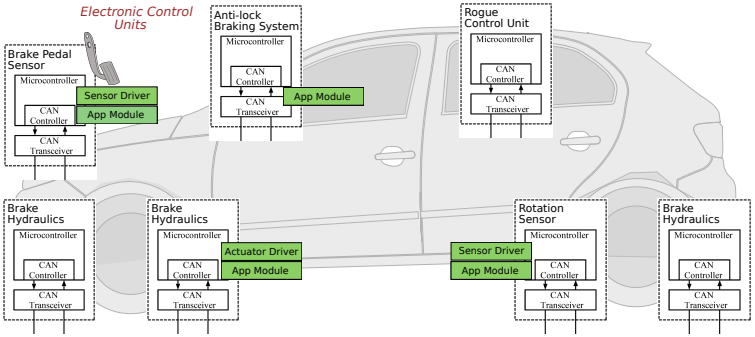
Source: `http://wapo.st/1eguLvk`

Controller Area Network (CAN) Bus



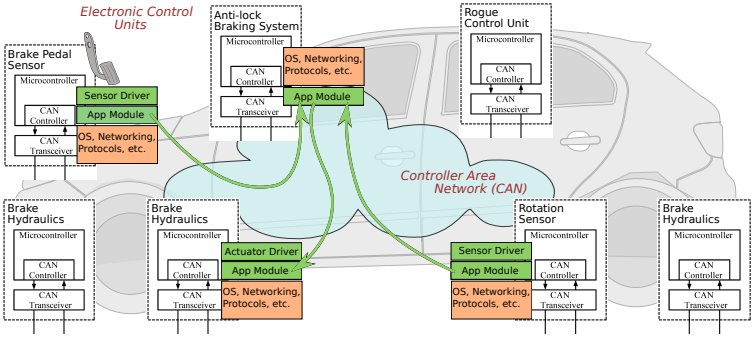
Source: https://homepages.thm.de/~hg10013/Lehre/MMS/WS0304/Ludwig/bild05.jpg

Overview: Vulcanising Distributed Automotive Applications



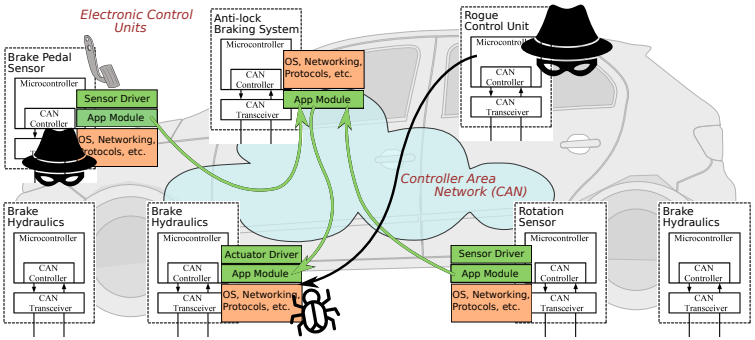
- Critical application components in **enclaves**: software isolation + attestation

Overview: Vulcanising Distributed Automotive Applications



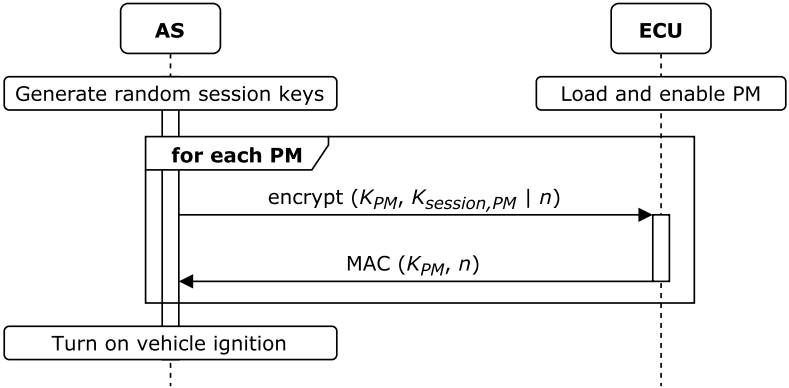
- Critical application components in **enclaves**: software isolation + attestation
- Authenticated CAN messages over untrusted system software/network

Overview: Vulcanising Distributed Automotive Applications



- Critical application components in **enclaves**: software isolation + attestation
- Authenticated CAN messages over untrusted system software/network
- Rogue ECUs, software attackers and errors in untrusted code cannot interfere with security, but may *harm availability*

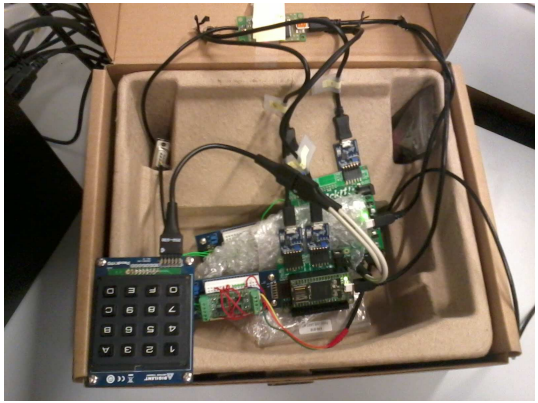
VulCAN Attestation Server: Boot + Session Key Provisioning



- Challenge-response attestation + encrypted session key distribution
- Preserve motorist safety via secure boot + exclusive vehicle ignition

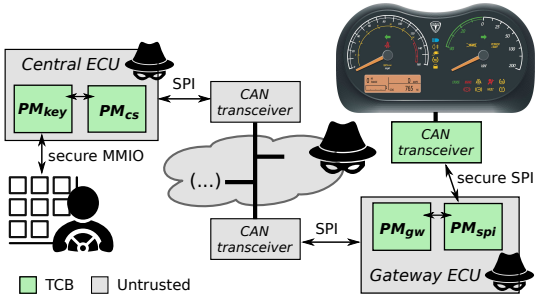
VulCAN Demo Scenario (https://distrinet.cs.kuleuven.be/software/vulcan/)

⇒ *distributed authenticated path from keypad to shielded instrument cluster*



VulCAN Demo Scenario (https://distrinet.cs.kuleuven.be/software/vulcan/)

⇒ *distributed authenticated path from keypad to shielded instrument cluster*



Future Work & Research Challenges

Port Sancus to **RISC-V** CPU:

→ replace SPONGEWRAP/SPONGENT with AES authenticated encryption hardware unit

Improve **LLVM compiler** support:

→ unified framework to interact/deploy enclaves on heterogeneous networked platforms

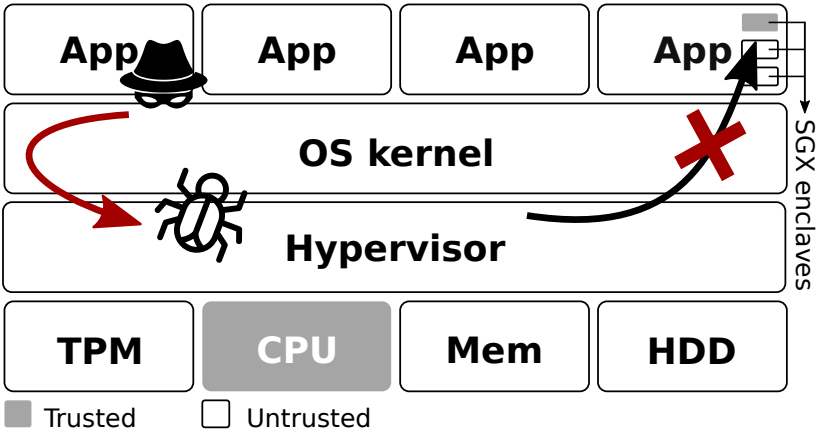
Availability and real-time guarantees on compromised ECUs

→ protected scheduler [VBNMP16]; network availability out of scope (or partition via gateway)

Formal verification of software modules [JSP+11] + hardware designs [ZWSM15]

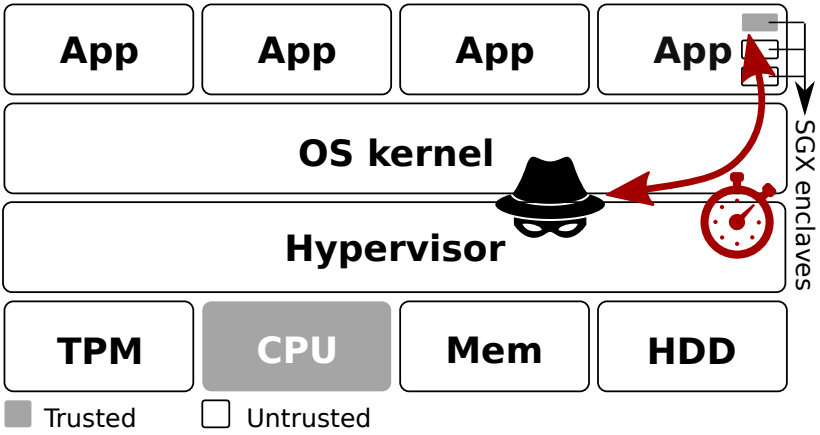
Road Map

The Big Picture: Application Attack Surface



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

The Big Picture: Application Attack Surface

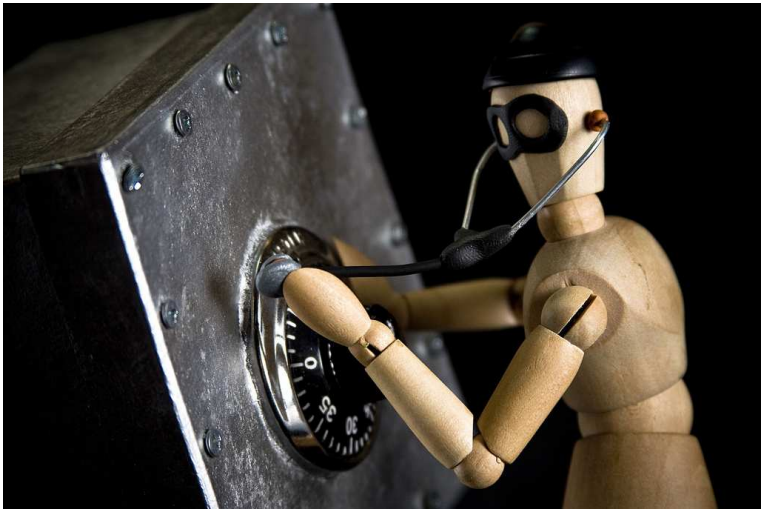


Untrusted OS → new class of powerful **side-channels**

Side-Channel Attack Principle

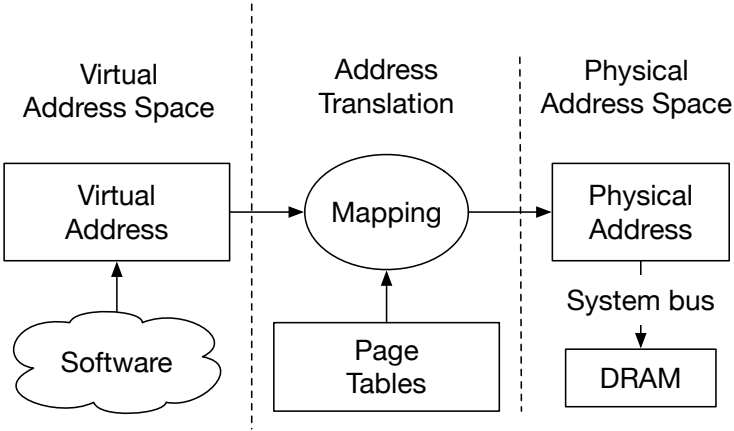
Source: `https://commons.wikimedia.org/wiki/File:WinonaSavingsBankVault.JPG`

Side-Channel Attack Principle



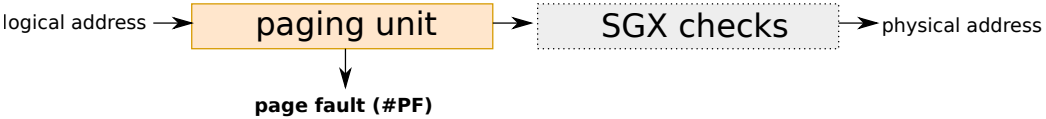
Source: https://flic.kr/p/69sHDa

The Virtual Memory Abstraction



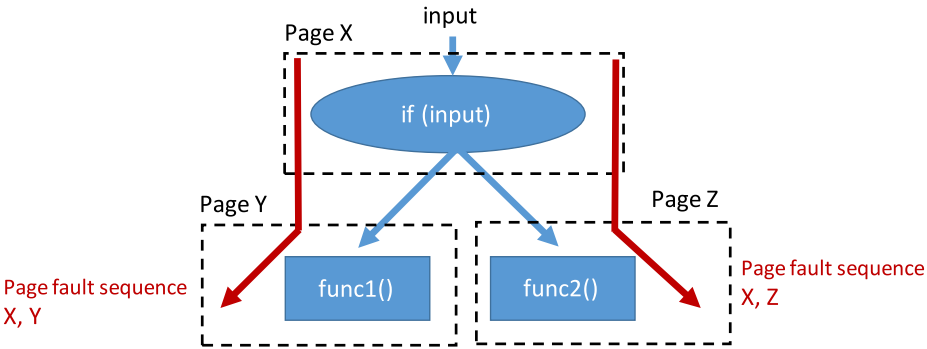
Costan et al. "Intel SGX explained", IACR 2016 [CD16]

Page Faults as a Side-Channel



⇒ Untrusted address translation may **fault** during enclaved execution (!)

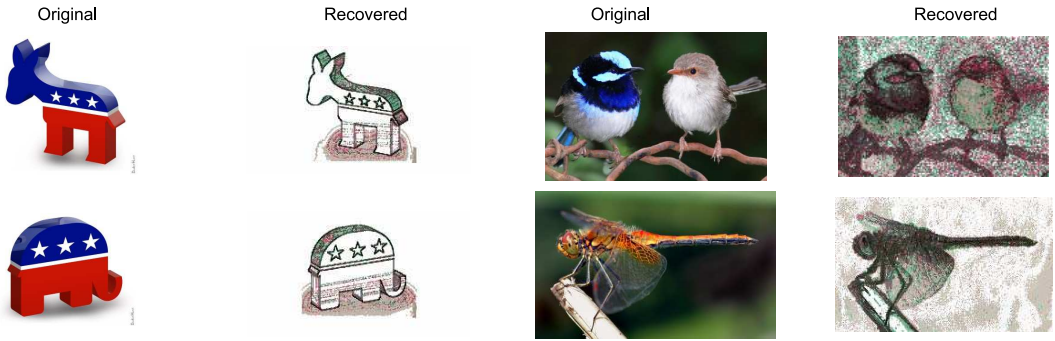
Page Faults as a Side-Channel



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

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

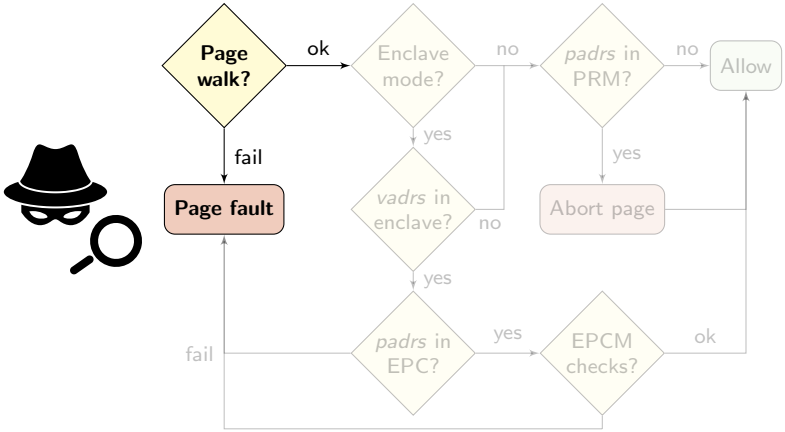
Page Faults as a Side-Channel



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

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

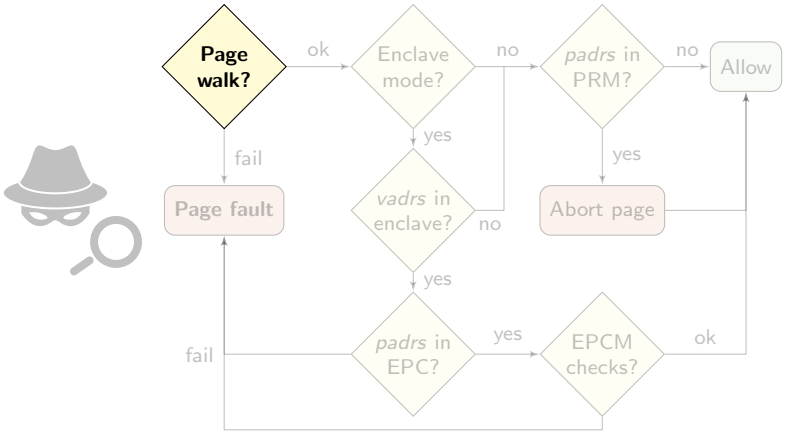
Current Solutions: Hiding Enclave Page Faults



Shih et al. "T-SGX: Eradicating controlled-channel attacks against enclave programs", NDSS 2017 [SLKP17]

Shinde et al. "Preventing page faults from telling your secrets", AsiaCCS 2016 [SCNS16]

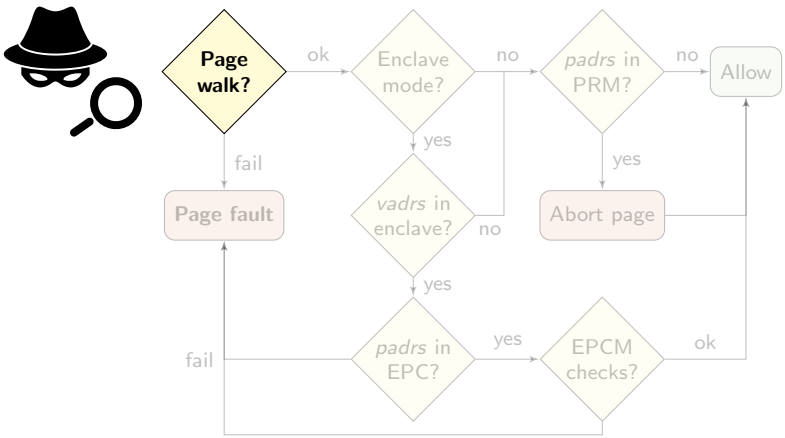
Current Solutions: Hiding Enclave Page Faults



Shih et al. "T-SGX: Eradicating controlled-channel attacks against enclave programs", NDSS 2017 [SLKP17]

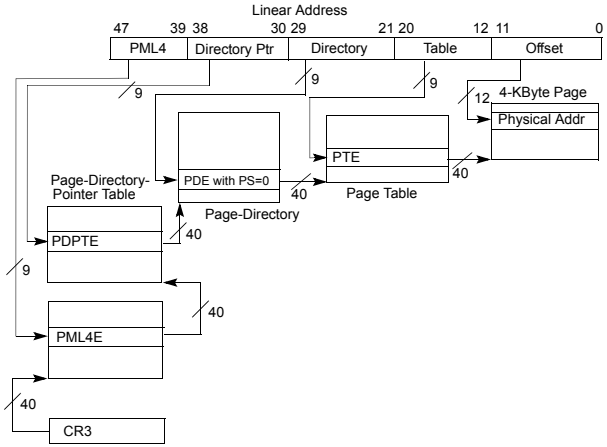
Shinde et al. "Preventing page faults from telling your secrets", AsiaCCS 2016 [SCNS16]

Current Solutions: Hiding Enclave Page Faults



Defenses do not hold when attacker learns page accesses without triggering faults!

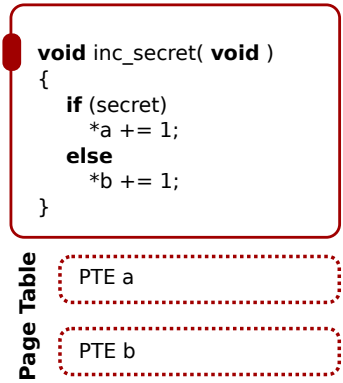
Current Solutions: Hiding Enclave Page Faults



Defenses do not hold when attacker learns page accesses without triggering faults!

Telling Your Secrets Without Page Faults

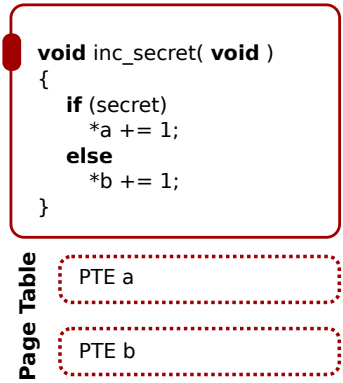
1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit



Telling Your Secrets Without Page Faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

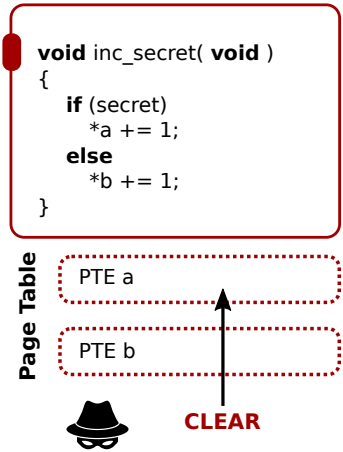
   ⇝ Also updated in enclave mode!



Telling Your Secrets Without Page Faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

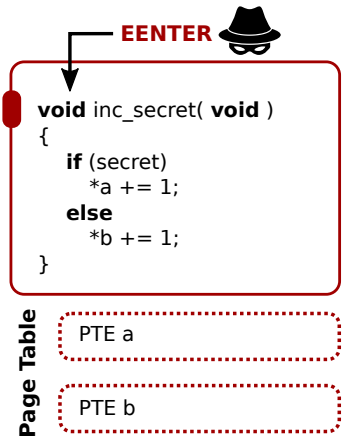
   ⇝ Also updated in enclave mode!



Telling Your Secrets Without Page Faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

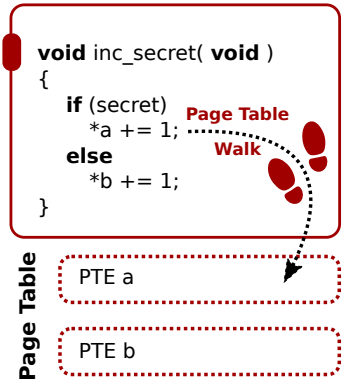
   ⇝ Also updated in enclave mode!



Telling Your Secrets Without Page Faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

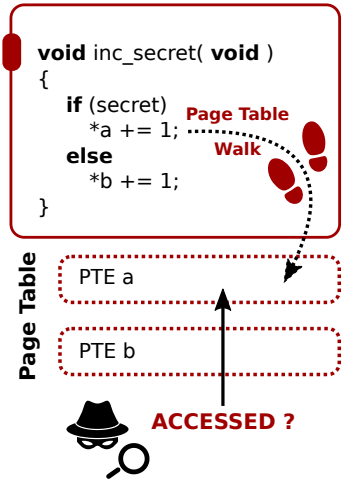
   ⇝ Also updated in enclave mode!



Telling Your Secrets Without Page Faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

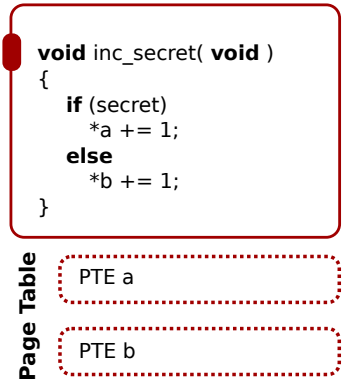


Telling Your Secrets Without Page Faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

2. Attack vector: Unprotected **page table memory**:
   - Cached as regular data
   - Accessed during address translation



Telling Your Secrets Without Page Faults

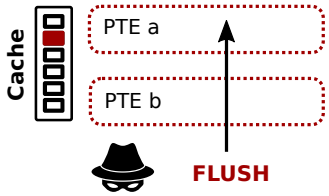
1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

2. Attack vector: Unprotected **page table memory**:
   - Cached as regular data
   - Accessed during address translation

   ⇝ Flush+Reload cache timing attack!

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



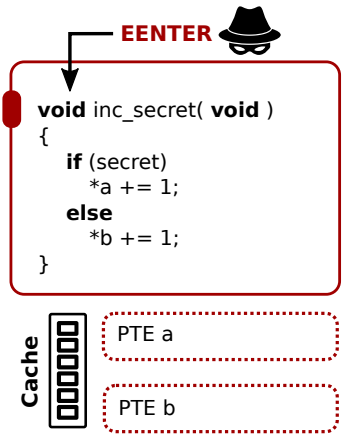
Telling Your Secrets Without Page Faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

2. Attack vector: Unprotected **page table memory**:
   - Cached as regular data
   - Accessed during address translation

   ⇝ Flush+Reload cache timing attack!



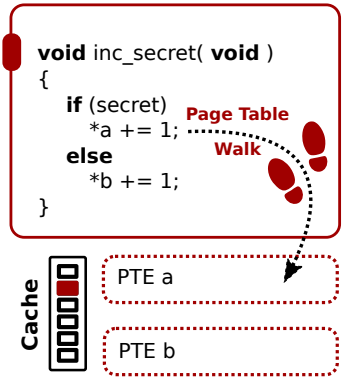
Telling Your Secrets Without Page Faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

2. Attack vector: Unprotected **page table memory**:
   - Cached as regular data
   - Accessed during address translation

   ⇝ Flush+Reload cache timing attack!



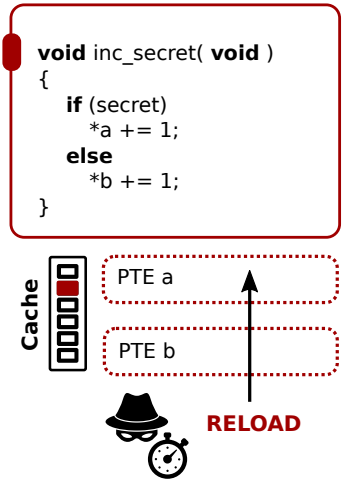
Telling Your Secrets Without Page Faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

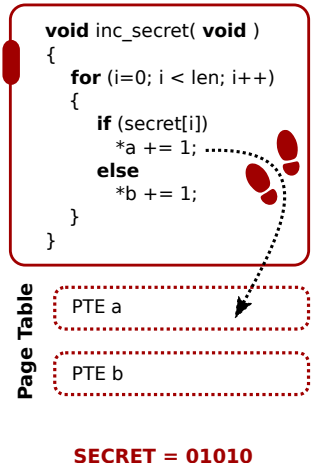
2. Attack vector: Unprotected **page table memory**:
   - Cached as regular data
   - Accessed during address translation

   ⇝ Flush+Reload cache timing attack!



#PF-Less Challenges: Monitoring Repeated Accesses

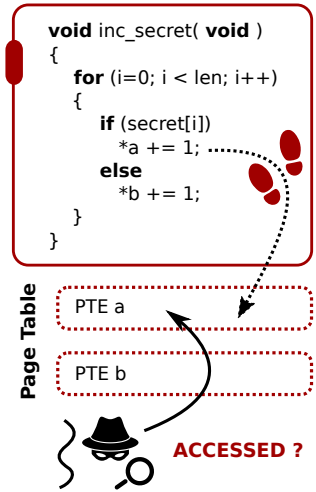
1. Challenge: No #PF on memory access



#PF-Less Challenges: Monitoring Repeated Accesses

1. Challenge: No #PF on memory access

   ⇝ Monitor PTEs from concurrent **spy thread**

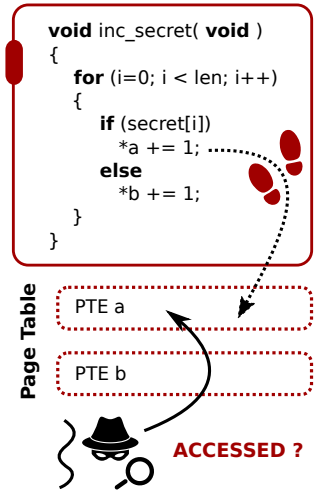


#PF-Less Challenges: Monitoring Repeated Accesses

1. Challenge: No #PF on memory access

   ⇝ Monitor PTEs from concurrent **spy thread**

2. Challenge: Translation Lookaside Buffer (TLB)



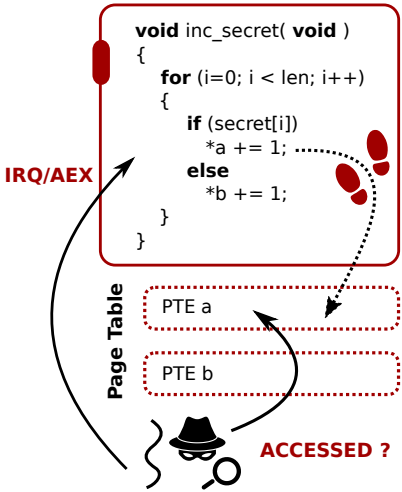
#PF-Less Challenges: Monitoring Repeated Accesses

1. Challenge: No #PF on memory access

   ⇝ Monitor PTEs from concurrent **spy thread**

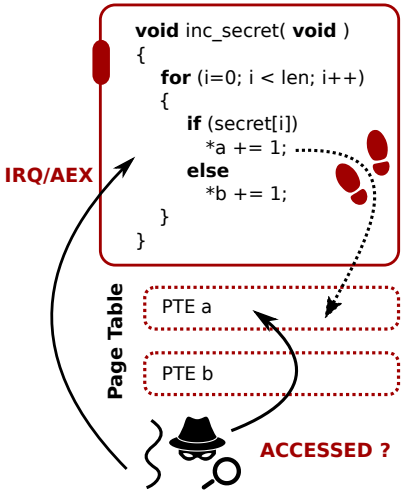
2. Challenge: Translation Lookaside Buffer (TLB)

   ⇝ Directed **Inter-Processor Interrupt**



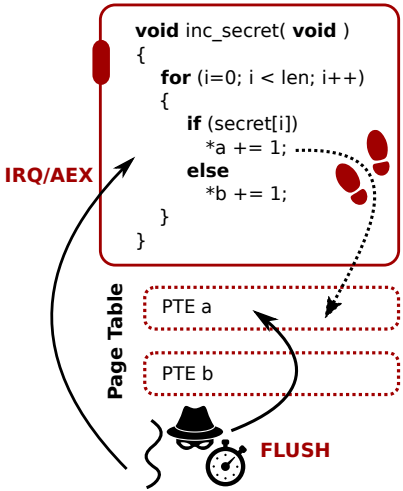
#PF-Less Challenges: Monitoring Repeated Accesses

1. Challenge: No #PF on memory access
   ⇝ Monitor PTEs from concurrent **spy thread**
2. Challenge: Translation Lookaside Buffer (TLB)
   ⇝ Directed **Inter-Processor Interrupt**
3. Challenge: Temporal resolution (IPI latency)



#PF-Less Challenges: Monitoring Repeated Accesses

1. Challenge: No #PF on memory access
   ⇝ Monitor PTEs from concurrent **spy thread**
2. Challenge: Translation Lookaside Buffer (TLB)
   ⇝ Directed **Inter-Processor Interrupt**
3. Challenge: Temporal resolution (IPI latency)
   ⇝ Precise **Flush+Flush** technique



PTE Flush+Flush: A High-Resolution, Low-Latency Channel

Challenge: monitoring repeated accesses

Interrupt victim enclave via **spy thread**

$\exists$ access **detection latency** $\leftrightarrow$ #PF-attacks

PTE Flush+Flush: A High-Resolution, Low-Latency Channel

Challenge: monitoring repeated accesses

Interrupt victim enclave via **spy thread**
$\exists$ access **detection latency** $\leftrightarrow$ #PF-attacks

Interrupt granularity:

☹ A/D monitoring: $\sim$ 430 `nop` / $\sim$ 175 `add`

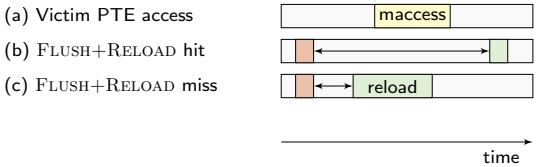
PTE Flush+Flush: A High-Resolution, Low-Latency Channel

Challenge: monitoring repeated accesses

Interrupt victim enclave via **spy thread**
$\exists$ access **detection latency** $\leftrightarrow$ #PF-attacks

Interrupt granularity:

- ☹ A/D monitoring: $\sim$ 430 `nop` / $\sim$ 175 `add`
- ☹ Flush+Reload: might miss victim access (TLB!)



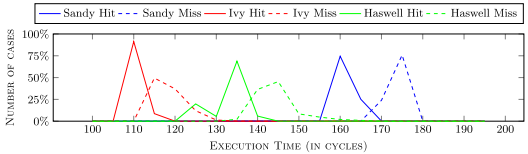
PTE Flush+Flush: A High-Resolution, Low-Latency Channel

Challenge: monitoring repeated accesses

Interrupt victim enclave via **spy thread**
$\exists$ access **detection latency** $\leftrightarrow$ #PF-attacks

Interrupt granularity:

- ☹ A/D monitoring: $\sim$ 430 `nop` / $\sim$ 175 `add`
- ☹ Flush+Reload: might miss victim access (TLB!)
- ☺ Flush+Flush: `clflush` completes earlier for uncached data



Gruss et al. "Flush+Flush: a fast and stealthy cache attack", DIMVA 2016 [GMWM16]

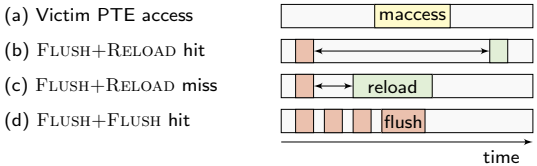
PTE Flush+Flush: A High-Resolution, Low-Latency Channel

Challenge: monitoring repeated accesses

Interrupt victim enclave via **spy thread**
$\exists$ access **detection latency** $\leftrightarrow$ #PF-attacks

Interrupt granularity:

- ☹ A/D monitoring: $\sim$ 430 `nop` / $\sim$ 175 `add`
- ☹ Flush+Reload: might miss victim access (TLB!)
- ☺ Flush+Flush: interrupt *within trigger instruction* ($> 99.8\%$)

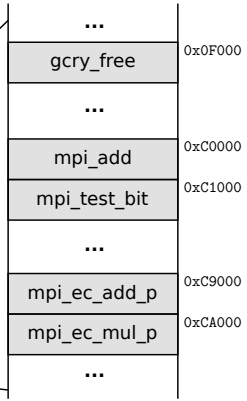


Attacking Libgcrypt EdDSA (simplified)

```
1  if (mpi_is_secure (scalar)) {
2      /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4      point_init (&tmppnt);
5
6      for (j=nbits-1; j >= 0; j--) {
7          _gcry_mpi_ec_dup_point (result, result, ctx);
8          _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
9          point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
10     }
11     point_free (&tmppnt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```

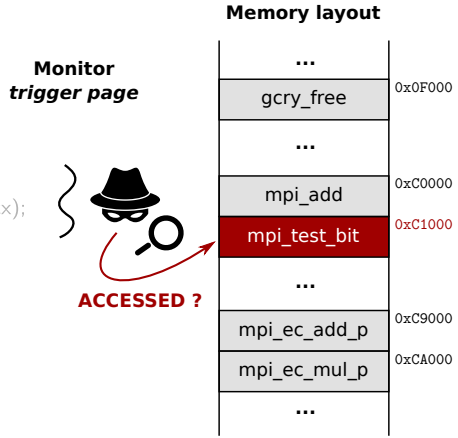
22 Code pages per iteration

Memory layout



Attacking Libgcrypt EdDSA (simplified)

```
1  if (mpi_is_secure (scalar)) {
2      /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4      point_init (&tmppnt);
5
6      for (j=nbits-1; j >= 0; j--) {
7          _gcry_mpi_ec_dup_point (result, result, ctx);
8          _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
9          point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
10     }
11     point_free (&tmppnt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```

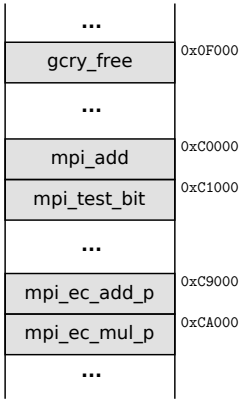


Attacking Libgcrypt EdDSA (simplified)

```
1  if (mpi_is_secure (scalar)) {
2      /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4      point_init (&tmppnt);
5
6      for (j=nbits-1; j >= 0; j--) {
7          _gcry_mpi_ec_dup_point (result, result, ctx);
8          _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
9          point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
10     }
11     point_free (&tmppnt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```

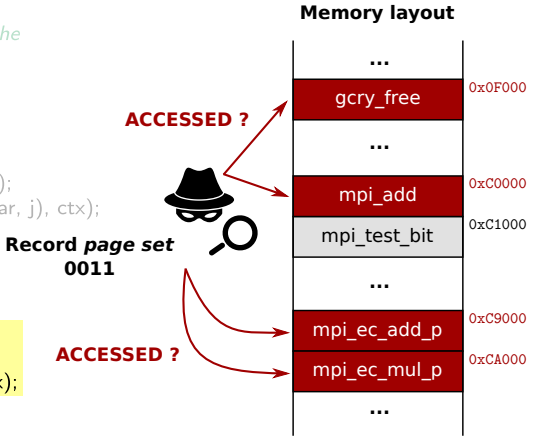
INTERRUPT

Memory layout



Attacking Libgcrypt EdDSA (simplified)

```
1  if (mpi_is_secure (scalar)) {
2      /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4      point_init (&tmppnt);
5
6      for (j=nbits-1; j >= 0; j--) {
7          _gcry_mpi_ec_dup_point (result, result, ctx);
8          _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
9          point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
10     }
11     point_free (&tmppnt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```



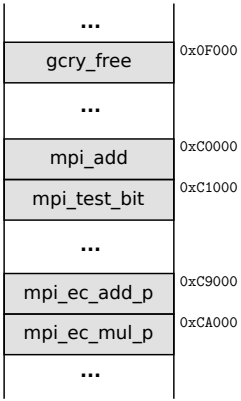
Attacking Libgcrypt EdDSA (simplified)

```
1  if (mpi_is_secure (scalar)) {
2      /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4      point_init (&tmppnt);
5
6      for (j=nbits-1; j >= 0; j--) {
7          _gcry_mpi_ec_dup_point (result, result, ctx);
8          _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
9          point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
10     }
11     point_free (&tmppnt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```

RESUME

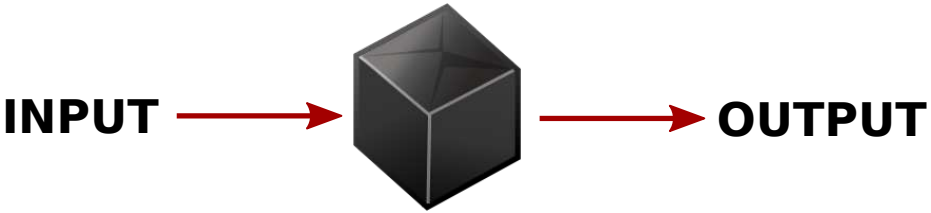
Full 512-bit key recovery, single run

Memory layout

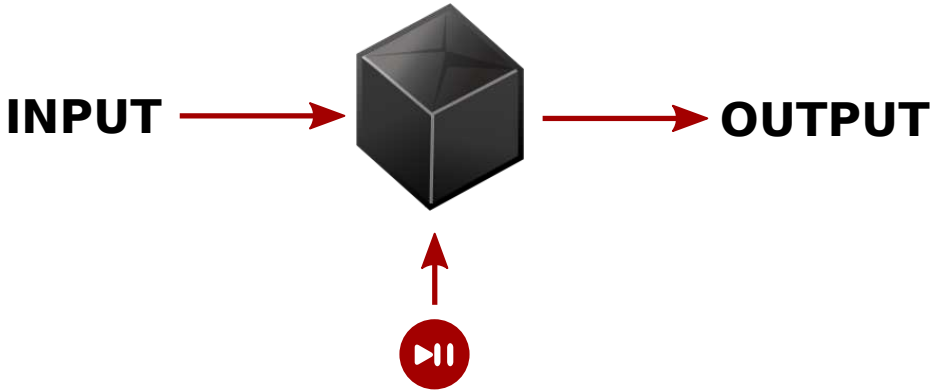


Road Map

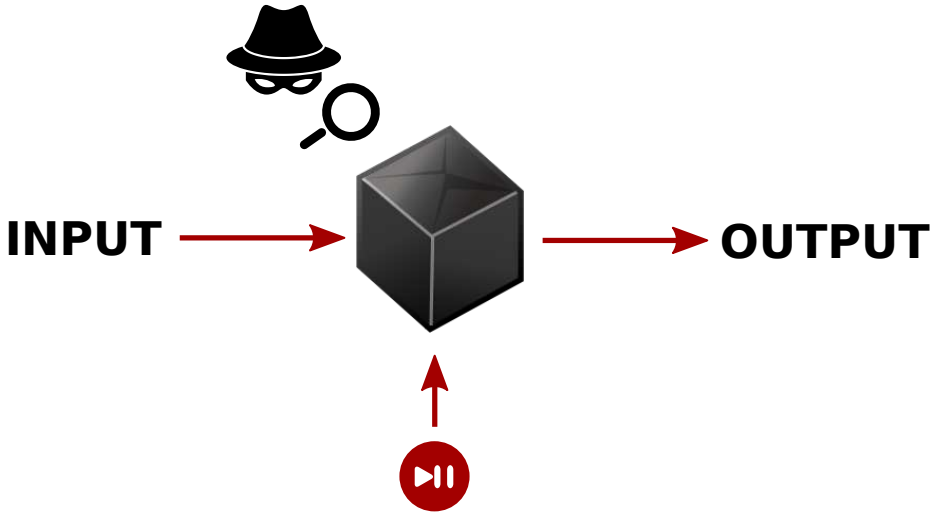
Enclaves as a Black Box



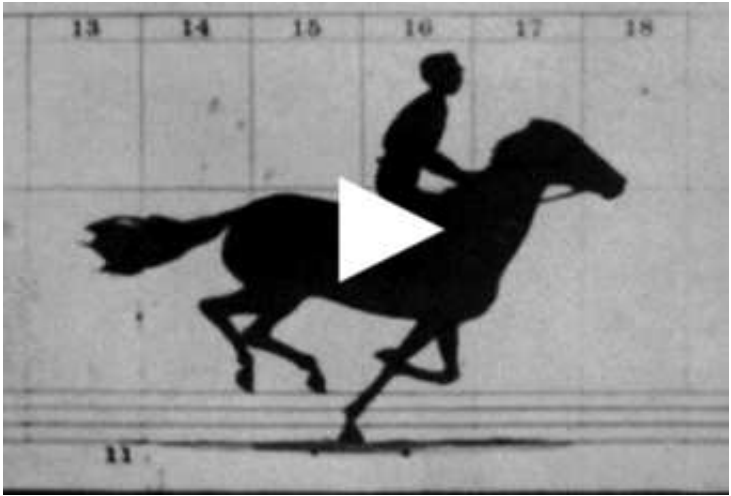
Enclaves as a Black Box



Enclaves as a Black Box

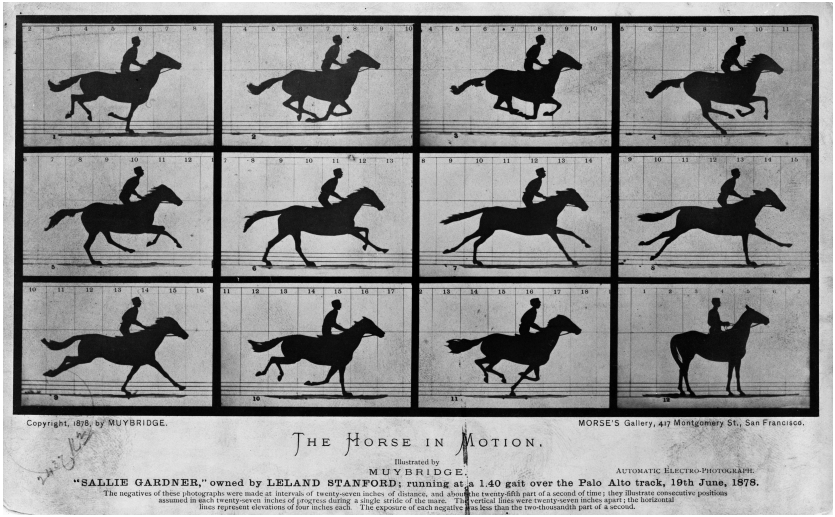


The Galloping Enclave Analogy

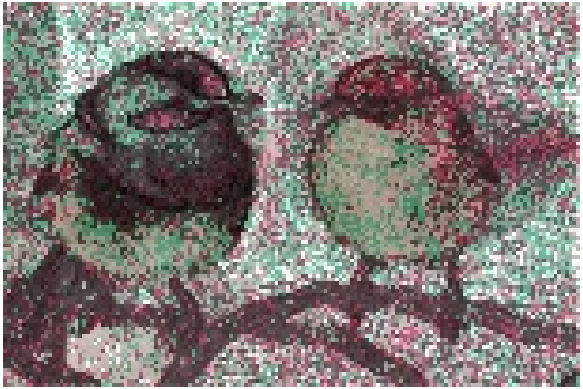


Source: https://en.wikipedia.org/wiki/Sallie_Gardner_at_a_Gallop

The Galloping Enclave Analogy



High Resolution Side-Channels in Practice



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

$\Rightarrow$ *Coarse-grained preemption (4 KB page leakage)*

High Resolution Side-Channels in Practice

Hähnel et al.: "High-resolution side channels for untrusted operating systems", ATC 2017 [HCP17]

$\Rightarrow$ *Fine-grained preemption (64 B cache line leakage)*

Timer-Based Attacks

SGX-Step Goal: executing enclaves one instruction at a time

Timer-Based Attacks

SGX-Step Goal: executing enclaves one instruction at a time

Frequent enclave preemption challenge:

- ☹ x86 HW *debug features* disabled in enclave mode
- ☺ . . . but full control over **timer devices**/scheduling

Timer-Based Attacks

SGX-Step Goal: executing enclaves one instruction at a time

Frequent enclave preemption challenge:

- ☹ x86 HW *debug features* disabled in enclave mode
- ☺ . . . but full control over **timer devices**/scheduling

⇒ User space virtual **memory mappings** for x86 APIC + page table entries

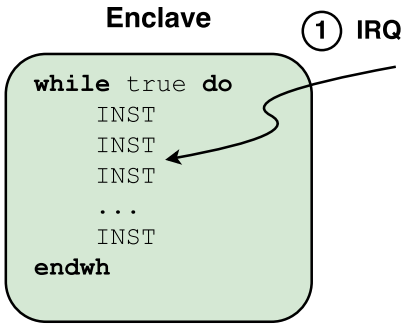
```
jo@sgx-laptop:~$ cat /proc/iomem | grep "Local APIC"
fee00000-fee00fff : Local APIC
jo@sgx-laptop:~$ sudo devmem2 0xFEE00030 h
/dev/mem opened.
Memory mapped at address 0x7f37dc187000.
Value at address 0xFEE00030 (0x7f37dc187030): 0x15
jo@sgx-laptop:~$
```

Interrupting and Resuming Enclaves

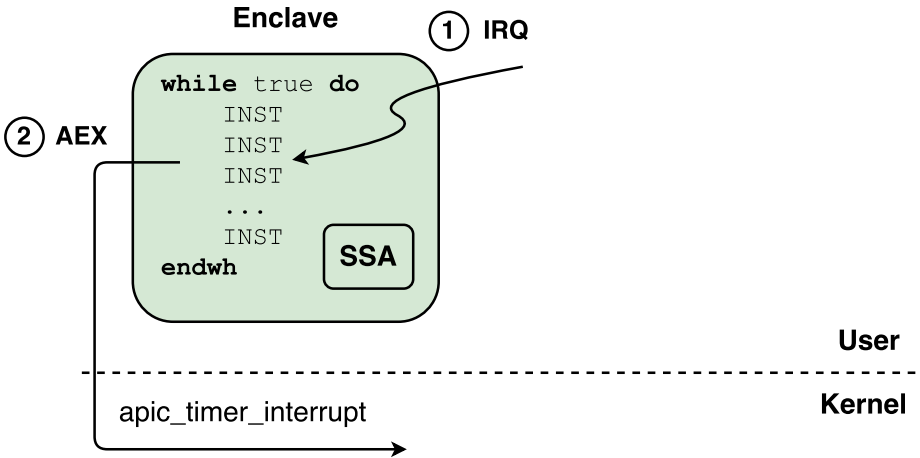
Enclave

```
while true do
    INST
    INST
    INST
    ...
    INST
endwh
```

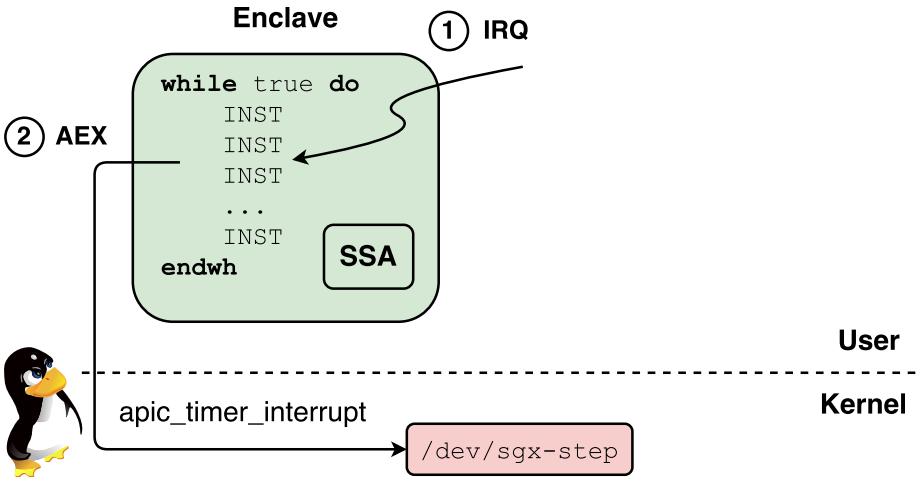
Interrupting and Resuming Enclaves



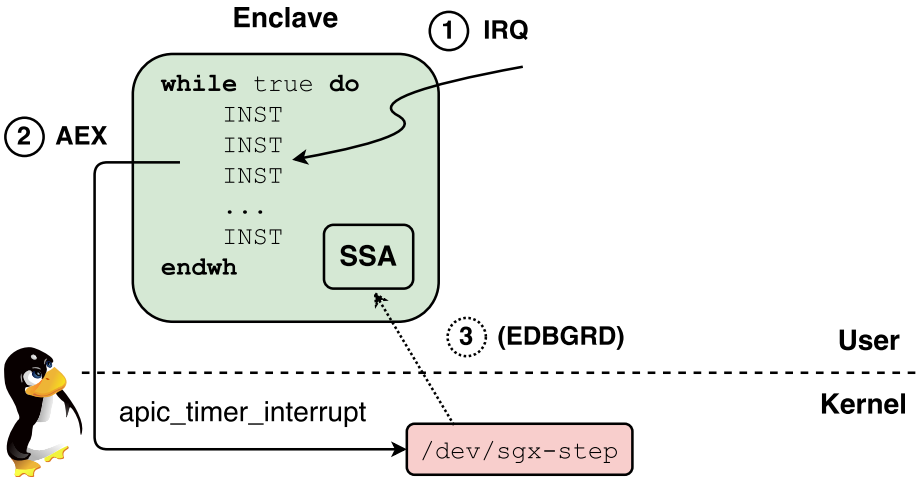
Interrupting and Resuming Enclaves



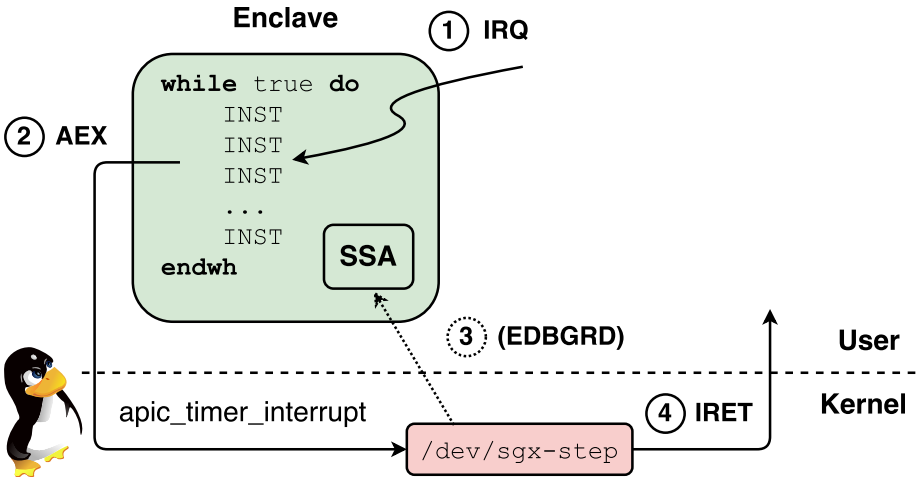
Interrupting and Resuming Enclaves



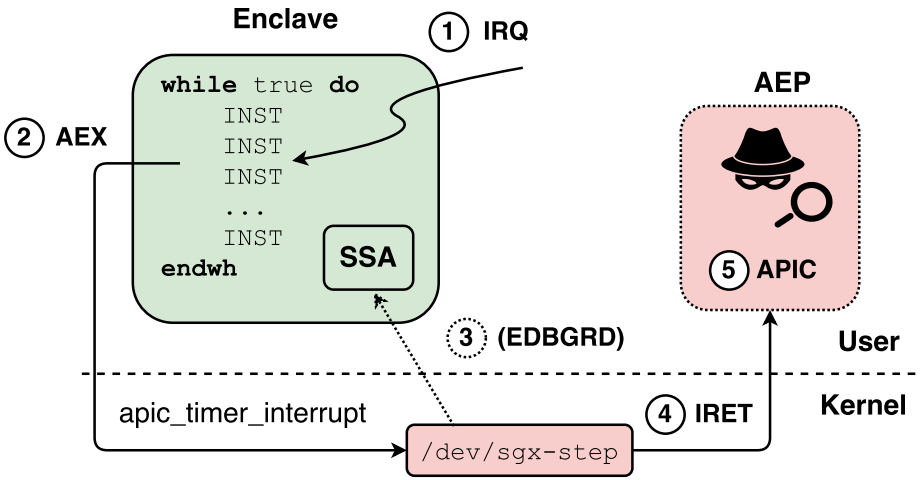
Interrupting and Resuming Enclaves



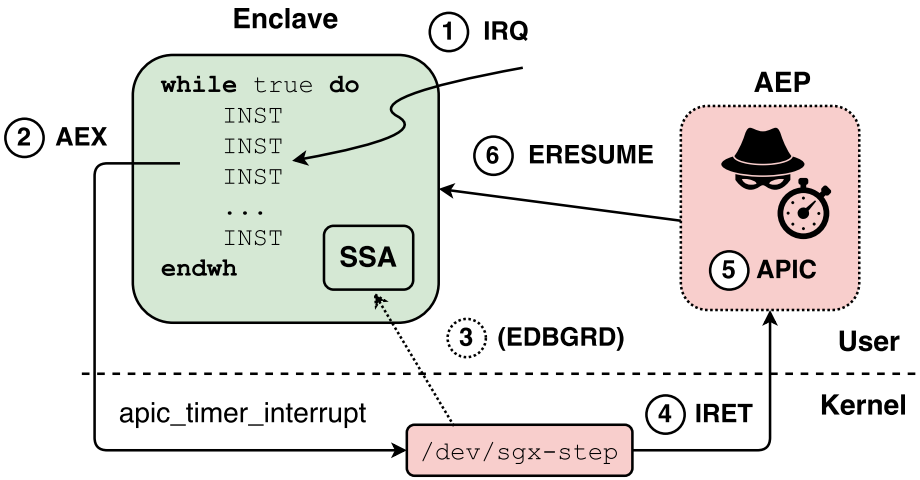
Interrupting and Resuming Enclaves



Interrupting and Resuming Enclaves



Interrupting and Resuming Enclaves



Increasing the Resolution of Page Table-Based Attacks

`strlen` loop

Note: page fault-driven attacks cannot make progress

```
1  size_t strlen (char *str)
2  {
3    char *s;
4
5    for (s = str; *s; ++s);
6    return (s - str);
7  }
```

```
1      mov  %rdi,%rax
2  1:  cmpb $0x0,(%rax)
3      je   2f
4      inc  %rax
5      jmp  1b
6  2:  sub  %rdi,%rax
7      retq
```

Increasing the Resolution of Page Table-Based Attacks

strlen loop

Note: page fault-driven attacks cannot make progress

```
1  size_t strlen (char *str)
2  {
3    char *s;
4
5    for (s = str; *s; ++s);
6    return (s - str);
7  }
```

```
1     mov  %rdi,%rax
2  1: cmpb $0x0,(%rax)
3     je   2f
4     inc  %rax
5     jmp  1b
6  2: sub  %rdi,%rax
7     retq
```

⇒ tight loop: 4 instructions, single memory operand

Increasing the Resolution of Page Table-Based Attacks

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.

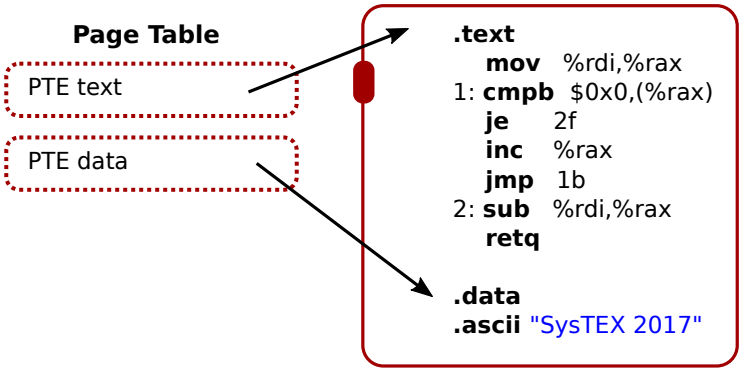
More important, the application enclave should use an appropriate crypto implementation that is side channel attack resistant inside the enclave if side-channel attacks are a concern.

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

Attacking `strlen`

Page fault adversary

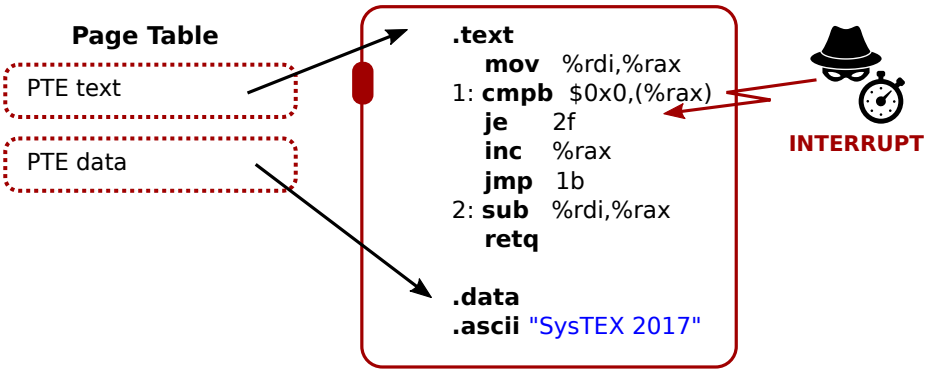
Progress ⇒ both code + data pages present ☹



Attacking `strlen`

Single-stepping adversary

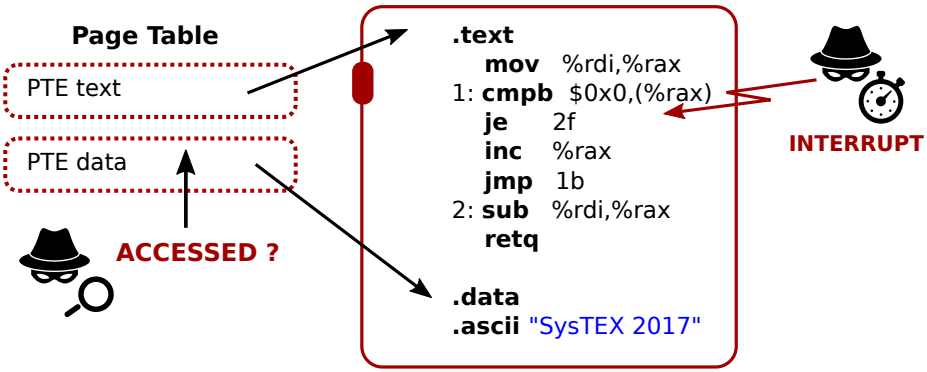
Execute + interrupt ⇒ data page accessed ? ☺



Attacking `strlen`

Single-stepping adversary

Execute + interrupt ⇒ data page accessed ? ☺



High-Resolution Side-Channels in Practice (revisited)

2 Background

On February 16th, 2018, a team of security researchers at Catholic University of Leuven (KU Leuven) disclosed to Intel Corporation an issue with Edger8r Tool within the Intel® Software Guard Extensions (Intel® SGX) Software Developer's Kit (SDK). This issue could cause the Edger8r tool to generate source code that could, when used as intended within an SGX enclave, expose the enclave to a side-channel attack. The attack would then have the potential to disclose confidential data within the enclave.

Source: `https://software.intel.com/sites/default/files/managed/e1/ec/180309_SGX_SDK_Developer_Guidance_Edger8r.pdf`

High-Resolution Side-Channels in Practice (revisited)

```
static sgx_status_t SGX_CDECL sgx_ecall_pointer_string(void* pms)
{
    CHECK_REF_POINTER(pms, sizeof(ms_ecall_pointer_string_t));
    ms_ecall_pointer_string_t* ms =
                SGX_CAST(ms_ecall_pointer_string_t*, pms);
    sgx_status_t status = SGX_SUCCESS;
    char* _tmp_str = ms->ms_str;
    size_t _len_str = _tmp_str ? strlen(_tmp_str) + 1 : 0;
    char* _in_str = NULL;

    CHECK_UNIQUE_POINTER(_tmp_str, _len_str);
```

⇒ *Reliably locate all 0x00 bytes in enclave memory (!)*

Road Map

Conclusion

Take-Away Message

Hardware-based security primitives to *isolate* and *attest* enclaved application logic

Conclusion

Take-Away Message

Hardware-based security primitives to *isolate* and *attest* enclaved application logic

But . . .

⇒ Need trustworthy hardware:

- Free and open-source processors (e.g., Sancus [NVBM+17], RISC-V, CHERI [WWC+14])
- Formal verification (e.g., secVerilog [ZWSM15])

Conclusion

Take-Away Message

Hardware-based security primitives to *isolate* and *attest* enclaved application logic

But . . .

⇒ Need trustworthy hardware:

- Free and open-source processors (e.g., Sancus [NVBM+17], RISC-V, CHERI [WWC+14])
- Formal verification (e.g., secVerilog [ZWSM15])

⇒ Watch out for side-channels:

- Architecture: e.g., unprotected page table memory
- Application: e.g., secret-dependent control/data flow

Thank you! Questions?

https://distrinet.cs.kuleuven.be/software/sancus/

https://github.com/jovanbulck

References I

P. Agten, R. Strackx, B. Jacobs, and F. Piessens.
Secure compilation to modern processors.
In *Computer Security Foundations Symposium*, pp. 171–185. IEEE, 2012.

V. Costan and S. Devadas.
Intel SGX explained.
Technical report, Computer Science and Artificial Intelligence Laboratory MIT, 2016.
`https://eprint.iacr.org/2016/086.pdf`.

D. Gruss, C. Maurice, K. Wagner, and S. Mangard.
Flush+flush: A fast and stealthy cache attack.
In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2016.

M. Hähnel, W. Cui, and M. Peinado.
High-resolution side channels for untrusted operating systems.
In *2017 USENIX Annual Technical Conference*, ATC '17. USENIX Association, 2017.

B. Jacobs, J. Smans, P. Philippaerts, F. Vogels, W. Penninckx, and F. Piessens.
VeriFast: A powerful, sound, predictable, fast verifier for C and Java.
In *NASA Formal Methods 2011*, vol. 6617 of *LNCS*, pp. 41–55, Heidelberg, 2011. Springer.

P. Maene, J. Götzfried, R. de Clercq, T. Müller, F. Freiling, and I. Verbauwhede.
Hardware-based trusted computing architectures for isolation and attestation.
IEEE Transactions on Computers, (99), 2017.

References II

C. Miller and C. Valasek.
Remote exploitation of an unaltered passenger vehicle.
Black Hat USA, 2015.

J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens.
Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base.
In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pp. 479–494, Berkeley, CA, USA, 2013. USENIX Association.

J. Noorman, J. Van Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. Freiling.
Sancus 2.0: A low-cost security architecture for IoT devices.
ACM Transactions on Privacy and Security (TOPS), 2017.

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 (ASIA CCS)*, pp. 317–328. ACM, 2016.

M. Seaman.
Powering an msp430 from a single battery cel.
Technical report, Texas Instruments, September 2008.
`http://www.ti.com.cn/cn/lit/an/slaa398/slaa398.pdf`.

M.-W. Shih, S. Lee, T. Kim, and M. Peinado.
T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs.
In *24th Annual Network and Distributed System Security Symposium (NDSS)*, 2017.

References III

R. Strackx, F. Piessens, and B. Preneel.
Efficient isolation of trusted subsystems in embedded systems.
In *Security and Privacy in Communication Networks*, pp. 344–361. Springer, 2010.

J. Van Bulck, J. T. Mühlberg, and F. Piessens.
VulCAN: Efficient component authentication and software isolation for automotive control networks.
In *Proceedings of the 33th Annual Computer Security Applications Conference (ACSAC'17)*. ACM, 2017.

J. Van Bulck, J. Noorman, J. T. Mühlberg, and F. Piessens.
Secure resource sharing for embedded protected module architectures.
In *9th WISTP International Conference on Information Security Theory and Practice (WISTP'15)*, vol. 9311 of *LNCS*, pp. 71–87. Springer, 2015.

J. Van Bulck, J. Noorman, J. T. Mühlberg, and F. Piessens.
Towards availability and real-time guarantees for protected module architectures.
In *Companion Proceedings of the 15th International Conference on Modularity (MASS'16)*, pp. 146–151. ACM, 2016.

J. Van Bulck, F. Piessens, and R. Strackx.
SGX-Step: A practical attack framework for precise enclave execution control.
In *Proceedings of the 2nd Workshop on System Software for Trusted Execution (SysTEX '17)*. ACM, October 2017.

J. Van 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 *Proceedings of the 26th USENIX Security Symposium*. USENIX Association, August 2017.

References IV

Wikileaks Vault 7: CIA Hacking Tools Revealed.
2014-10-23 branch direction meeting notes.
https://wikileaks.org/ciav7p1/cms/page_13763790.html.

J. Woodruff, R. N. Watson, D. Chisnall, S. W. Moore, J. Anderson, B. Davis, B. Laurie, P. G. Neumann, R. Norton, and M. Roe.
The cheri capability model: Revisiting risc in an age of risk.
In *ACM SIGARCH Computer Architecture News*, vol. 42, pp. 457–468. IEEE Press, 2014.

Y. Xu, W. Cui, and M. Peinado.
Controlled-channel attacks: Deterministic side channels for untrusted operating systems.
In *2015 IEEE Symposium on Security and Privacy*, pp. 640–656. IEEE, 2015.

D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers.
A hardware design language for timing-sensitive information-flow security.
ACM SIGPLAN Notices, 50(4):503–516, 2015.