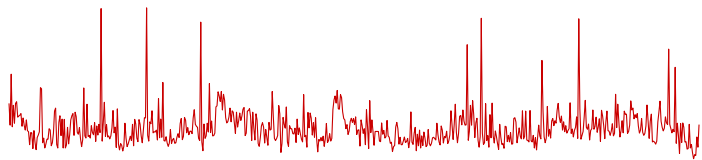
Tutorial: Uncovering Side-Channels in Intel SGX Enclaves

Part 1: Reconstructing enclave code and data accesses

Jo Van Bulck

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

SPACE 2018, December 15, 2018



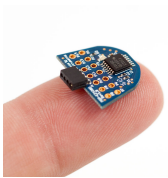
About imec-DistriNet enclave research

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

- Enclave security **across the system stack:** hardware, compiler, OS, application
- Integrated **attack-defense** perspective and **open-source** prototypes

Foreshadow vulnerability
[VBMW+18]

SGX-Step framework
[VBPS17]



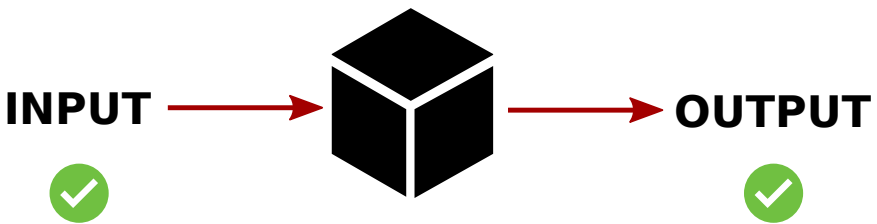
Sancus enclave processor
[NAD+13, NVBM+17]

Tutorial organization

1. **Part 1** (09:00 – 10:30): Reconstructing enclave code and data accesses
   - Lecture: Introduction to Intel SGX and software side-channel attacks
   - Hands-on: Exploiting elementary example applications
2. **Part 2** (11:00 – 12:30): Stealing enclave secrets with transient execution
   - Lecture: Introduction to transient execution attacks (Meltdown, Foreshadow, Spectre)
   - Hands-on: Exploiting elementary example applications

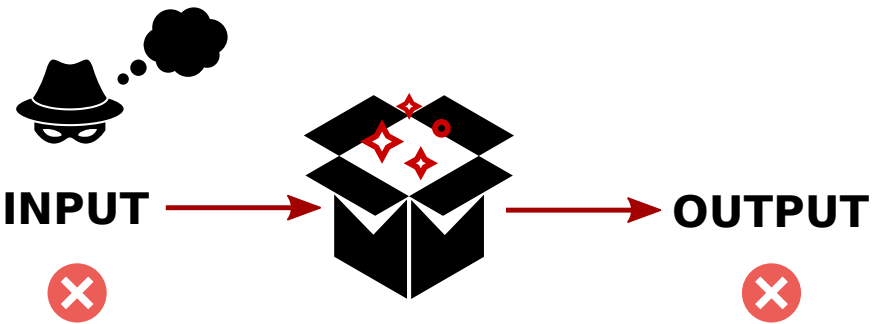
A primer on software security

Secure program: convert all input to *expected output*

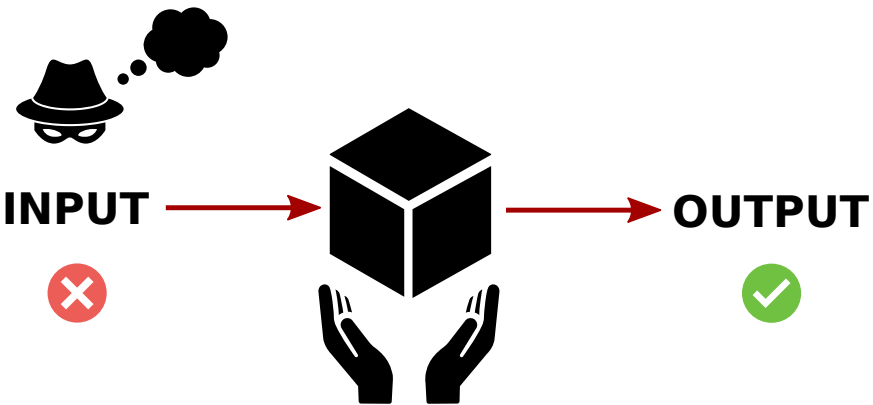


A primer on software security

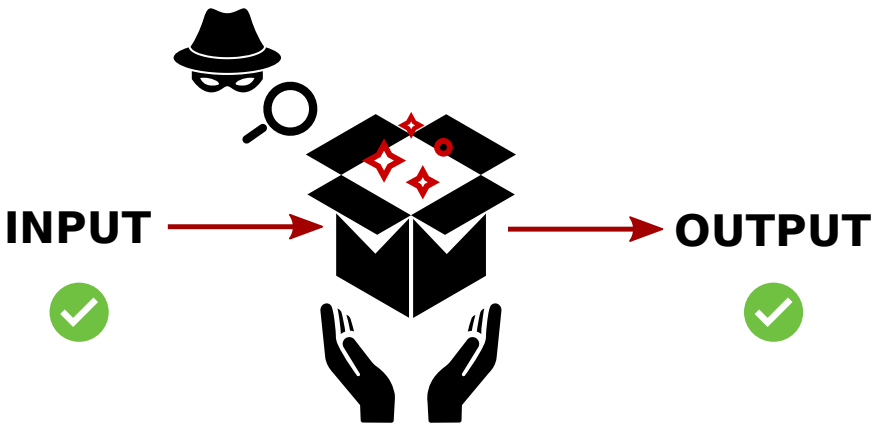
Buffer overflow vulnerabilities: trigger *unexpected behavior*



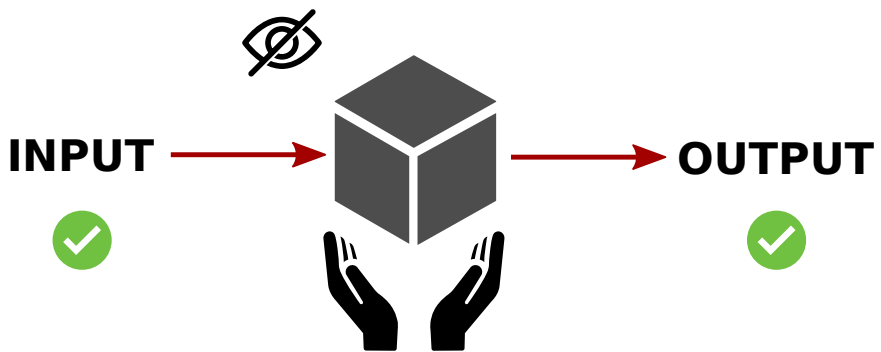
Safe languages & formal verification: preserve *expected behavior*



Side-channels: observe *side-effects* of the computation

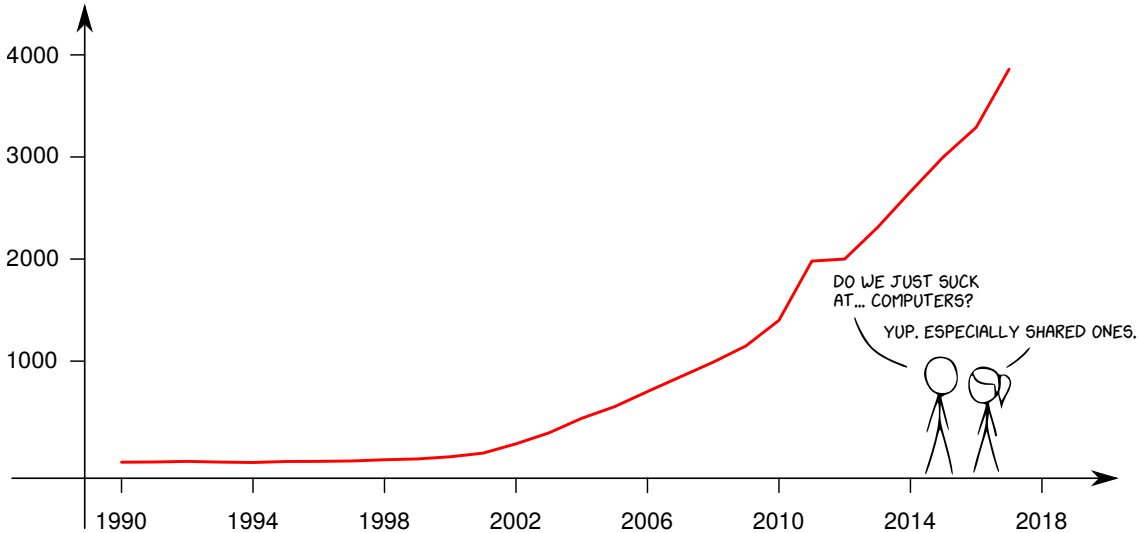


Constant-time code: eliminate *secret-dependent* side-effects



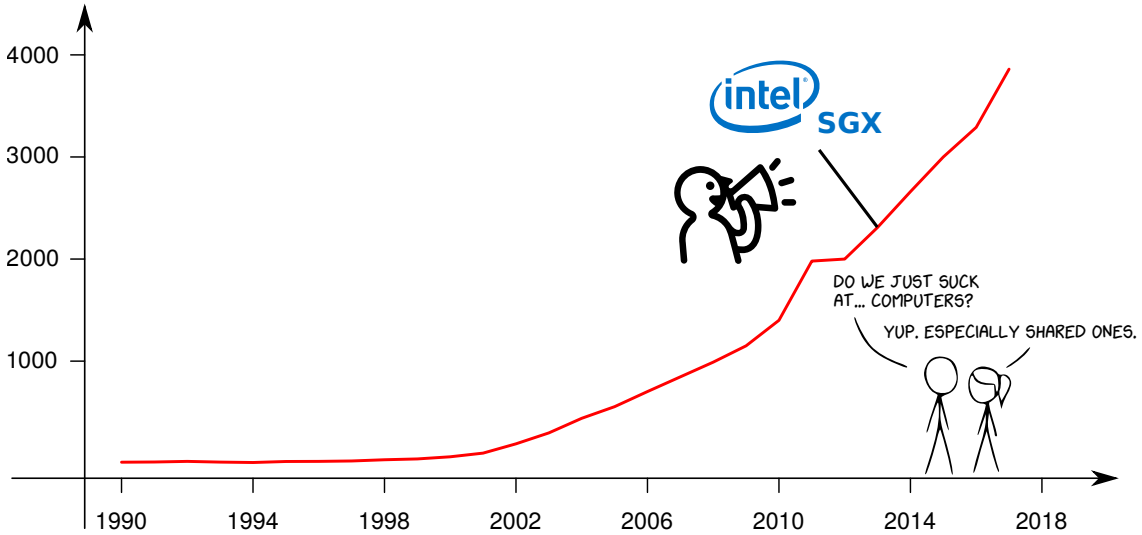

VAULT DOOR
WEIGHT: 22 1/2 Tons
THICKNESS: 22 Inches
STEEL: 11 Layers of Special Cutting and Drill Resistant
LOCKS: 4 Hamilton Watch Movements for Time Locks

Evolution of “side-channel attack” occurrences in Google Scholar



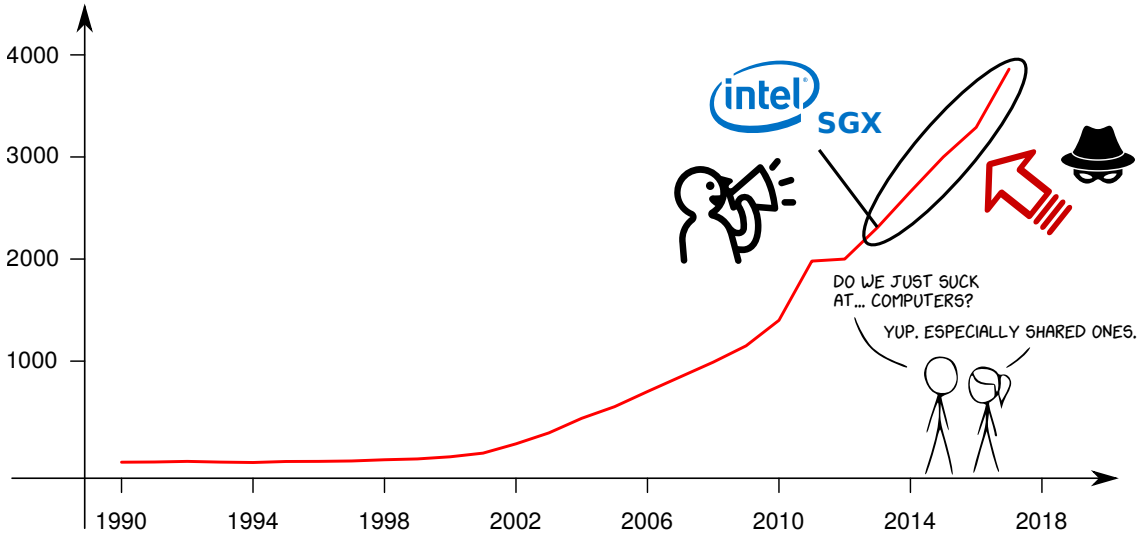
Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

Evolution of “side-channel attack” occurrences in Google Scholar

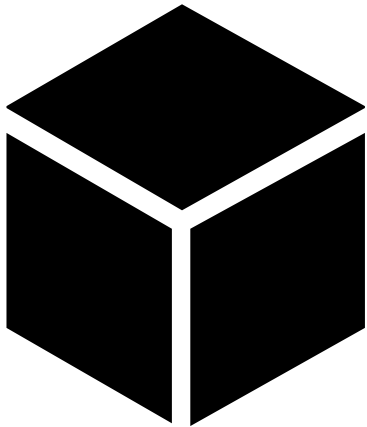


Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

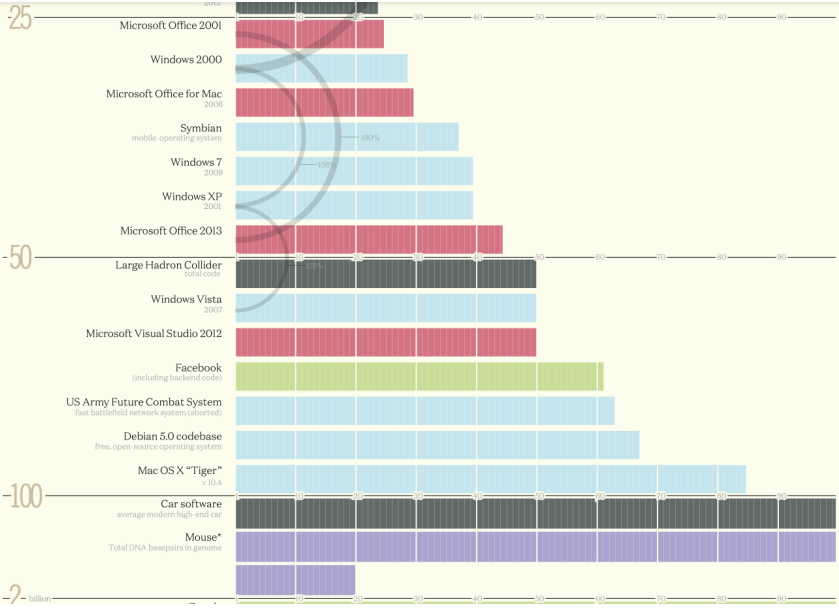
Evolution of “side-channel attack” occurrences in Google Scholar



Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

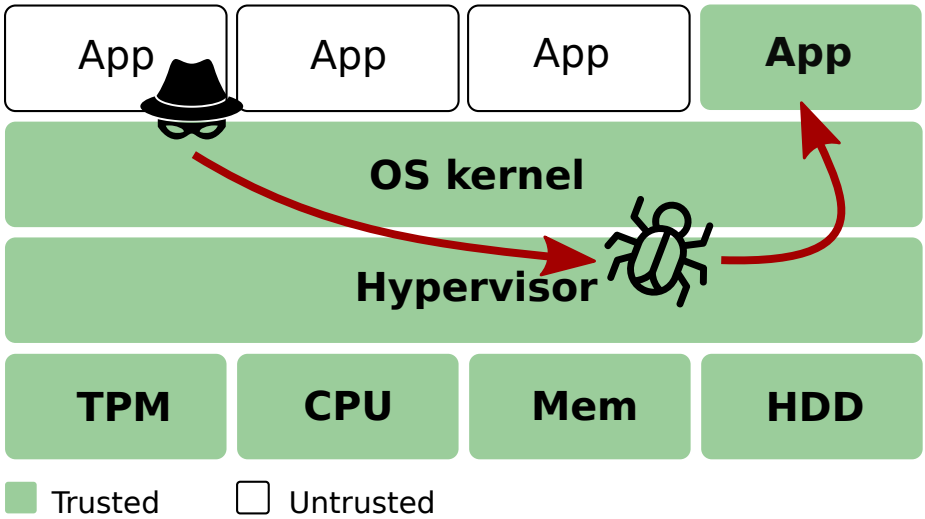


What's inside the black box?

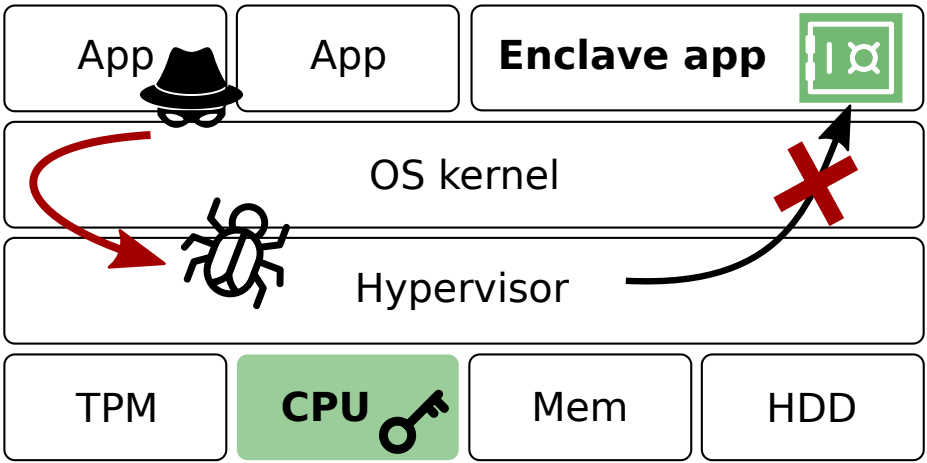


https://informationisbeautiful.net/visualizations/million-lines-of-code/

Enclaved execution: Reducing attack surface

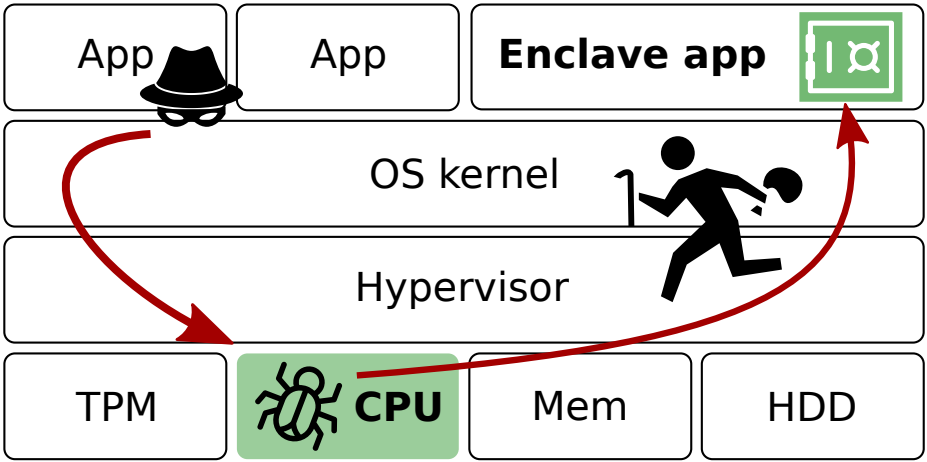


Enclaved execution: Reducing attack surface



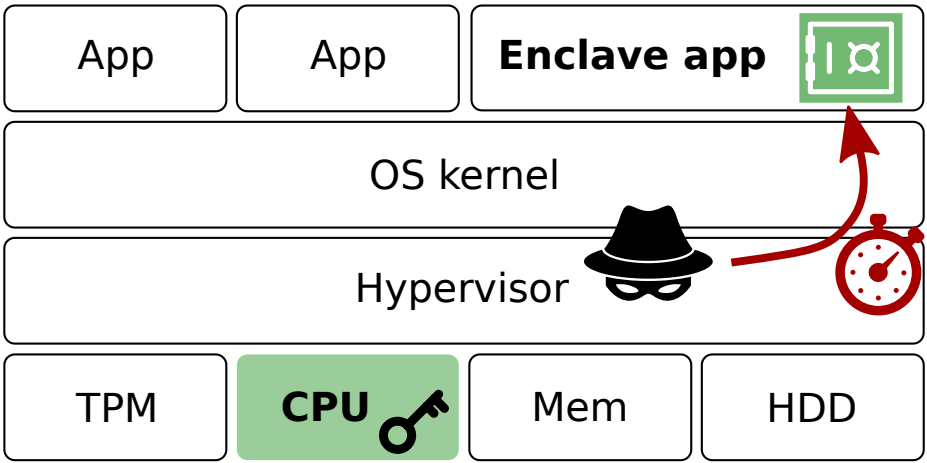
Intel SGX promise: hardware-level **isolation and attestation**

Tutorial part 2: Transient execution attacks



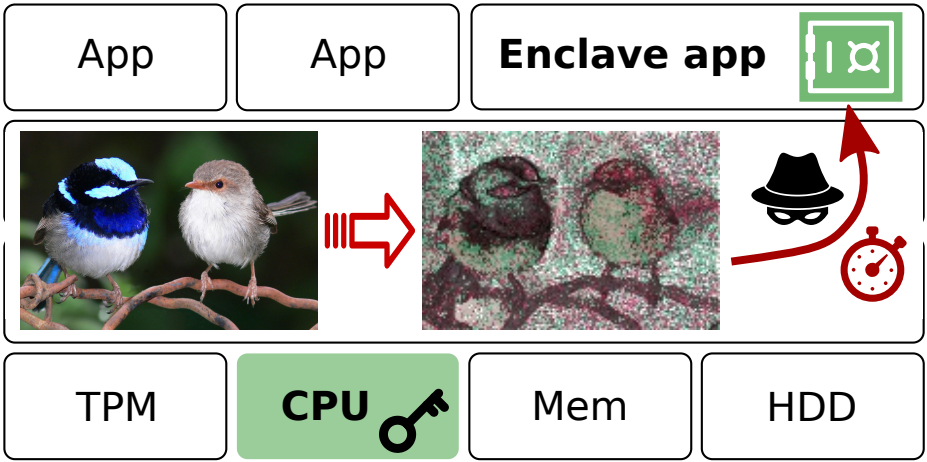
Trusted CPU → exploit **microarchitectural bugs/design flaws**

Van Bulck et al. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution", USENIX 2018 [VBMW+18]



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

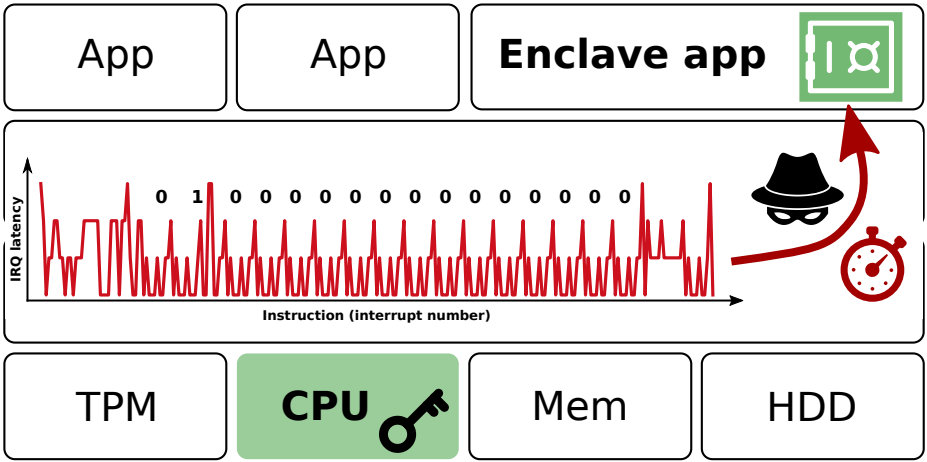
Tutorial part 1: Privileged side-channel attacks



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

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

Tutorial part 1: Privileged side-channel attacks



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

Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018 [VBPS18]

KEEP CALM

IT IS

OUT OF SCOPE

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.

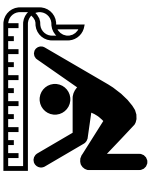
An attacker with access to the platform can see what pages are being executed or accessed. This side-channel vulnerability can be mitigated by aligning specific code and data blocks to exist entirely within a single page.

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.

Research landscape: Understanding side-channel leakage in enclaves

- Which **side-channels** exist?
- Which enclave **applications** are vulnerable? (Not only crypto!)
- How can we **defend** against them, and at what cost?

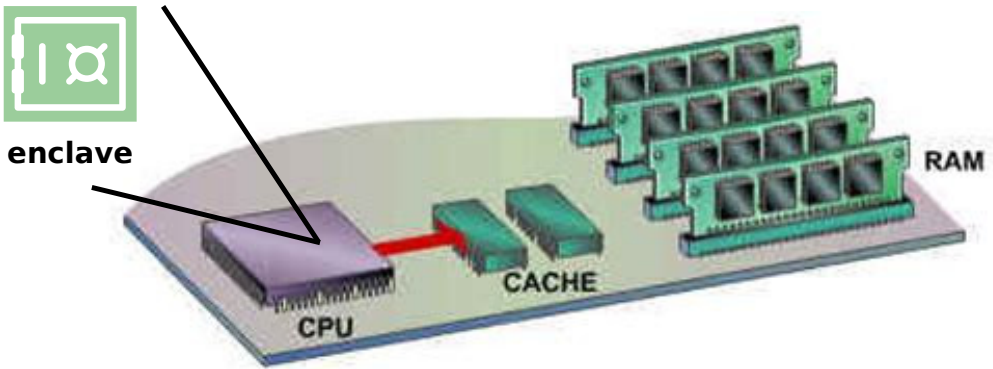
Research landscape: Understanding side-channel leakage in enclaves



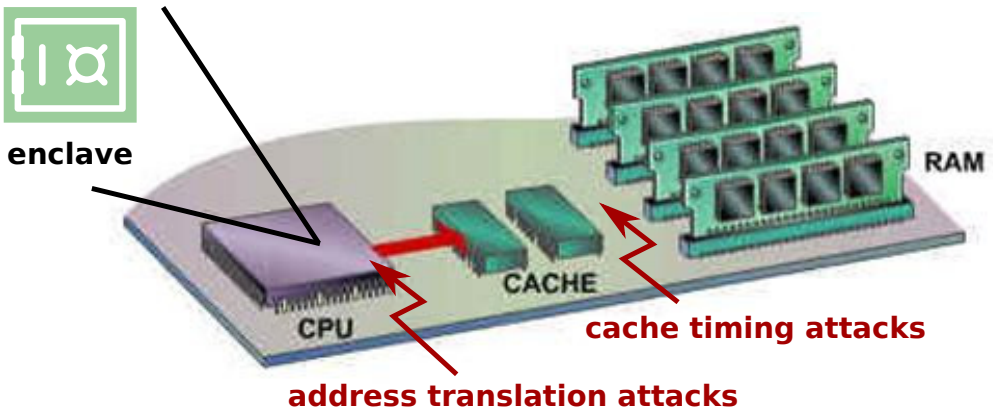
- Which **side-channels** exist?
- Which enclave **applications** are vulnerable? (Not only crypto!)
- How can we **defend** against them, and at what cost?

⇒ Educate developers to raise awareness and avoid side-channel pitfalls (= this tutorial!)

Overview: Spying on enclave memory accesses



Overview: Spying on enclave memory accesses



Secret-dependent code/data memory accesses

```
1 void secret_vote(char candidate)
2 {
3     if (candidate == 'a')
4         vote_candidate_a();
5     else
6         vote_candidate_b();
7 }
```

```
1 int secret_lookup(int s)
2 {
3     if (s > 0 && s < ARRAY_LEN)
4         return array[s];
5     return -1;
6
7 }
```

Secret-dependent code/data memory accesses

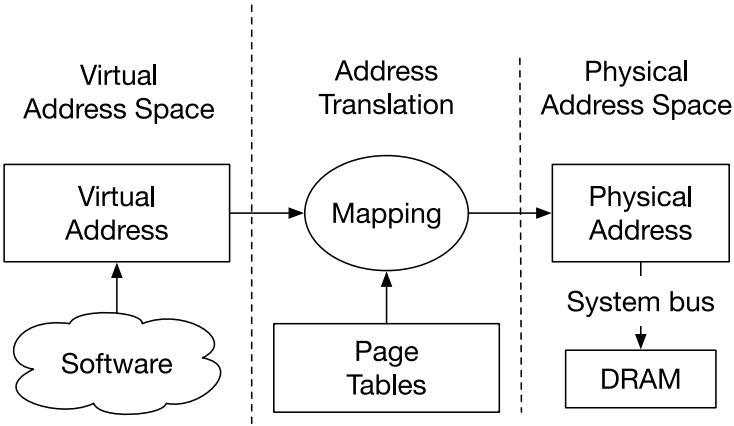
```
1 void secret_vote(char candidate)
2 {
3     if (candidate == 'a')
4         vote_candidate_a();
5     else
6         vote_candidate_b();
7 }
```

```
1 int secret_lookup(int s)
2 {
3     if (s > 0 && s < ARRAY_LEN)
4         return array[s];
5     return -1;
6
7 }
```

What if the adversary obtains a perfect "oracle" for all enclaved code+data memory access sequences?

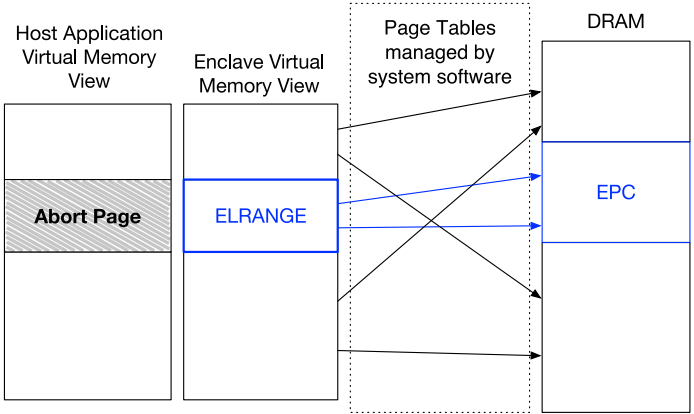
Address translation attacks

The virtual memory abstraction



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

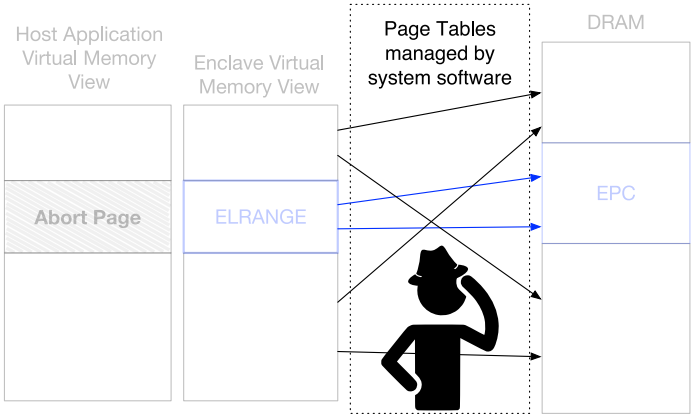
How enclave accesses are enforced



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

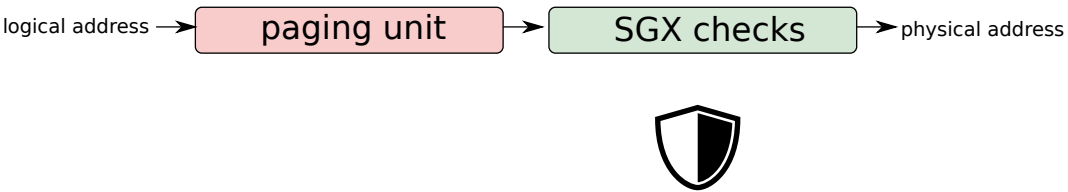
How enclave accesses are enforced

Note: Untrusted OS controls *virtual-to-physical mapping*



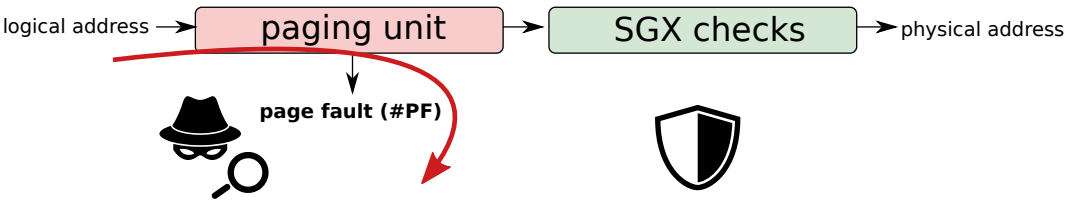
Costan et al. "Intel SGX explained", IACR 2016 [CD16]

Page faults as a side-channel



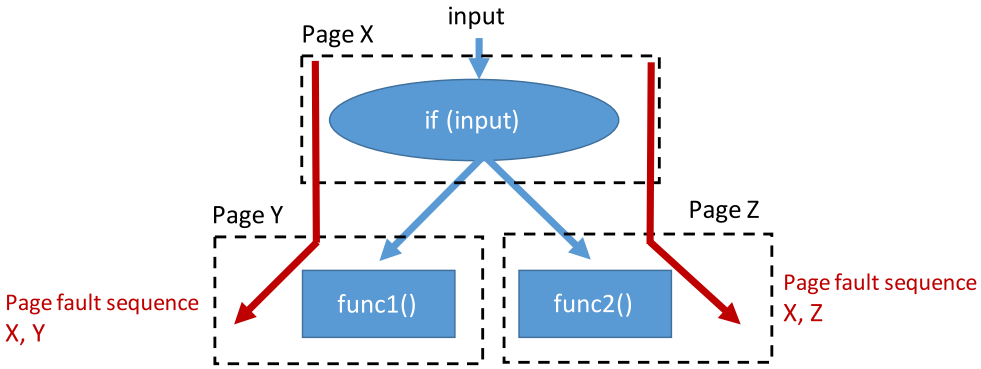
SGX machinery protects against direct address remapping attacks

Page faults as a side-channel



. . . but untrusted address translation may **fault** during enclaved execution (!)

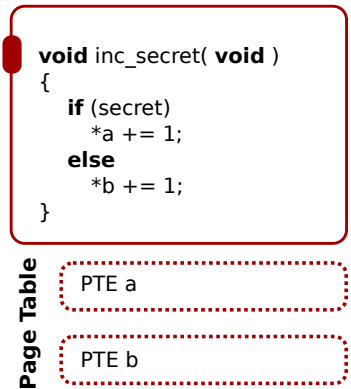
Page faults as a side-channel



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

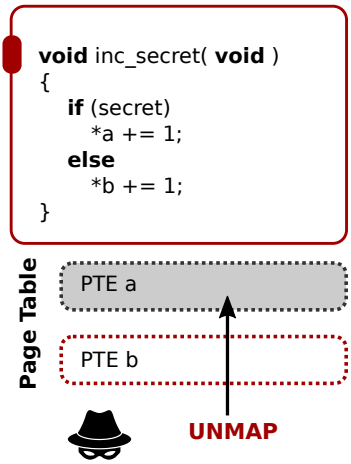
⇒ Page fault traces leak **private control data/flow**

#PF attacks: An end-to-end example



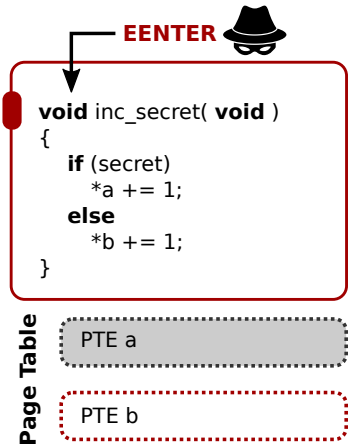
#PF attacks: An end-to-end example

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



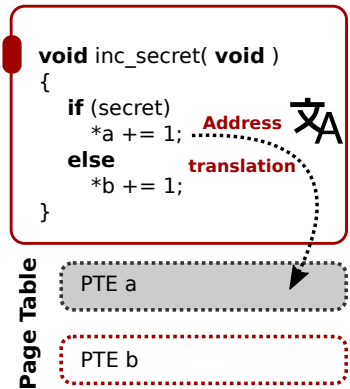
#PF attacks: An end-to-end example

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



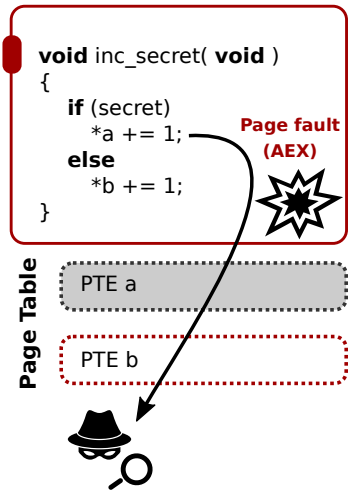
#PF attacks: An end-to-end example

1. Revoke access rights on *unprotected* enclave page table entry
2. Enter victim enclave
3. Secret-dependent data memory access
   - ⇝ Processor performs virt-to-phys address translation!



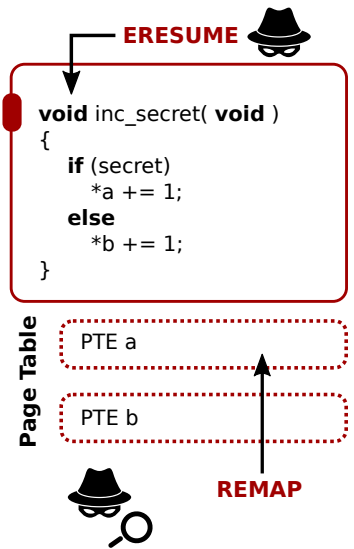
#PF attacks: An end-to-end example

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

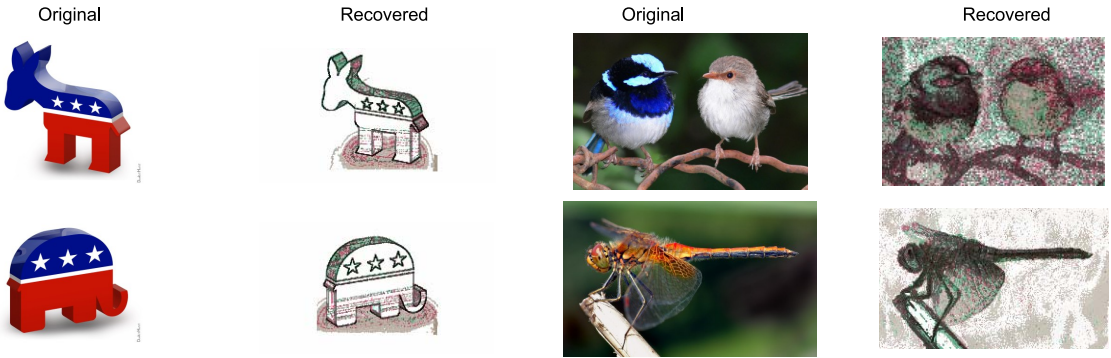


#PF attacks: An end-to-end example

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



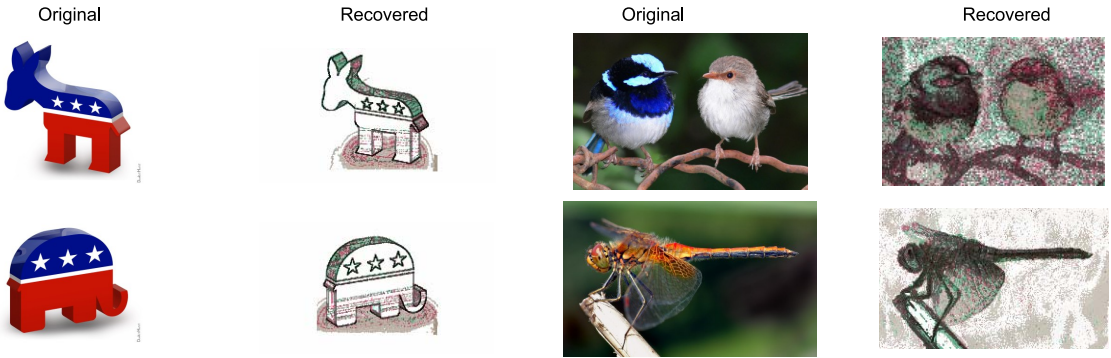
Page table-based attacks in practice



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

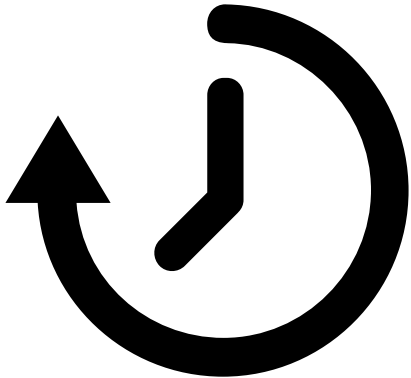
⇒ **Low-noise, single-run** exploitation of legacy applications

Page table-based attacks in practice



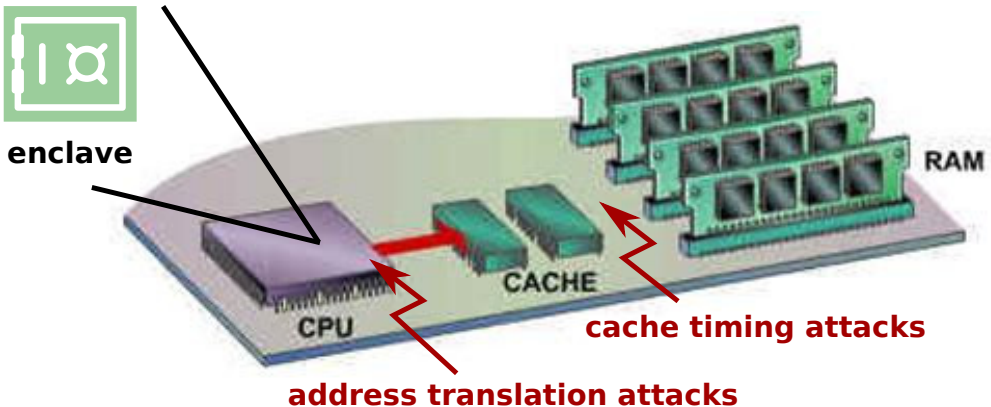
Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

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

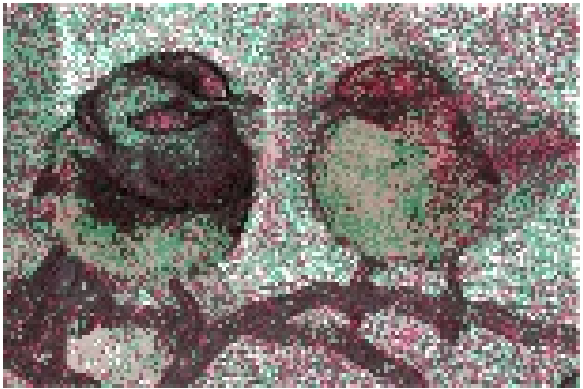


Cache timing attacks

Overview: Spying on enclave code/data accesses (revisited)



High resolution side-channels in practice



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

⇒ Coarse-grained preemption **(4 KB page leakage)**

High resolution side-channels in practice

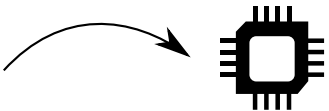
Hähnel et al.: "High-resolution side channels for untrusted operating systems", ATC 2017 [HCP17]

⇒ Fine-grained preemption **(64 B cache line leakage)**

CPU cache timing side-channel

Cache principle: CPU speed ≫ DRAM latency → *cache code/data*

```
while true do
    maccess(&a);
endwh
```



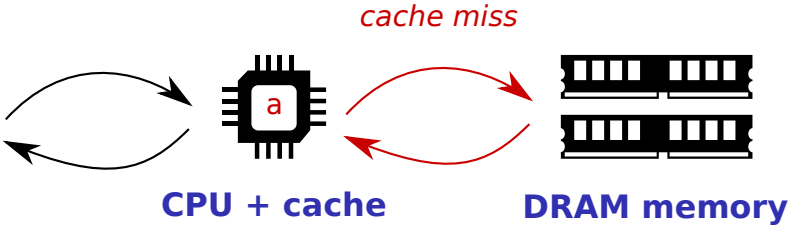
CPU + cache

DRAM memory

CPU cache timing side-channel

Cache miss: Request data from (slow) DRAM upon first use

```
while true do
    maccess(&a);
endwh
```



CPU cache timing side-channel

Cache hit: No DRAM access required for subsequent uses

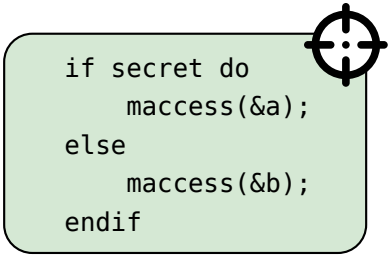
```
while true do
    maccess(&a);
endwh
```

cache hit

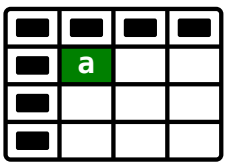
CPU + cache

DRAM memory

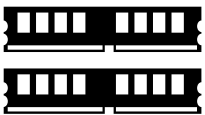
Flush+Reload: Cache timing attacks on shared memory



'a' is accessible to attacker

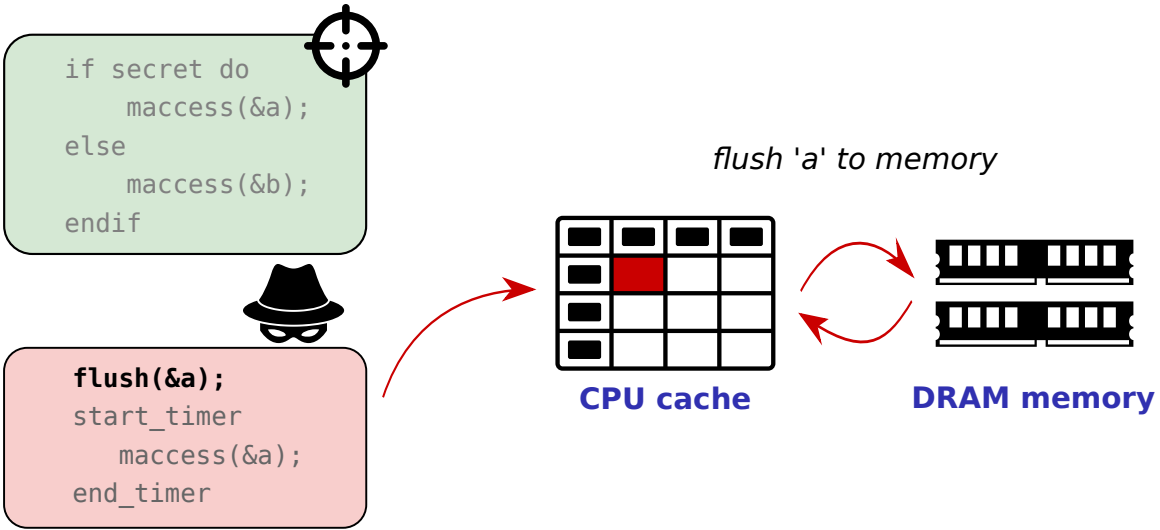


CPU cache

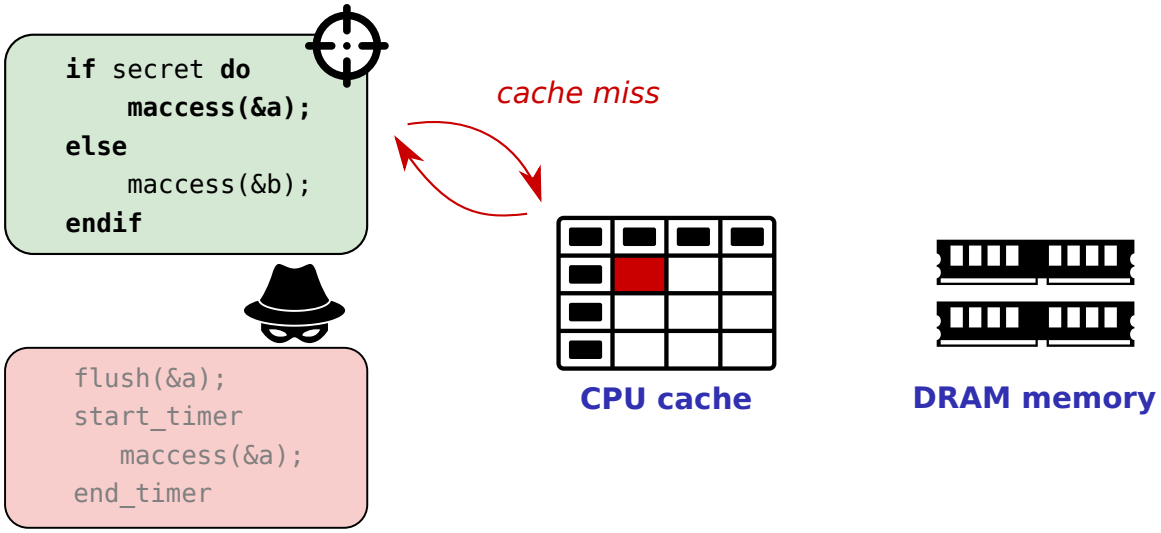


DRAM memory

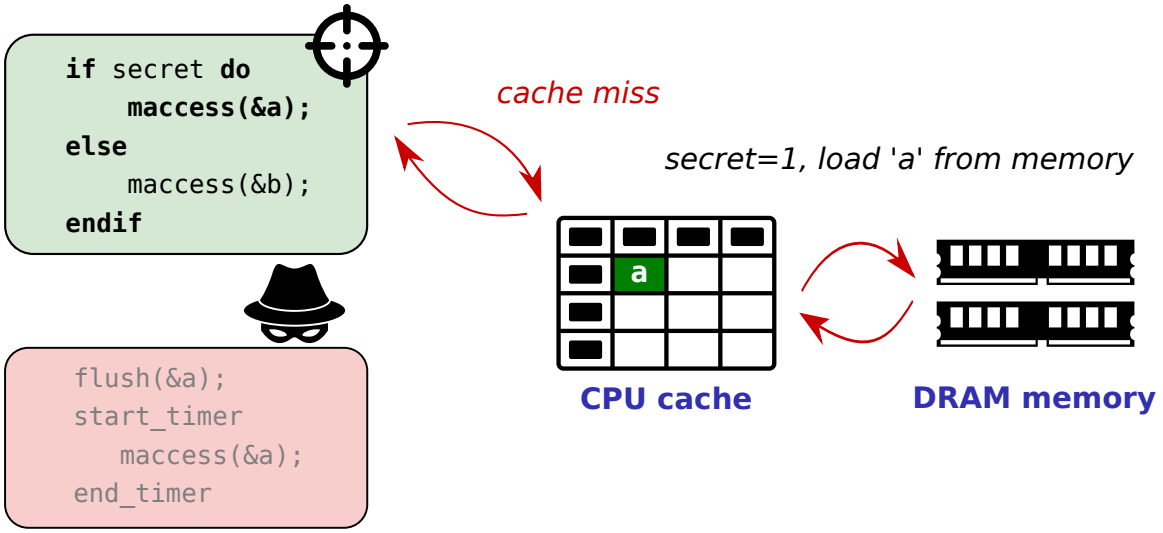
Flush+Reload: Cache timing attacks on shared memory



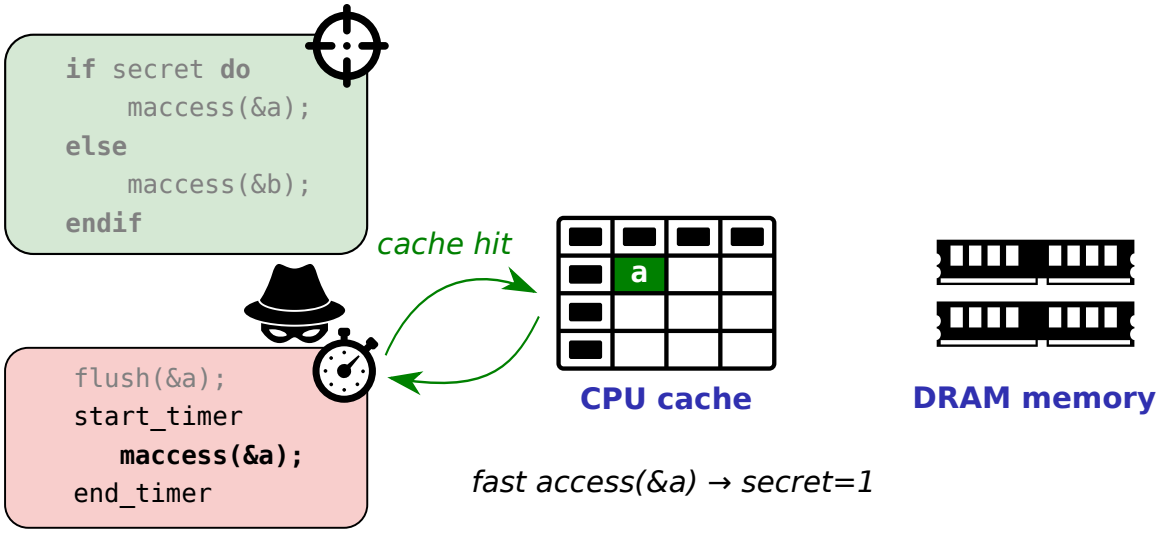
Flush+Reload: Cache timing attacks on shared memory



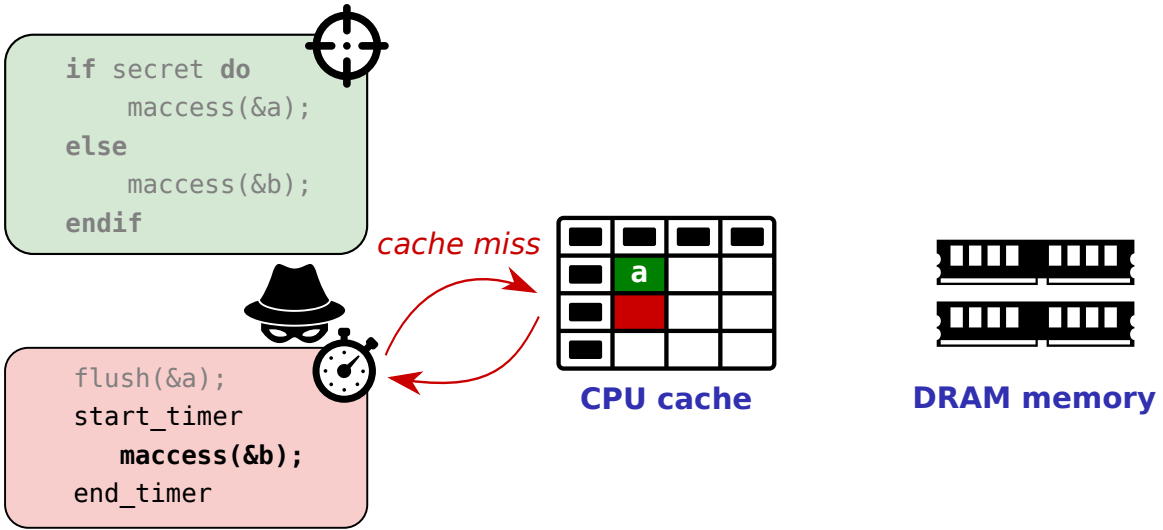
Flush+Reload: Cache timing attacks on shared memory



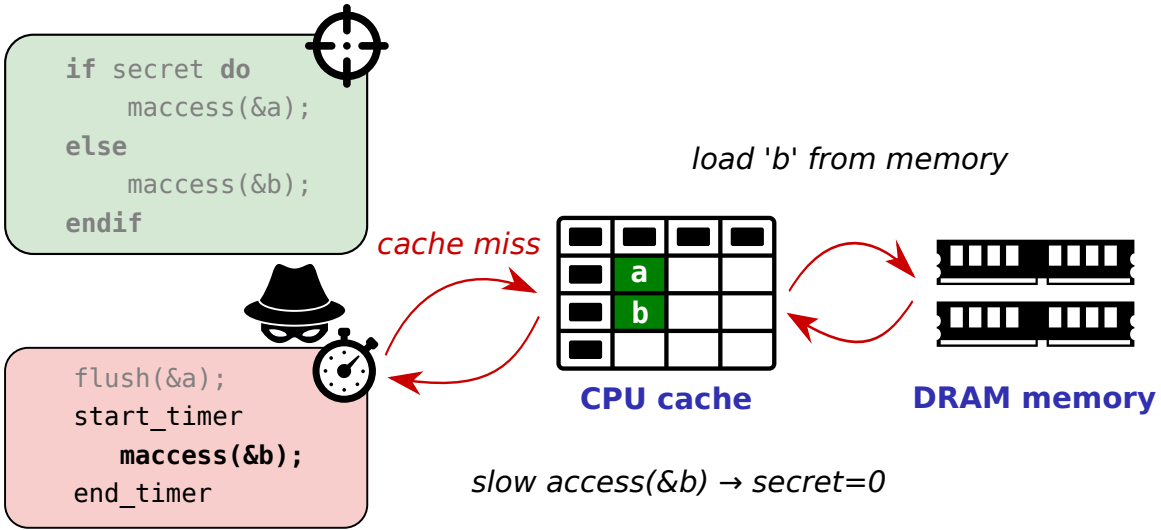
Flush+Reload: Cache timing attacks on shared memory



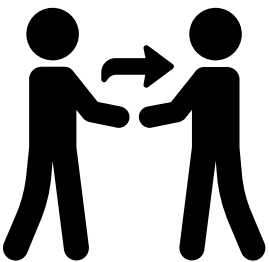
Flush+Reload: Cache timing attacks on shared memory



Flush+Reload: Cache timing attacks on shared memory



Flush+Reload limitations

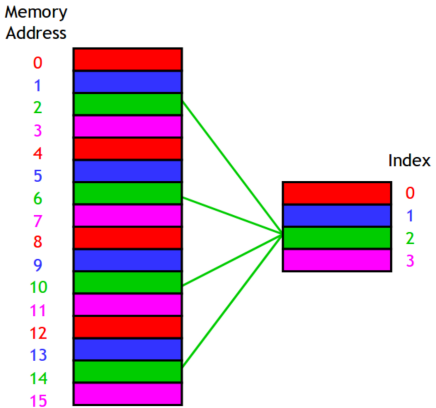


- Very **reliable** attack + easy to mount
- . . . but relies on **shared memory** ($\leftrightarrow$ enclaves)!


prime day

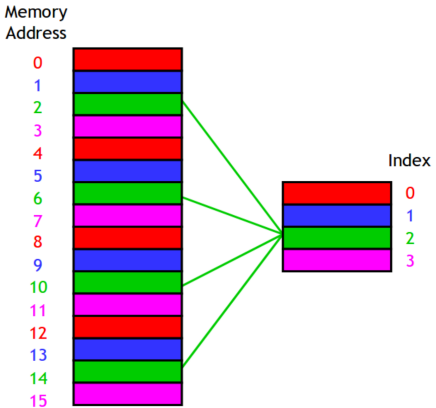
CPU cache organization 101

- **Shared** among all protection domains ☺
- Cache size ≪ addressable memory size
- **Cache line:** unit of caching (64 bytes)
- **Mapping scheme:** memory address → cache line



CPU cache organization 101

- **Shared** among all protection domains ☺
- Cache size ≪ addressable memory size
- **Cache line:** unit of caching (64 bytes)
- **Mapping scheme:** memory address → cache line
- **Cache collision:** replace cache line with new data requested from memory

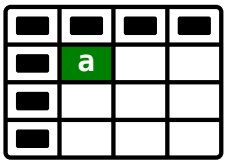


Prime+Probe: Cache timing attacks across protection domains

```
if secret do
    maccess(&a);
else
    maccess(&b);
endif
```

```
maccess(&c);
start_timer
   maccess(&c);
end_timer
```

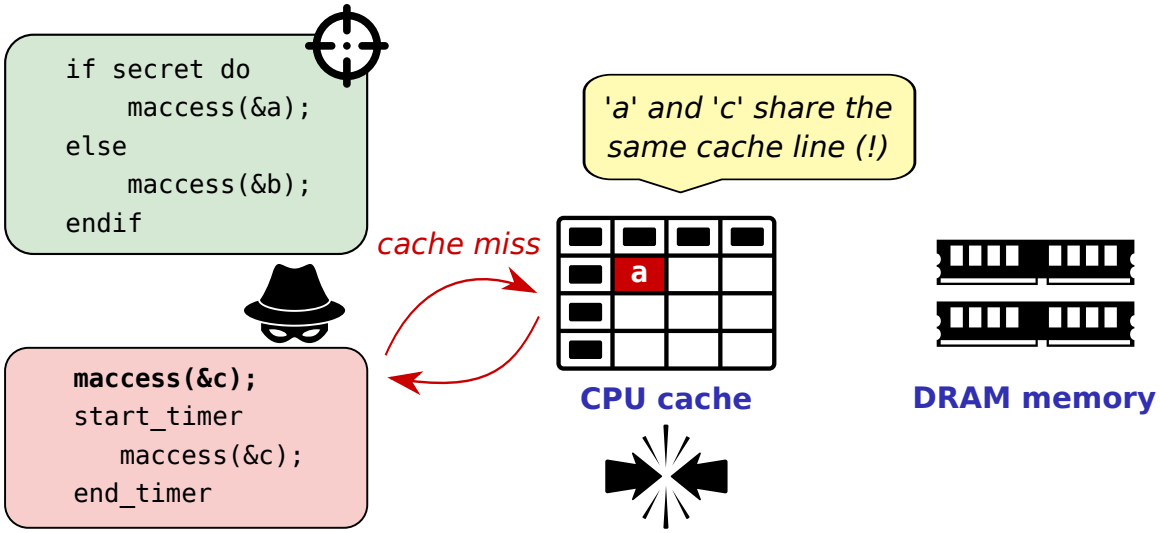
*'a' is **not** accessible to attacker*



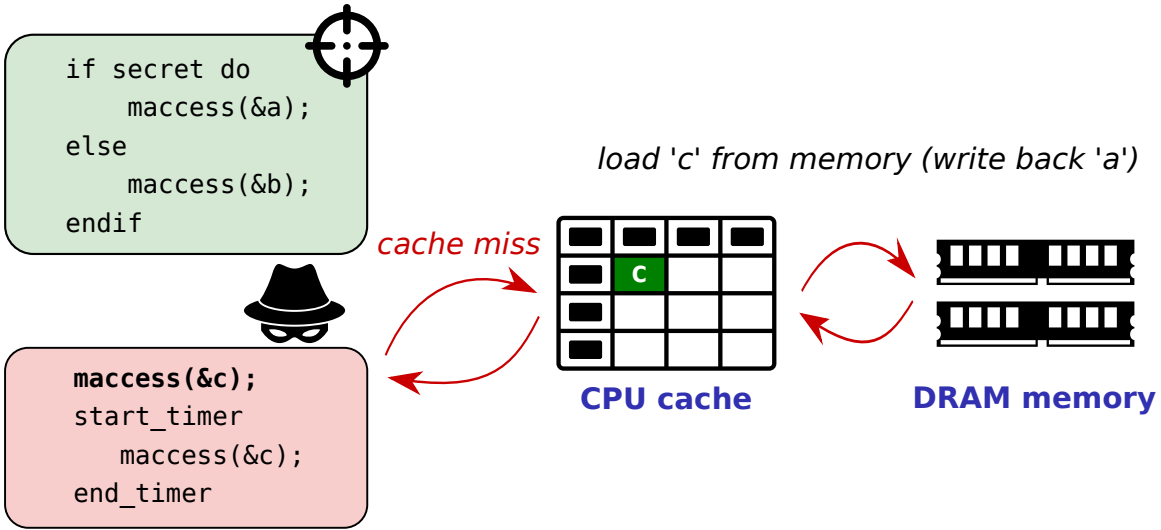
CPU cache

DRAM memory

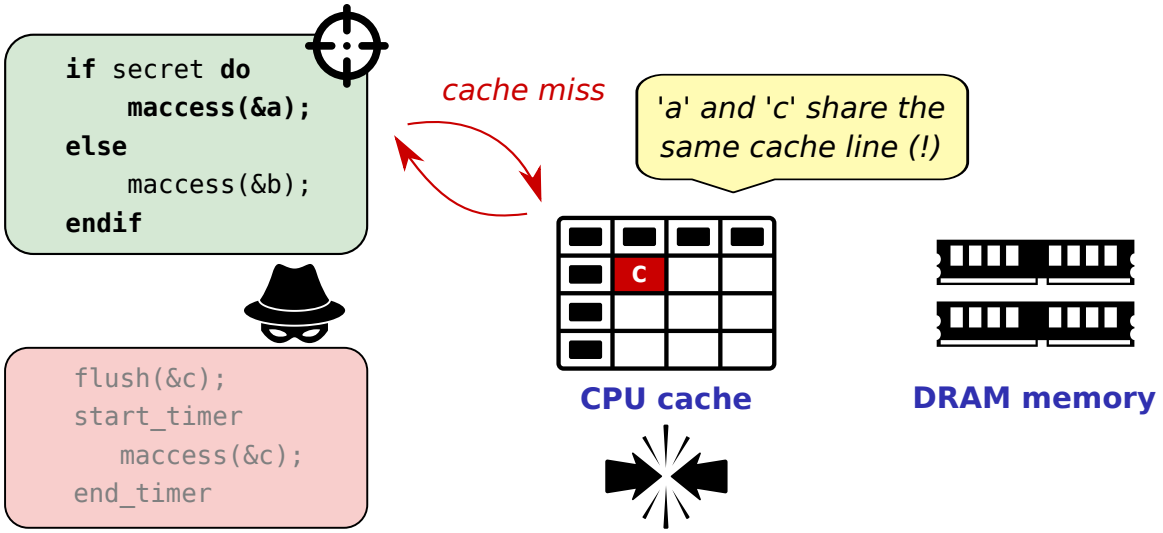
Prime+Probe: Cache timing attacks across protection domains



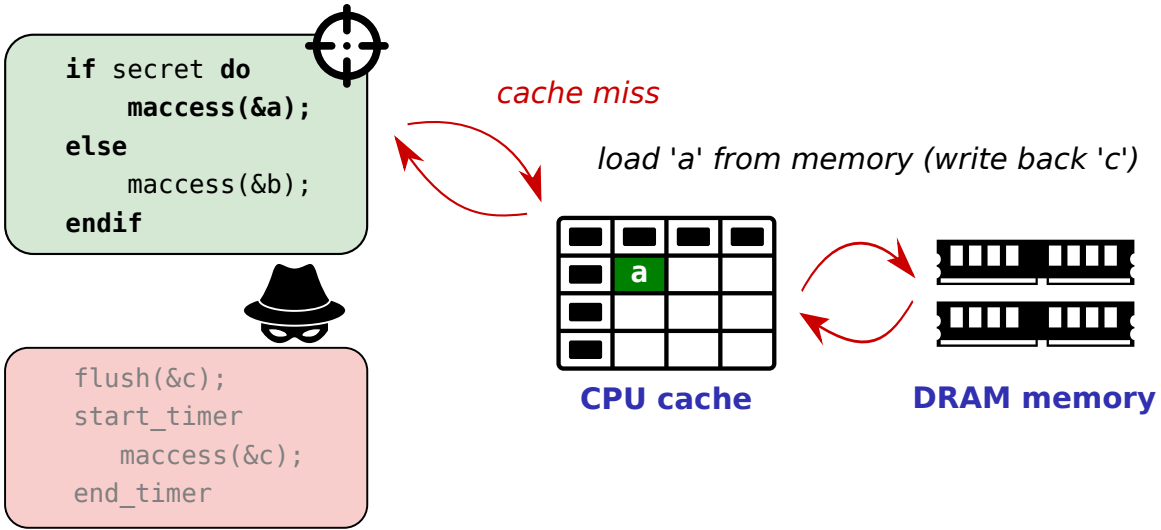
Prime+Probe: Cache timing attacks across protection domains



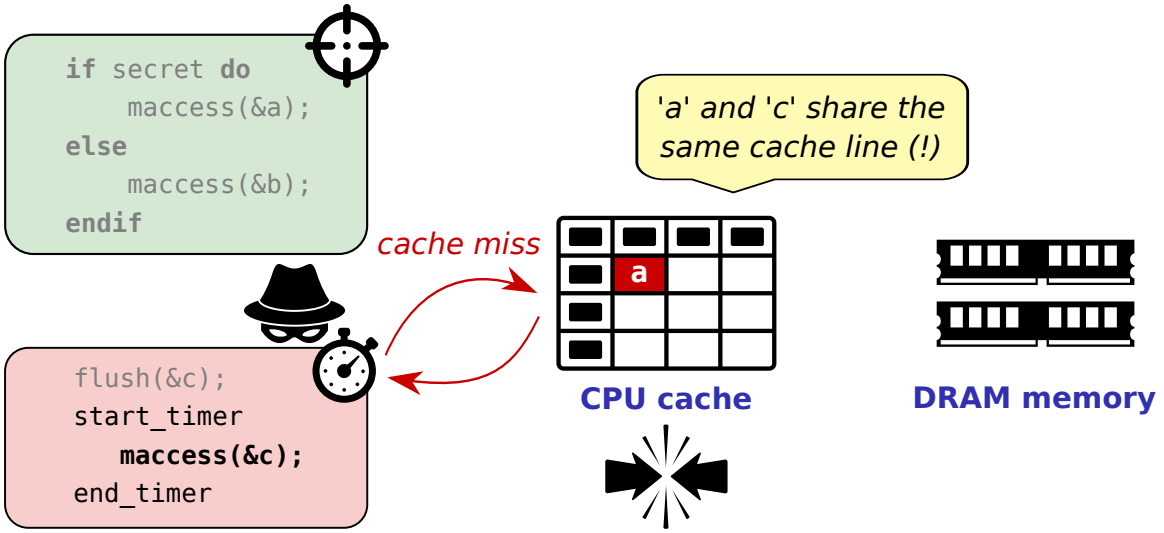
Prime+Probe: Cache timing attacks across protection domains



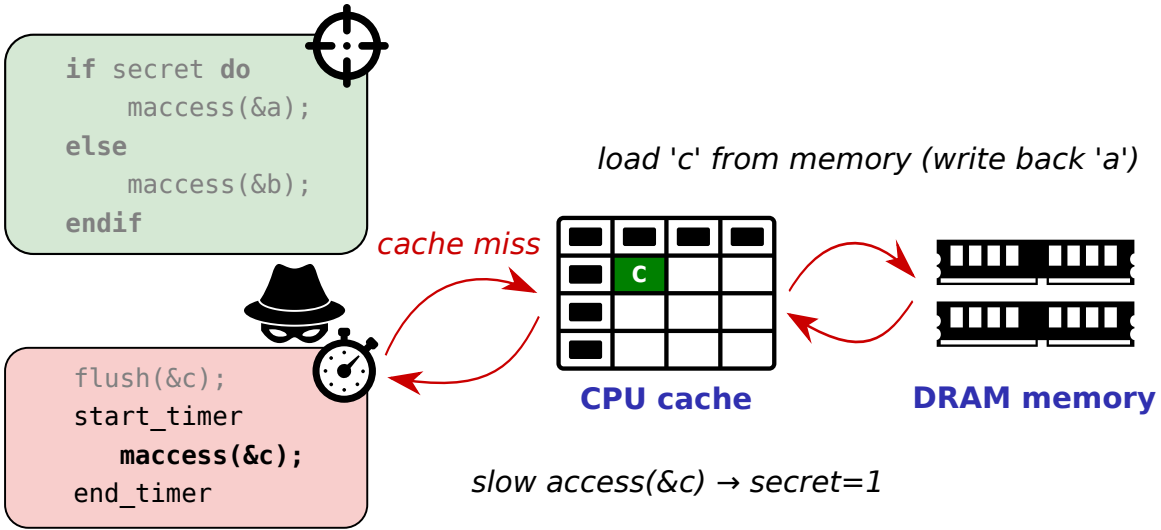
Prime+Probe: Cache timing attacks across protection domains



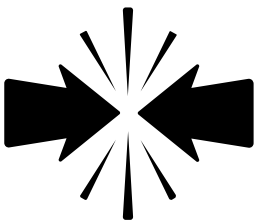
Prime+Probe: Cache timing attacks across protection domains



Prime+Probe: Cache timing attacks across protection domains



Prime+Probe Challenges

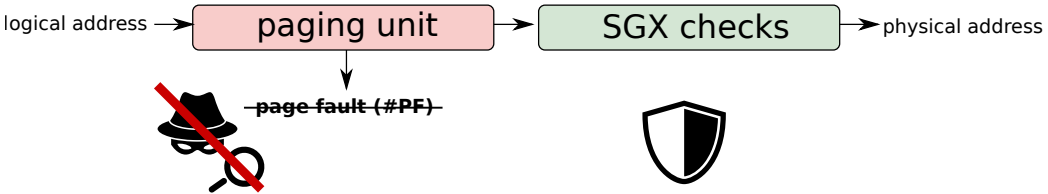


- Exploit **contention** on shared cache resource
- Very **generic** attack applicable to many cache designs + protection domains
- . . . but relies on detailed understanding of **cache mapping** scheme → complex for real-world set-associative caches (e.g., reverse engineering Intel last-level cache)

Defense idea #1

What about hiding enclave page faults?

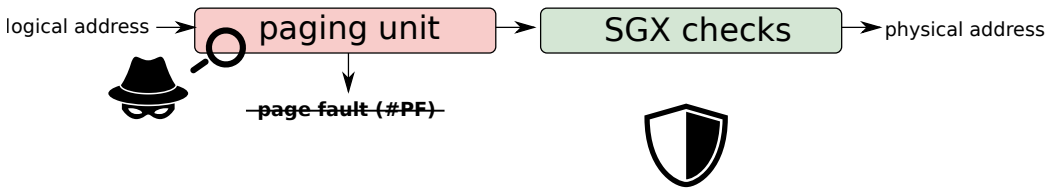
Current solutions: Hiding enclave page faults



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

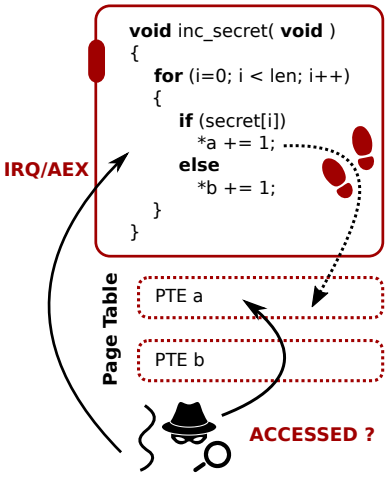


. . . But stealthy attacker can still learn page accesses without triggering faults!

Telling your secrets without page faults

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

   ⇝ Also updated in enclave mode!



Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017 [VBWK+17]

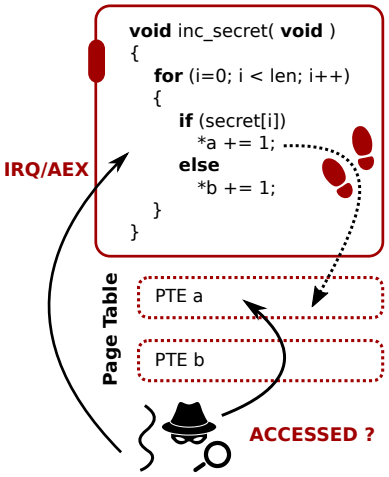
Telling your secrets without page faults

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

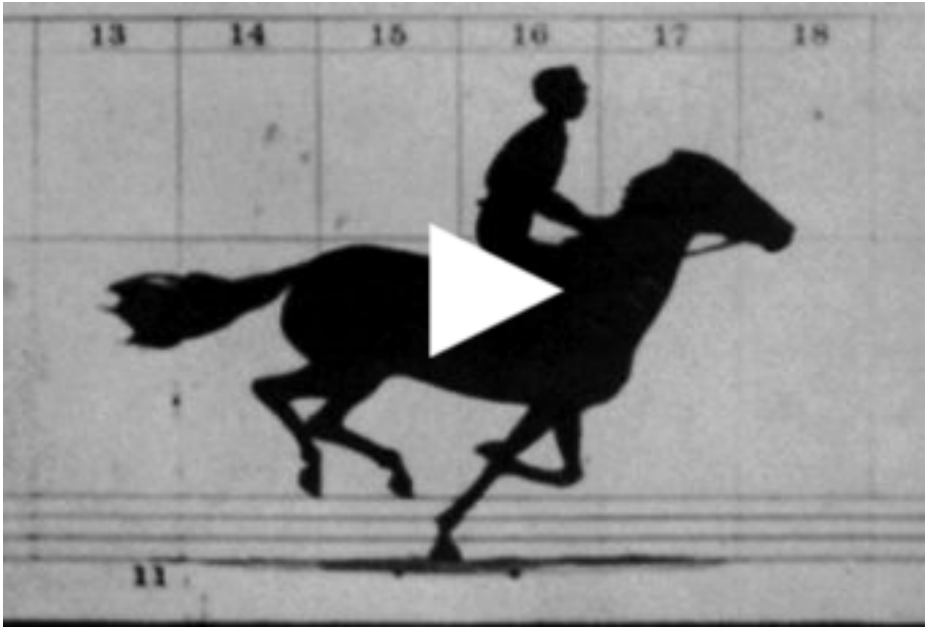
   ⇝ Flush+Reload cache timing attack!



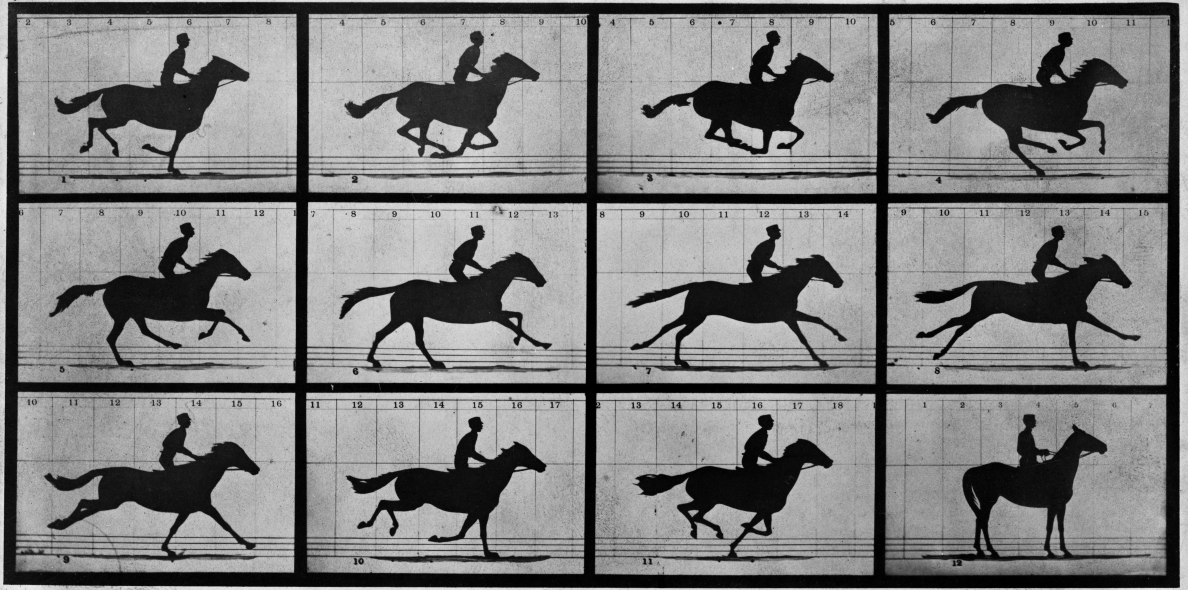
Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017 [VBWK+17]

Defense idea #2

What about limiting the temporal resolution?



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



Copyright, 1878, by MUYBRIDGE.

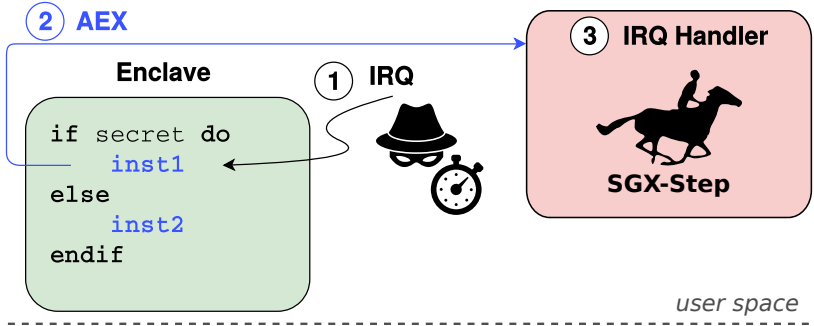
MORSE'S Gallery, 417 Montgomery St., San Francisco.

THE HORSE IN MOTION.

Illustrated by

SGX-Step: Executing enclaves one instruction at a time

SGX-Step: user space APIC timer + interrupt handling ☺



Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017 [VBPS17]

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

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.

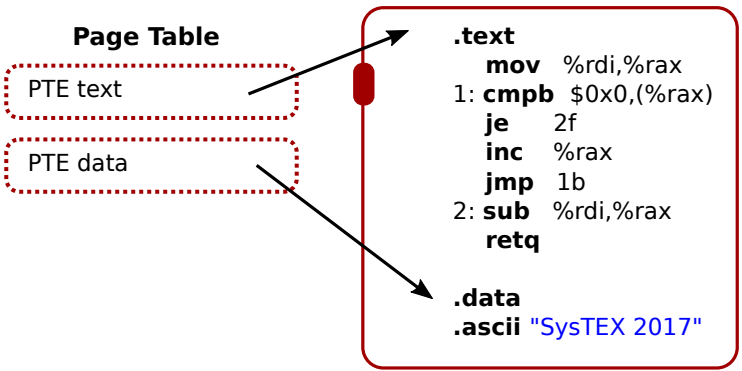
More important, the application enclave should use an appropriate crypto implementation that is side channel attack resistant inside the enclave if side-channel attacks are a concern.

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

High-resolution attacks in practice: Attacking `strlen`

Page fault adversary

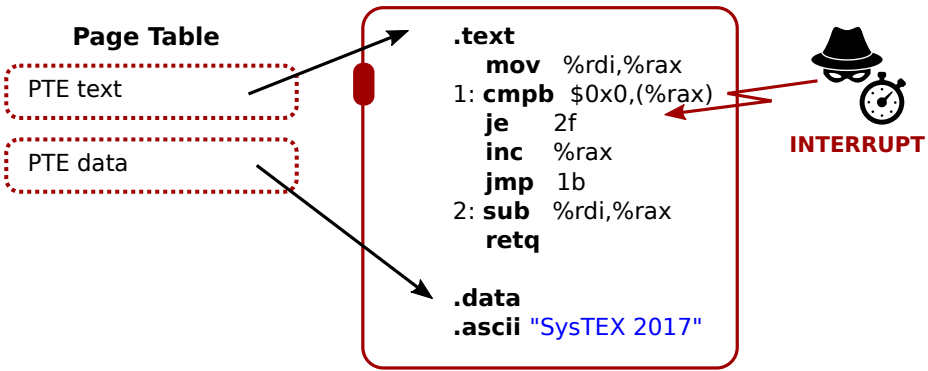
Progress ⇒ both code + data pages present ☹



High-resolution attacks in practice: Attacking `strlen`

Single-stepping adversary

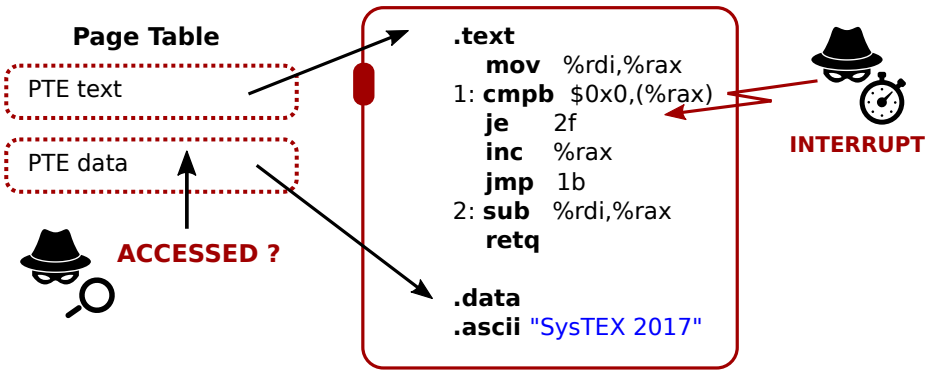
Execute + interrupt ⇒ data page accessed ? ☺

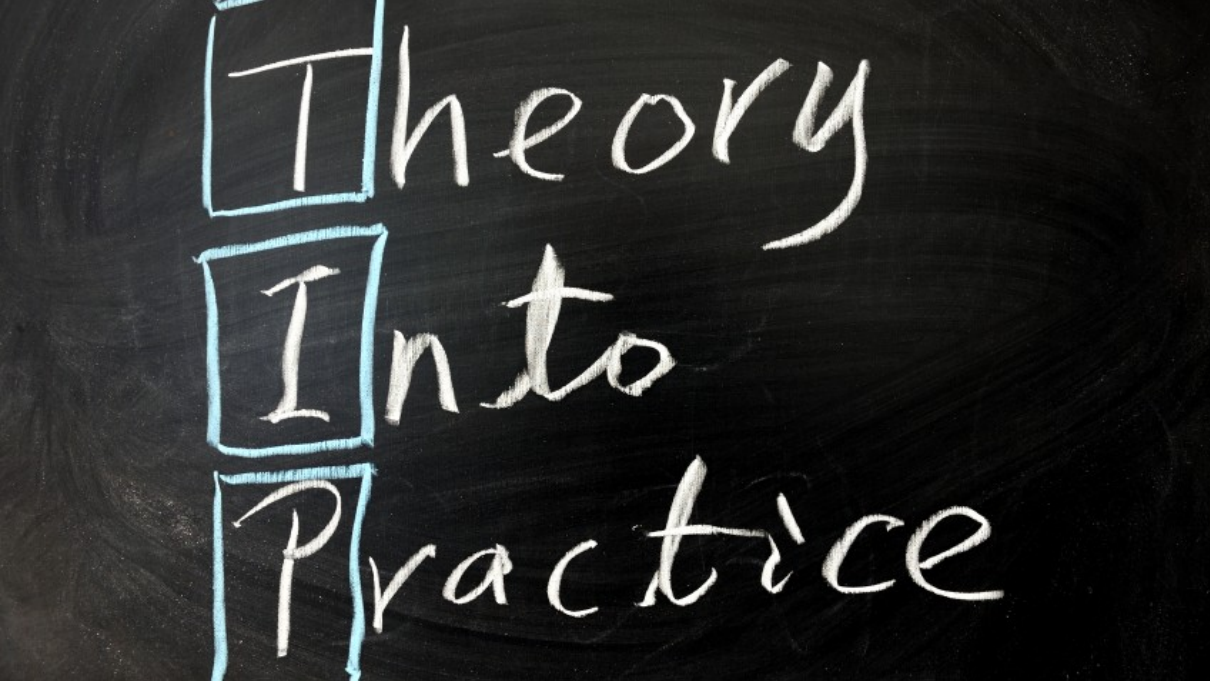


High-resolution attacks in practice: Attacking `strlen`

Single-stepping adversary

Execute + interrupt ⇒ data page accessed ? ☺




Theory
Into
Practice

Hands-on exercises

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

Important note

First develop the *unprotected attack scenario on your local x86 machine*, before testing the enclaved version on the remote SGX machine via SSH (!)

1. Connect to the `space18-sgx` **WiFi network**
   - WPA2 passphrase "space2018-sgx-tutorial"
2. Now ssh into the **SGX machine:** `ssh sgx@10.45.160.95`
   - User: "sgx"
   - Password: "space18"
   - Make sure to work in your own directory to avoid interference

References I

V. Costan and S. Devadas.
Intel SGX explained.
Cryptology ePrint Archive, Report 2016/086, 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.

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 *22nd USENIX Security symposium*, pp. 479–494. USENIX Association, 2013.

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

J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx.
Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution.
In *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, August 2018.

References II

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, pp. 4:1–4:6. ACM, 2017.

J. Van Bulck, F. Piessens, and R. Strackx.
Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic.
In *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS'18)*. ACM, October 2018.

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.

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