

# Attacks and Defenses for Intel SGX

Taesoo Kim



# About Myself



- 03-09: B.S. from KAIST in CS/EE
- 09-11: S.M. from MIT in CS
- 11-14: Ph.D. from MIT in CS
- 14- : Assistant Professor at Gatech

Research interests:

Operating Systems, Systems Security, Bug Finding, etc

<https://taesoo.kim/>

# Systems Software & Security Lab

We build practical systems with focuses on security, performance, robustness, or often just for fun. Our research projects have been published in top academic conferences, and have made great impacts on real programs, such as Firefox, Android, and the Linux kernel, that you might be using every day. If you are interested in hacking with us, please drop us an email via [<sslabs@cc.gatech.edu>](mailto:<sslabs@cc.gatech.edu>).

In particular, we have one or two openings for postdocs and two positions for PhDs in this coming 2018 Fall.

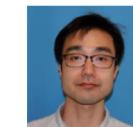
## News (all/18/17/16/15/14)

- [08/15/2018] QSYM got a Distinguished Paper Award at USENIX Security'18!
- [08/12/2018] DEFKOROOT won DEF CON CTF 2018!!
- [07/24/2018] uCFI is accepted to CCS 2018!
- [05/02/2018] QSYM and RTAG are accepted to USENIX Security!
- [04/18/2018] eCS is accepted to ATC'18
- [03/10/2018] Kaleidoscope accepted at the EuroSys Doctoral Workshop
- [02/07/2018] Steffen got the best poster award at the KAUST OBD Workshop!
- [01/22/2018] Solros and Ordo are accepted to EuroSys'18!
- [11/27/2017] Deadline is accepted to S&P'18!
- [11/14/2017] LATR is accepted to ASPLOS'18
- [11/11/2017] Insu has been selected as one of finalists for the MSR PhD fellowship
- [11/03/2017] Gift by Intel to support our SGX research (\$90K)!
- [09/20/2017] SGX-Bomb is accepted to SysTEX'17!
- [08/02/2017] RAIN, OSSPolice and OS for Fuzzing are accepted to CCS 2017!
- [08/01/2017] SAMSUNG Global Research Outreach (GRO) 2017 Awarded
- [07/25/2017] AVPASS is on DARK Reading 1/2/3 and WIRED
- [05/11/2017] PlatPal, PITTYPAT, Branch Shadowing Attack, and Dark ROP are accepted to CCS 2017!
- [05/04/2017] AVPASS is accepted to Black Hat USA 2017
- [05/03/2017] Gift by Mozilla to support our research on fuzzing (\$60K)!
- [04/24/2017] Mosaic has won the best student paper award at EuroSys'17!

Group Leaders



Taesoo Kim



Sangho Lee (w/ Wenke Lee)



Hong Hu (w/ Wenke Lee)

PhD Students



Sandhya Kashyap



Meng Xu



Insu Yun



Steffen Maasz



Ming-Wei Shih



Mohan Kumar



Wen Xu



Jinho Jung



Ren Ding



ChangSeok Oh



Soyeon Park



Fan Sang



Seulbae Kim



Chanil Jeon



ChulWon Kang

# Our Group's Research Interests

- **Bug finding:**  
e.g., static analysis, fuzzing, symbolic execution, etc.
- **System security:**  
e.g., system updates, Intel SGX, sandboxing, etc.
- **System scalability:**  
e.g., file system, graph processing, scalable lock, etc.

# Our Group's Research Interests

(> 300 bugs in Linux, Firefox, OpenSSL, etc.)

## CVEs

We frequently report and fix security-critical vulnerabilities that we find as a byproduct of our research. Some of bugs that have an explicitly assigned CVE or references are listed here:

Date	Description	Ref.
2018/07/27	Linux HFS+ memory corruption ( <a href="#">link</a> )	CVE-2018-14617
2018/07/27	Linux F2FS memory corruptions ( <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> )	CVE-2018-14614,14615,14616
2018/07/27	Linux Btrfs memory corruptions ( <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> )	CVE-2018-14609,14610,14611,14612,14613
2018/07/16	Linux ext4 memory corruptions ( <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> )	CVE-2018-10879,10880,10881,10882,10883
2018/07/16	Linux ext4 memory corruptions ( <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> )	CVE-2018-10840,10876,10877,10878
2018/07/03	Linux F2FS memory corruptions ( <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> )	CVE-2018-13096,13097,13098,13099,13100
2018/07/03	Linux XFS memory corruption ( <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> )	CVE-2018-13093,13094,13095
2018/04/24	Linux XFS memory corruptions ( <a href="#">link</a> , <a href="#">link</a> )	CVE-2018-10322,10323
2018/04/01	Linux ext4 memory corruptions ( <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> , <a href="#">link</a> )	CVE-2018-1092,1093,1094,1095
2017/11/30	FFmpeg out-of-bound read in gmc_mm <sub>x</sub> ( <a href="#">link</a> )	CVE-2017-17081

# DEFKOROOT: Won DEF CON CTF'18 (DEFKOR + R00tmentary)



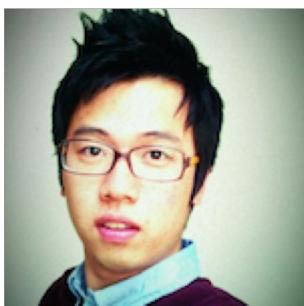
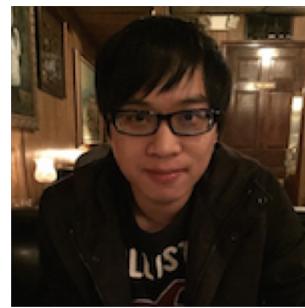
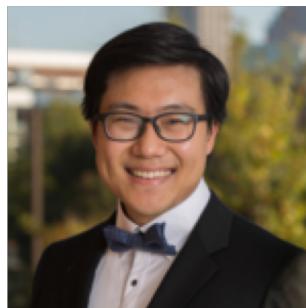
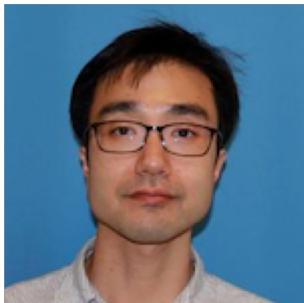
(R00tmentary)

# Attacks and Defenses for Intel SGX

Taesoo Kim

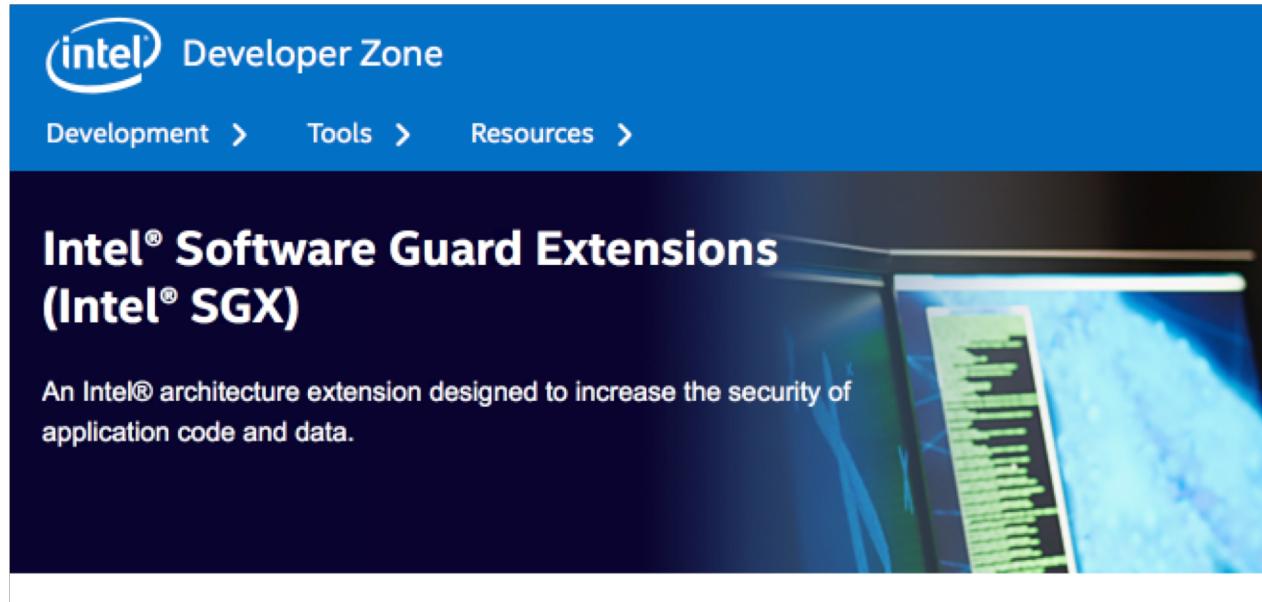


# The Team



# Disclaimer

<https://software.intel.com/en-us/sgx/academic-research>



# Outline

- Threat model / assumption
- Traditional attack vectors
- New attack vectors
- On-going approaches
- Summary

# Outline

- Threat model / assumption
- **Traditional attack vectors**
  - Cache-based side channel
  - Memory safety
  - Weak mitigation techniques (e.g., ASLR)
  - Uninitialized padding in EDL
- New attack vectors
- On-going approaches
- Summary

# Outline

- Threat model / assumption
- Traditional attack vectors
- **New attack vectors**
  - Page table attack
  - Branch shadowing attack
  - Rowhammer against SGX
  - L1 terminal fault against SGX (i.e., [Foreshadow](#))
- On-going approaches
- Summary

# Revisited: Intel SGX 101

- “Practical” TEE implementation by Intel
- Extending x86 Instruction Set Architecture (ISA)
  - Native performance
  - Compatible to x86
  - Commodity (i.e., cheap)



Lenovo T560

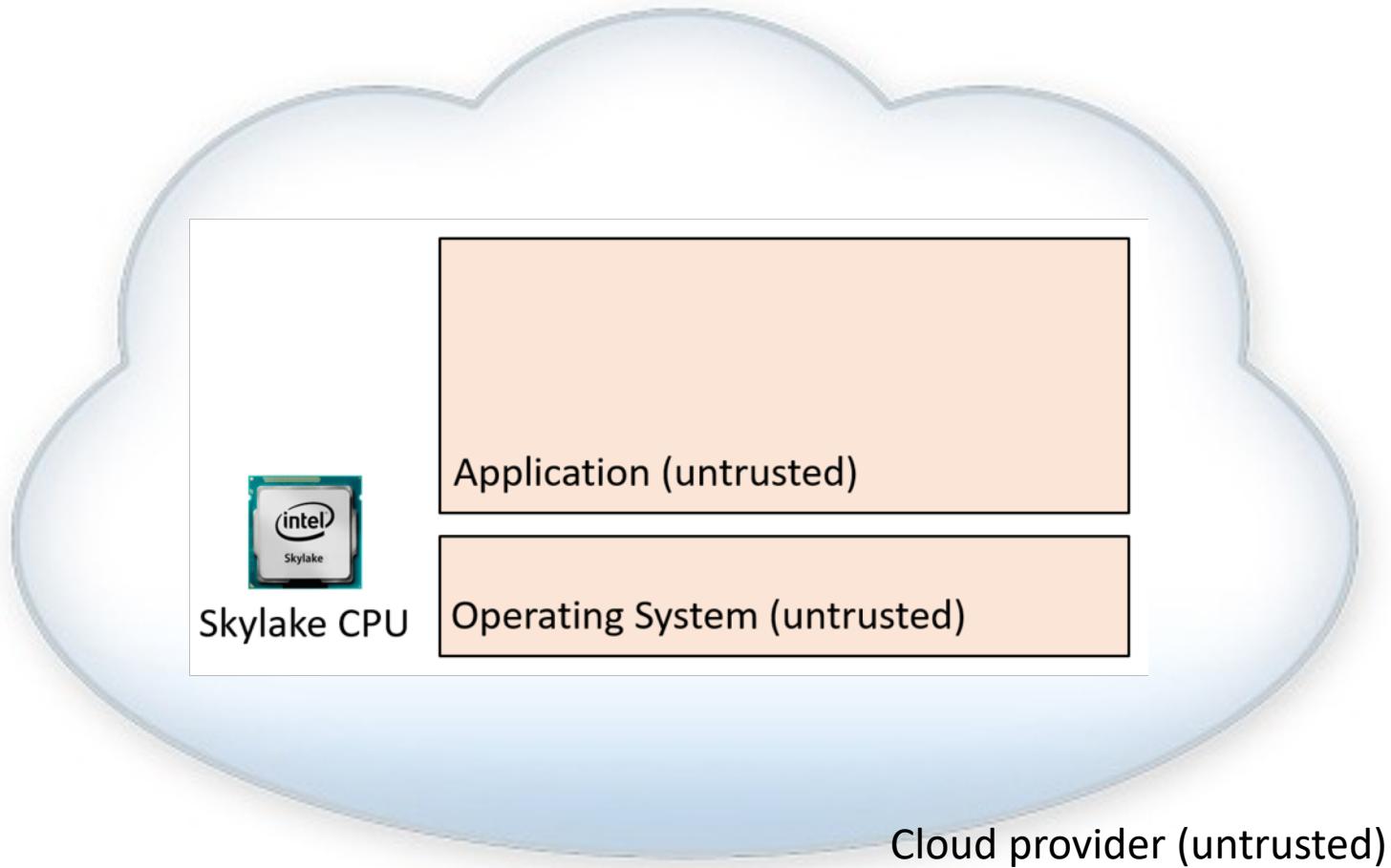


Dell OptiPlex 5040

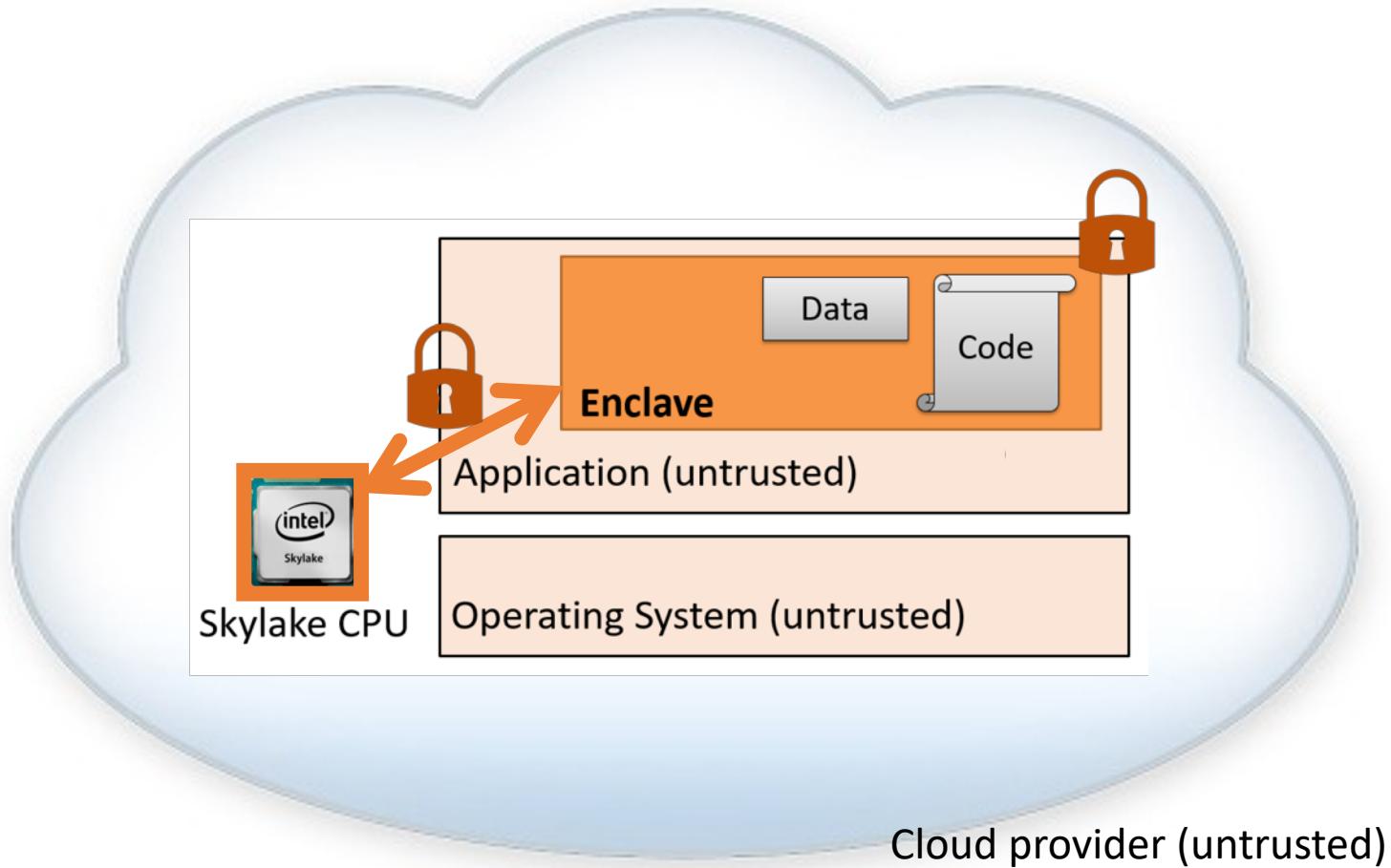


Supermicro Server

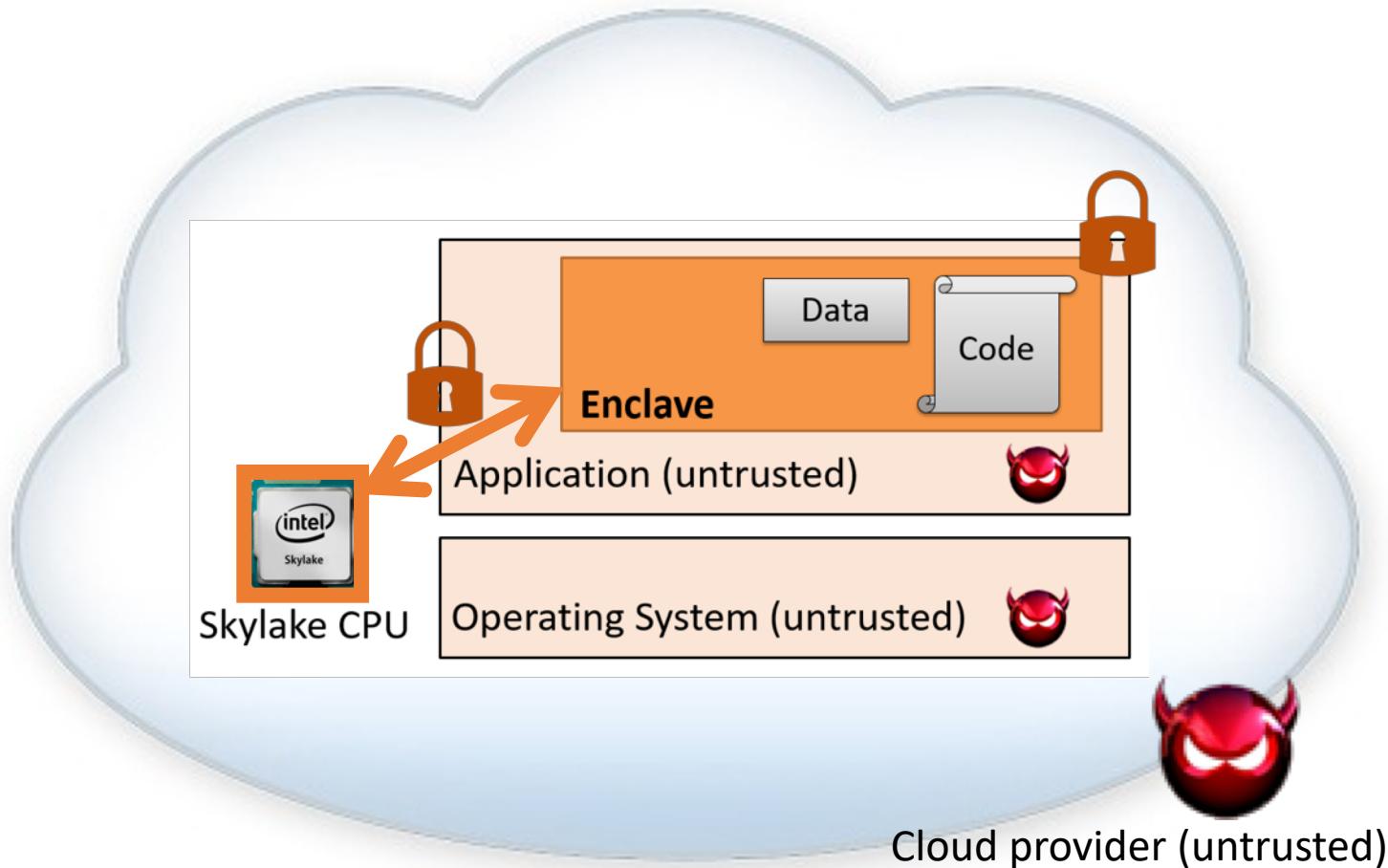
# Revisited: SGX for Cloud



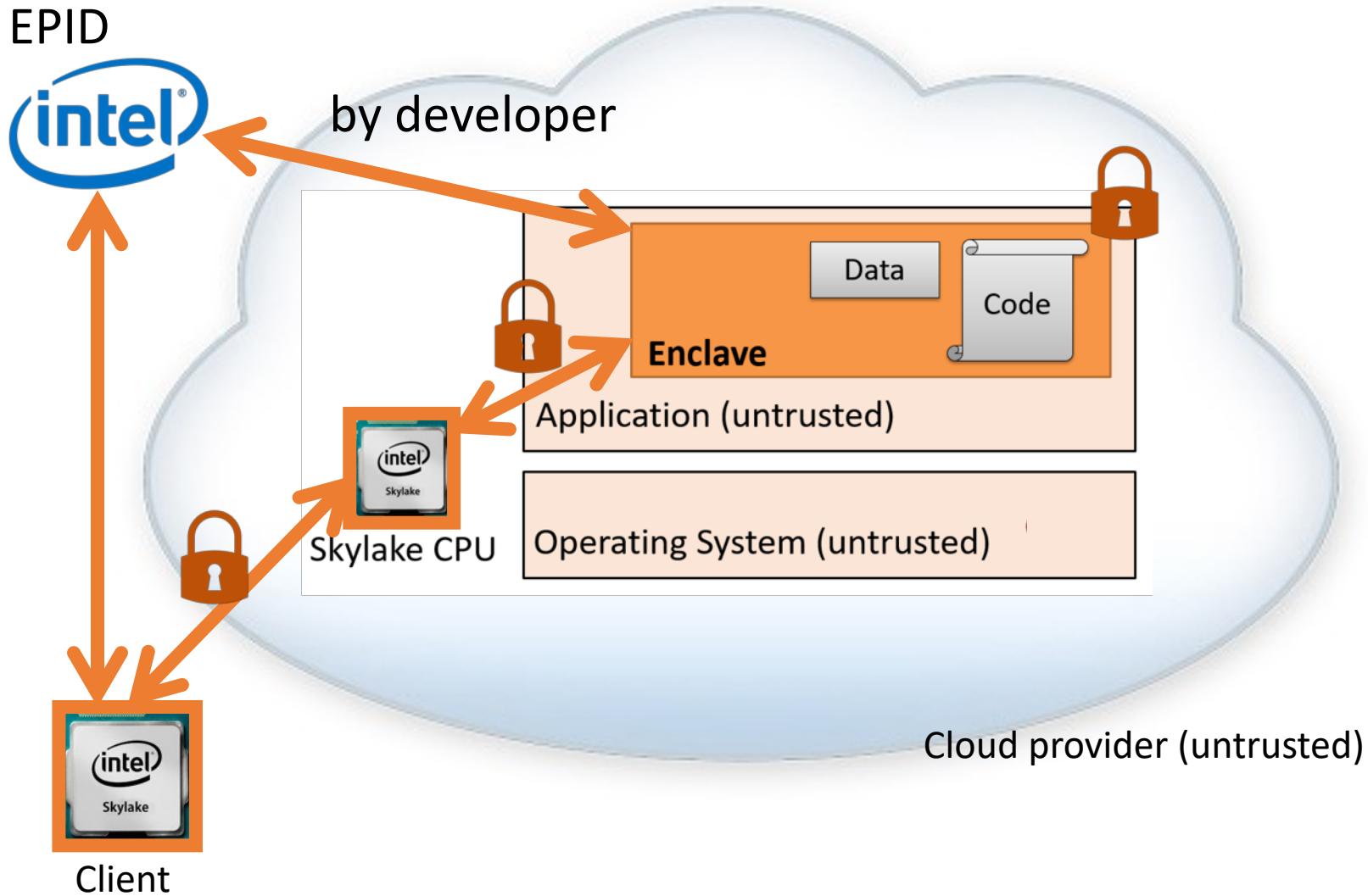
# Revisited: SGX for Cloud



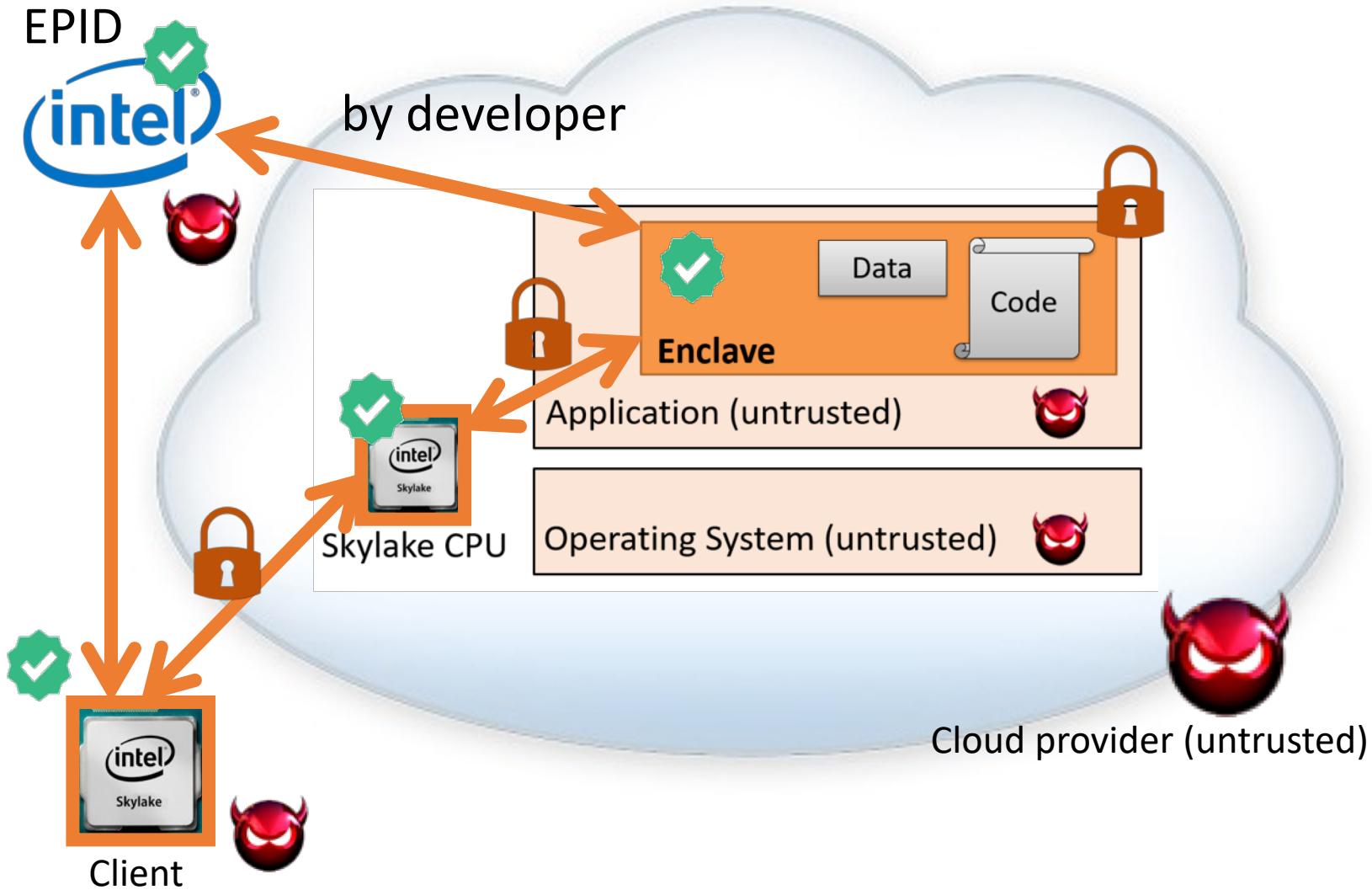
# Revisited: SGX for Cloud (Isolation)



# Revisited: SGX for Cloud (Remote attestation)



# Revisited: SGX for Cloud (Remote attestation)



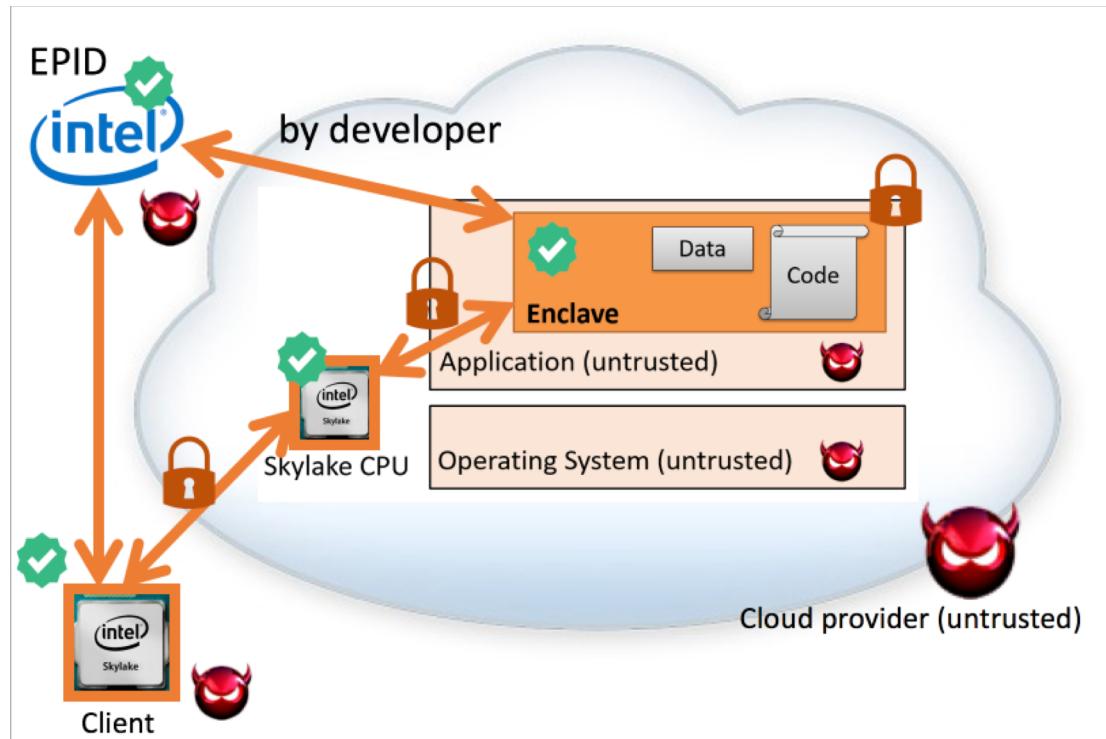
# SGX Ecosystem for Attackers



: Trusted components (i.e., where we should attack)



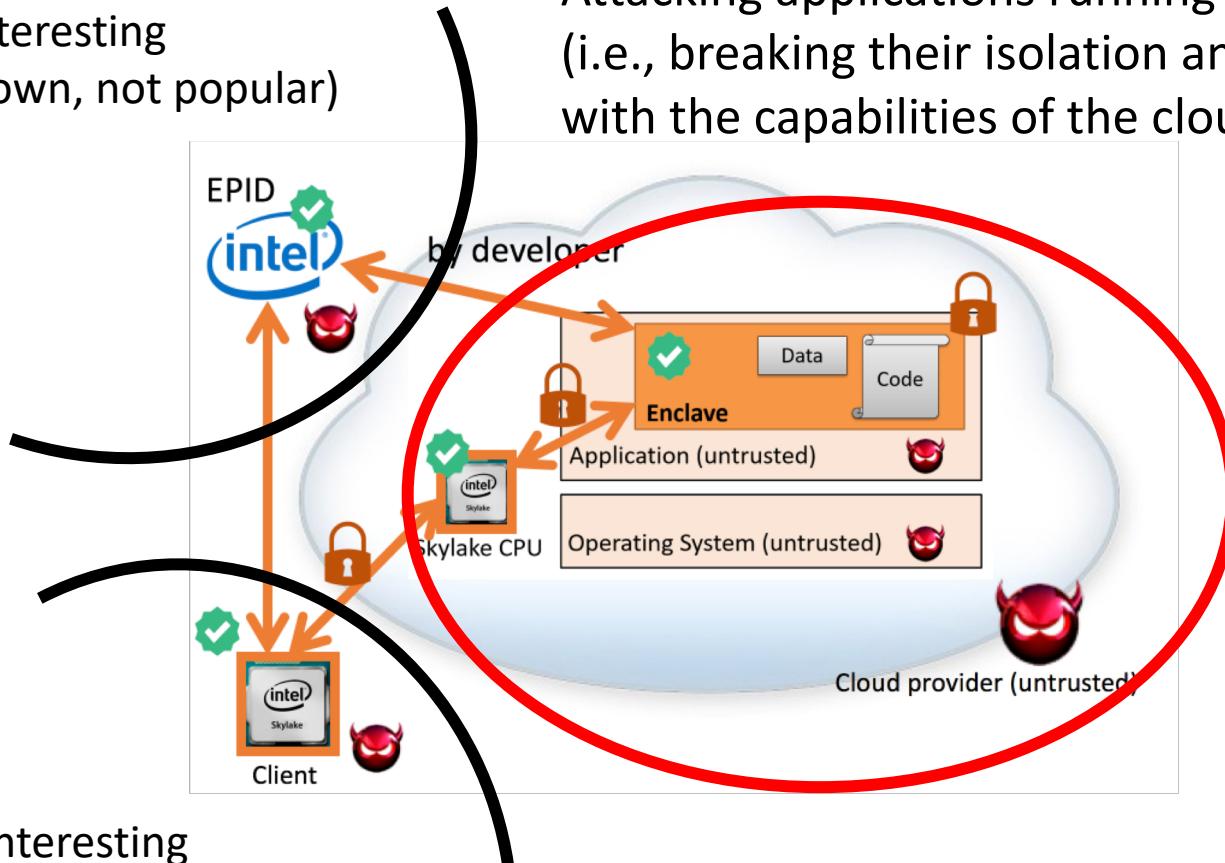
: Attacker's capabilities (i.e., what attackers can do)



# Our Initial Interests as Attacker

Not interesting  
(unknown, not popular)

Attacking applications running on enclaves  
(i.e., breaking their isolation and confidentiality)  
with the capabilities of the cloud provider



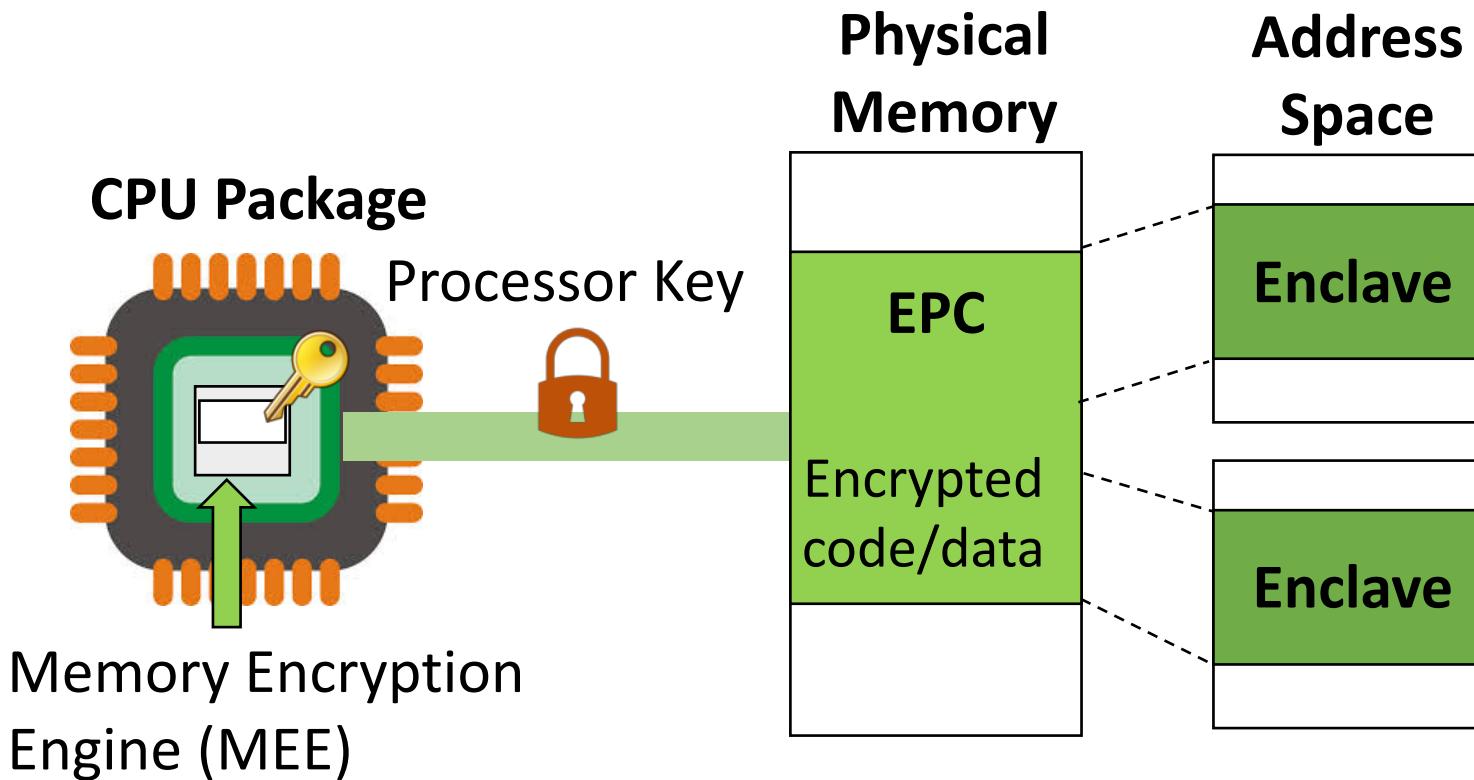
Not interesting  
(non technical issues)

# Summary: Intel SGX 101

- Two important design goals:
  - Performance (i.e., native speed, multithread)
  - General purpose (i.e., x86 ISA)
- Two important security primitives:
  - Isolated execution → confidentiality, integrity
  - Remote attestation → integrity

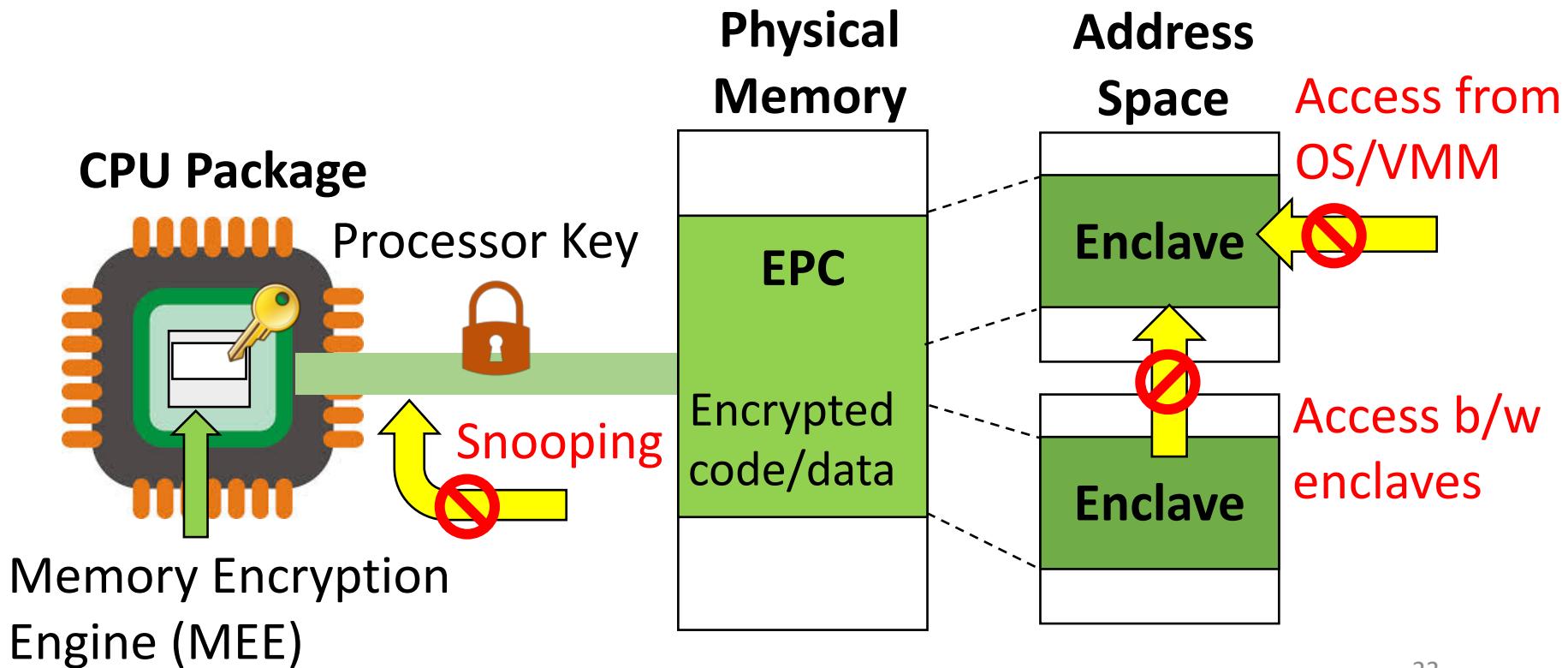
# Isolated Execution

- Protect enclaves from untrusted privilege software
- Small attack surface (TCB: App + CPU)



# Isolated Execution

- Protect enclaves from untrusted privilege software
- Small attack surface (TCB: App + CPU)



# SGX's Threat Model (very strong!)

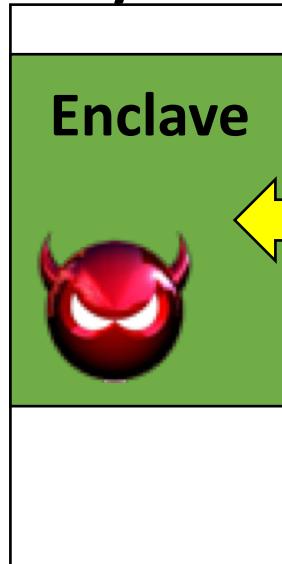
- *All* except the core package can be malicious
  - Device, firmware, ...
  - Operating systems, hypervisor ...
- DoS (availability) is naturally out of concern
- Intel excludes cache-based side-channel (due to performance)

# What if Enclave is Compromised?

- Leak sensitive information
- Prevent attackers from being audited/analyzed
- Permanently parasite to the enclave program

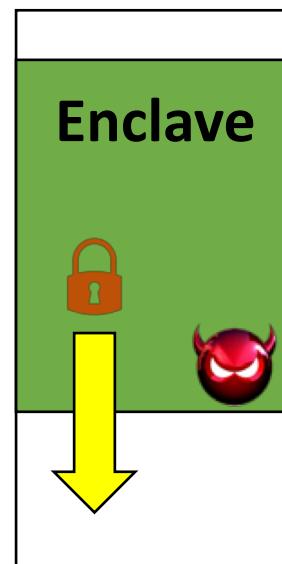
**Protected?**

by SGX

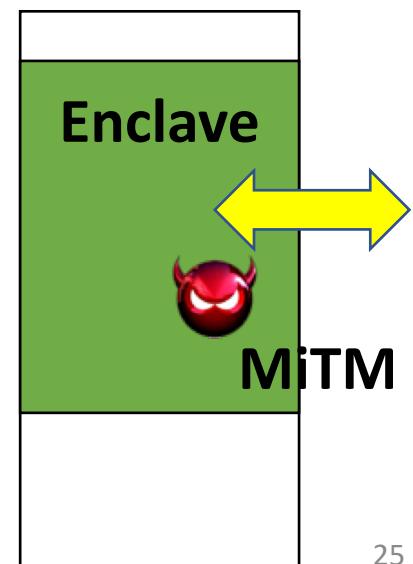


No access  
from  
OS/VMM

**Leak secret**



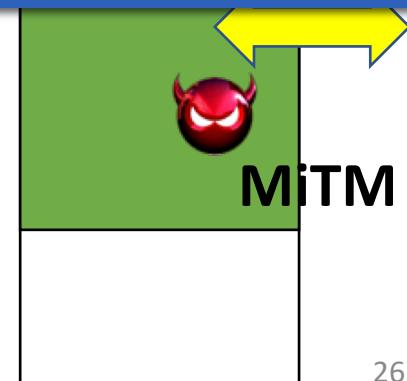
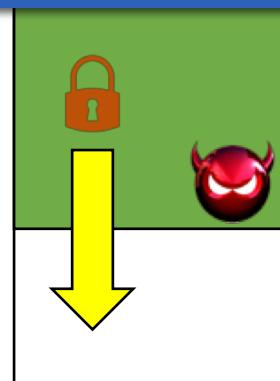
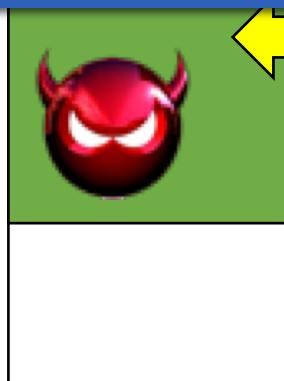
**Rootkit**



# What if Enclave is Compromised?

- Leak sensitive information

Due to 1) its strong threat model and  
2) consequences of compromises, developing a  
secure enclave program is ***much more difficult***  
than a typical program!



# Demonstrated Post Exploitation

- Dumping confidential data
  - e.g., memcpy(non-enclave region, enclave, size)
- Permanent parasite
  - e.g., MiTM on the remote attestation
- Breaking ecosystem
  - e.g., leaking attestation keys for Quoting enclaves

## Hacking in Darkness: Return-oriented Programming against Secure Enclaves

Jaehyuk Lee<sup>†</sup> Jinsoo Jang<sup>†</sup> Yeongjin Jang<sup>\*</sup> Nohyun Kwak<sup>‡</sup> Yesul Choi<sup>†</sup> Changho Choi<sup>‡</sup>  
Taesoo Kim<sup>\*</sup> Marcus Peinado<sup>¶</sup> Brent Byunghoon Kang<sup>‡</sup>

<sup>†</sup>KAIST

<sup>\*</sup>Georgia Institute of Technology

<sup>‡</sup>Microsoft Research

### Abstract

Intel Software Guard Extensions (SGX) is a hardware-based Trusted Execution Environment (TEE) that is widely seen as a promising solution to traditional security threats. While SGX promises strong protection to bug-free software, decades of experience show that we have to expect vulnerabilities in any non-trivial application. In a traditional environment, such vulnerabilities often allow attackers to take complete control of vulnerable systems. Efforts to evaluate the security of SGX have focused on

The consequences of Dark ROP are alarming; the attacker can completely breach the enclave's memory protections and trick the SGX hardware into disclosing the enclave's encryption keys and producing measurement reports that defeat remote attestation. This result strongly suggests that SGX is not a silver bullet for improving traditional security. It also highlights the need for enclave developers to understand the security implications of trusted computing hardware, such as SGX, Intel ME, and Qualcomm Haven).

SEC'17

## FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution

Jo Van Bulck<sup>1</sup>, Marina Minkin<sup>2</sup>, Ofir Weisse<sup>3</sup>, Daniel Genkin<sup>3</sup>, Baris Kasikci<sup>3</sup>, Frank Piessens<sup>1</sup>,  
Mark Silberstein<sup>2</sup>, Thomas F. Wenisch<sup>3</sup>, Yuval Yarom<sup>4</sup>, and Raoul Strackx<sup>1</sup>

<sup>1</sup>imec-DistriNet, KU Leuven, <sup>2</sup>Technion, <sup>3</sup>University of Michigan, <sup>4</sup>University of Adelaide and Data61

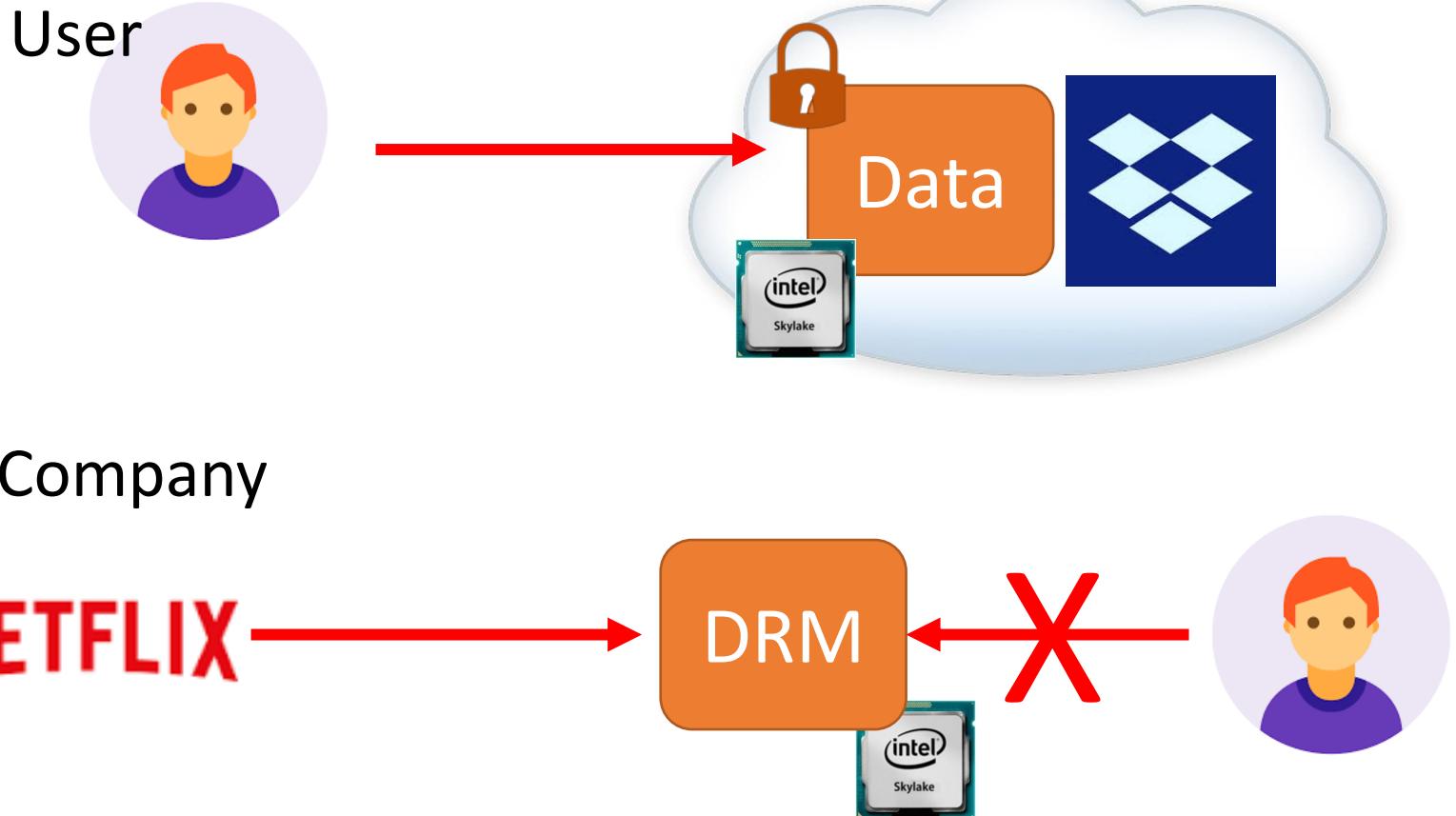
### Abstract

Trusted execution environments, and particularly the Software Guard eXtensions (SGX) included in recent Intel x86 processors, gained significant traction in recent years. A long track of research papers, and increasingly also real-world industry applications, take advantage of the strong hardware-enforced confidentiality and integrity guarantees provided by SGX. However, SGX is not immune to potentially malicious attacks.

distrusting enclaves with a minimal Trusted Computing Base (TCB) that includes only the processor package and microcode. Enclave-private CPU and memory state is exclusively accessible to the code running inside it, and remains explicitly out of reach of all other enclaves and software running outside the TCB. However, SGX is not immune to potentially malicious attacks.

SEC'18

# Thinking of SGX Usages



e.g., prevent reverse engineering  
(or DRM data)

# Traditional Attack Vectors

- Cache-based side channel
- Memory safety
- Weak mitigation techniques
- Uninitialized padding in EDL

# Traditional Attack Vectors

- Cache-based side channel
  - e.g., inferring a private key
- Memory safety
  - e.g., control flow hijacking
- Weak mitigation techniques
  - e.g., breaking ALSR
- Uninitialized padding in EDL
  - e.g., leaking security sensitive information

# Cache-based Side-channel Attacks

CacheZoom: How SGX Amplifies  
The Power of Cache Attacks

arXiv'17

Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth

Worcester  
{anog}

## Software Grand Exposure: SGX Cache Attacks Are Practical

**Abstract.** In modern systems, memory is commonly shared, a fact that can cause privacy and security problems. Intel proposes to mitigate these issues within the processor by introducing SGX.

Ferdinand Brasser<sup>1</sup>, Urs Müller<sup>2</sup>, Alexandra Dmitrienko<sup>2</sup>, Kari Kostiainen<sup>2</sup>, Srdjan Capkun<sup>2</sup>, and Ahmad-Reza Sadeghi<sup>1</sup>

EuroSec'17

## Cache Attacks on Intel SGX

Johannes Götzfried  
FAU Erlangen-Nürnberg  
johannes.gotzfried@fau.de

Moritz Eckert  
FAU Erlangen-Nürnberg  
moritz.eckert@fau.de

Sebastian Schinzel  
FH Münster  
schinzel@fh-muenster.de

arXiv'17

[.CR] 24 Feb 2017

### ABSTRACT

For the first time, we practical demonstrate that SGX enclaves are vulnerable against side-channel attacks. As a case study, we present an efficient cache attack on AES when running in SGX enclaves. Using Neve and Seifert's elimination of the cache probing mechanism relying on the SGX cache probing mechanism, we extract the AES secret key in less than 10 minutes by investigating 480 encrypted blocks.

## Malware Guard Extension: Using SGX to Conceal Cache Attacks (Extended Version)

Michael Schwarz  
Graz University of Technology  
Email: michael.schwarz@iaik.tugraz.at

Samuel Weiser  
Graz University of Technology  
Email: samuel.weiser@iaik.tugraz.at

Daniel Gruss  
Graz University of Technology  
Email: daniel.gruss@iaik.tugraz.at

Clémentine Maurice  
Graz University of Technology  
Email: clementine.maurice@iaik.tugraz.at

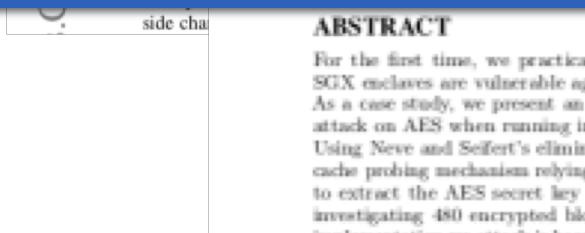
Stefan Mangard  
Graz University of Technology  
Email: stefan.mangard@iaik.tugraz.at

ar 2017

# Cache-based Side-channel Attacks

CacheZoom: How SGX Amplifies  
The Power of Cache Attacks

*Cache attacks are possible and often, makes it “easier” to launch the attack due to its strong threat model (e.g., using PMC)*  
→ Known defenses (e.g., coloring ...)



## Using SGX to Conceal Cache Attacks (Extended Version)

Michael Schwarz  
Graz University of Technology  
Email: michael.schwarz@iaik.tugraz.at

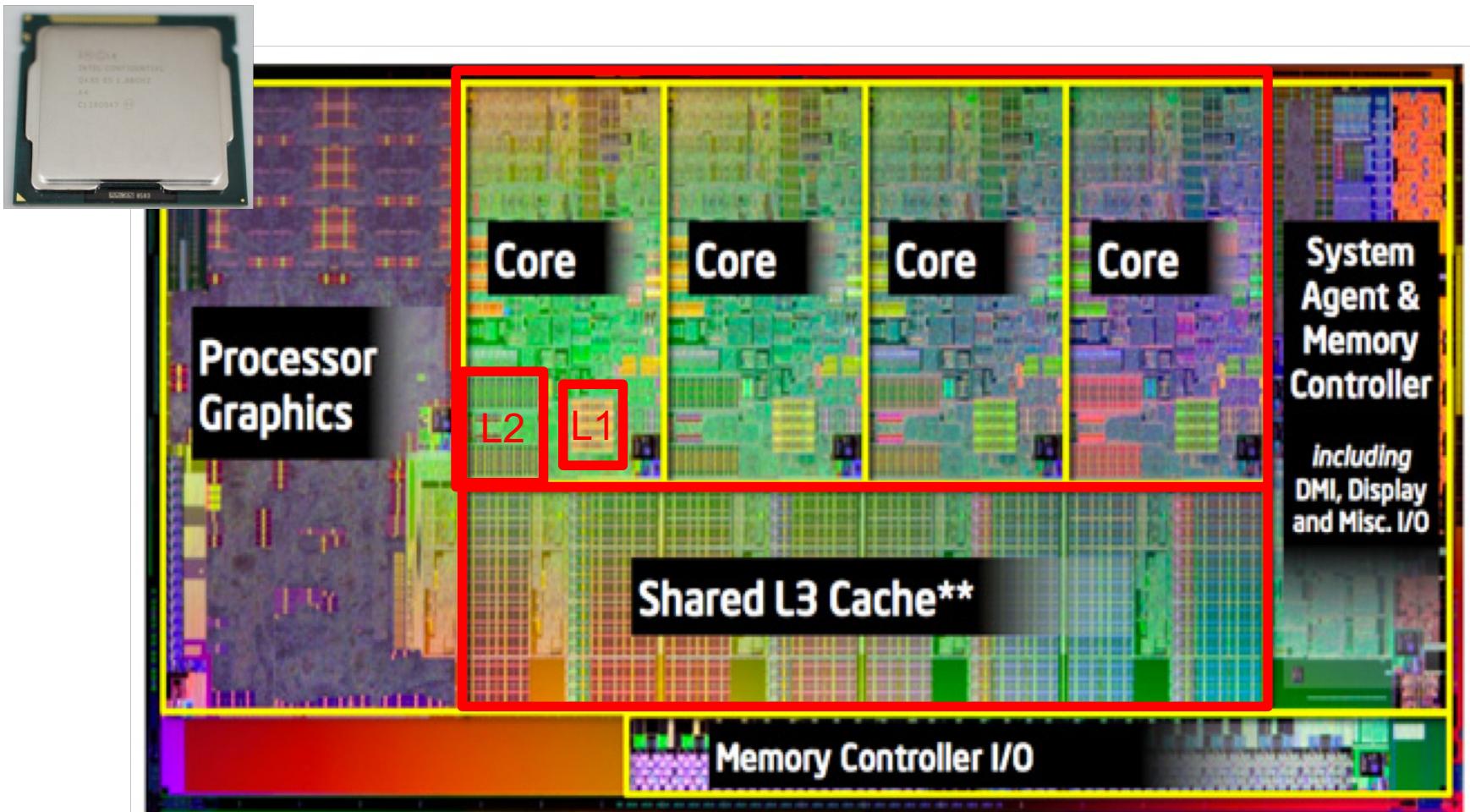
Samuel Weiser  
Graz University of Technology  
Email: samuel.weiser@iaik.tugraz.at

Daniel Gruss  
Graz University of Technology  
Email: daniel.gruss@iaik.tugraz.at

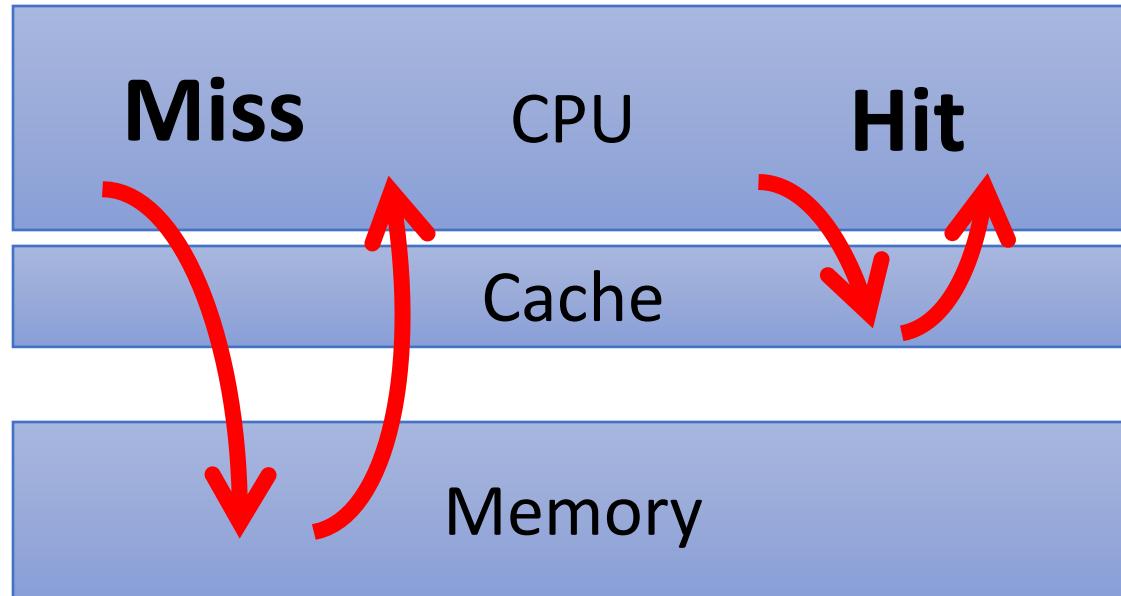
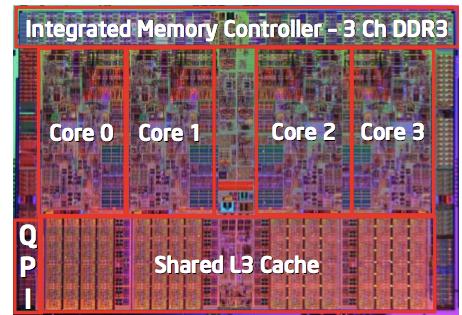
Clémentine Maurice  
Graz University of Technology  
Email: clementine.maurice@iaik.tugraz.at

Stefan Mangard  
Graz University of Technology  
Email: stefan.mangard@iaik.tugraz.at

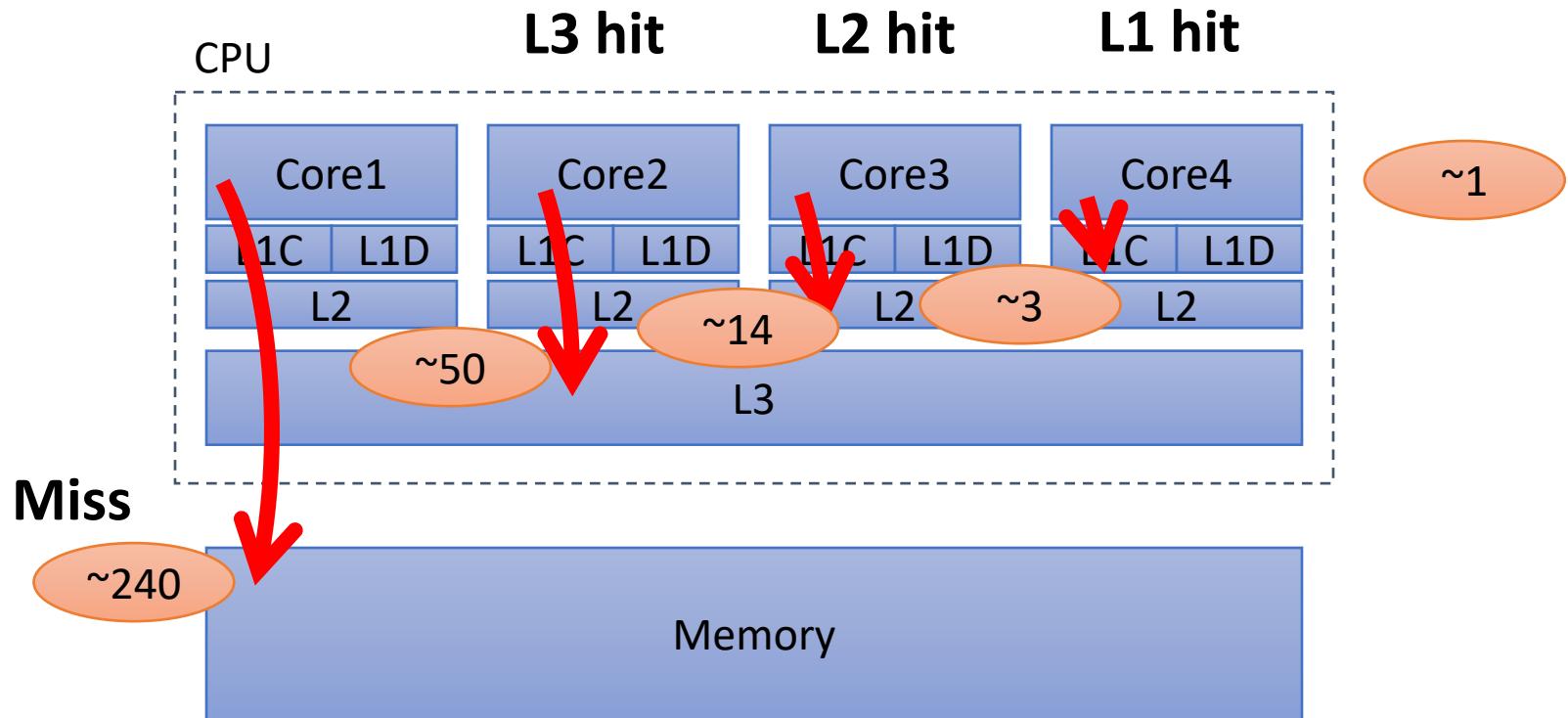
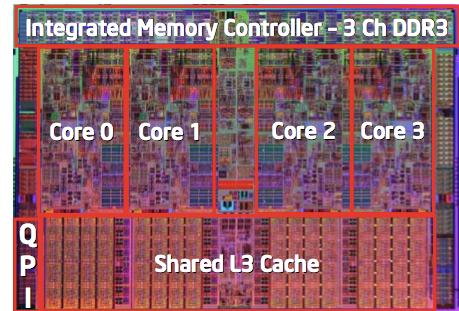
# CS101: Cache Structure



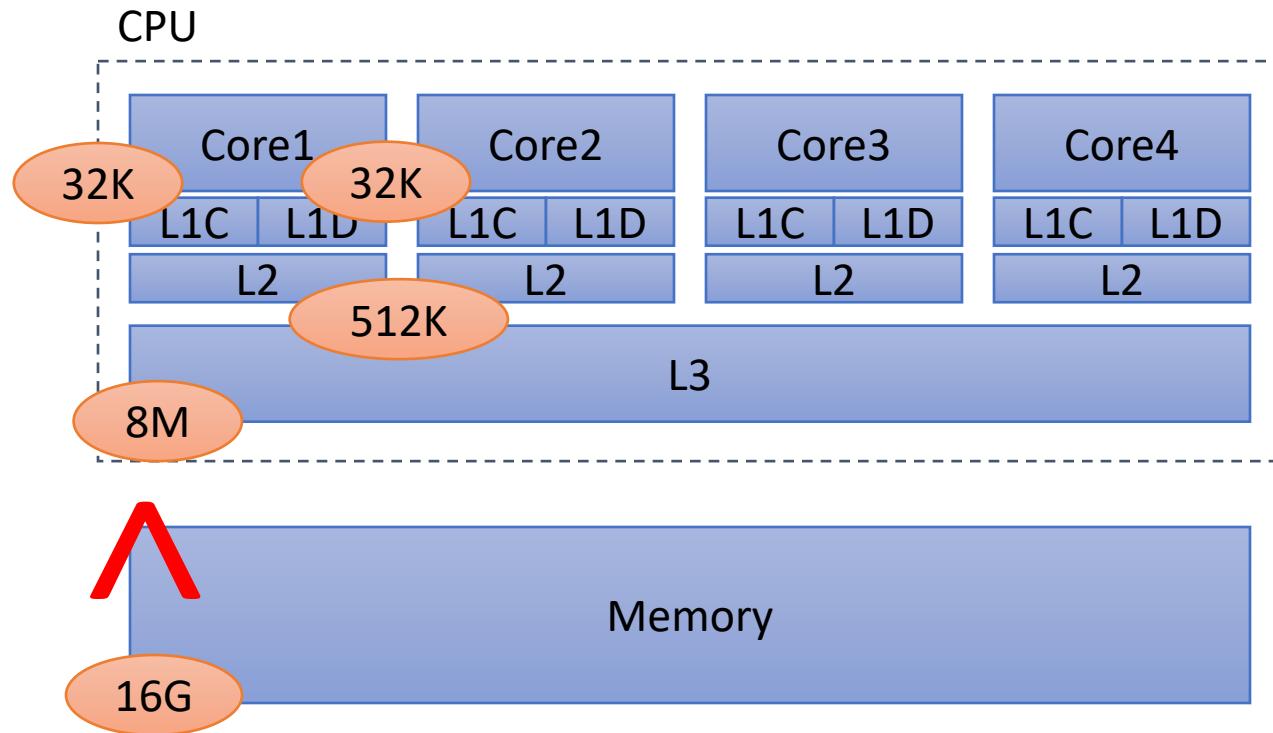
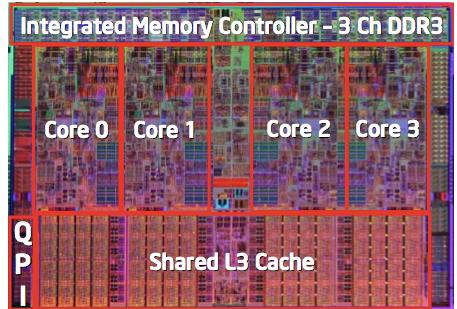
# CS101: Cache



# CS101: Cache

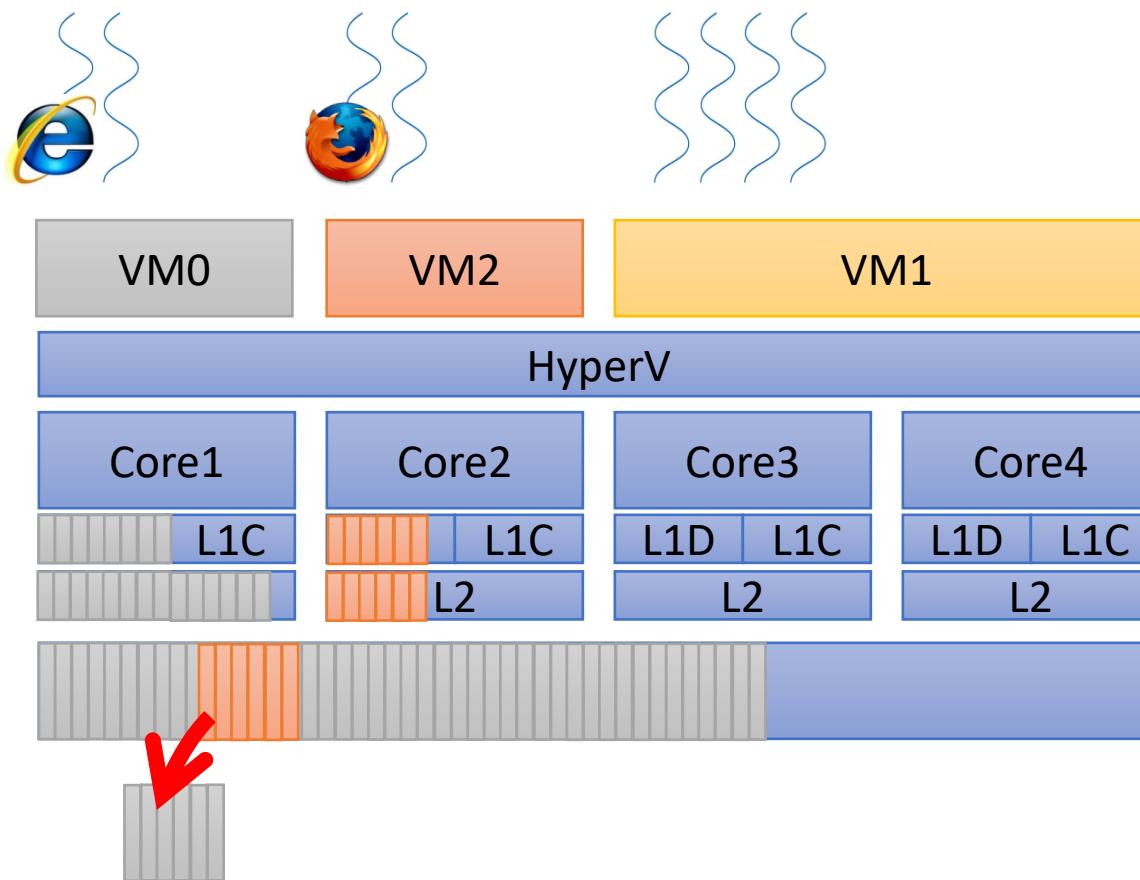


# CS101: Cache



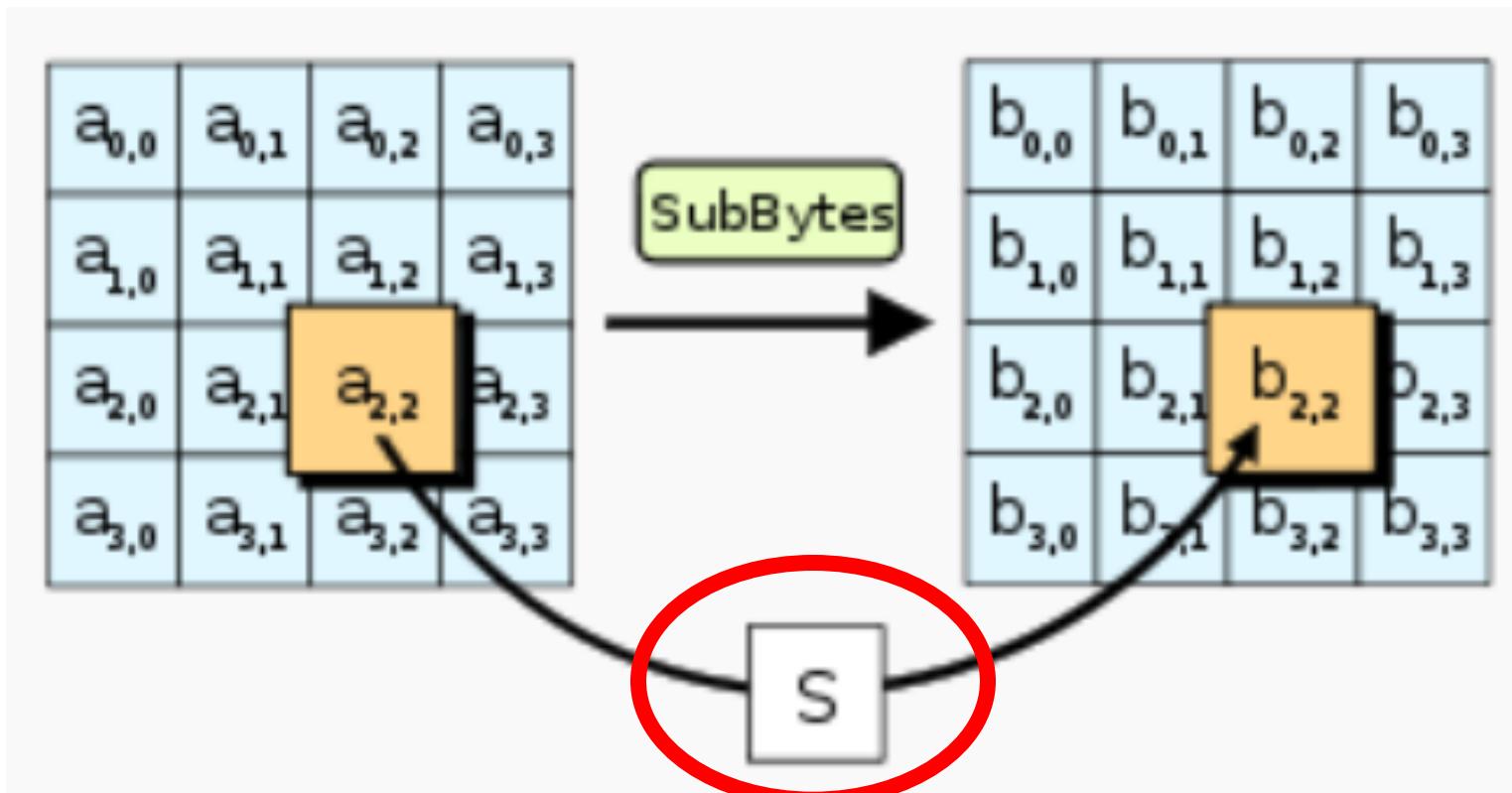
Which cacheline do we have to keep/evict (policy)?  
How to organize cacheline (structure)?

# Basic Idea: Cache Side-channel



# Real Attack: AES?

{SubBytes} + ShiftRows + MixColumns + AddRoundKeys} x {10, 12, 14}

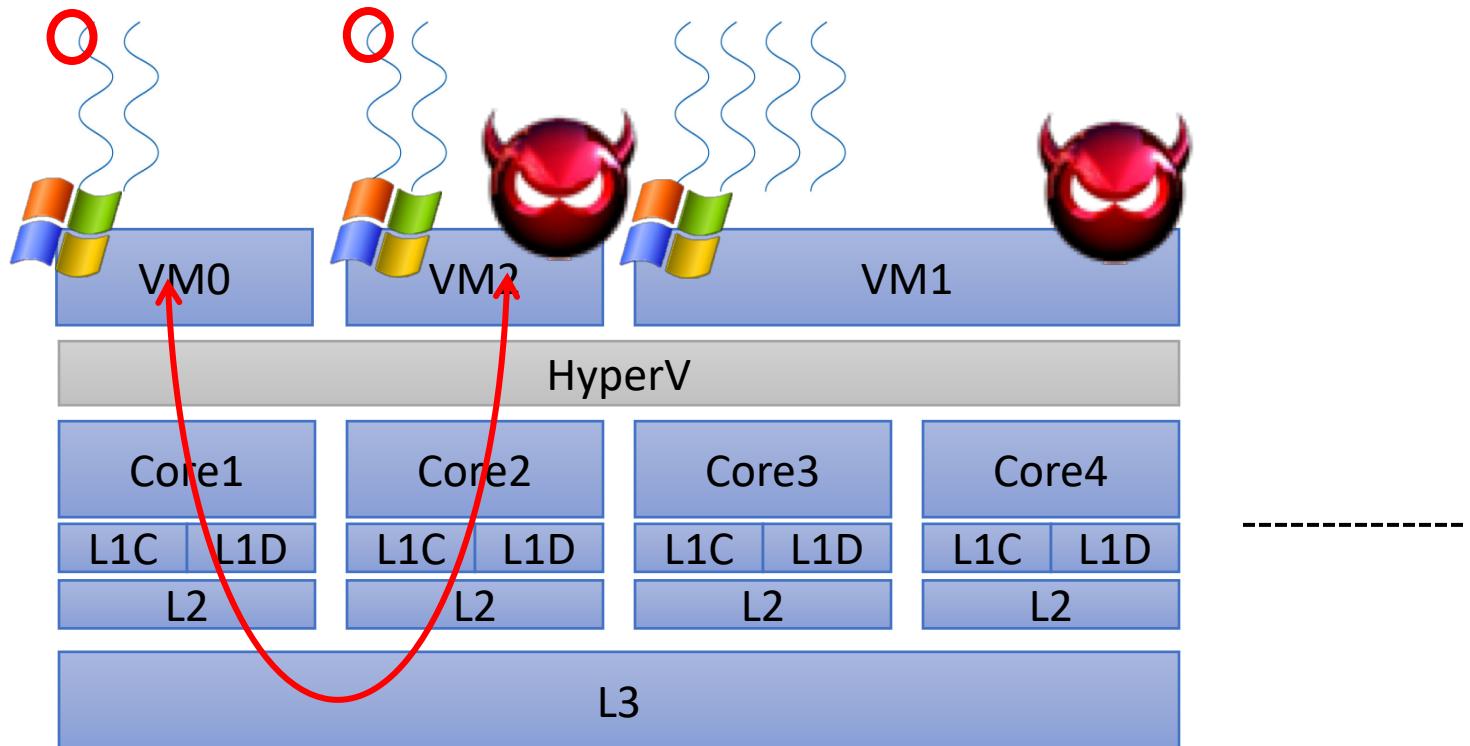


Monitoring cacheline access of Lookup Table!

# Known Attack Demonstrations

- Known cache-based side channel attacks:
  - 2003 DES by Tsunoo et al. (with 226.0 samples)
  - 2005 AES by Bernstein et al. (with 218.9 samples)
  - 2005 RSA by Percival et al. (-)
  - ...
  - 2011 AES by Gullasch et al. (with 26.6 samples)
  - ...
  - 2017 AES by Ahmad et al. (with 10 samples against SGX)

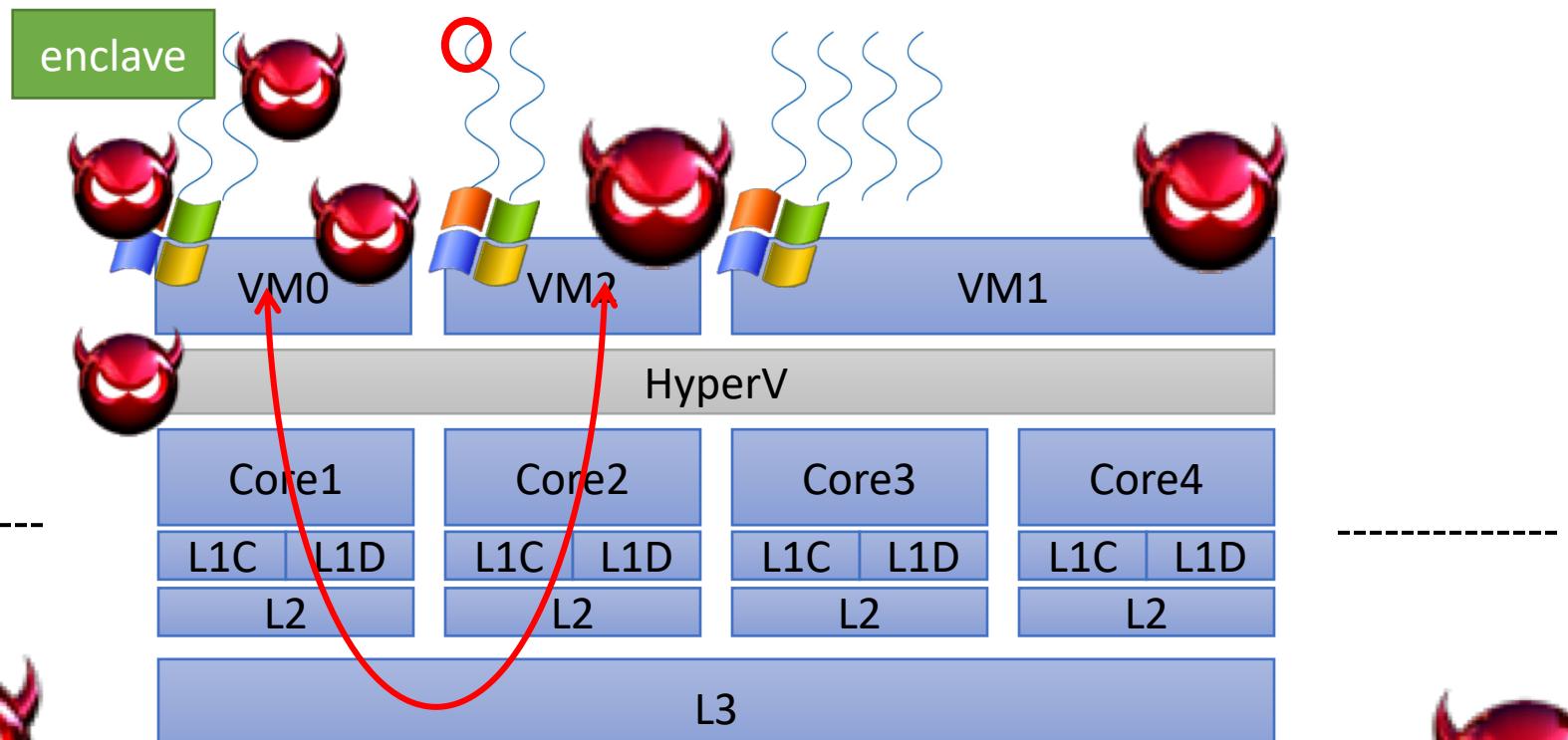
# Cache Side-channel (in Cloud)



 Windows Azure™



# Cache Side-channel against SGX



# Thinking of SGX Adversaries: SGX Makes Cache Attack Easier

- Accurate intervention (i.e., scheduling/exception)
- Controlled environment (i.e., OS, hyperthread)
- Rich information available (e.g., physical mapping, PMC)

## CacheZoom: How SGX Amplifies The Power of Cache Attacks

Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth

Worcester Polytechnic Institute, Worcester, MA, USA  
[{ahmoghimi,girazoki,teisenbarth}@wpi.edu](mailto:{ahmoghimi,girazoki,teisenbarth}@wpi.edu)

**Abstract.** In modern computing environments, computation is commonly shared, and parallel computation can cause privacy and security problems. Intel SGX enforces. Intel proposed SGX to create a trusted execution environment within the processor. SGX relies on the hardware, and claims runtime

arXiv'17

## Software Grand Exposure: SGX Cache Attacks Are Practical

Ferdinand Brasser<sup>1</sup>, Urs Müller<sup>2</sup>, Alexandra Dmitrienko<sup>2</sup>, Kari Kostiainen<sup>2</sup>, Srdjan Capkun<sup>2</sup>, and Ahmad-Reza Sadeghi<sup>1</sup>

<sup>1</sup>System Security Lab, Technische Universität Darmstadt, Germany  
[{ferdinand.brasser,ahmad.sadeghi}@trust.tu-darmstadt.de](mailto:{ferdinand.brasser,ahmad.sadeghi}@trust.tu-darmstadt.de)

<sup>2</sup>Institute of Information Security, ETH Zurich, Switzerland  
[{alexandra.dmitrienko,kari.kostiainen,srdjan.capkun}@inf.ethz.ch](mailto:muurs@student.ethz.ch,{alexandra.dmitrienko,kari.kostiainen,srdjan.capkun}@inf.ethz.ch)

### Abstract

Side-channel information leakage is a known limitation of SGX. Researchers have demonstrated that secret-dependent information can be extracted from enclave execution through page-fault access patterns. Consequently, various recent research efforts are actively seeking countermeasures to SGX side-channel attacks. It is widely assumed that SGX may be vulnerable to other side channels, such as cache access pattern monitoring,

that can issue remotely verifiable attestation statements on enclave software configuration. These SGX mechanisms (isolation, measurement) are part of application security. The SGX mechanism can be outsourced to an external computing infrastructure without having to fully trust the cloud provider and the entire software stack.<sup>1</sup>

WOOT'17

# Cache Attack is Practical Concern?

- Yes or no, depending on contexts and applications.
  - Think first: why considering SGX? on cloud?
- Performance (= cache) vs. potential risks!
- SGX can make the cache attack harder too
  - By leveraging isolation / randomization  
(security by obscurity practical)

→ Intel *explicitly* noted *that it's better to be addressed in SW* (if you wish) rather than HW (by default).

# Breaking Remote Attestation via Cache-based Side-channel Attacks

## CacheQuote: Efficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks

Fergus Dall<sup>1</sup>, Gabrielle De Micheli<sup>2</sup>, Thomas Eisenbarth<sup>3,4</sup>, Daniel Genkin<sup>2,5</sup>, Nadia Heninger<sup>2</sup>, Ahmad Moghimi<sup>4</sup> and Yuval Yarom<sup>1,6</sup>

<sup>1</sup> University of Adelaide

[fergus@beware.dropbear.id.au](mailto:fergus@beware.dropbear.id.au), [yval@cs.adelaide.edu.au](mailto:yval@cs.adelaide.edu.au)

<sup>2</sup> University of Pennsylvania

[{gmicheli,danielg3,nadia}@cis.upenn.edu](mailto:{gmicheli,danielg3,nadia}@cis.upenn.edu)

<sup>3</sup> University of Lübeck

[thomas.eisenbarth@uni-luebeck.de](mailto:thomas.eisenbarth@uni-luebeck.de)

<sup>4</sup> Worcester Polytechnic Institute

[amoghimi@wpi.edu](mailto:amoghimi@wpi.edu)

<sup>5</sup> University of Maryland

<sup>6</sup> Data61

**Abstract.** Intel Software Guard Extensions (SGX) allows users to perform secure computation on platforms that run untrusted software. To validate that the computation is correctly initialized and that it executes on trusted hardware, SGX supports attestation providers that can vouch for the user's computation. Communication with these attestation providers is based on the Extended Privacy ID (EPID).

IACR'18

# Defense: Cache Attacks

- Cache oblivious implementation of crypto algos
  - Fine-grained code/data randomization
  - Mitigating via contiguous monitoring (e.g., Varys)
  - Looking for better HW-based solutions!  
(e.g., partitioning/coloring)

Varys

Protecting SGX Enclaves From Practical Side-Channel Attack

Oleksii Oleksenko<sup>†</sup>, Bohdan Trach<sup>†</sup>, Robert Krah<sup>†</sup>, Andre Martin<sup>†</sup>  
Christof Fetzer<sup>†</sup>, Mark Silberstein<sup>‡</sup>  
<sup>†</sup>*TU Dresden*, <sup>‡</sup>*Technion*

## Abstract

Numerous recent works have experimentally shown that Intel Software Guard Extensions (SGX) are vulnerable to cache timing and page table side-channel attacks which could be used to circumvent the data confidentiality guarantees provided by SGX. Existing mechanisms that protect against these attacks either incur high execution costs, are ineffective against certain attack variants, or require significant code modifications.

cludes side channels from the SGX threat model, SCAs effectively circumvent the SGX confidentiality guarantees and impede SGX adoption in many real-world scenarios. More crucially, a privileged adversary against SGX can mount much more powerful SCAs compared to the unprivileged example, levels by slow

ATC'18

Sanctum: Minimal Hardware Extensions for Strong Software Isolation

*Victor Costan, Ilya Lebedev, and Srinivas Devadas*  
*victor@costan.us, ilebedev@mit.edu, devadas@mit.edu*

## Abstract

Sanctum offers the same promise as Intel's Software Guard Extensions (SGX), namely strong provable isolation of software modules running concurrently and sharing resources, but protects against an important class of additional software attacks that infer private information from a program's memory access patterns. Sanctum shuns unnecessary complexity, leading to a simpler security analysis. We follow a principled approach to eliminating entire attack surfaces through isolation, rather than

formal verification effort [26] spent 20 *man-years* to cover 9,000 lines of code.

Given Linux and Xen's history of vulnerabilities and uncertain prospects for formal verification, a prudent system designer cannot include either in a TCB (trusted computing base) and must look elsewhere for software isolation needs.

Fortuna [5, 36] having software

SEC'16

# Traditional Attack Vectors

- Cache-based side channel
  - e.g., inferring a private key
- Memory safety
  - e.g., control flow hijacking
- Weak mitigation techniques
  - e.g., breaking ALSR
- Uninitialized padding in EDL
  - e.g., leaking security sensitive information

# Memory Safety Issues

- SGX is not free from memory safety issues
- Current ecosystem is built on memory unsafe lang.

## Hacking in Darkness: Return-oriented Programming against Secure Enclaves

Jaehyuk Lee<sup>†</sup> Jinsoo Jang<sup>†</sup> Yeongjin Jang<sup>\*</sup> Nohyun Kwak<sup>‡</sup> Yeseul Choi<sup>†</sup> Changho Choi<sup>‡</sup>  
Taesoo Kim<sup>\*</sup> Marcus Peinado<sup>†</sup> Brent Byunghoon Kang<sup>‡</sup>

<sup>†</sup>KAIST

<sup>\*</sup>Georgia Institute of Technology

<sup>‡</sup>Microsoft Research

### Abstract

Intel Software Guard Extensions (SGX) is a hardware-based Trusted Execution Environment (TEE) that is widely seen as a promising solution to traditional security threats. While SGX promises strong protection to bug-free software, recent research shows that we have to exercise care when using it. In a traditional application, memory corruption bugs often allow attackers to compromise the system. Efforts to mitigate such bugs have focused on

The consequences of Dark ROP are alarming; the attacker can completely breach the enclave's memory protections and trick the SGX hardware into disclosing the enclave's encryption keys and producing measurement reports that defeat remote attestation. This result strongly suggests that SGX research should focus more on traditional security mitigations rather than on making enclave development more convenient by expanding the trusted computing base and the attack surface (e.g., Graphene, Haven).

SEC'17

## The Guard's Dilemma: Efficient Code-Reuse Attacks Against Intel SGX

Andrea Biondo, Mauro Conti

*University of Padua, Italy*

Lucas Davi

*University of Duisburg-Essen, Germany*

Tommaso Frassetto, Ahmad-Reza Sadeghi  
*TU Darmstadt, Germany*

### Abstract

Intel Software Guard Extensions (SGX) isolate security-critical code inside a protected memory area called enclave. Previous research on SGX has demonstrated that memory corruption vulnerabilities within enclave code can be exploited to extract secret keys and bypass remote attestation. However, these attacks require kernel privileges, and rely on frequently probing enclave code which results in many enclave crashes. Further, they assume a constant, not randomized memory layout.

In this paper, we present novel exploitation techniques against SGX that do not require any enclave crashes and

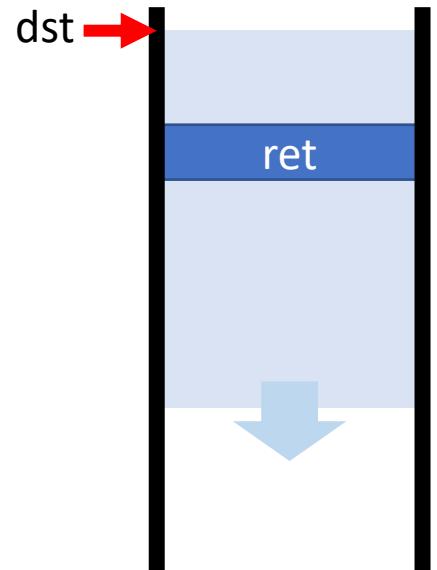
using one of the pre-defined entry points. The enclave can subsequently perform sensitive computations, call pre-defined functions in the host, and return to the caller.

In the ideal scenario, the enclave code only includes minimal carefully-inspected code, which could be formally verified to be free of vulnerabilities. However, legacy software is not always written in this style. SGX does not provide any protection against memory corruption in legacy code. Furthermore, SGX does not provide any protection against memory-corruption vulnerabilities that plague legacy software are also very likely to occur in those complex

SEC'18

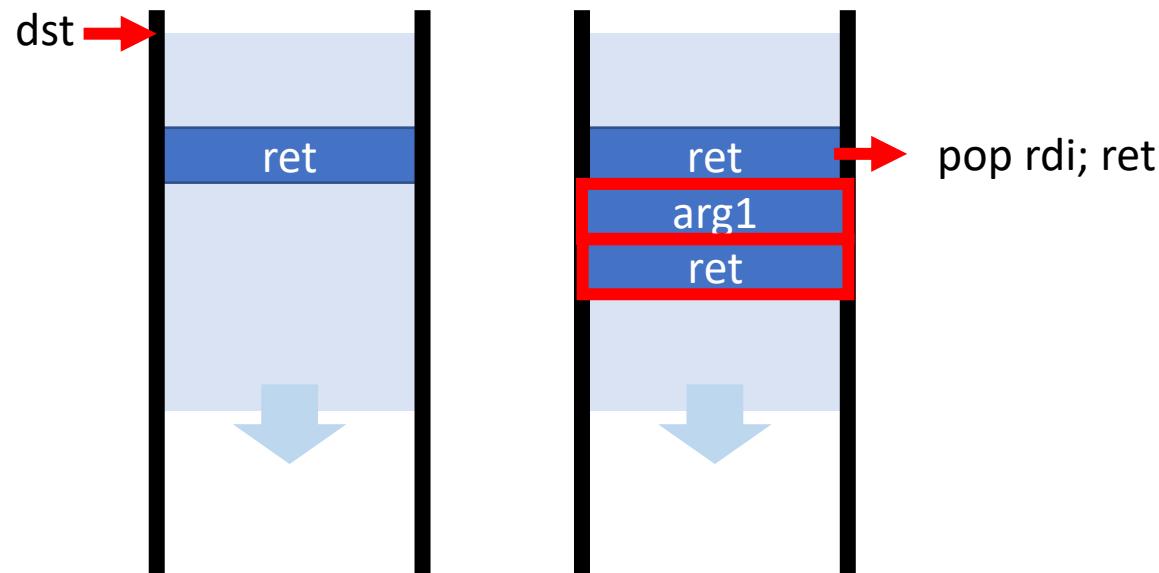
# Return-oriented Programming (ROP)

```
void vuln(char *input) {  
    char dst[0x100];  
    memcpy(dst, input, 0x200);  
}
```



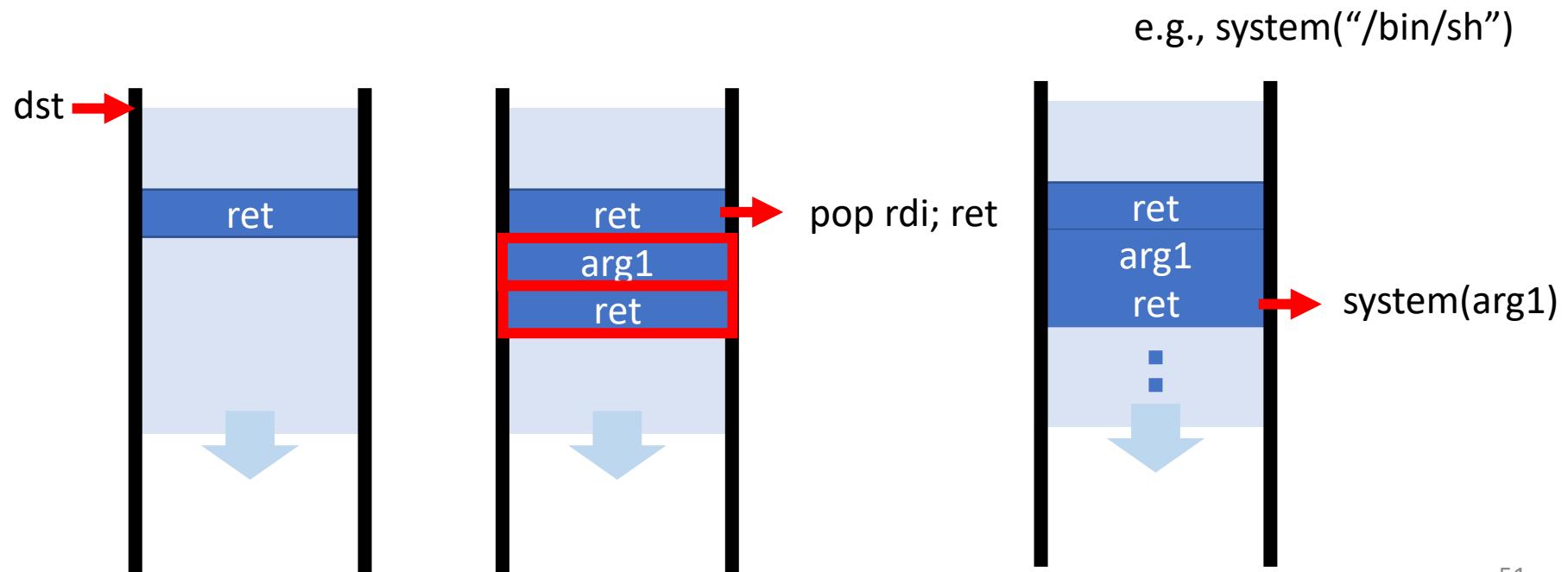
# Return-oriented Programming (ROP)

```
void vuln(char *input) {  
    char dst[0x100];  
    memcpy(dst, input, 0x200);  
}
```



# Return-oriented Programming (ROP)

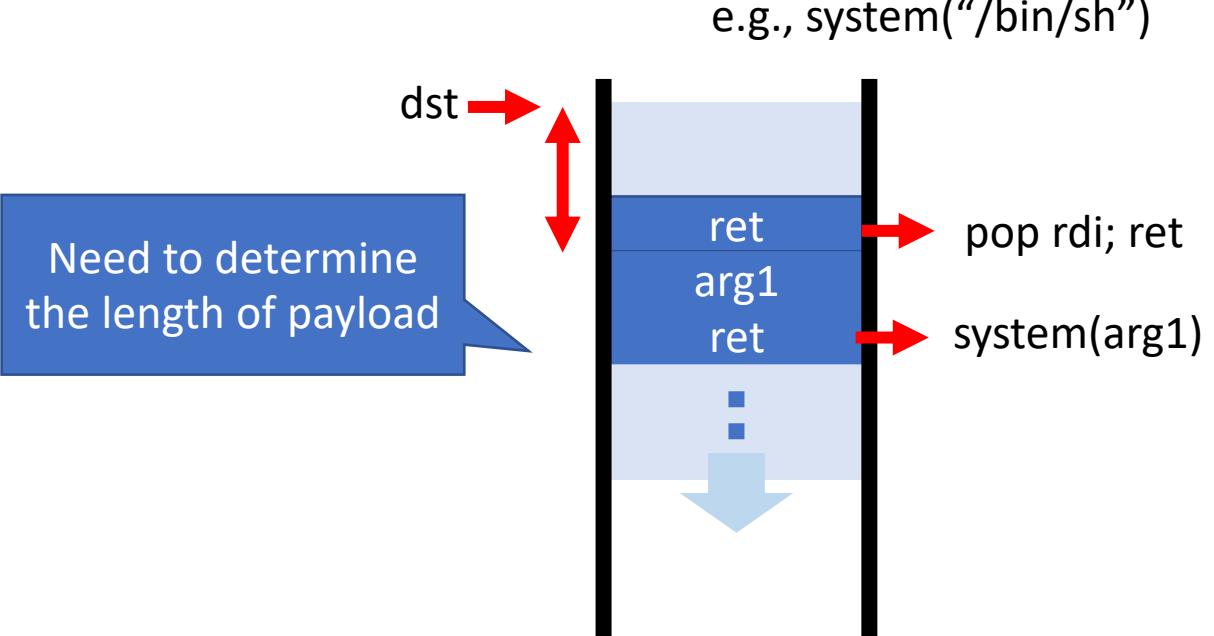
```
void vuln(char *input) {  
    char dst[0x100];  
    memcpy(dst, input, 0x200);  
}
```



# Typical Requirements for ROP

```
void vuln(char *input) {  
    char dst[0x100];  
    memcpy(dst, input, 0x200);  
}
```

Code (via reverse engineering)



# ROP Inside an Enclave

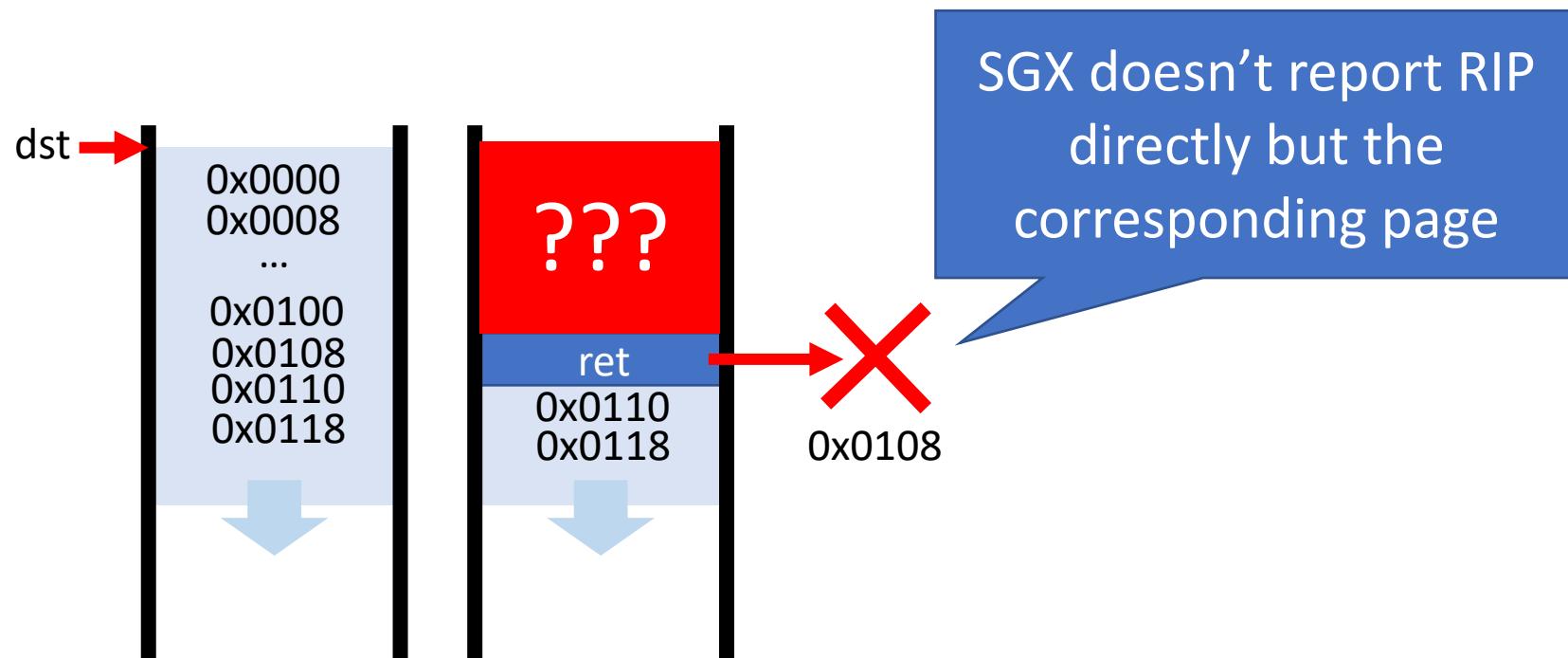
```
void vuln(char *input) {  
    char dst[ ??? ];  
    memcpy(dst, input, ??? );  
}
```

Code is not visible!  
(e.g., loaded in an encrypted form)

# ROP Inside an Enclave

```
void vuln(char *input) {  
    char dst[ ??? ];  
    memcpy(dst, input, ??? );  
}
```

Code is not visible!  
(e.g., loaded in an encrypted form)



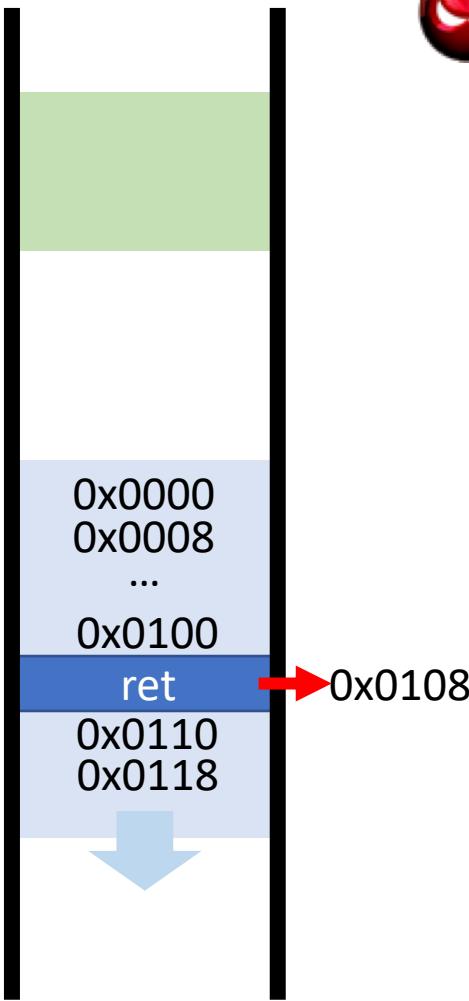
# ROP in Darkness: Dark ROP

- Step 1. Debunking the locations of pop gadgets
- Step 2. Locating ENCLU + pop rax (i.e., EEXIT)
- Step 3. Deciphering all pop gadgets
- Step 4. Locating memcpy()

# Threat Model for DarkROP

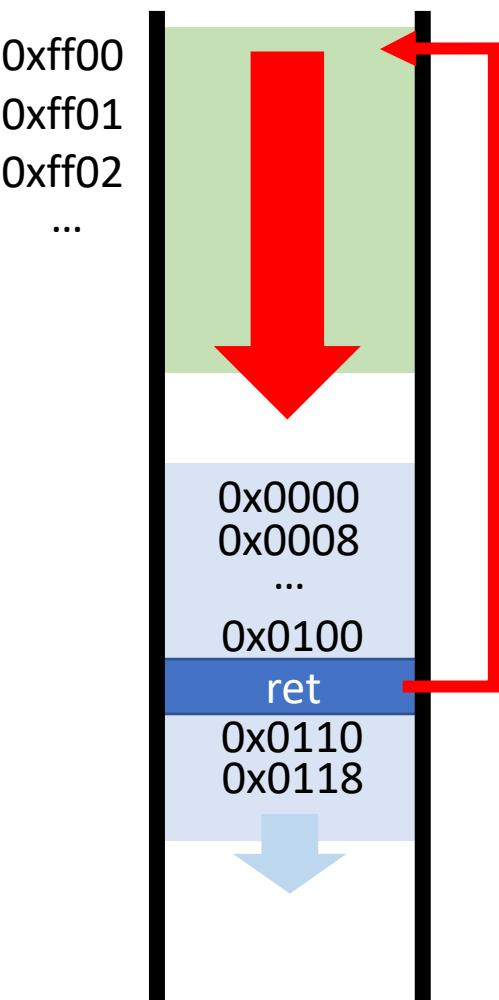
- Know existence of a buffer overflow (i.e., crash)
- Crashing the enclave arbitrarily times
- Built with standard libraries (e.g., SGX SDK)
- **Distributed in an encrypted form (like VC3)**

# Step 1. Looking for pop Gadgets



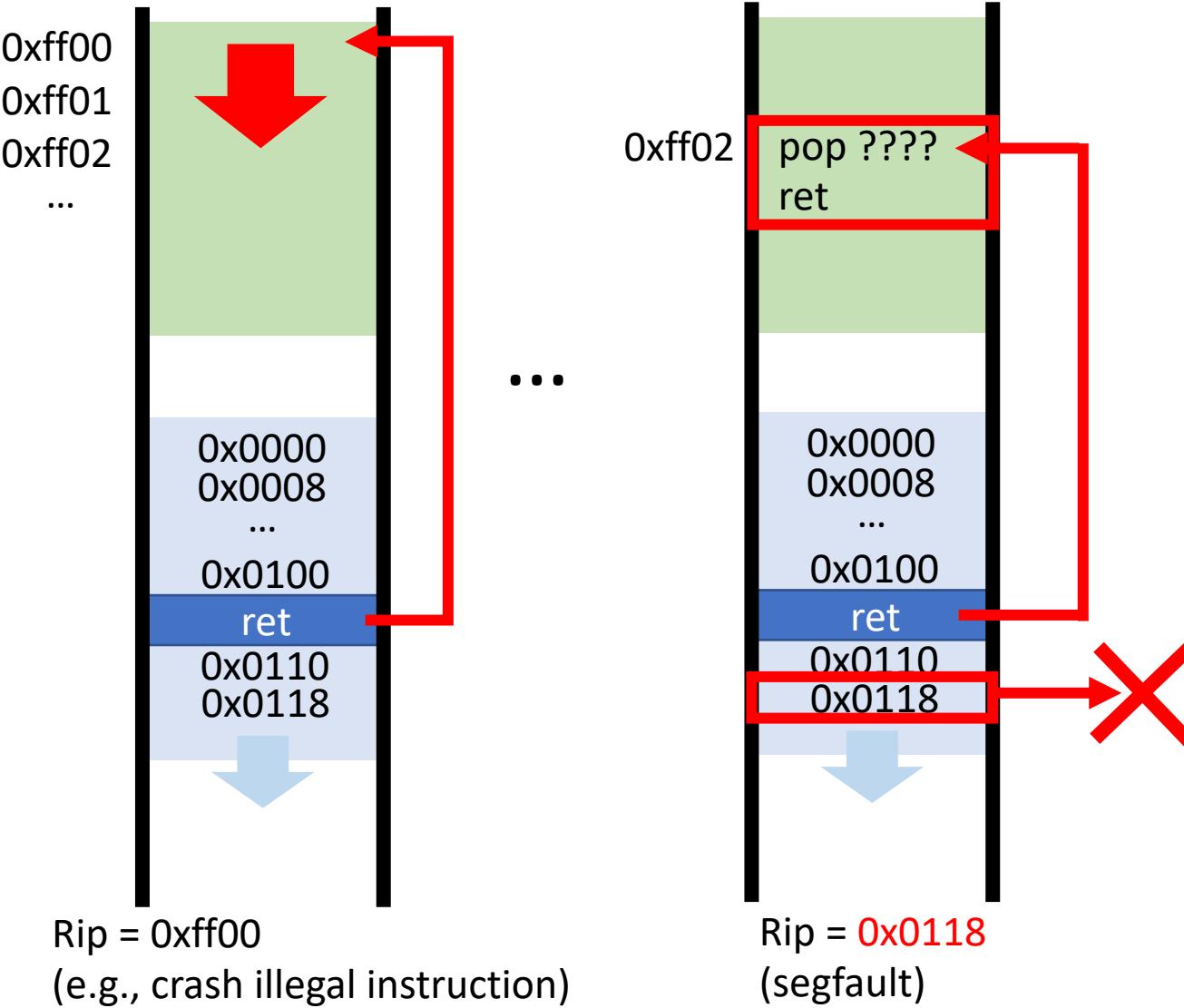
You have a full control over the layout of the enclave

# Step 1. Looking for pop Gadgets

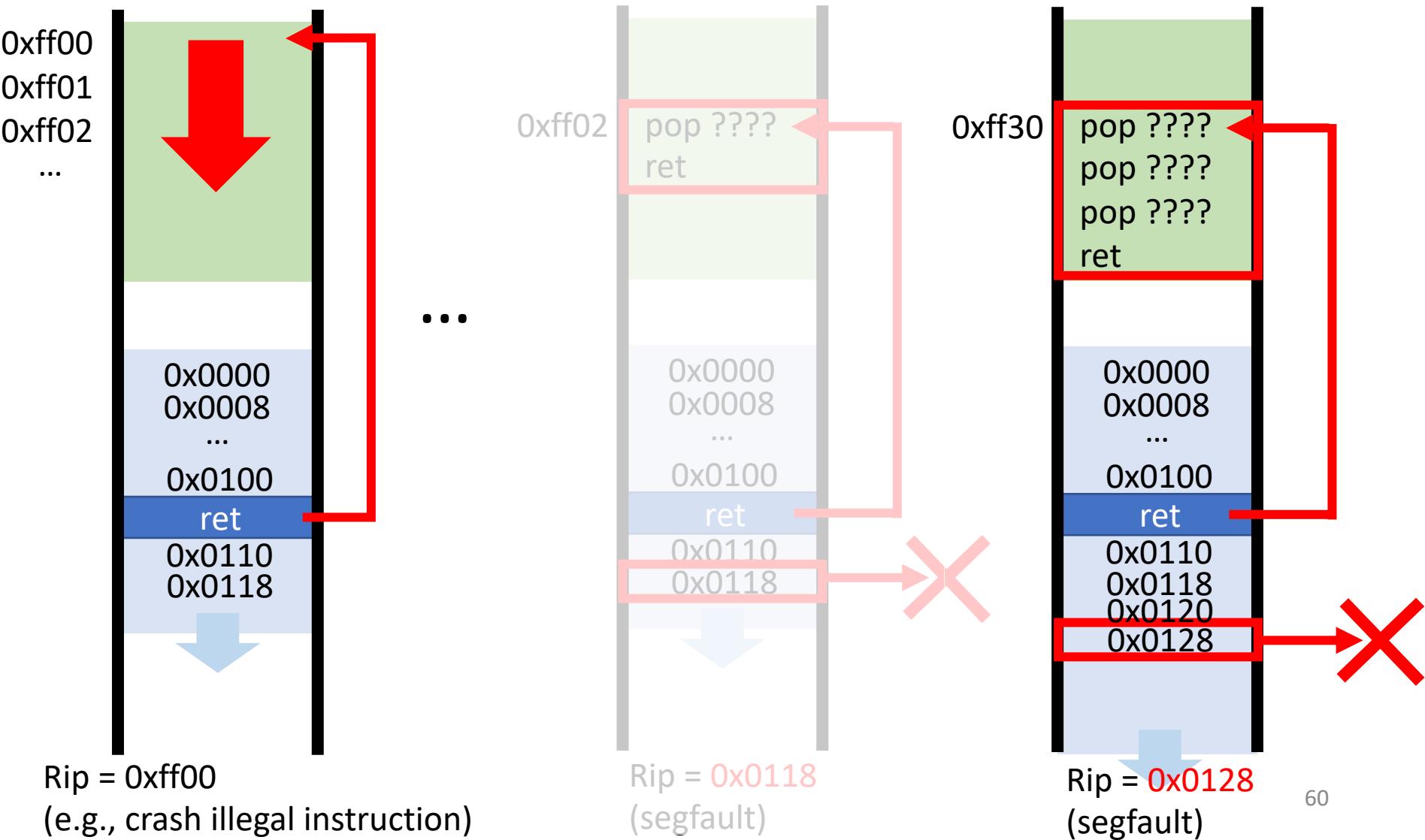


Rip = 0xff00  
(e.g., crash illegal instruction)

# Step 1. Looking for pop Gadgets



# Step 1. Looking for pop Gadgets



# Step 1. Looking for pop Gadgets

Catalog of pop gadgets  
(unknown args)

0xff02 → pop ?;ret  
0xff30 → pop ?;pop ?;pop ?;ret  
...

0xff02

pop ????  
ret

0x0000  
0x0008  
...  
0x0100

ret

0x0110  
0x0118

Rip = 0x0118  
(segfault)

0xff30

pop ????  
pop ????  
pop ????  
ret

0x0000  
0x0008  
...  
0x0100

ret

0x0110  
0x0118  
0x0120  
0x0128

Rip = 0x0128  
(segfault)



# Step 2. Looking for ENCLU

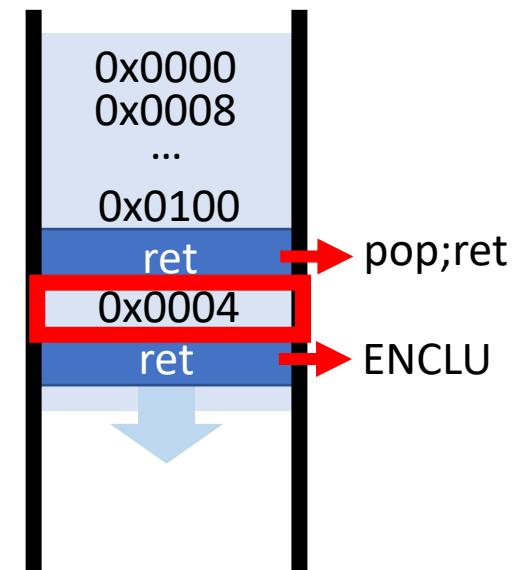
- ENCLU: an inst. dispatches to various leaf functions
  - $\text{rax} = 0$ : EREPORT
  - $\text{rax} = 1$ : EGETKEY
  - ...
  - $\text{rax} = 4$ : EEXIT

# Step 2. Looking for ENCLU

- ENCLU: an inst. dispatches to various leaf functions
  - $\text{rax} = 0$ : EREPORT
  - $\text{rax} = 1$ : EGETKEY
  - ...
  - $\text{rax} = 4$ : EEXIT

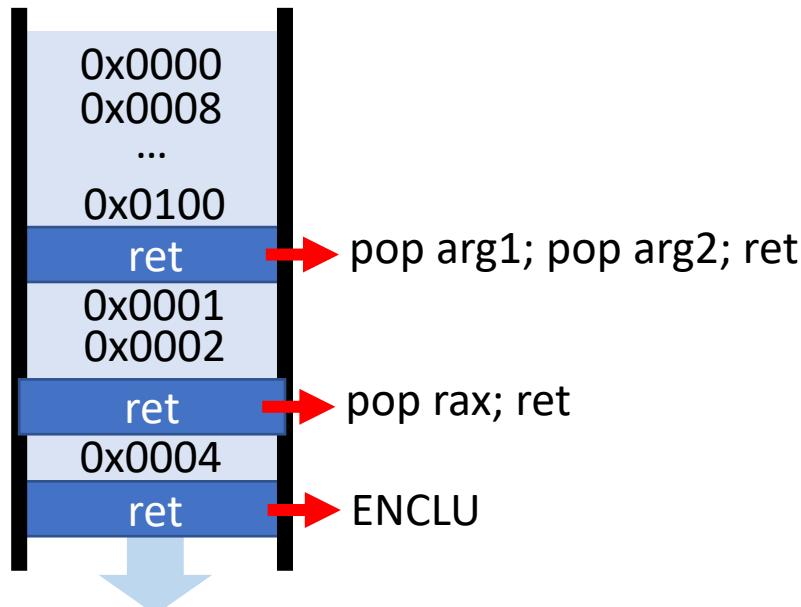
→ Scan code for each “pop????;ret”

→ If gracefully exit,  $\text{rip} = \text{ENCLU}$



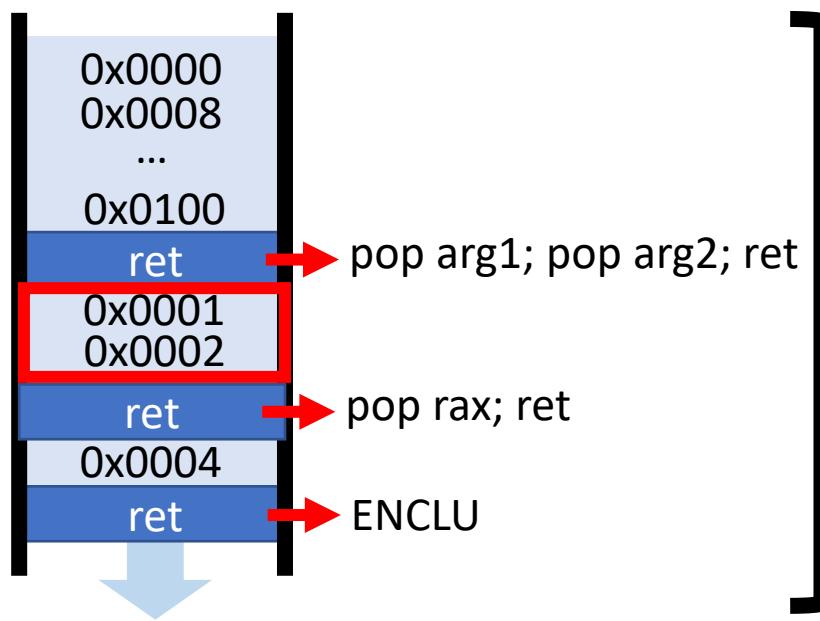
# Step 3. Deciphering pop Gadgets

- EEXIT (ENCLU & rax=4) left a register file uncleaned
  - Scan code for all pop gadgets
  - Check arguments



# Step 3. Deciphering pop Gadgets

- EEXIT (ENCLU & rax=4) left a register file uncleared
  - Scan code for all pop gadgets
  - Check arguments



Deciphering  
pop? pop? gadget

arg1 = 0x0001  
arg2 = 0x0002

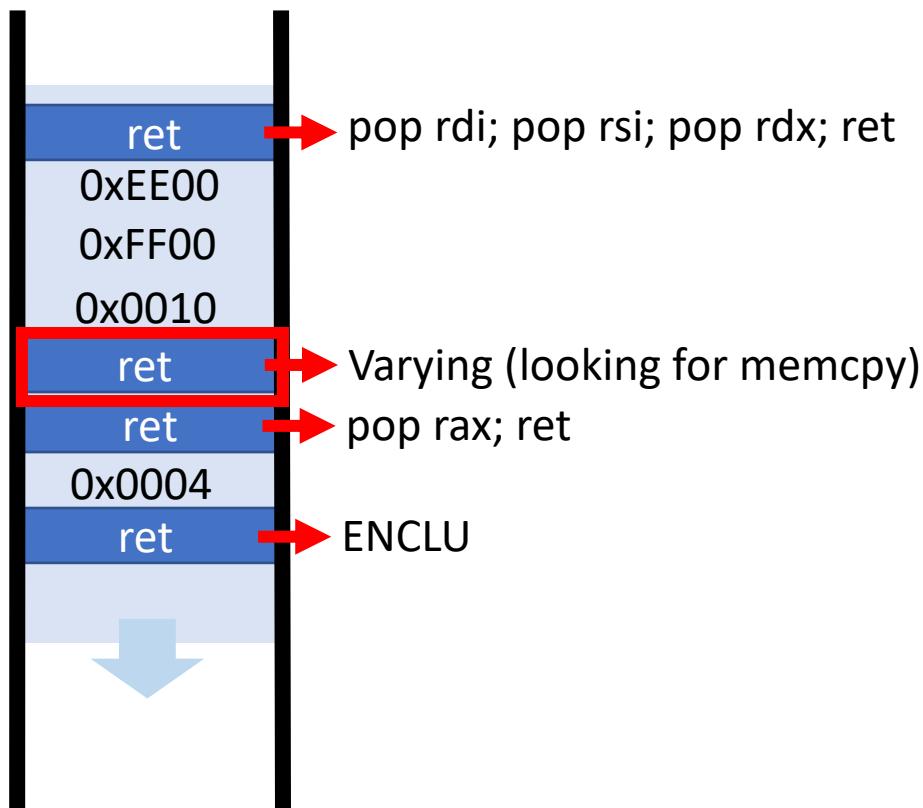
+  
Register file

=  
rax = 0x0004  
rsi = 0x0001  
rdi = 0x0002  
...

pop rsi  
pop rdi  
ret

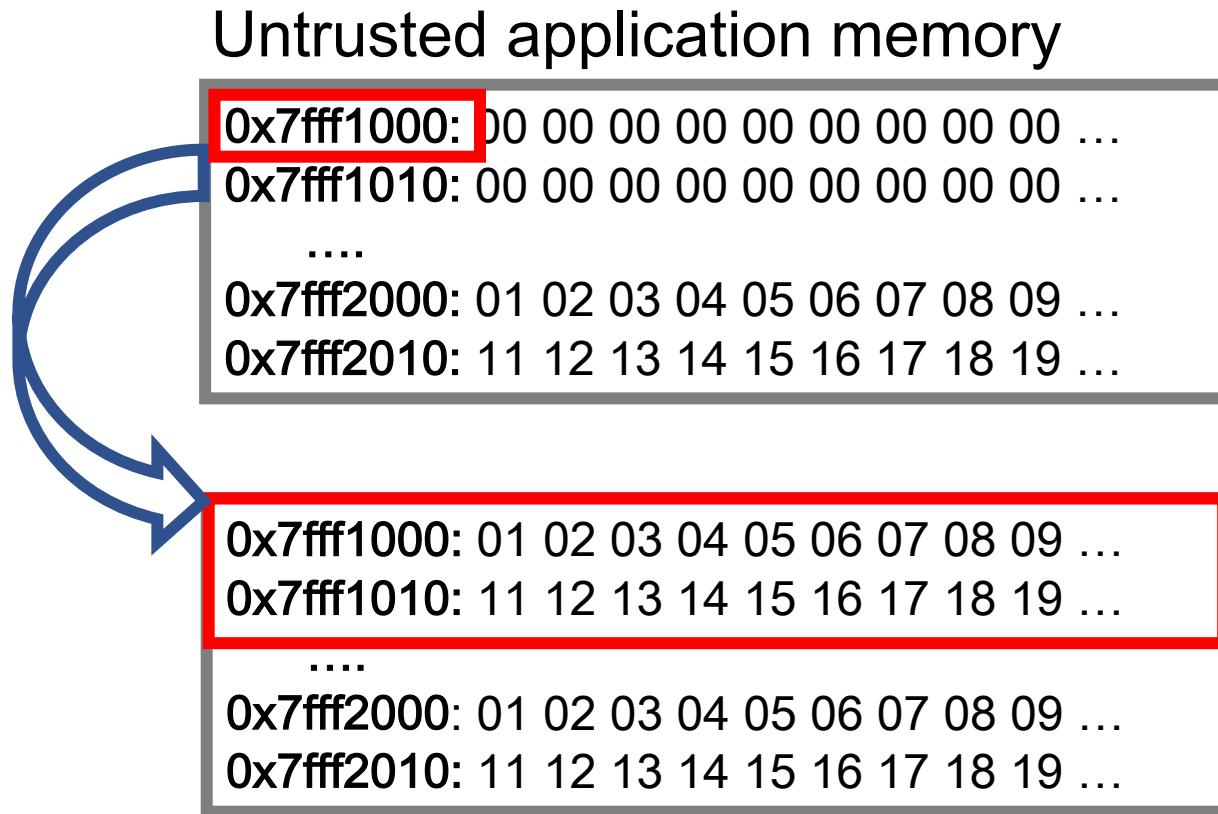
# Step 4. Looking for memcpy()

- Identifying `memcpy(dst*, valid, 0x10)`



# Step 4. Looking for memcpy()

- E.g., invoking memcpy(0x7ff1000, any valid, 0x10)



# Gadgets Everywhere (e.g., SDK)

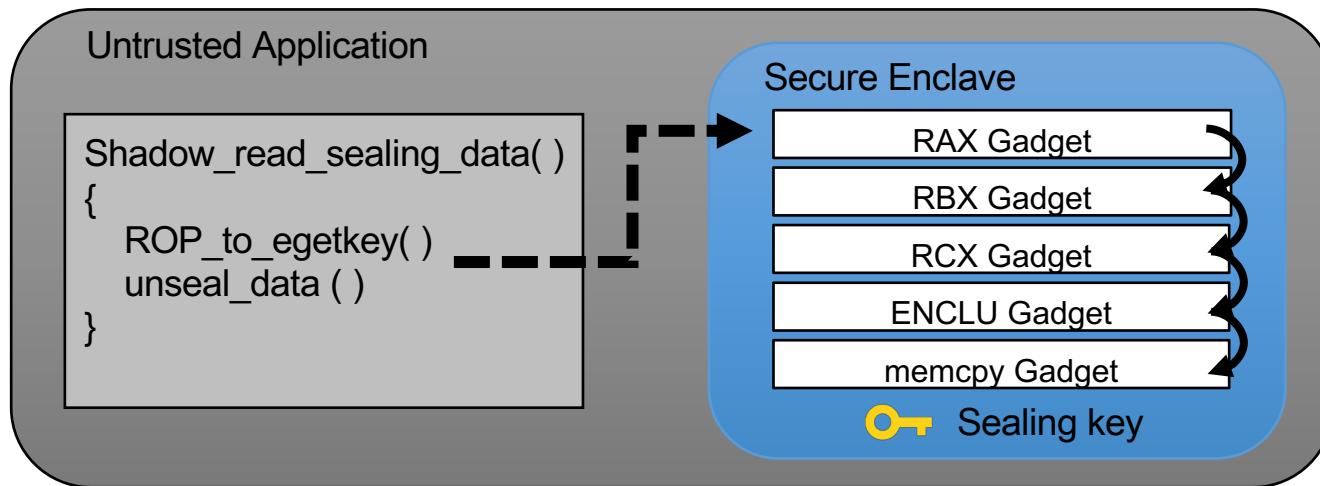
Gadget	From
<i>ENCLU Gadget</i> do_ereport: ENCLU  pop rdx pop rcx pop rbx ret	libsgx_trts.a
sgx_register_exception_handler: mov <u>rax</u> , <u>rbx</u> pop rbx pop rbp pop r12 ret	libsgx_trts.a
<i>Memcpy Gadget</i> memcpy:	libsgx_tstdc.a
sgx_sgx_ra_proc_msg2_trusted: pop rsi pop r15 ret pop rdi ret	libsgx_tkey_exchange.a

Gadget	From
<i>GPR Modification Gadget</i> __intel_cpu_indicator_init: pop r15 pop r14 pop r13 pop r12 pop r9 pop r8 pop rbp pop rsi pop rdi pop rbx pop rcx pop rdx pop rax ret	sgx_tstdc.lib
<i>ENCLU Gadget</i> do_ereport: enclu pop rax ret	sgx_trts.lib

# DEMO: PoC Dark ROP

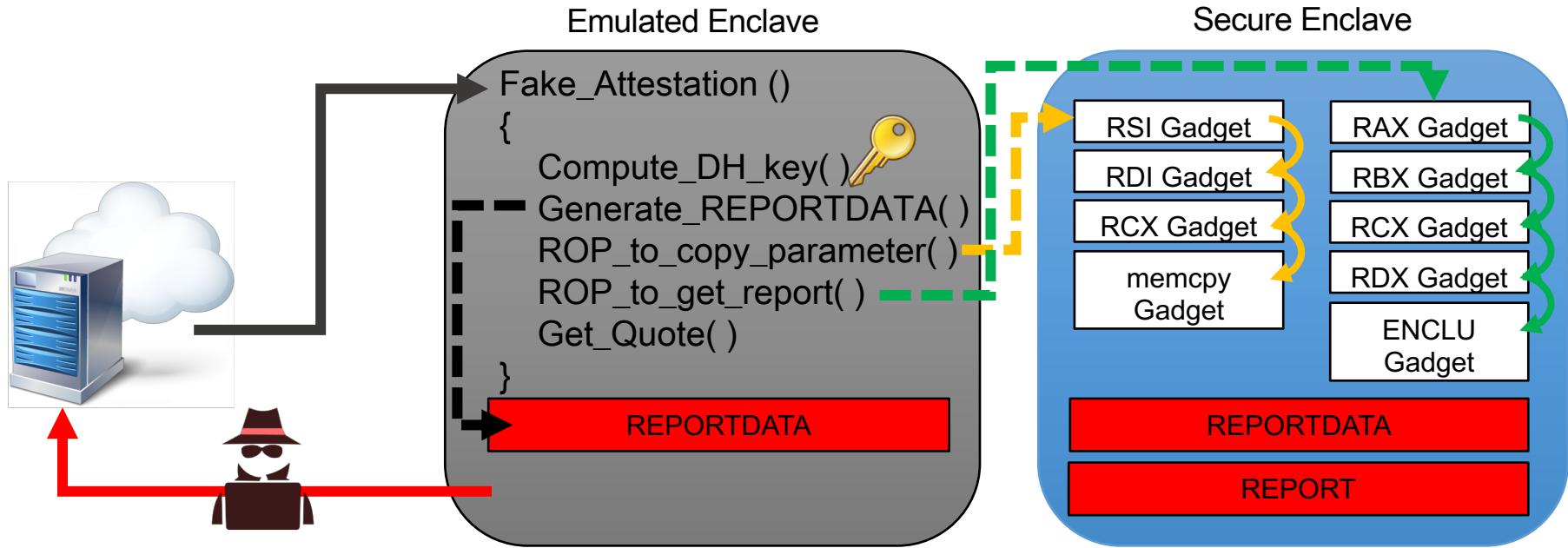
Step1. Looking for pop gadgets

# Case Study 1: Unsealing Data



- Unsealing and leaking confidential data
  - i.e., EGETKEY retrieves the hardware key bound to specific enclave

# Case Study 2: Hijacking Remote Attestation



- Breaking the Integrity guarantees of SGX
  - MiTM between secure enclave and attestation server
  - Masquerading to deceive remote attestation service

# Defense: SGXBounds

- Addressing *spatial memory* problems (bound chk)

## SGXBOUNDS: Memory Safety for Shielded Execution

Dmitrii Kuvaiskii<sup>†</sup> Oleksii Oleksenko<sup>†</sup> Sergei Arnautov<sup>†</sup> Bohdan Trach<sup>†</sup>  
Pramod Bhatotia<sup>\*</sup> Pascal Felber<sup>†</sup> Christof Fetzer<sup>†</sup>

<sup>†</sup>TU Dresden    <sup>\*</sup>The University of Edinburgh    <sup>‡</sup>University of Neuchâtel

### Abstract

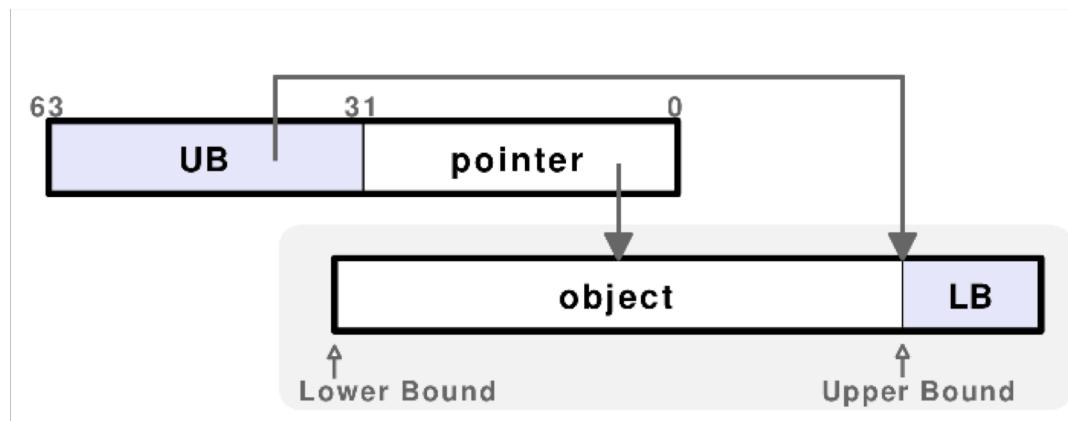
Shielded execution based on Intel SGX provides strong security guarantees for legacy applications running on untrusted platforms. However, memory safety attacks such as Heartbleed can render the confidentiality and integrity properties of shielded execution completely ineffective. To prevent these attacks, the state-of-the-art memory-safety approaches can be used in the context of shielded execution.

Shielded execution aims to protect confidentiality and integrity of applications when executed in an untrusted environment [19, 22]. The main idea is to isolate the application from the rest of the system (including privileged software), using only a narrow interface to communicate to the outside, potentially malicious world. Since this interface defines the security boundary, checks are performed to prevent the untrusted environment from in an attempt to leak conf

EuroSys'17

# Defense: SGXBounds

- Addressing *spatial memory* problems (bound chk)
- Key idea: an efficient tag representation thanks to smaller memory space!



# Defense: SGXBounds

```
1 int *s[N], *d[N]
2
3
4 for (i=0; i<M; i++):
5     si = s + i
6     di = d + i
7
8
9
10    val = load si
11
12
13    store val, di
14
15
```

```
int *s[N], *d[N]
s = specify_bounds(s, s + N)
d = specify_bounds(d, d + N)
for (i=0; i<M; i++):
    si = s + i
    di = d + i
    sp, sLB, sUB = extract(si)
    if bounds_violated(sp, sLB, sUB):
        crash(si)
    val = load si
    dp, dLB, dUB = extract(di)
    if bounds_violated(dp, dLB, dUB):
        crash(di)
    store val, di
```

# Done w/ Memory Safety on SGX?

- SGXBounds is a temporary solution
  - No temporal safety (i.e., UAF)
  - More address space in the future (e.g., large pages)
- What about traditional mitigations (required)?

# Traditional Attack Vectors

- Cache-based side channel
  - e.g., inferring a private key
- Memory safety
  - e.g., control flow hijacking
- Weak mitigation techniques
  - e.g., breaking ALSR
- Uninitialized padding in EDL
  - e.g., leaking security sensitive information

# SGX Mitigation Checklist

- Popular mitigation schemes:

Stack Canary

RELRO

DEP/NX

ASLR/PIE

# SGX Mitigation Checklist

- Popular mitigation schemes:

- Stack Canary

- RELRO

- DEP/NX

- ASLR/PIE

```
ecall_pointer_user_check():
```

```
push    %rbp
mov     %rsp,%rbp
sub    $0x90,%rsp
mov     %rdi,-0x88(%rbp)
mov     %rcx,-0x90(%rbp)
mov     %fs:0x28,%rax
mov     %rax,-0x8(%rbp)
```

*prologue*

```
xor    %fs:0x28,%rsi
je    4010 <ecall_pointer_user_check+0x118>
callq  8fb0 <__stack_chk_fail>
leaveq
retq
```

*epilogue*

# SGX Mitigation Checklist

- Popular mitigation schemes:

 Stack Canary

 RELRO

 DEP/NX

 ASLR/PIE

# Defense: ASLR/SW-DEP inside SGX

- Popular mitigation schemes:

- ✓ Stack Canary
- ✓ RELRO
- ✗ DEP/NX
- ✗ ASLR/PIE

## SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs

Jacback Sen<sup>\*†</sup>, Byounyoung Lee<sup>‡§</sup>, Seongmin Kim<sup>\*</sup>, Ming-Wei Shih<sup>‡</sup>,

Insik Shin<sup>\*</sup>, Dongsu Han<sup>\*</sup>, Tae soo Kim<sup>‡</sup>

<sup>\*</sup>KAIST

<sup>†</sup>Purdue University

<sup>‡</sup>Georgia Institute of Technology

{jacback, dallas1004, ishin, dongsu\_han}@kaist.ac.kr, blee@purdue.edu, {mingwei.shih, taecon}@gatech.edu

**Abstract**—Traditional execution environments deploy Address Space Layout Randomization (ASLR) to defend against memory corruption attacks. However, Intel Software Guard Extension (SGX), a new trusted execution environment designed to serve security-critical applications on the cloud, lacks such an effective, well-studied feature. In fact, we find that applying ASLR to SGX programs raises non-trivial issues beyond simple engineering for a number of reasons: 1) SGX is designed to defeat a stronger adversary than the traditional model, which requires the address space layout to be hidden from the kernel; 2) the limited memory uses in SGX programs present a new challenge in providing a sufficient degree of entropy; 3) remote attestation conflicts with the dynamic relocation required for ASLR; and 4) the SGX specification relies on known and fixed addresses for key data structures that cannot be randomized.

system and hypervisor. It also offers hardware-based measurement, attestation, and enclave page access control to verify the integrity of its application code.

Unfortunately, we observe that two properties, namely, confidentiality and integrity, do not guarantee the actual security of SGX programs, especially when traditional memory corruption vulnerabilities, such as buffer overflow, exist inside SGX programs. Worse yet, many existing SGX-based systems tend to have a large code base written in assembly language as library in Haven [12], or even C/C++ [28, 29]. This is because unsafe programming is often required to interact with SGX in an assembly language.

NDSS'17

# Challenges for Mitigation Schemes

It is non-trivial when an attacker is the kernel:

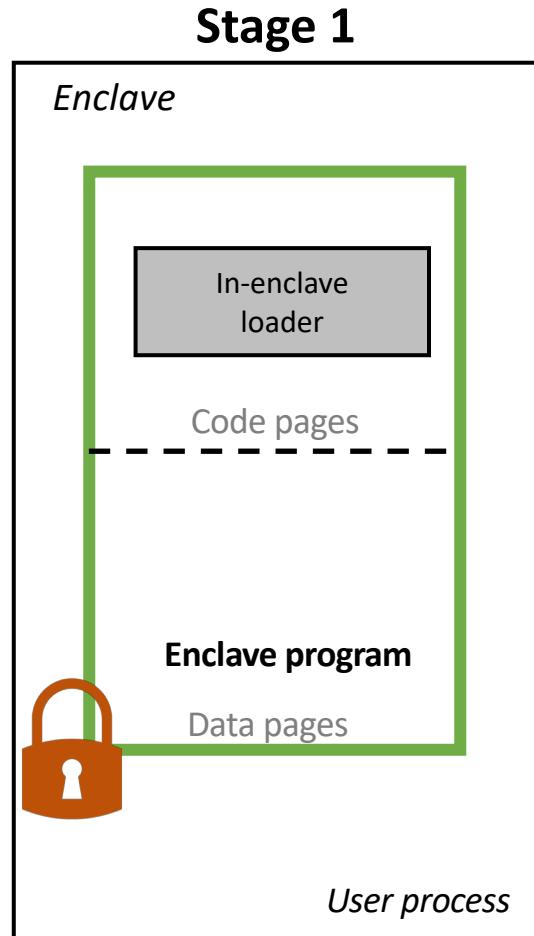
- Visible memory layout
- Small randomization entropy
- No runtime page permission change

# Challenges for Mitigation Schemes

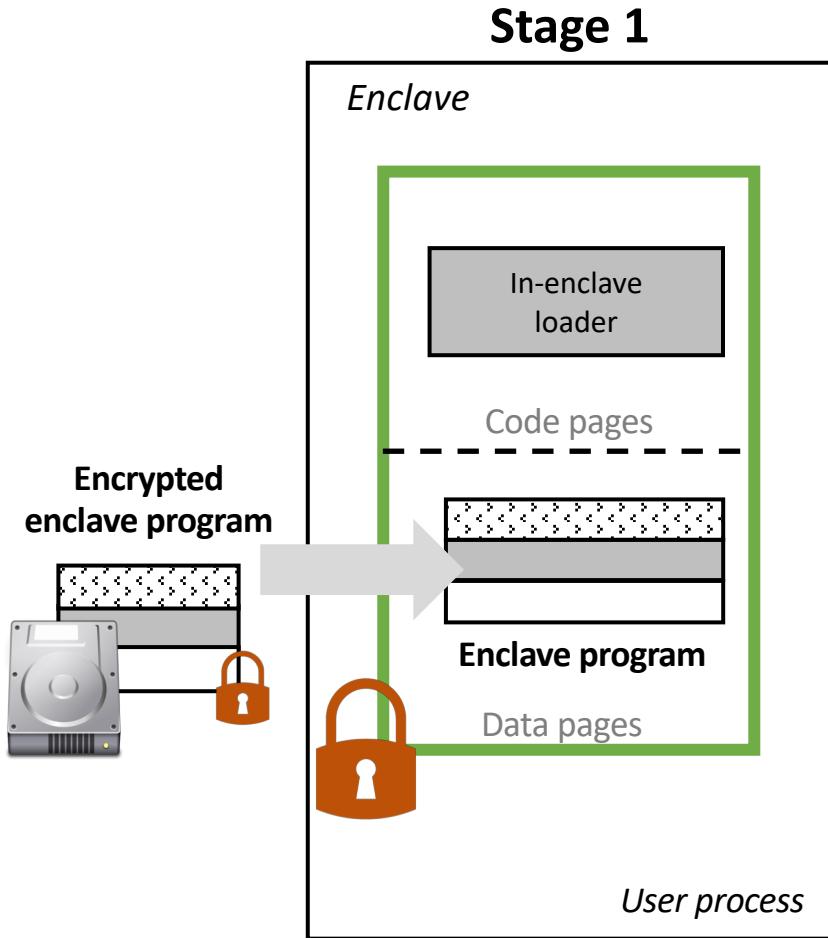
It is non-trivial when an attacker is the kernel:

- Visible memory layout
  - Secure in-enclave loading
- Small randomization entropy
  - Fine-grained ASLR
- No runtime page permission change
  - Soft-DEP/SFI

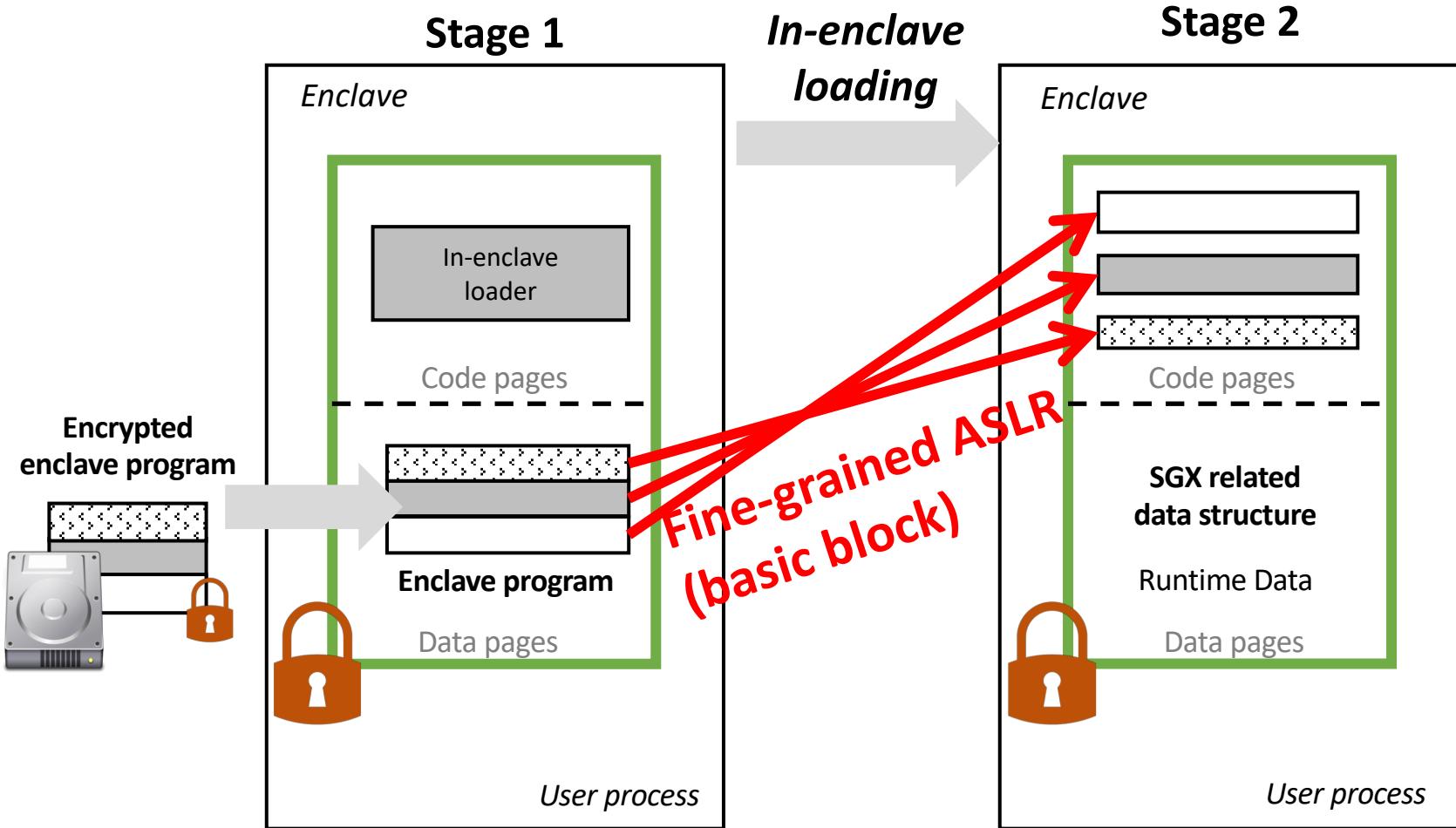
# SGX-Shield's Approach: In-enclave Loading



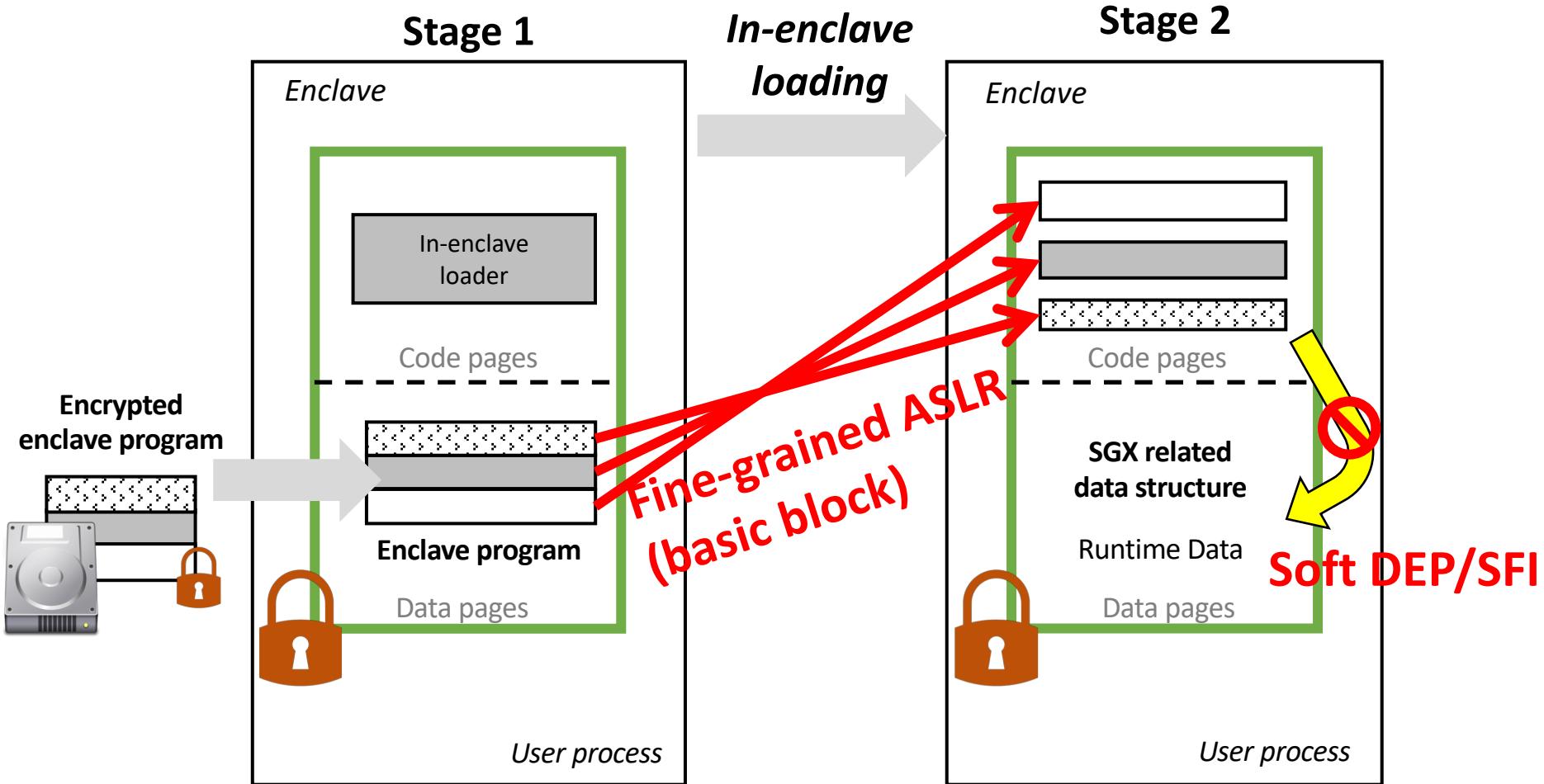
# SGX-Shield's Approach: In-enclave Loading



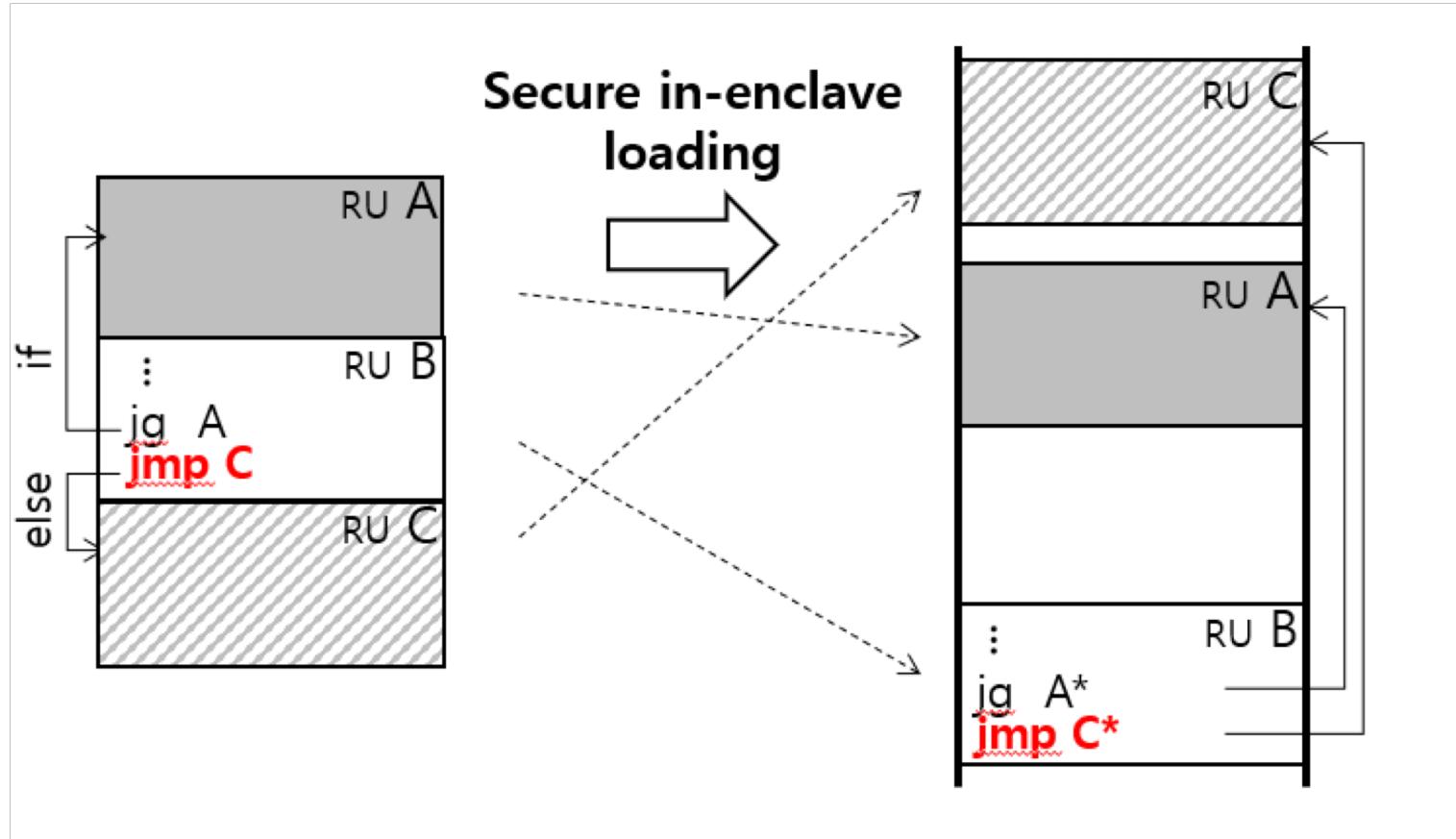
# SGX-Shield's Approach: In-enclave Loading



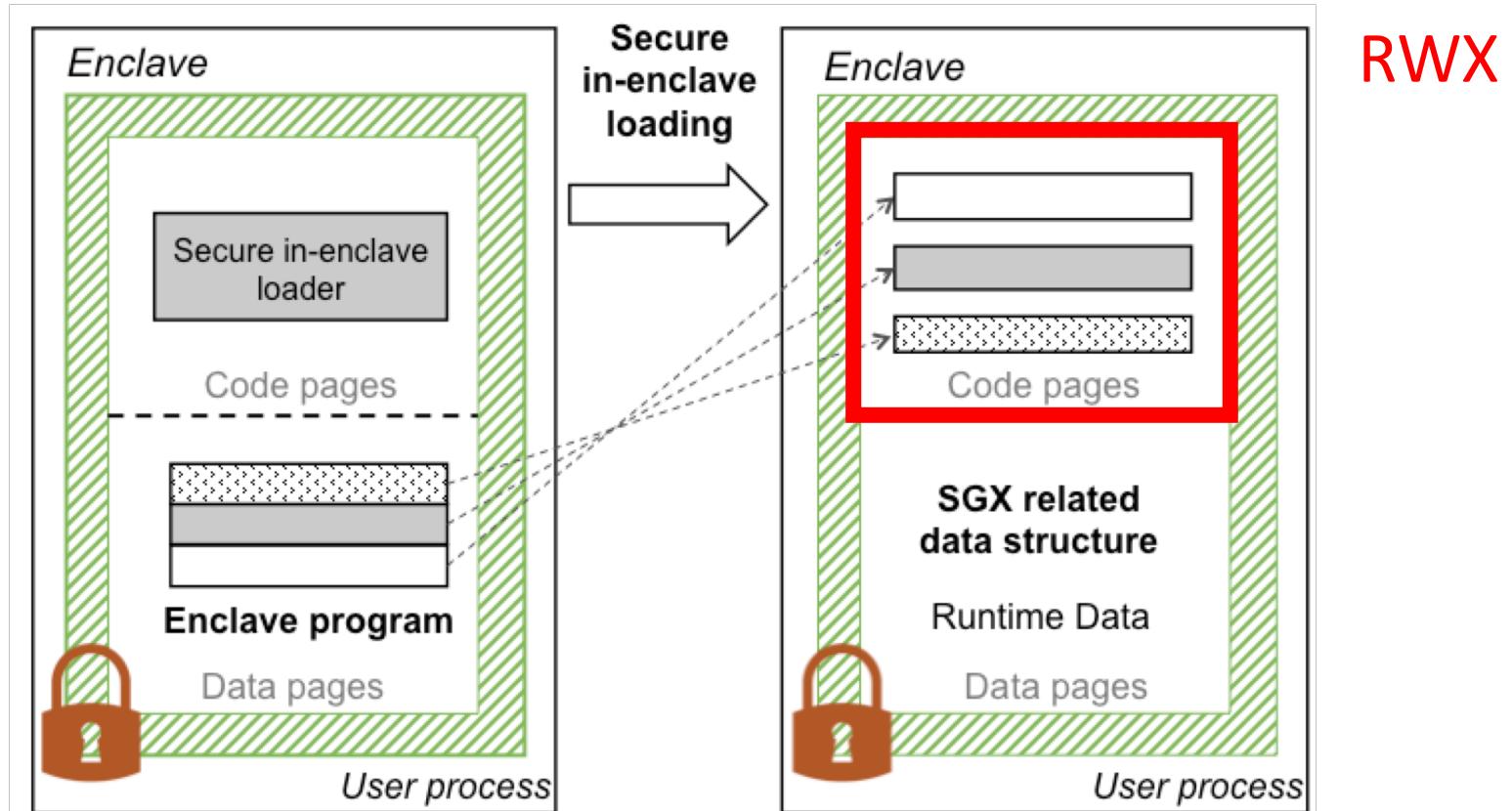
# SGX-Shield's Approach: In-enclave Loading



# SGX-Shield's Fine-grained ASLR



# No Runtime Permission Change

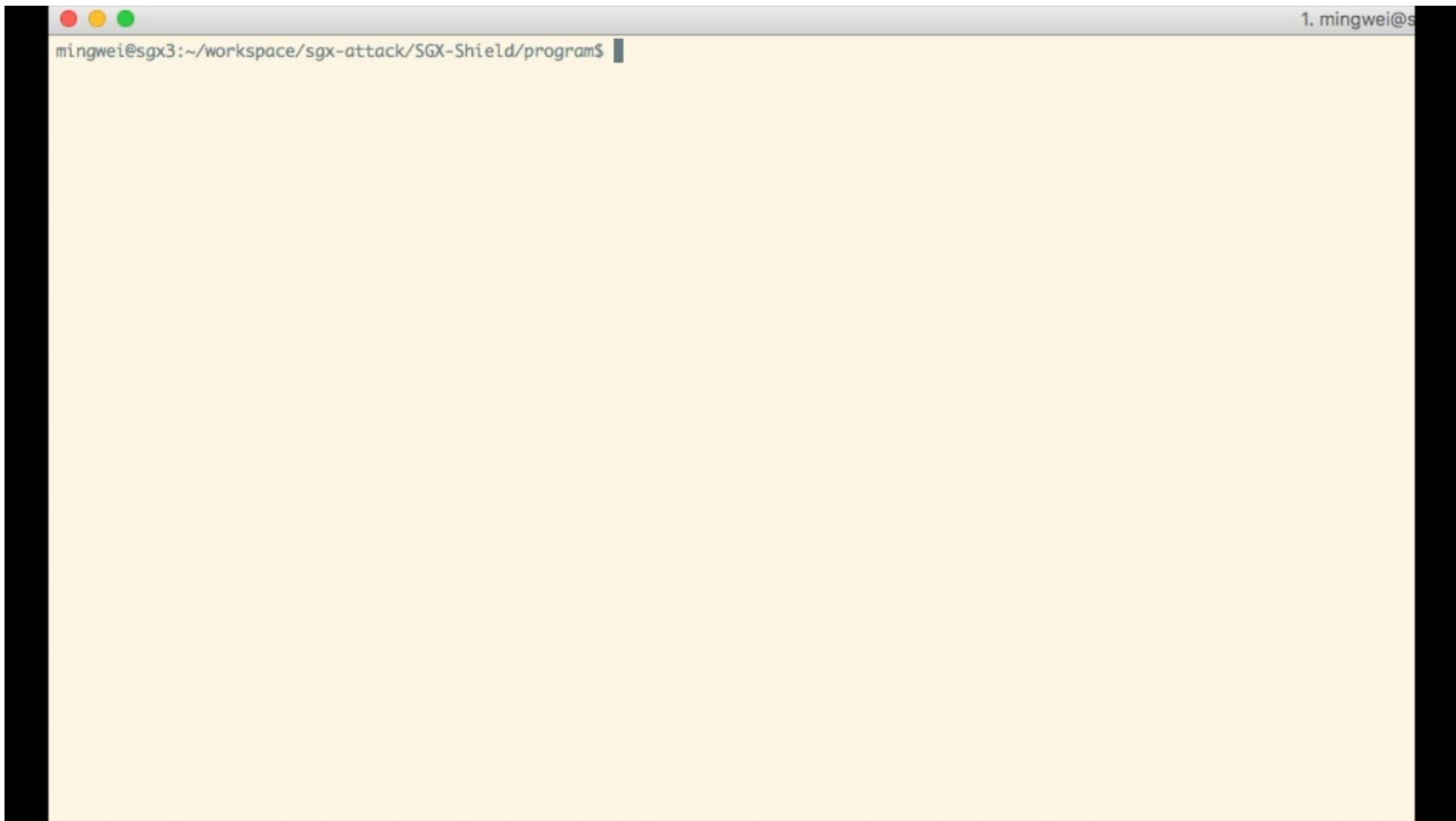


# SW-based Permission Enforcement (via SFI like Nacl)

		<b>Hardware-based permission</b>	<b>Software+Hardware permission</b>
<i>Out of enclave</i>	↑		
Code of loader		RWX	<i>No Permission</i>
Code		RWX	X
Data of loader		RW	<i>No Permission</i>
Data		RW	RW
<i>Out of enclave</i>	↓		

Virtual address space of an enclave

# DEMO: SGX-Shield



<https://github.com/sslab-gatech/SGX-Shield>

# SGX-Shield has Two Limitations

- 1) ASLR scheme is vulnerable against fine-grained side-channels (i.e., multifaceted)
- 2) No protections on backward edges and SDK libs

## Securing ASLR on SGX against Multifaceted Side-channel Attacks

Paper #233

**Abstract**—Intel Software Guard Extensions (SGX) allows security-sensitive applications to run in isolation, thus protecting their confidentiality and integrity. SGX protects applications from all other software on the platform, including the operating system. However, the trusted computing base of the application still includes the application code itself, and vulnerabilities in this code can have the same catastrophic consequences under SGX as they have elsewhere. Thus, it is desirable to deploy the known general defense schemes in SGX. In particular, address space layout randomization (ASLR) has been proposed as a general way to mitigate vulnerabilities in SGX code.

This paper investigates the potential security challenges of deploying ASLR in SGX-like environments which are subject to multiple side-channels. An SGX adversary who can observe the memory accesses of the code running under SGX at cache-line and/or page granularity may gain enough information to derandomize even fine-grained ASLR.

Our results include *multifaceted* side-channel attacks against SGX-Shield, the only published ASLR system for SGX. One of the attacks completely infers the ASLR code layout as the code

protect entire classes of vulnerabilities from being exploited highly desirable. Techniques such as control-flow integrity (CFI) [6], address space layout randomization (ASLR) [61], data execution prevention (DEP) [8], and stack canaries [20] are widely deployed in mass-market commercial systems and have kept countless bugs from becoming exploitable vulnerabilities.

Despite the undeniable benefits of these generic defenses, their deployment in TEEs is, at best, incomplete. The reason lies in the additional challenges posed by the TEE environment. Some of the defenses such as CFI or stack canaries are compiler-based and can be effortlessly deployed into TEEs. However, other defenses require system support that is not readily available in existing TEEs. In particular, ASLR, which is the focus of this paper, requires support from both the operating system and the SGX library to work correctly in a TEE. However, in the TEE environment, the SGX library is controlled by the attacker. SGX-Shield is the first ASLR system that loads a loader into the enclave, thus bypassing the control of the operating system.

Under  
submission

## The Guard's Dilemma: Efficient Code-Reuse Attacks Against Intel SGX

Andrea Biondo, Mauro Conti  
*University of Padua, Italy*

Lucas Davi  
*University of Duisburg-Essen, Germany*

Tommaso Frassetto, Ahmad-Reza Sadeghi  
*TU Darmstadt, Germany*

### Abstract

Intel Software Guard Extensions (SGX) isolate security-critical code inside a protected memory area called enclave. Previous research on SGX has demonstrated that memory corruption vulnerabilities within enclave code can be exploited to extract secret keys and bypass remote attestation. However, these attacks require kernel privileges, and rely on frequently probing enclave code which results in many enclave crashes. Further, they assume a constant, not randomized memory layout.

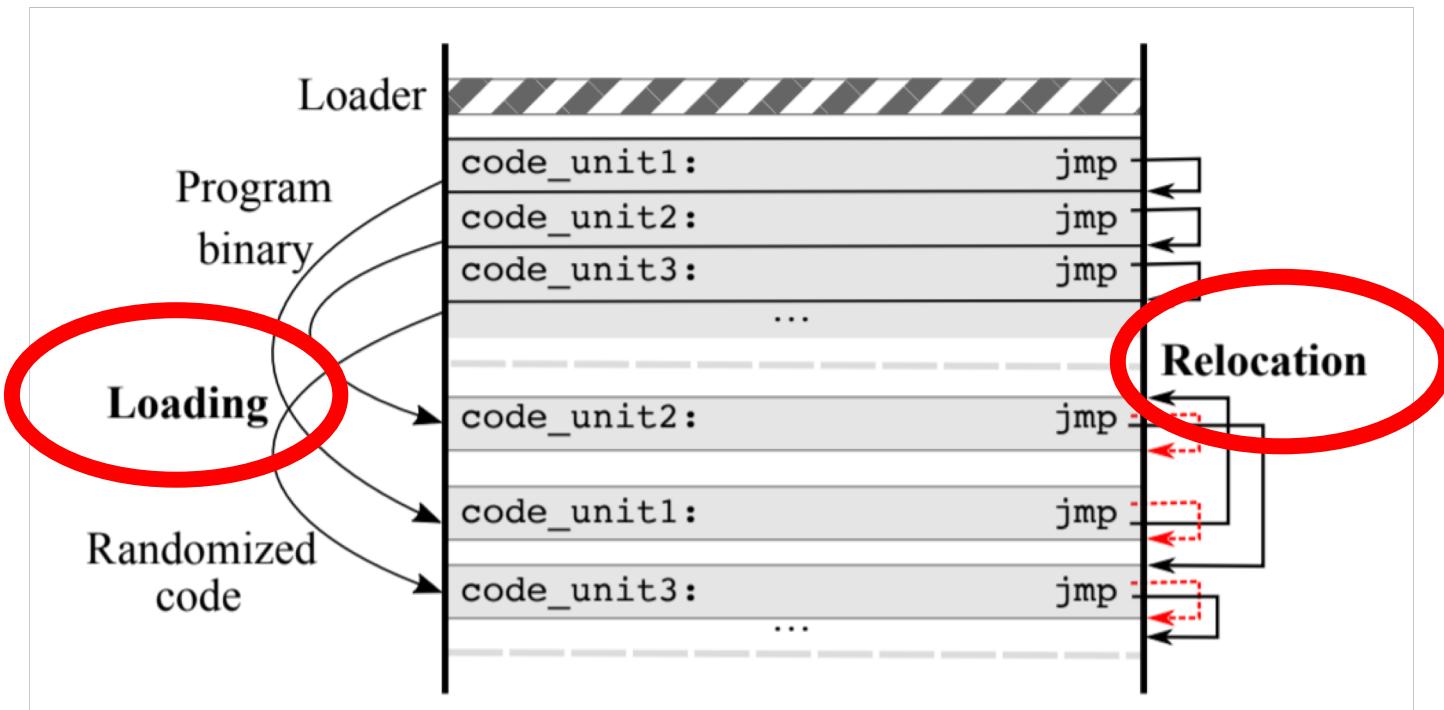
In this paper, we present novel exploitation techniques against SGX that do not require any enclave crashes and

using one of the pre-defined entry points. The enclave can subsequently perform sensitive computations, call pre-defined functions in the host, and return to the caller.

In the ideal scenario, the enclave code only includes minimal carefully-inspected code, which could be formally verified before being loaded. However, legacy software is not always built this way. Legacy SGX applications, for example, often contain legacy memory-corruption vulnerabilities that plague legacy software are also very likely to occur in those complex

SEC'18

# Breaking Fine-grained ASLR



# Attacking Randomization Process

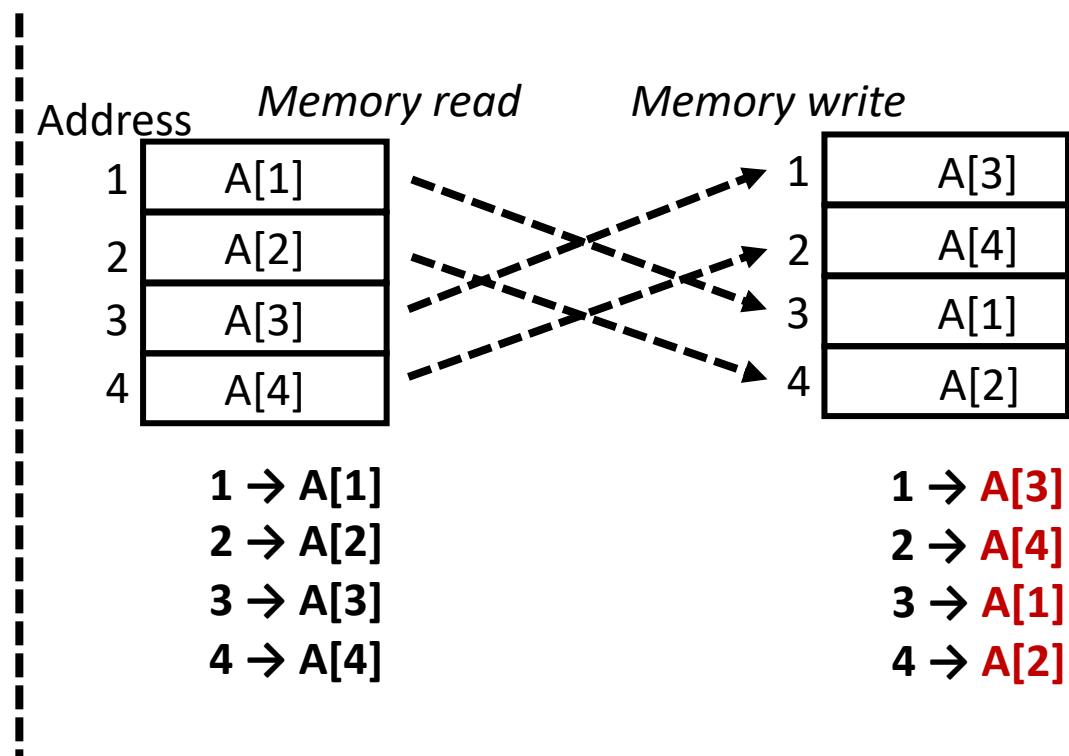
*Side-channel observations*

$1 \rightarrow 3$  ( $A[1]$ )

$2 \rightarrow 4$  ( $A[2]$ )

$3 \rightarrow 1$  ( $A[3]$ )

$4 \rightarrow 2$  ( $A[4]$ )

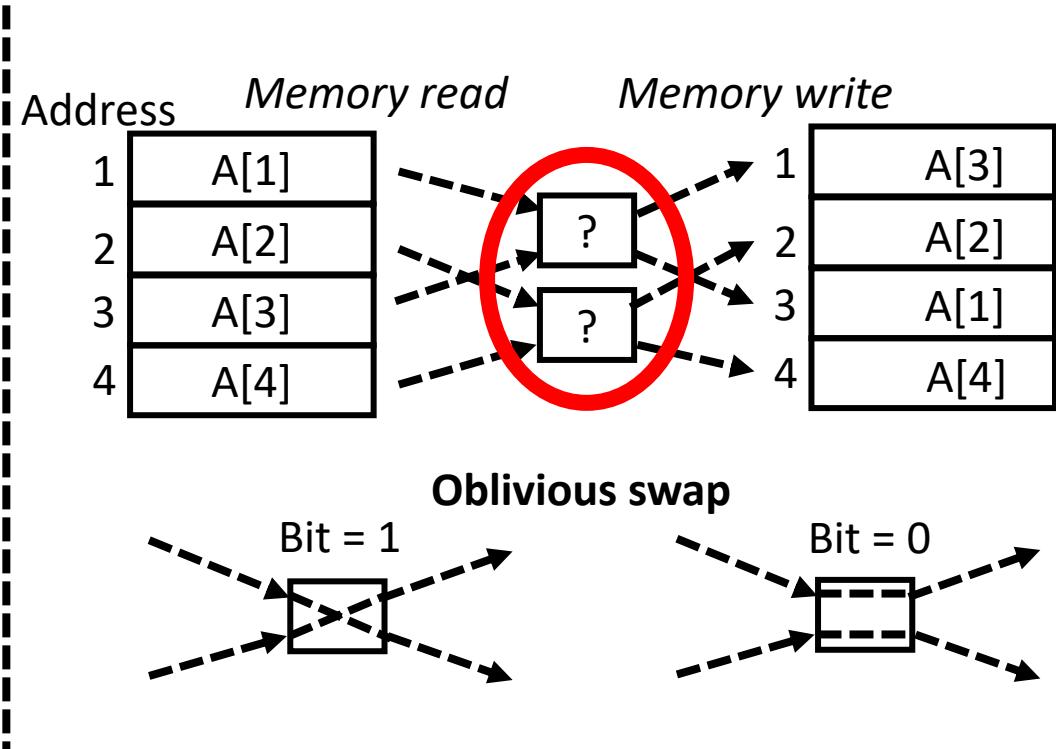


# SGX-Armor: Obfuscating Randomization via Oblivious Swap

*Side-channel observations*

$1 \rightarrow 3 \rightarrow 1 \rightarrow 3$  (Swap)  
 $2 \rightarrow 4 \rightarrow 2 \rightarrow 4$  (No swap)

**Swap or not  
reveals the same  
patterns**



# Oblivious Swap Primitive

```
1 # swap(a1,      a2,      b)
2 #           %rsi,    %rdi,    %edx
3 swap:
4   cmpl    $0x1, %edx
5   jne     no_swap
6   movq    (%rdi), %rax
7   movq    (%rsi), %rdx
8   movq    %rdx,  (%rdi)
9   movq    %rax,  (%rsi)
10
11 no_swap:
12         retq
```

```
1 # oswap(a1,      a2,      b)
2 #           %rsi,    %rdi,    %edx
3 oswap:
4   movq    (%rdi), %rax
5   movq    (%rsi), %rcx
6   cmpl    $0x1, %edx
7   cmovez  %rax,  %rdx
8   cmovez  %rcx,  %rax
9   cmovez  %rdx,  %rcx
10
11
12         retq
```

# SGX-Shield has Two Limitations

- 1) ALSR scheme is vulnerable against fine-grained side-channels (i.e., multifaceted)
  - 2) No protections on backward edges and SDK libs

# Securing ASLR on SGX against Multifaceted Side-channel Attacks

Paper #233

**Abstract**—Intel Software Guard Extensions (SGX) allows security-sensitive applications to run in isolation, thus protecting their confidentiality and integrity. SGX protects applications from all other software on the platform, including the operating system. However, the trusted computing base of the application still includes the application code itself, and vulnerabilities in this code can have the same catastrophic consequences under SGX as they have elsewhere. Thus, it is desirable to deploy the known general defense schemes in SGX. In particular, address space layout randomization (ASLR) has been proposed as a general way to mitigate vulnerabilities in SGX code.

protect entire classes of vulnerabilities from being exploited highly desirable. Techniques such as control-flow integrity (CFI) [6], address space layout randomization (ASLR) [61], data execution prevention (DEP) [8], and stack canaries [20] are widely deployed in mass-market commercial systems and have kept countless bugs from becoming exploitable vulnerabilities.

Despite the undeniable benefits of these generic defenses, their deployment in TEEs is, at best, incomplete. The reason lies in the additional challenges posed by the TEE environment. Some of the defenses such as CFI or stack canaries are compiler-based and can be effortlessly deployed into TEEs. However, other defenses require system support that is not readily available in existing TEEs. In particular, ASLR, which is the focus of this paper, is not supported by most modern operating systems.

However, in the TCG-based SGX, the user-space loader can inject the enclave's memory layout into the SGX-SRAM, thus bypassing the ASLR protection. This is done by injecting the memory layout of the enclave into the SGX-SRAM via the SGX-PL, which is controlled by the SGX-SRAM loader into the enclave. This allows the user-space loader to control the memory layout of the enclave, thus bypassing the ASLR protection.

Under  
submission

## The Guard's Dilemma: Efficient Code-Reuse Attacks Against Intel SGX

Andrea Biondo, Mauro Conti  
*University of Padua, Italy*

Lucas Davi  
*University of Duisburg-Essen, Germany*

Tommaso Frassetto, Ahmad-Reza Sadeghi  
*TU Darmstadt, Germany*

## Abstract

Intel Software Guard Extensions (SGX) isolate security-critical code inside a protected memory area called an enclave. Previous research on SGX has demonstrated that memory corruption vulnerabilities within enclaves' code can be exploited to extract secret keys and bypass remote attestation. However, these attacks require kernel privileges, and rely on frequently probing enclave code, which results in many enclave crashes. Further, they assume a constant, not randomized memory layout.

In this paper, we present novel exploitation techniques against SGX that do not require any enclave crashes and

using one of the pre-defined entry points. The enclave can subsequently perform sensitive computations, call pre-defined functions in the host, and return to the caller.

SEC'18

SEC'18

*memory-corruption vulnerabilities* that plague legacy software are also very likely to occur in those complex

Andrea Biondo, Mauro Conti

*University of Padua, Italy*

Lucas Davi

*University of Duisburg-Essen, Germany*

Tommaso Frassetto, Ahmad-Reza Sadeghi  
*TU Darmstadt, Germany*

#### Abstract

Intel Software Guard Extensions (SGX) isolate security-critical code inside a protected memory area called enclave. Previous research on SGX has demonstrated that memory corruption vulnerabilities within enclave code can be exploited to extract secret keys via remote attestation. However, these attacks require privileges, and rely on frequently problematic behaviors which results in many enclave crashes. Moreover, they assume a constant, not randomized memory layout.

In this paper, we present novel exploitation techniques against SGX that do not require any enclave

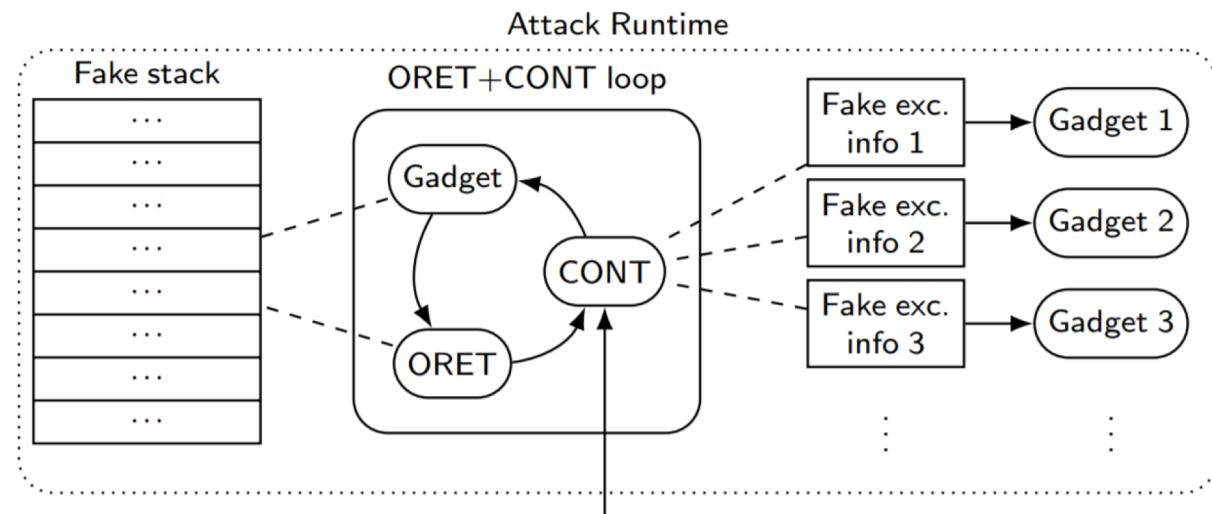
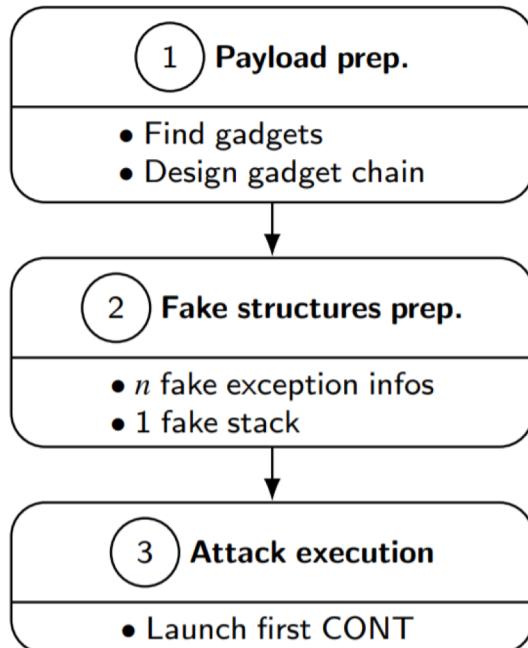
using one of the pre-defined entry points. The enclave can subsequently perform sensitive computations, call pre-defined functions in the host, and return to the caller.

In the ideal scenario, the enclave code only includes minimal carefully-inspected code, which could be extremely easier to be free of vulnerabilities. However,

SEC'18

# Another ROP

- Similar to Signal Oriented Programming
- SGX has ORET/CONT gadgets in SDK



# Traditional Attack Vectors

- Cache-based side channel
  - e.g., inferring a private key
- Memory safety
  - e.g., control flow hijacking
- Weak mitigation techniques
  - e.g., breaking ALSR
- Uninitialized padding in EDL
  - e.g., leaking security sensitive information

# Uninitialized Padding Problem

```
struct usbdevfs_connectinfo {  
    unsigned int devnum;  
    unsigned char slow;  
};
```

# Uninitialized Padding Problem

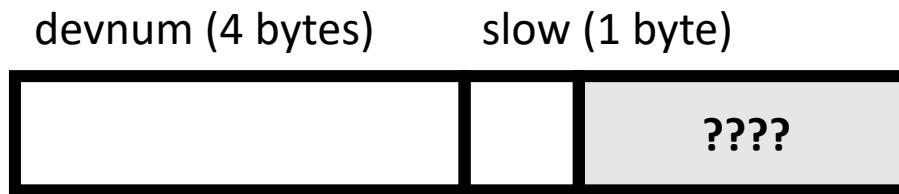
```
struct usbdevfs_connectinfo {  
    unsigned int devnum;  
    unsigned char slow;  
};
```

---

```
struct usbdevfs_connectinfo {  
    .devnum = 1,  
    .slow = 0,  
};
```

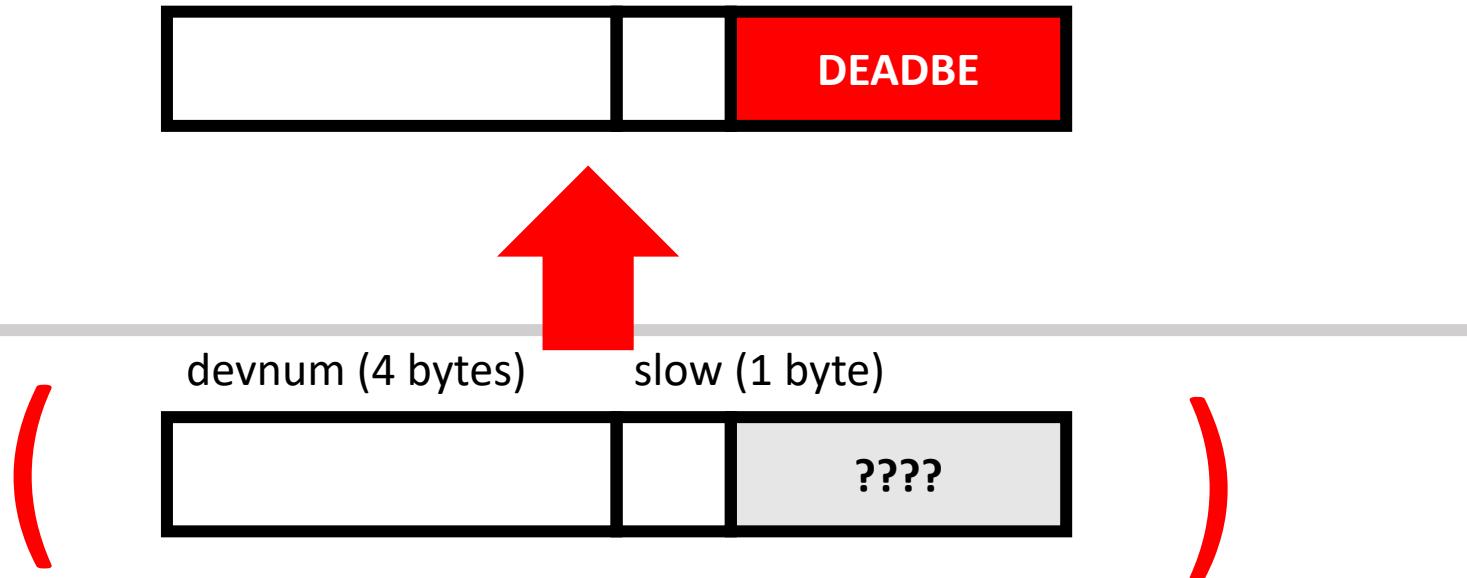
# Uninitialized Padding Problem

```
struct usbdevfs_connectinfo {  
    unsigned int devnum;  
    unsigned char slow;  
};
```



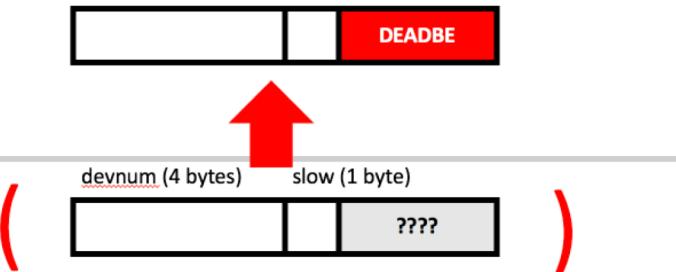
```
struct usbdevfs_connectinfo {  
    .devnum = 1,  
    .slow = 0,  
};
```

# Uninitialized Padding Problem



```
struct usbdevfs_connectinfo {  
    .devnum = 1,  
    .slow = 0,  
};
```

# Uninitialized Padding Problem



## UniSan: Proactive Kernel Memory Initialization to Eliminate Data Leakages

Kangjie Lu, Chengyu Song, Taesoo Kim, and Wenke Lee  
School of Computer Science, Georgia Institute of Technology

### ABSTRACT

The operating system kernel is the de facto trusted computing base for most computer systems. To secure the OS kernel, many security mechanisms, e.g., kASLR and StackGuard, have been increasingly deployed to defend against attacks (e.g., code reuse attack). However, the effectiveness of these protections has been proven to be inadequate—there are many information leak vulnerabilities in the kernel to leak the randomized pointer or canary, thus bypassing kASLR and StackGuard. Other sensitive data in the kernel, such as

### 1. INTRODUCTION

As the de facto trusted computing base (TCB) of computer systems, the operating system (OS) kernel has always been a prime target for attackers. By compromising the kernel, attackers can escalate their privilege to steal sensitive data in the system and control the whole computer. There are three main approaches to launch privilege escalation attacks: 1) direct code injection attacks [2, 3, 4, 5, 6]; 2) user space to user space (U2U) attacks [7, 8, 9, 10, 11, 12, 13, 14, 15, 16]; and 3) code reuse attacks [17]. In addition, the recent trend of memory safety protection has been developed to prevent memory corruption attacks [18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 5510, 5511, 5512, 5513, 5514, 5515, 5516, 5517, 5518, 5519, 5520, 5521, 5522, 5523, 5524, 5525, 5526, 5527, 5528, 5529, 55210, 55211, 55212, 55213, 55214, 55215, 55216, 55217, 55218, 55219, 55220, 55221, 55222, 55223, 55224, 55225, 55226, 55227, 55228, 55229, 552210, 552211, 552212, 552213, 552214, 552215, 552216, 552217, 552218, 552219, 552220, 552221, 552222, 552223, 552224, 552225, 552226, 552227, 552228, 552229, 5522210, 5522211, 5522212, 5522213, 5522214, 5522215, 5522216, 5522217, 5522218, 5522219, 5522220, 5522221, 5522222, 5522223, 5522224, 5522225, 5522226, 5522227, 5522228, 5522229, 55222210, 55222211, 55222212, 55222213, 55222214, 55222215, 55222216, 55222217, 55222218, 55222219, 55222220, 55222221, 55222222, 55222223, 55222224, 55222225, 55222226, 55222227, 55222228, 55222229, 552222210, 552222211, 552222212, 552222213, 552222214, 552222215, 552222216, 552222217, 552222218, 552222219, 552222220, 552222221, 552222222, 552222223, 552222224, 552222225, 552222226, 552222227, 552222228, 552222229, 5522222210, 5522222211, 5522222212, 5522222213, 5522222214, 5522222215, 5522222216, 5522222217, 5522222218, 5522222219, 5522222220, 5522222221, 5522222222, 5522222223, 5522222224, 5522222225, 5522222226, 5522222227, 5522222228, 5522222229, 55222222210, 55222222211, 55222222212, 55222222213, 55222222214, 55222222215, 55222222216, 55222222217, 55222222218, 55222222219, 55222222220, 55222222221, 55222222222, 55222222223, 55222222224, 55222222225, 55222222226, 55222222227, 55222222228, 55222222229, 552222222210, 552222222211, 552222222212, 552222222213, 552222222214, 552222222215, 552222222216, 552222222217, 552222222218, 552222222219, 552222222220, 552222222221, 552222222222, 552222222223, 552222222224, 552222222225, 552222222226, 552222222227, 552222222228, 552222222229, 5522222222210, 5522222222211, 5522222222212, 5522222222213, 5522222222214, 5522222222215, 5522222222216, 5522222222217, 5522222222218, 5522222222219, 5522222222220, 5522222222221, 5522222222222, 5522222222223, 5522222222224, 5522222222225, 5522222222226, 5522222222227, 5522222222228, 5522222222229, 55222222222210, 55222222222211, 55222222222212, 55222222222213, 55222222222214, 55222222222215, 55222222222216, 55222222222217, 55222222222218, 55222222222219, 55222222222220, 55222222222221, 55222222222222, 55222222222223, 55222222222224, 55222222222225, 55222222222226, 55222222222227, 55222222222228, 55222222222229, 552222222222210, 552222222222211, 552222222222212, 552222222222213, 552222222222214, 552222222222215, 552222222222216, 552222222222217, 552222222222218, 552222222222219, 552222222222220, 552222222222221, 552222222222222, 552222222222223, 552222222222224, 552222222222225, 552222222222226, 552222222222227, 552222222222228, 552222222222229, 5522222222222210, 5522222222222211, 5522222222222212, 5522222222222213, 5522222222222214, 5522222222222215, 5522222222222216, 5522222222222217, 5522222222222218, 5522222222222219, 5522222222222220, 5522222222222221, 5522222222222222, 5522222222222223, 5522222222222224, 5522222222222225, 5522222222222226, 5522222222222227, 5522222222222228, 5522222222222229, 55222222222222210, 55222222222222211, 55222222222222212, 55222222222222213, 55222222222222214, 55222222222222215, 55222222222222216, 55222222222222217, 55222222222222218, 55222222222222219, 55222222222222220, 55222222222222221, 55222222222222222, 55222222222222223, 55222222222222224, 55222222222222225, 55222222222222226, 55222222222222227, 55222222222222228, 55222222222222229, 552222222222222210, 552222222222222211, 552222222222222212, 552222222222222213, 552222222222222214, 552222222222222215, 552222222222222216, 552222222222222217, 552222222222222218, 552222222222222219, 552222222222222220, 552222222222222221, 552222222222222222, 552222222222222223, 552222222222222224, 552222222222222225, 552222222222222226, 552222222222222227, 552222222222222228, 552222222222222229, 5522222222222222210, 5522222222222222211, 5522222222222222212, 5522222222222222213, 5522222222222222214, 5522222222222222215, 5522222222222222216, 5522222222222222217, 5522222222222222218, 5522222222222222219, 5522222222222222220, 5522222222222222221, 5522222222222222222, 5522222222222222223, 5522222222222222224, 5522222222222222225, 5522222222222222226, 5522222222222222227, 5522222222222222228, 5522222222222222229, 55222222222222222210, 55222222222222222211, 55222222222222222212, 55222222222222222213, 55222222222222222214, 55222222222222222215, 55222222222222222216, 55222222222222222217, 55222222222222222218, 55222222222222222219, 55222222222222222220, 55222222222222222221, 55222222222222222222, 55222222222222222223, 55222222222222222224, 55222222222222222225, 55222222222222222226, 55222222222222222227, 55222222222222222228, 55222222222222222229, 552222222222222222210, 552222222222222222211, 552222222222222222212, 552222222222222222213, 552222222222222222214, 552222222222222222215, 552222222222222222216, 552222222222222222217, 552222222222222222218, 552222222222222222219, 552222222222222222220, 552222222222222222221, 552222222222222222222, 552222222222222222223, 552222222222222222224, 552222222222222222225, 552222222222222222226, 552222222222222222227, 552222222222222222228, 552222222222222222229, 5522222222222222222210, 5522222222222222222211, 5522222222222222222212, 5522222222222222222213, 5522222222222222222214, 5522222222222222222215, 5522222222222222222216, 5522222222222222222217, 5522222222222222222218, 5522222222222222222219, 5522222222222222222220, 5522222222222222222221, 5522222222222222222222, 5522222222222222222223, 5522222222222222222224, 5522222222222222222225, 5522222222222222222226, 5522222222222222222227, 5522222222222222222228, 5522222222222222222229, 55222222222222222222210, 55222222222222222222211, 55222222222222222222212, 55222222222222222222213, 55222222222222222222214, 55222222222222222222215, 55222222222222222222216, 55222222222222222222217, 55222222222222222222218, 55222222222222222222219, 55222222222222222222220, 55222222222222222222221, 55222222222222222222222, 55222222222222222222223, 55222222222222222222224, 55222222222222222222225, 55222222222222222222226, 55222222222222222222227, 55222222222222222222228, 55222222222222222222229, 552222222222222222222210, 552222222222222222222211, 552222222222222222222212, 552222222222222222222213, 552222222222222222222214, 552222222222222222222215, 552222222222222222222216, 552222222222222222222217, 552222222222222222222218, 552222222222222222222219, 552222222222222222222220, 552222222222222222222221, 552222222222222222222222, 552222222222222222222223, 552222222222222222222224, 552222222222222222222225, 552222222222222222222226, 552222222222222222222227, 552222222222222222222228, 552222222222222222222229, 5522222222222222222222210, 5522222222222222222222211, 5522222222222222222222212, 5522222222222222222222213, 5522222222222222222222214, 5522222222222222222222215, 5522222222222222222222216, 5522222222222222222222217, 5522222222222222222222218, 5522222222222222222222219, 5522222222222222222222220, 5522222222222222222222221, 5522222222222222222222222, 5522222222222222222222223, 5522222222222222222222224, 5522222222222222222222225, 5522222222222222222222226, 5522222222222222222222227, 5522222222222222222222228, 5522222222222222222222229, 55222222222222222222222210, 55222222222222222222222211, 55222222222222222222222212, 55222222222222222222222213, 55222222222222222222222214, 55222222222222222222222215, 55222222222222222222222216, 55222222222222222222222217, 55222222222222222222222218, 55222222222222222222222219, 55222222222222222222222220, 55222222222222222222222221, 55222222222222222222222222, 55222222222222222222222223, 55222222222222222222222224, 55222222222222222222222225, 55222222222222222222222226, 55222222222222222222222227, 55222222222222222222222228, 55222222222222222222222229, 552222222222222222222222210, 552222222222222222222222211, 552222222222222222222222212, 552222222222222222222222213, 552222222222222222222222214, 552222222222222222222222215, 552222222222222222222222216, 552222222222222222222222217, 552222222222222222222222218, 552222222222222222222222219, 552222222222222222222222220, 552222222222222222222222221, 552222222222222222222222222, 552222222222222222222222223, 552222222222222222222222224, 552222222222222222222222225, 552222222222222222222222226, 552222222222222222222222227, 552222222222222222222222228, 552222222222222222222222229, 5522222222222222222222222210, 5522222222222222222222222211, 5522222222222222222222222212, 5522222222222222222222222213, 5522222222222222222222222214, 5522222222222222222222222215, 5522222222222222222222222216, 5522222222222222222222222217, 5522222222222222222222222218, 5522222222222222222222222219, 5522222222222222222222222220, 5522222222222222222222222221, 5522222222222222222222222222, 5522222222222222222222222223, 5522222222222222222222222224, 5522222222222222222222222225, 5522222

# Ecall/Ocall: EDL Interface for SGX

```
// Enclave.edl
untrusted {
    test_struct e/ocall_test_struct(void);
}
```

If there is a padding issue in `test_struct`, it leaks (or inject) potentially sensitive data (e.g., *a private key like HeartBleed*)

# Ecall/Ocall: EDL Interface for SGX

// Enclave.edl

```
untrusted {  
    test_struct ocall test_struct(void);  
}
```

Host

Ecall

Ocall

Enclave

PWNED!

CS.CR] 25 Oct 2017

Breaking the Isolation of Secure Enclave Memory via Structure Padding (Extend-Abstract)

Lee Taesoo Kim

Georgia Institute of Technology

Abstract

Software Guard Extensions (SGX) aims to provide an isolated execution environment, known as an enclave, for a user-level process to maximize its confidentiality and integrity. In this paper, we study how uninitialized data inside a secure enclave can be leaked via structure padding. We found that, during ECALL and OCALL, proxy functions that are automatically generated by the Intel SGX Software Development Kit (SDK) fully copy structure variables from an enclave to the normal memory to return the result of an ECALL function and to pass input parameters to an OCALL function. If the

trusted functions (e.g., system calls). Their any other attempts to execute untrusted functions (e.g., jumping into non-enclave code) result in faults.

Intel SGX Software Development Kit (SDK) is shipped with a tool called Edger8r [1] that automatically and securely generated code for ECALL and OCALL interfaces. Although SGX enclaves can access both EPCs and normal memory, non-enclave applications can only access the normal memory. Thus, all input and output values for the between them need to be copied first and then copied caller later. The Edger8r tool creates all such edge

arXiv'17

# DEMO: SGX Bleed POC

⋮

# Implication 1: Using memory-safe language doesn't solve the problem

**Rust SGX SDK**

Rust SGX SDK helps developers write Intel SGX applications in Rust programming language.

**v0.9.0 Release**

Almost there! Rust SGX SDK 0.9.0 is ready up to a version of the future 0.10.0, with the as well as many new features. Also, it does support for programming SGX untrusted part in it's easy to port Rust crate to the SGX trust execution environment and write the whole SGX application refer to [release\\_notes](#) for further details.

**PWNED!**



# Implication 2: Using certified C compilers doesn't help either

C11 (ISO/IEC 9899:201x), 701 page



§6.2.6.1/6

When a value is stored in an object or structure (...),  
the bytes of the object in its representation that correspond  
to any padding or alignment bytes take unspecified  
values.



# New Attack Vectors

- Page table attack
- Branch shadowing attack
- Rowhammer against SGX
- L1 terminal fault against SGX (i.e., Foreshadow)

# New Attack Vectors

- Page table attack
  - e.g., leaking image data
- Branch shadowing attack
  - e.g., breaking RSA
- Rowhammer against SGX
  - e.g., freezing machines
- L1 terminal fault against SGX (i.e., Foreshadow)
  - e.g., breaking SGX ecosystem (and more!)

# Page Table Attack (controlled-channel attack)

- Page level access pattern → reveal sensitive info.  
(e.g., page faults, page access bits, ...)

2015 IEEE Symposium on Security and Privacy

## Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems

Yuanzhong Xu  
The University of Texas at Austin  
[yxu@cs.utexas.edu](mailto:yxu@cs.utexas.edu)

Weidong Cui  
Microsoft Research  
[wdcui@microsoft.com](mailto:wdcui@microsoft.com)

Marcus Peinado  
Microsoft Research  
[marcuspe@microsoft.com](mailto:marcuspe@microsoft.com)

*Abstract*—The presence of large numbers of side channels in popular feature-rich commodity processors has inspired a long line of work on extracting sensitive information from the trusted computation base.

SP'15

## Telling Your Secrets Without Page Faults: Stealthy Page Table-Based Attacks on Enclaved

Jo Van Bulck  
imec-DistriNet, KU Leuven  
[jo.vanbulck@cs.kuleuven.be](mailto:jo.vanbulck@cs.kuleuven.be)

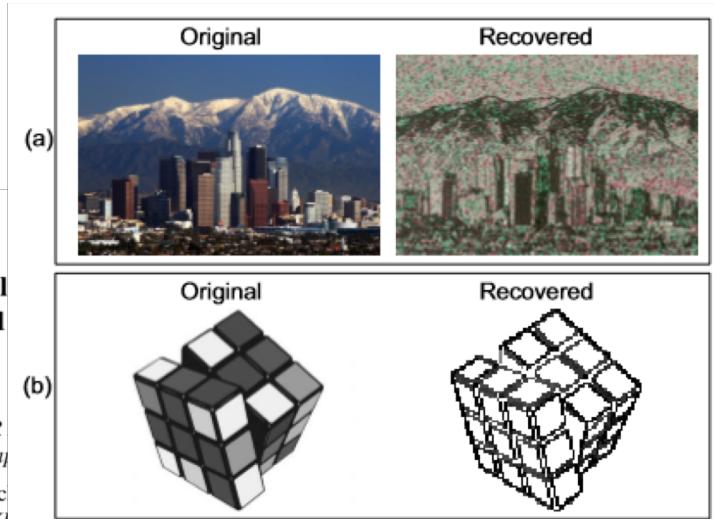
Nico Weichbrodt  
IBR DS, TU Braunschweig  
[weichbr@ibr.cs.tu-bs.de](mailto:weichbr@ibr.cs.tu-bs.de)

Frank Piessens  
imec-DistriNet, KU Leuven  
[frank.piessens@cs.kuleuven.be](mailto:frank.piessens@cs.kuleuven.be)

Raoul Strackx  
imec-DistriNet, KU Leuven  
[raoul.strackx@cs.kuleuven.be](mailto:raoul.strackx@cs.kuleuven.be)

### Abstract

Protected module architectures, such as Intel SGX, enable strong trusted computing guarantees for hardware-enforced enclaves on top of a potentially malicious operating system. However, such enclaved execution environ-



ware to make it relatively easy to run untrusted legacy applications without exposing them to the host environment.

An essential part of these guarantees is that the hardware prevents the host from reading memory or writing to memory while an enclave is running.

Sec'17

# DEMO: Page Fault Attack



The screenshot shows an iTerm2 terminal window with the following details:

- Menu Bar:** iTerm2, Shell, Edit, View, Profiles, Toolbelt, Window, Help.
- Toolbar:** Standard red, yellow, and green circular buttons.
- User Information:** 1. mingwei@...
- Terminal Path:** mingwei@sgx3:~/workspace/t-sgx/test/sgx-pf-attack\$
- Text Area:** The main area is mostly blank, indicating the state of the page fault attack.

# Defense: T-SGX

- Using Intel Transactional Synchronization Extension (TSX) to isolate page faults inside SGX

## T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs

Ming-Wei Shih<sup>†,\*</sup>, Sangho Lee<sup>†</sup>, and Taesoo Kim  
Georgia Institute of Technology  
[\[mingwei.shih, sangho, taesoo\]@gatech.edu](mailto:[mingwei.shih, sangho, taesoo]@gatech.edu)

Marcus Peinado  
Microsoft Research  
[marcuspe@microsoft.com](mailto:marcuspe@microsoft.com)

*Abstract*—Intel Software Guard Extensions (SGX) is a hardware-based trusted execution environment (TEE) that enables secure execution of a program in an isolated environment, an *enclave*. SGX hardware protects the running enclave against malicious software, including an operating system (OS), a hypervisor, and even low-level firmwares. This strong security

### I. INTRODUCTION

Hardware-based trusted execution environments (TEEs) have become one of the most various security threats, including kernel exploits, hardware Trojans,

NDSS'17

# Key Idea: TSX Isolates Faults!

- Unexpected side-effects (see, DrK [CCS'16])
- Any faults → invokes an abort handler

```
1 unsigned status;
2
3 // begin a transaction
4 if ((status = _xbegin()) == _XBEGIN_STARTED) {
5     // execute a transaction
6     [code]
7     // atomic commit
8     _xend();
9 } else {
10     // abort
11 }
```

## Breaking Kernel Address Space Layout Randomization with Intel TSX

Yeongjin Jang, Sangho Lee, and Taesoo Kim  
School of Computer Science, Georgia Institute of Technology  
{yeongjin.jang, sangho, taesoo}@gatech.edu

### ABSTRACT

Kernel hardening has been an important topic since many applications and security mechanisms often consider the kernel as part of their Trusted Computing Base (TCB). Among various hardening techniques, Kernel Address Space Layout Randomization (KASLR) is the most effective and widely adopted defense mechanism that can practically mitigate various memory corruption vulnerabilities, such as buffer overflow and use-after-free. In principle, KASLR is a simple yet effective technique that randomizes the base addresses of memory regions, which makes it difficult for an attacker to predict the memory layout of the kernel.

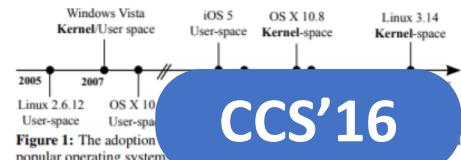
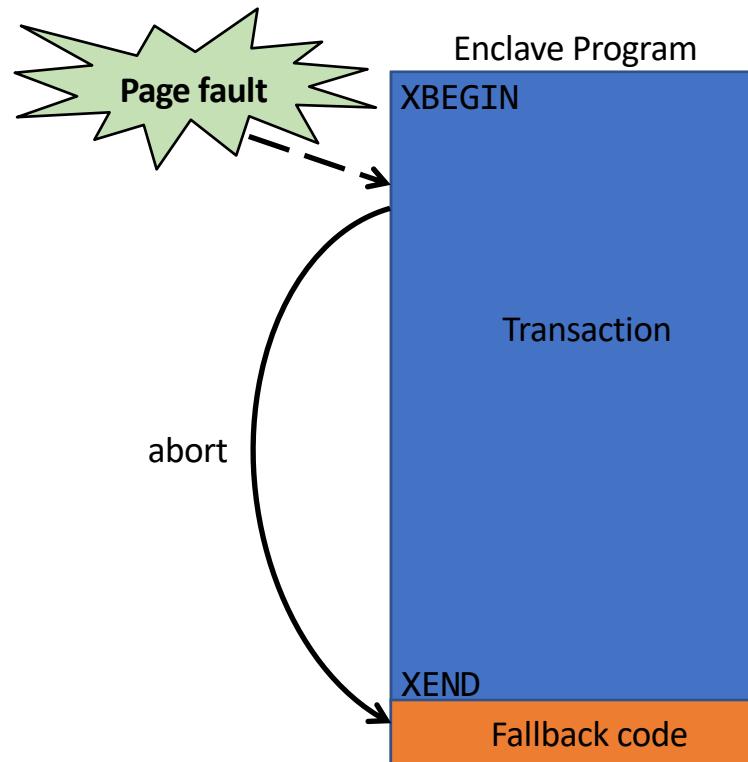


Figure 1: The adoption of KASLR in popular operating systems.

CCS'16

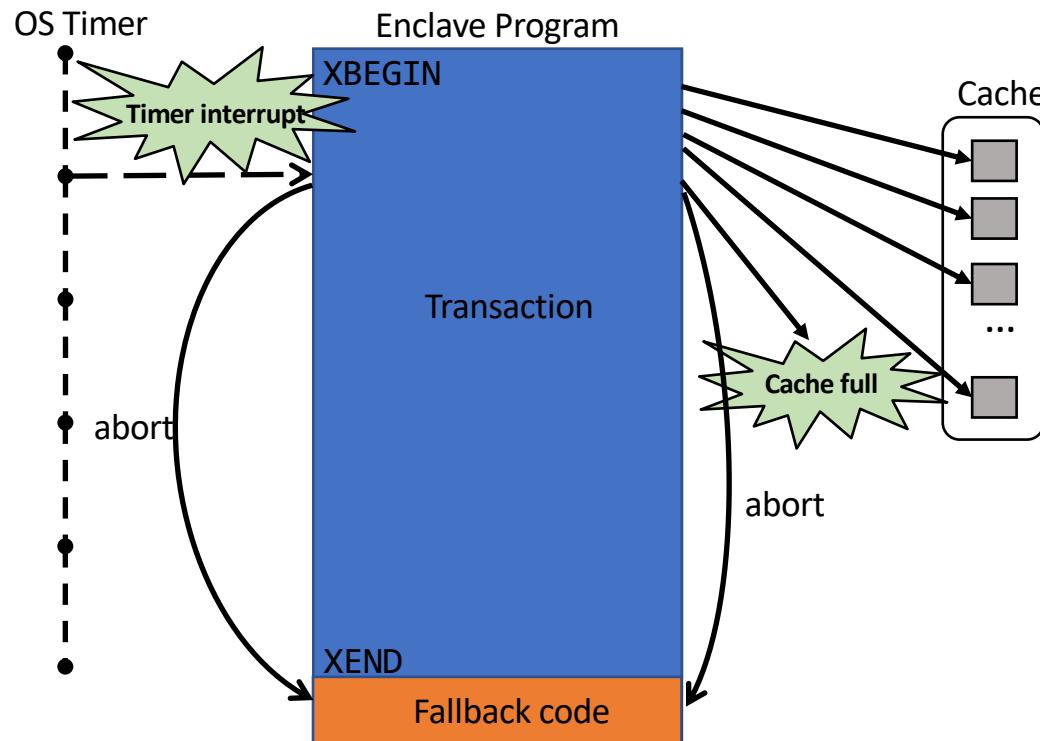
# A Strawman Solution

- Protect the entire program with TSX!



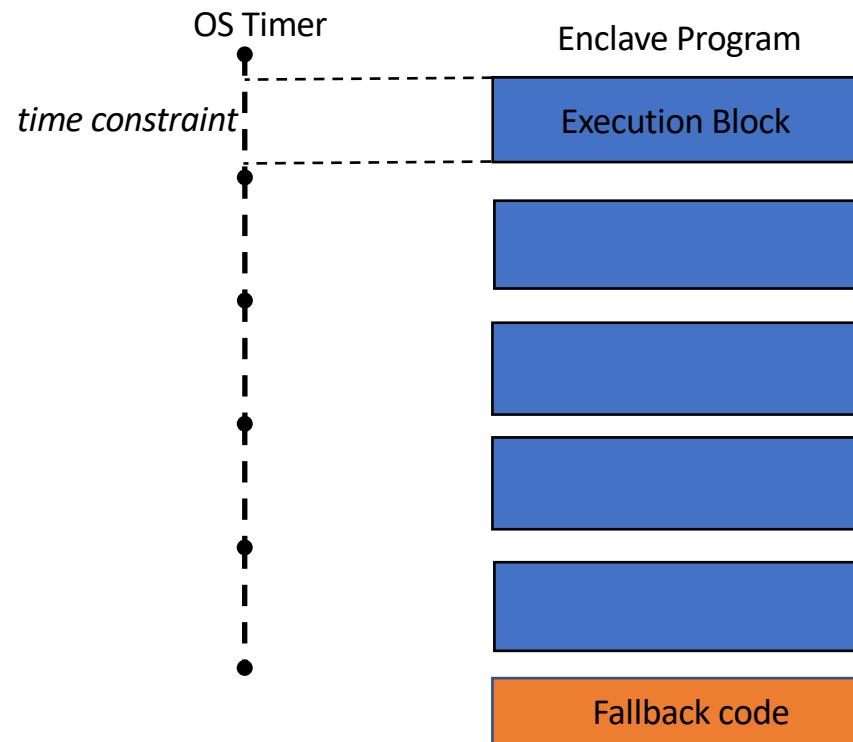
# Challenge: Not Progressing!

- 1) Timer interrupt (i.e., external faults)
- 2) False TSX aborts (e.g., capacity)

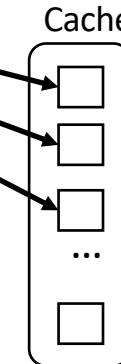


# Approach: Smaller Execution Units

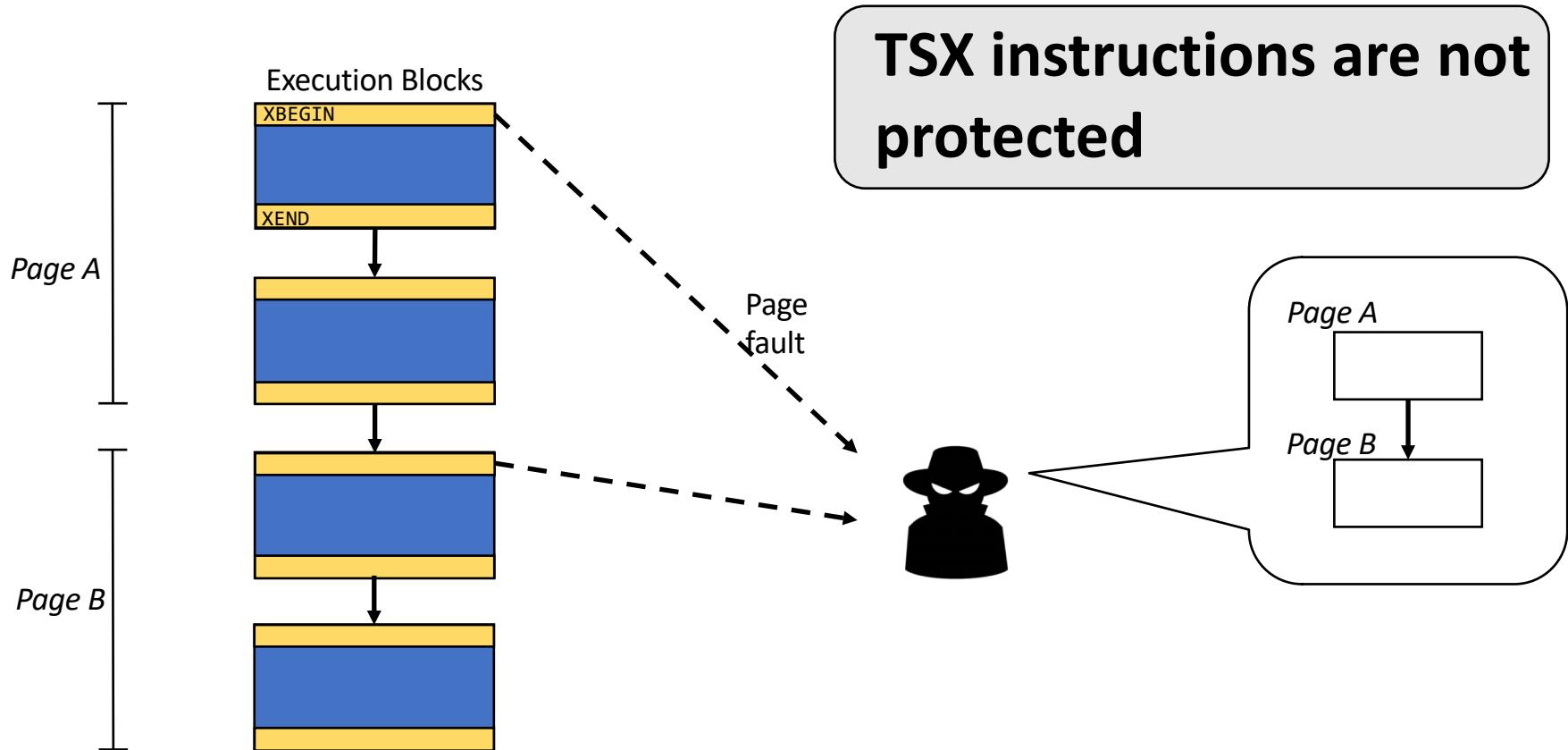
## 1) *Execution time analysis*



## 2) *Cache analysis*

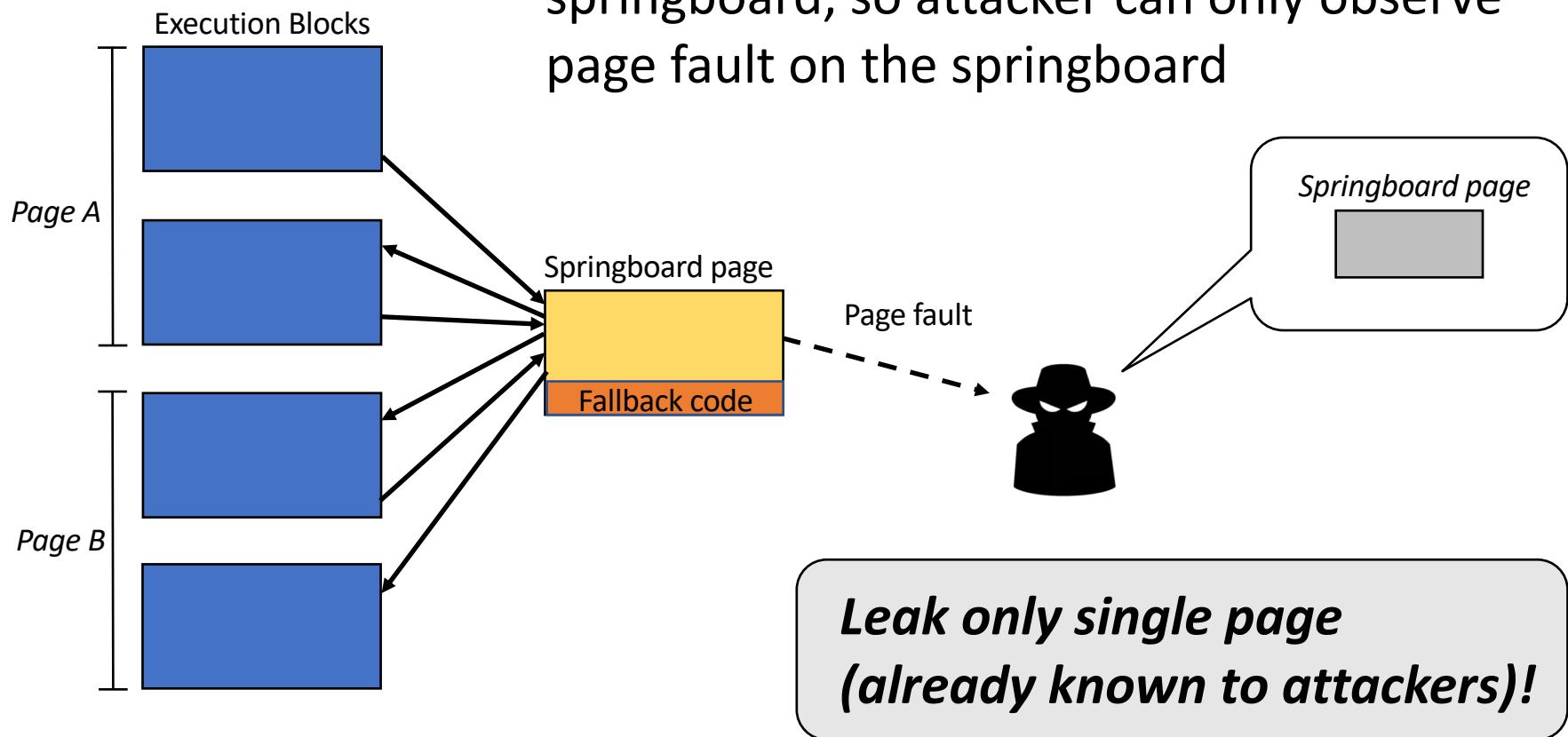


# This design still leaks information

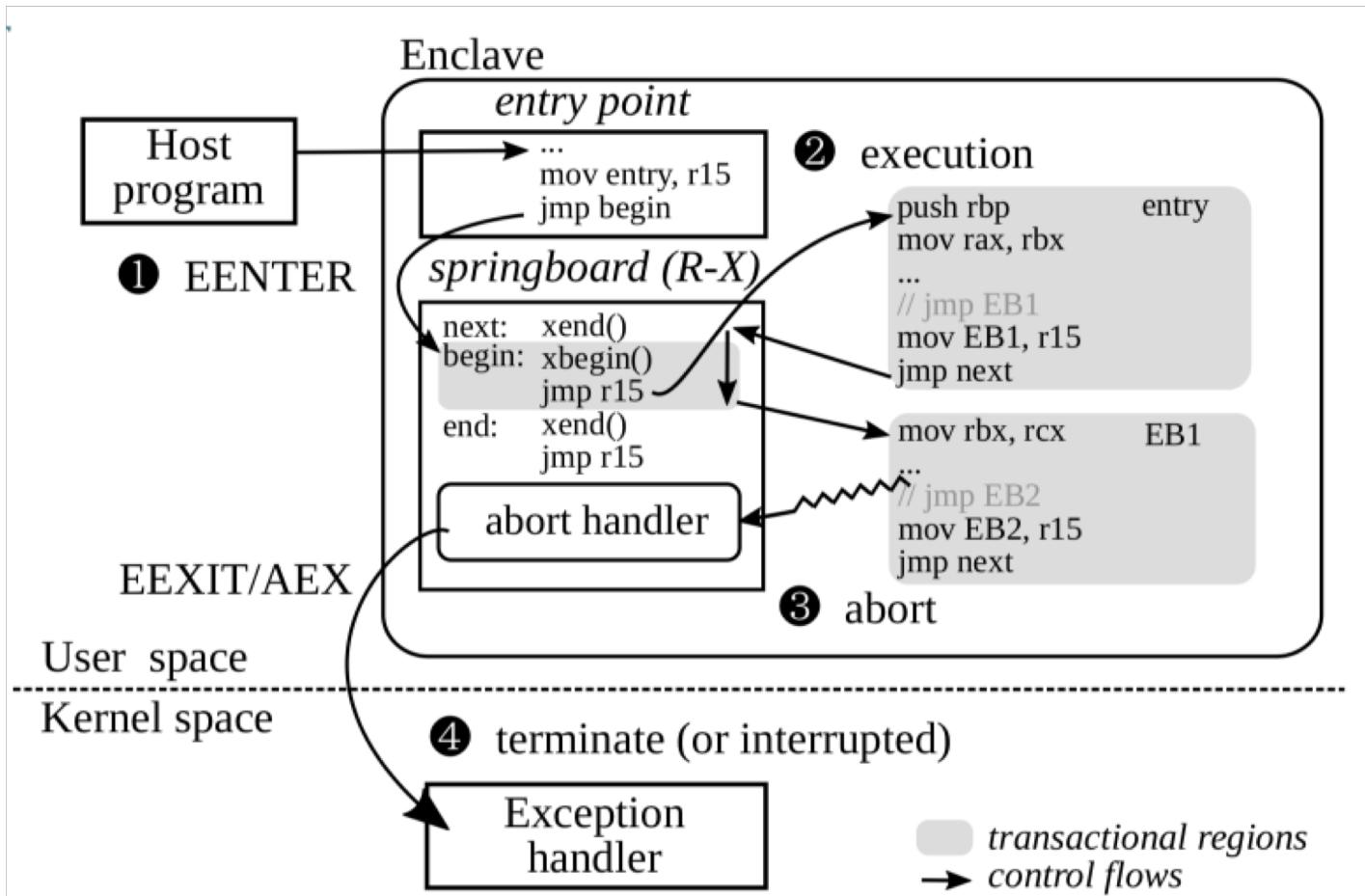


# Solution: Springboard

All transactions begin and end on the springboard, so attacker can only observe page fault on the springboard

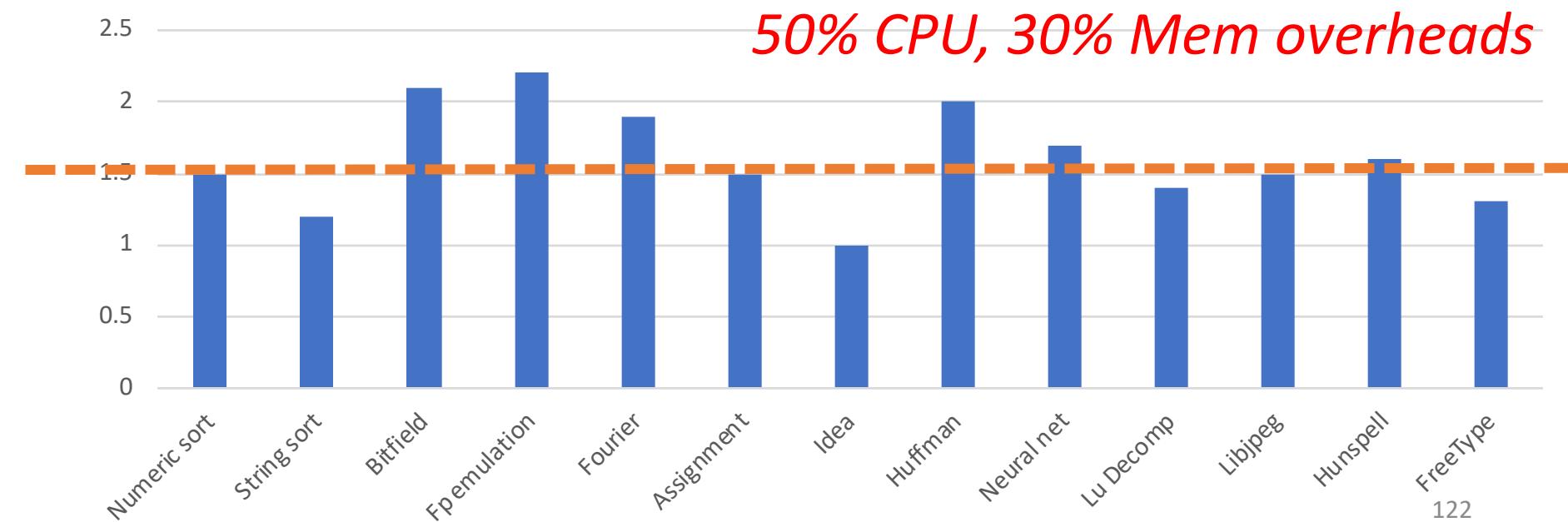


# Design of T-SGX (Compiler)



# T-SGX: Eradicating Page Faults

- Technique to avoid *false* aborts (e.g., capacity)
- Security analysis → springboard design
- Performance optimizations



# DEMO: T-SGX

```
mingwei@sgx3:~/workspace/t-sgx/test/sgx-pf-attack$ █
```

{

# New Attack Vectors

- Page table attack
  - e.g., leaking image data
- Branch shadowing attack
  - e.g., breaking RSA
- Rowhammer against SGX
  - e.g., freezing machines
- L1 terminal fault against SGX (i.e., Foreshadow)
  - e.g., breaking SGX ecosystem (and more!)

# New Side Channel: Branch Shadowing Attack

- Finer-grained, yet noise-free!  
(unlike page faults / cache attacks, respectively)
- Observation:
  - **Branch history is shared** between SGX and non-SGX
    - Execution history of an enclave affects the performance of non-SGX execution

# New Side Channel: Branch Shadowing Attack

- Finer-grained, yet noise-free!  
(unlike page faults / cache attacks, respectively)

## Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing

Sangho Lee<sup>†</sup> Ming-Wei Shih<sup>†</sup> Prasun Gera<sup>†</sup> Taesoo Kim<sup>†</sup> Hyesoon Kim<sup>†</sup> Marcus Peinado\*

<sup>†</sup> Georgia Institute of Technology

\* Microsoft Research

### Abstract

Intel has introduced a hardware-based trusted execution environment, Intel Software Guard Extensions (SGX), that provides a secure, isolated execution environment, or enclave, for a user program without trusting any underlying software (e.g., an operating system) or firmware.

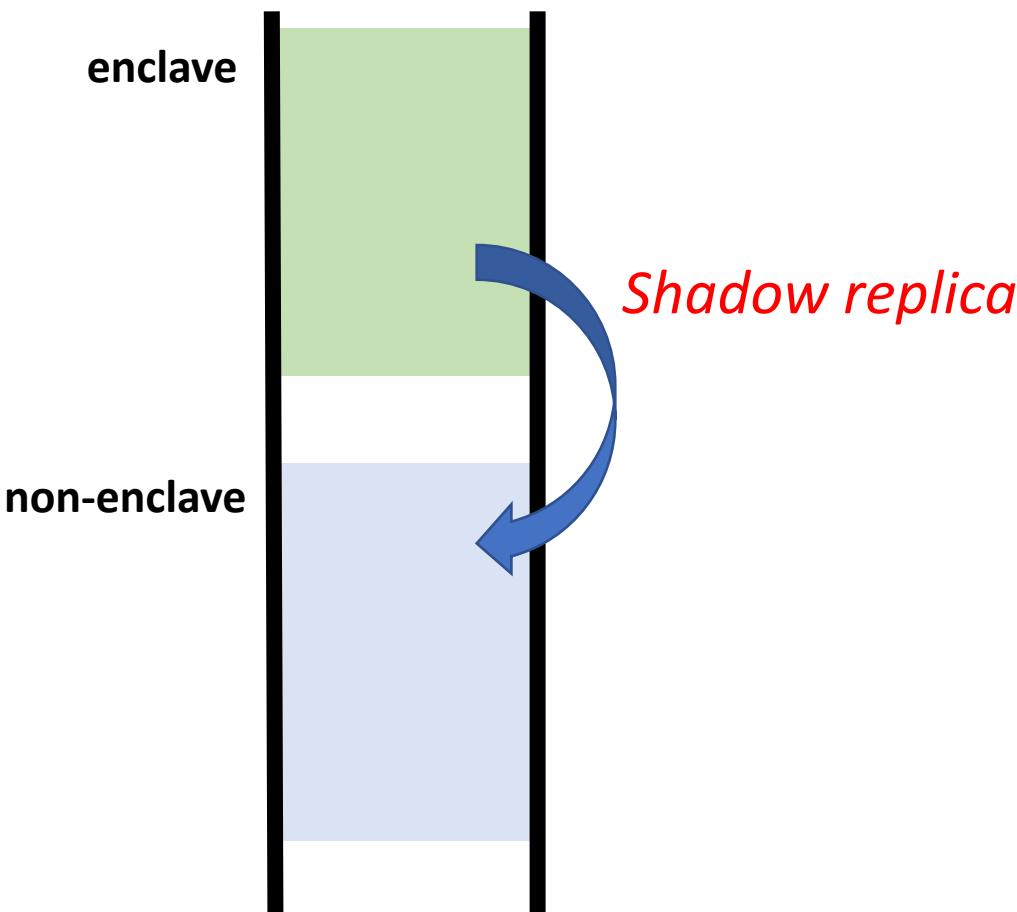
we need either to fully trust the operator, which is problematic [16], or encrypt all data before uploading them to the cloud and perform computations directly on the encrypted data. The latter can be broken by homomorphic encryption, which is still slower than fully homomorphic encryption, which is not yet practical. When we use a private cloud or personal workstation,

SEC'17

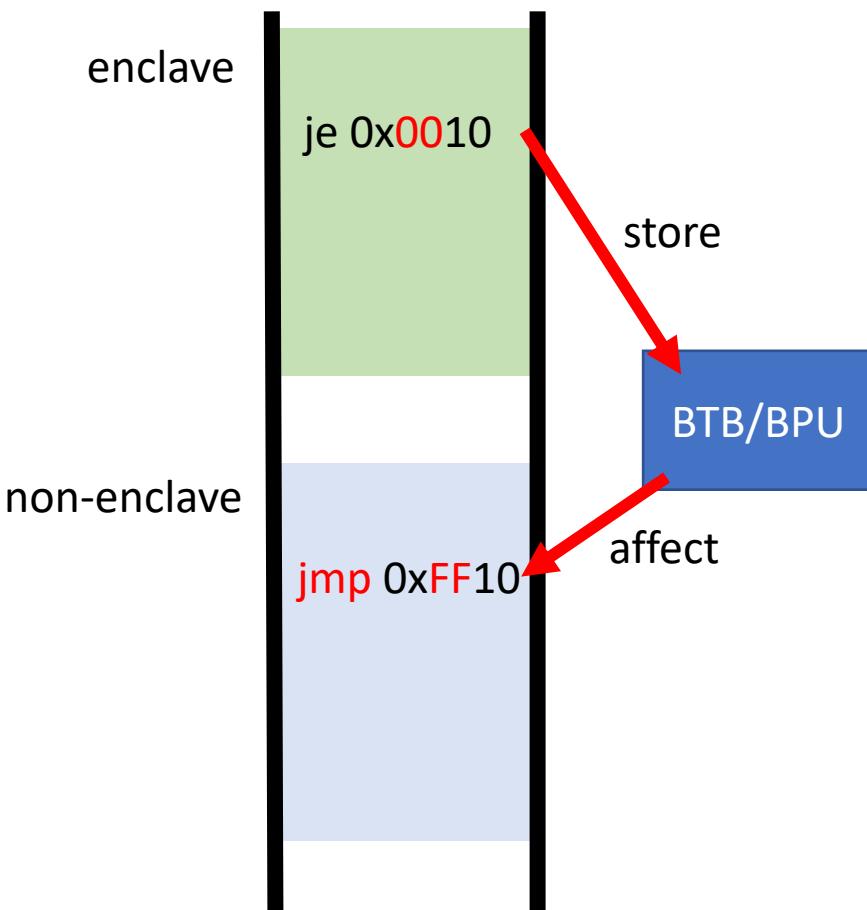
# Idea: Exploiting New HW Features

- Intel Skylake (and Broadwell) introduced two new debugging features that report prediction results
    - Last Branch Record (LBR)
    - Intel Processor Trace (PT)
- But only for *non-enclave* programs  
(or enclave on a debug mode)

# Our Approach: Branch Shadowing

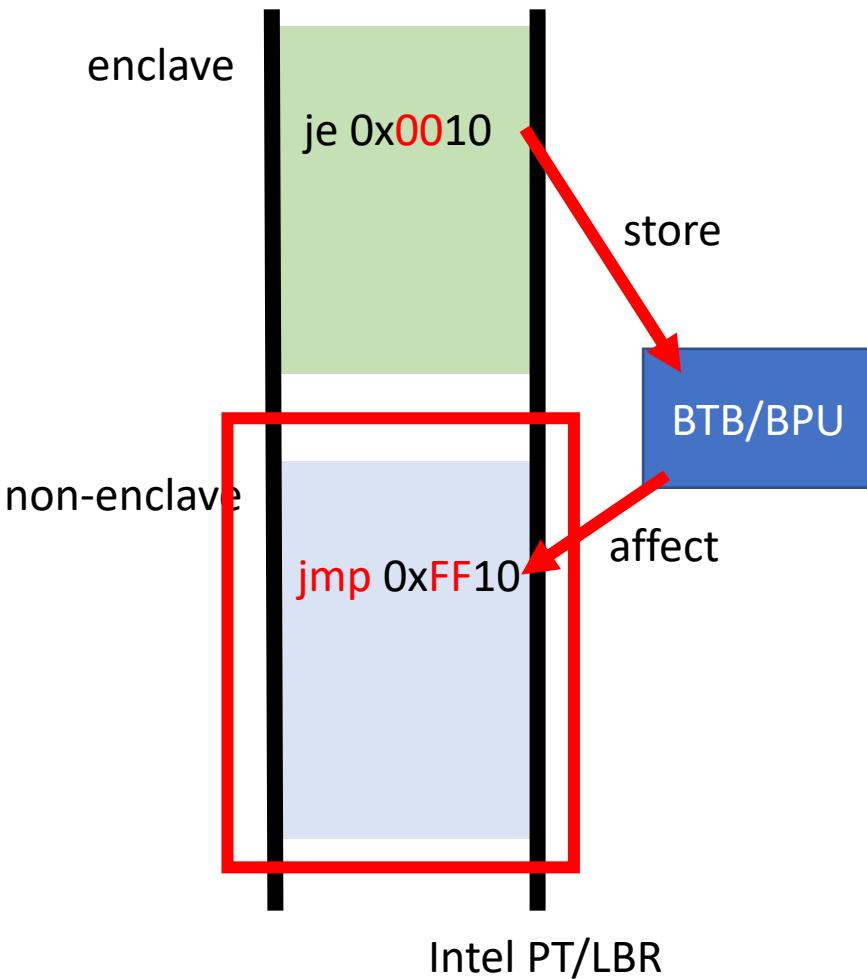


# Our Approach: Branch Shadowing



- are mapped onto the same branch prediction buffer
- is a shadow copy of an enclave program forced to take all branches (e.g., `je` → `jmp`)

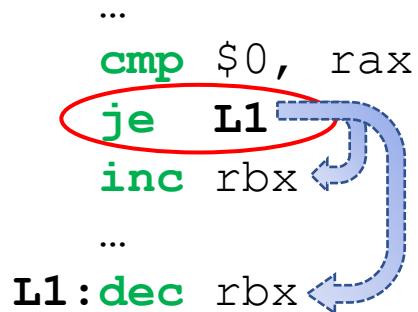
# Our Approach: Branch Shadowing



- are mapped onto the same branch prediction buffer
- is a shadow copy of an enclave program forced to take all branches (e.g., `je` → `jmp`)
- Monitor with LBR/PT and extract branch prediction results indirectly

# Branch Prediction 101

Predict the next instr. of a branch instr. to avoid pipeline stalls



*Which one would be the next instr.  
to be predicted?*

# Branch Prediction 101

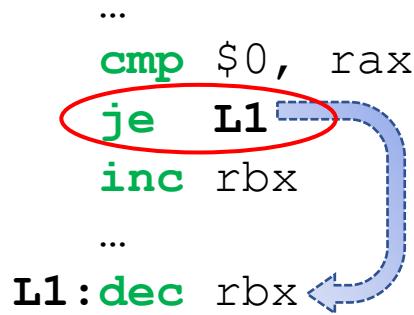
Predict the next instr. of a branch instr. to avoid pipeline stalls

```
...  
cmp $0, rax  
je L1   
inc rbx   
...  
L1:dec rbx
```

**Make this prediction if**  
**1) there is no history or**  
**2) the branch has not been taken**

# Branch Prediction 101

Predict the next instr. of a branch instr. to avoid pipeline stalls



Make this prediction if  
the branch has been taken

*Conditional behavior → Reveal history*

*How can we know which branch was taken?*

# Branch Prediction vs. Misprediction

- Measure branch execution time
  - Take **longer** if a branch is **incorrectly predicted** (e.g., roll back, clear pipeline, jump to the correct target)

	Prediction		Misprediction	
	mean	stdev	mean	stdev
RDTSCP	94.21	13.10	120.61	806.56
PT CYC	59.59	14.44	90.64	191.48
LBR cycle	25.69	9.72	35.04	10.52

→ **Observable difference but high measurement noise**

# Exploiting New HW Features

- Intel LBR/PT *explicitly report* the prediction result, but only *taken branches* (w/ limited buf size)
- Approach:
  - Translating all cond. to be taken in the shadow copy
  - Synchronization b/w enclave and its shadow

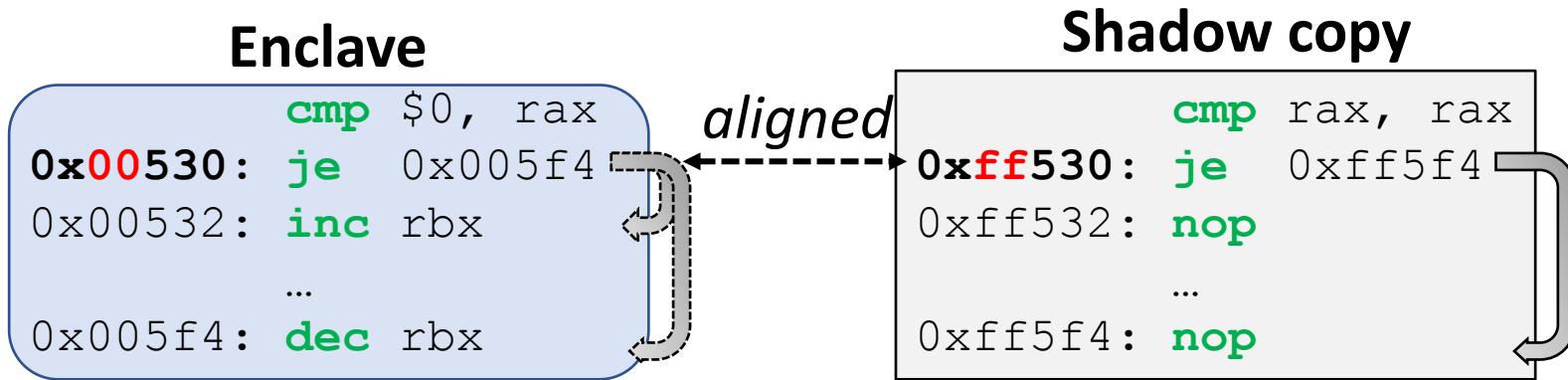
# Example: Inferring Cond. Branch

## Enclave

```
        cmp    $0,  rax  
0x00530: je     0x005f4  
0x00532: inc    rbx  
...  
0x005f4: dec    rbx
```

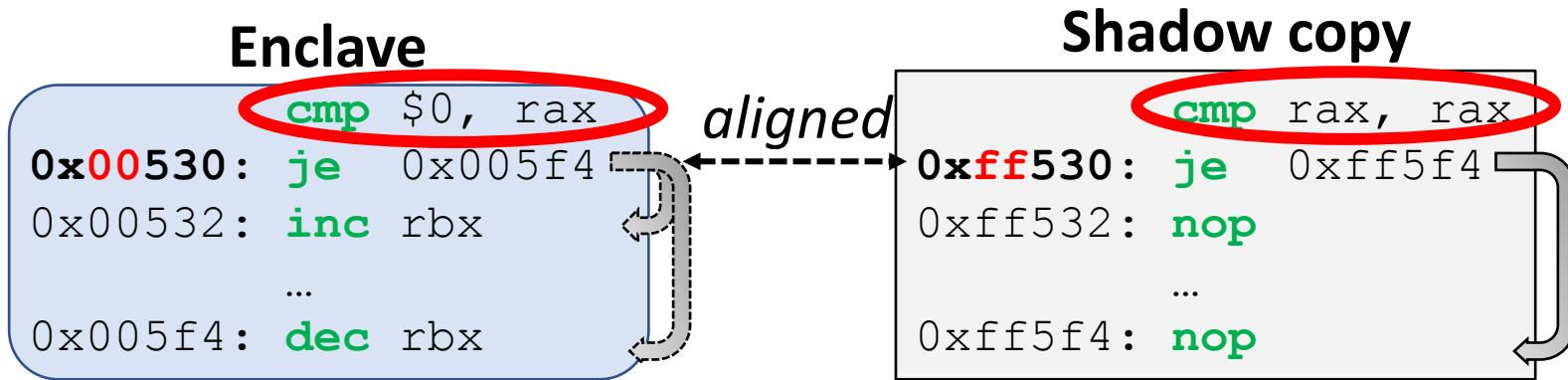


# Example: Inferring Cond. Branch



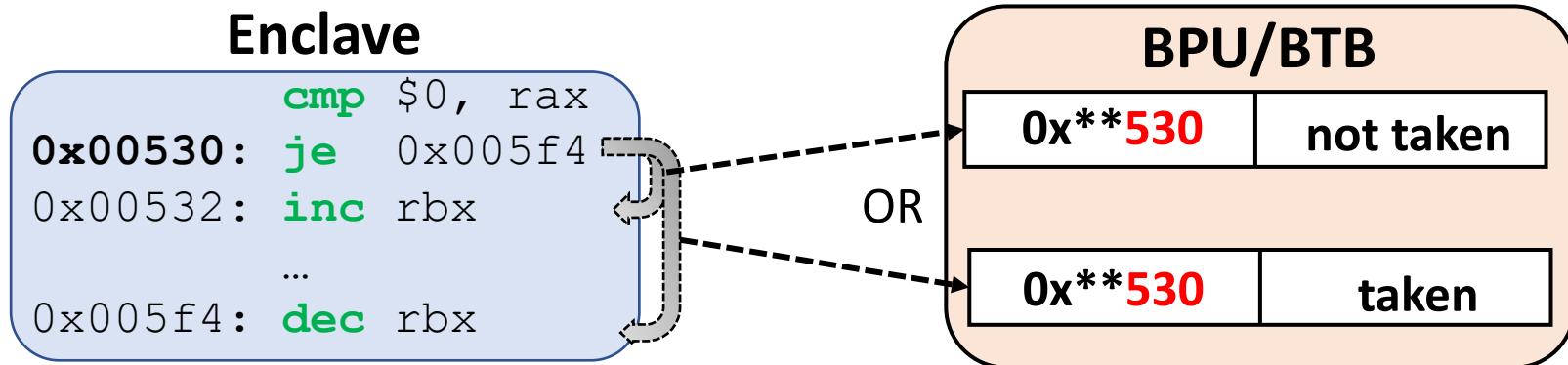
- Prepare a shadow copy w/
  - Colliding conditional branches

# Example: Inferring Cond. Branch

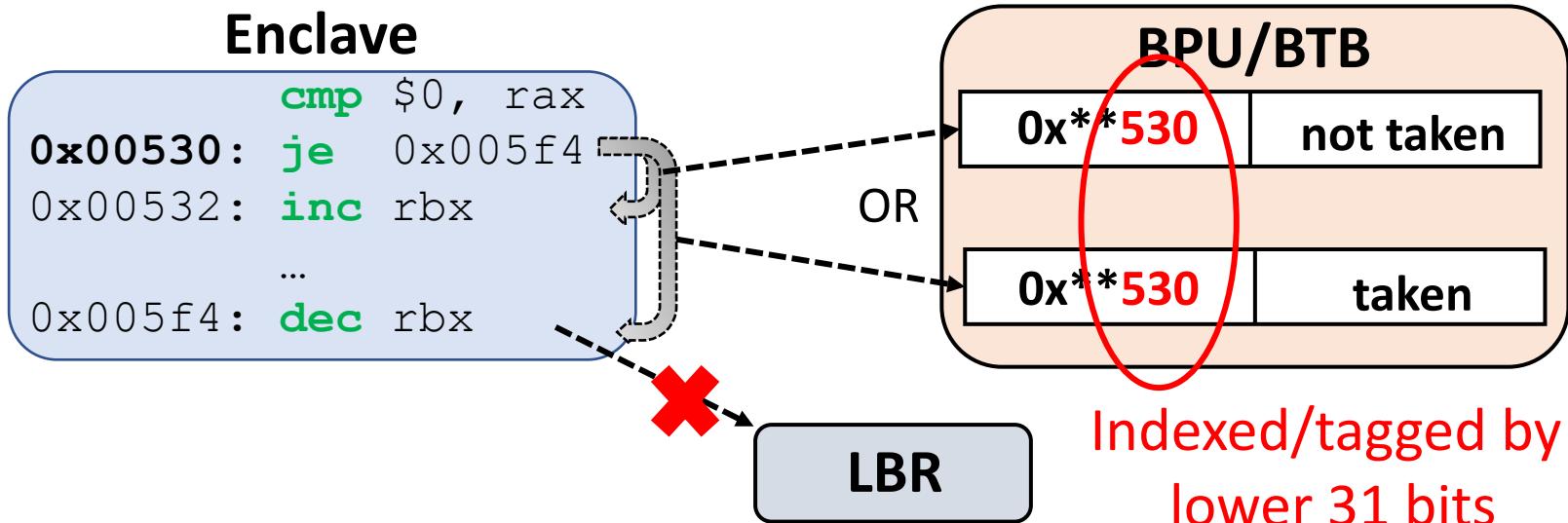


- Prepare a shadow copy w/
  - Colliding conditional branches
  - Always to be taken (to be monitored by LBR)

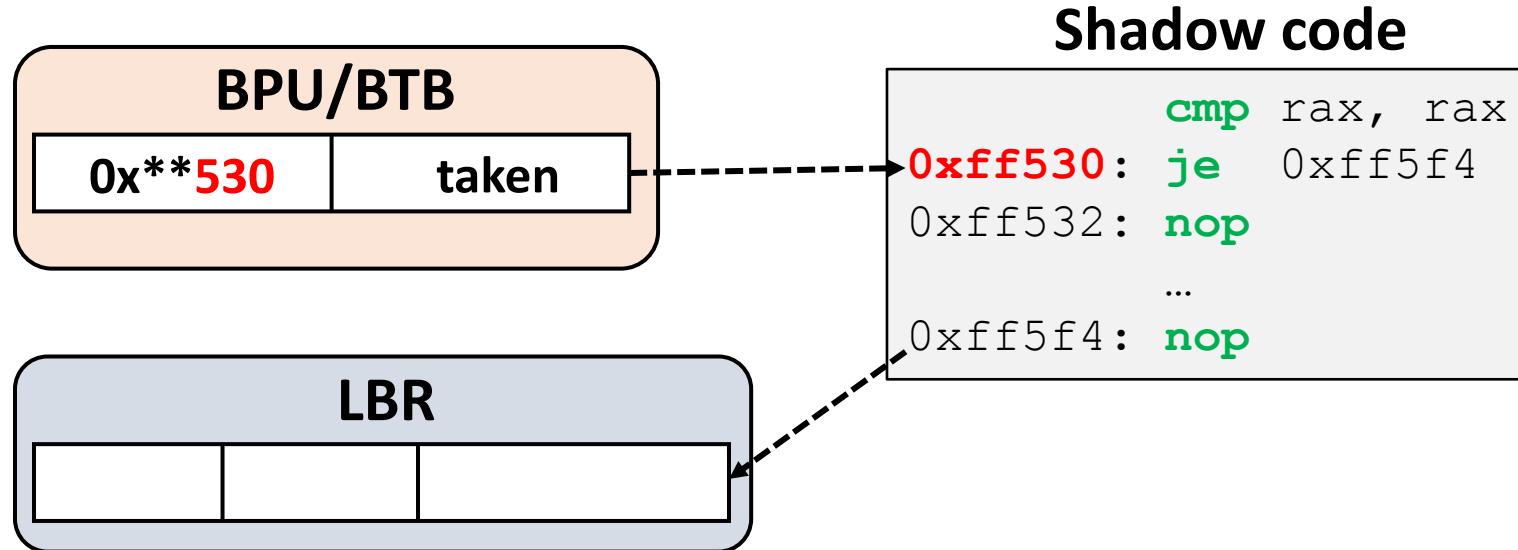
# Example: Inferring Cond. Branch



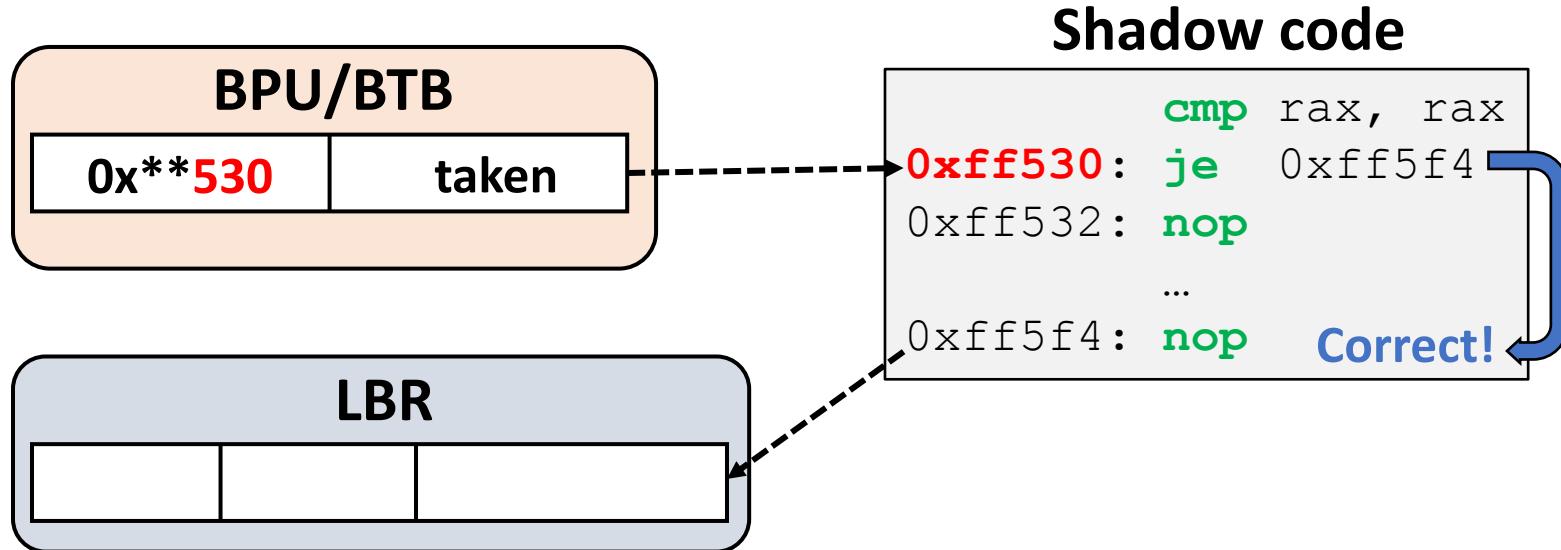
# Example: Inferring Cond. Branch



# Example: Inferring Taken Branch

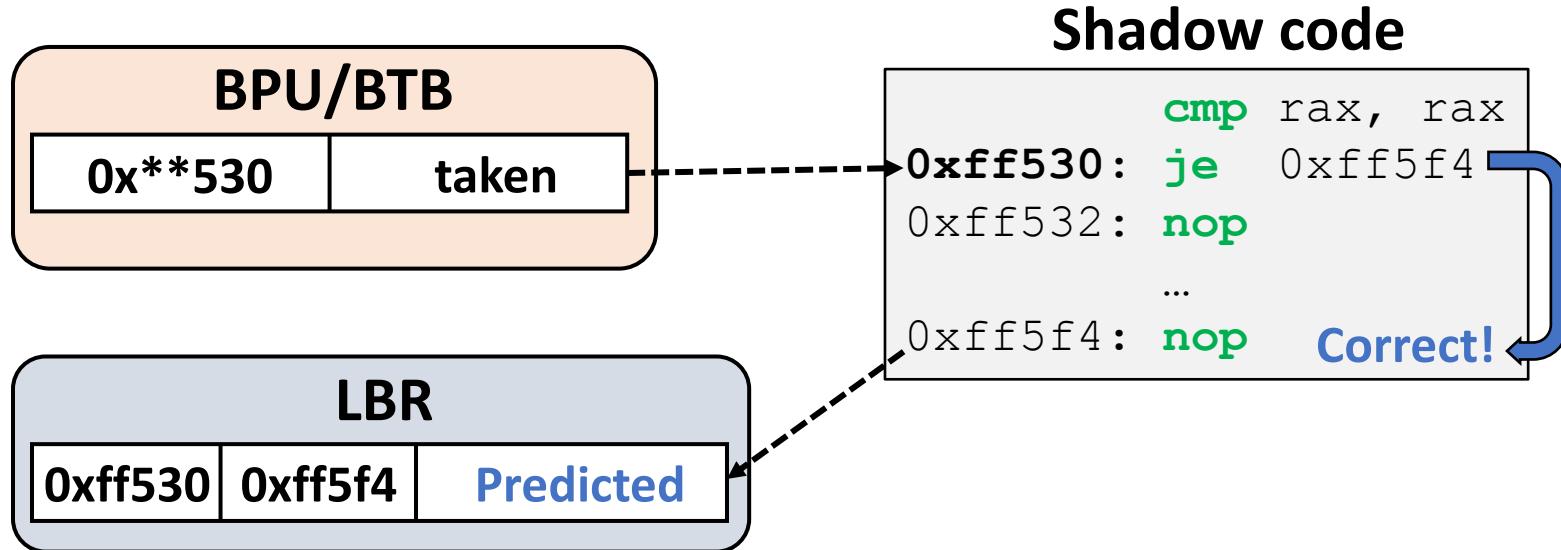


# Example: Inferring Taken Branch



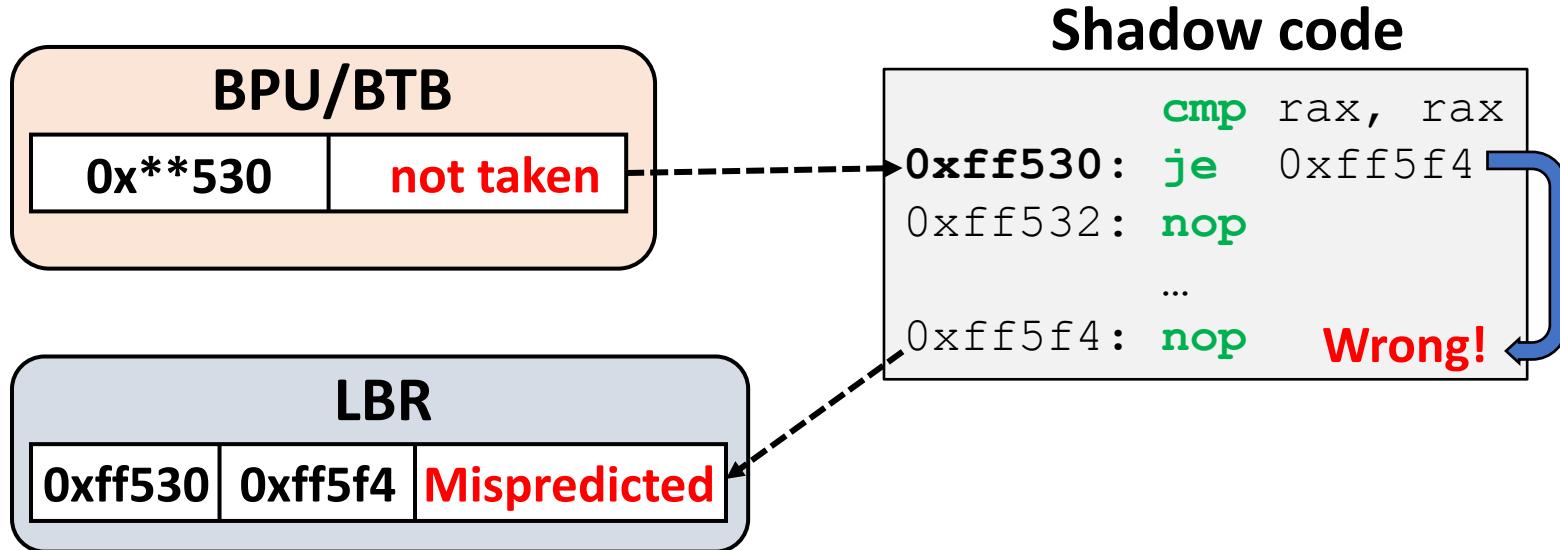
- BPU/BTB *correctly predicts* the execution of the shadow branch using the history

# Example: Inferring Taken Branch



- If LBR reports:
  - **Predicted** → The target branch **has been taken**

# Example: Inferring Not-taken Branch



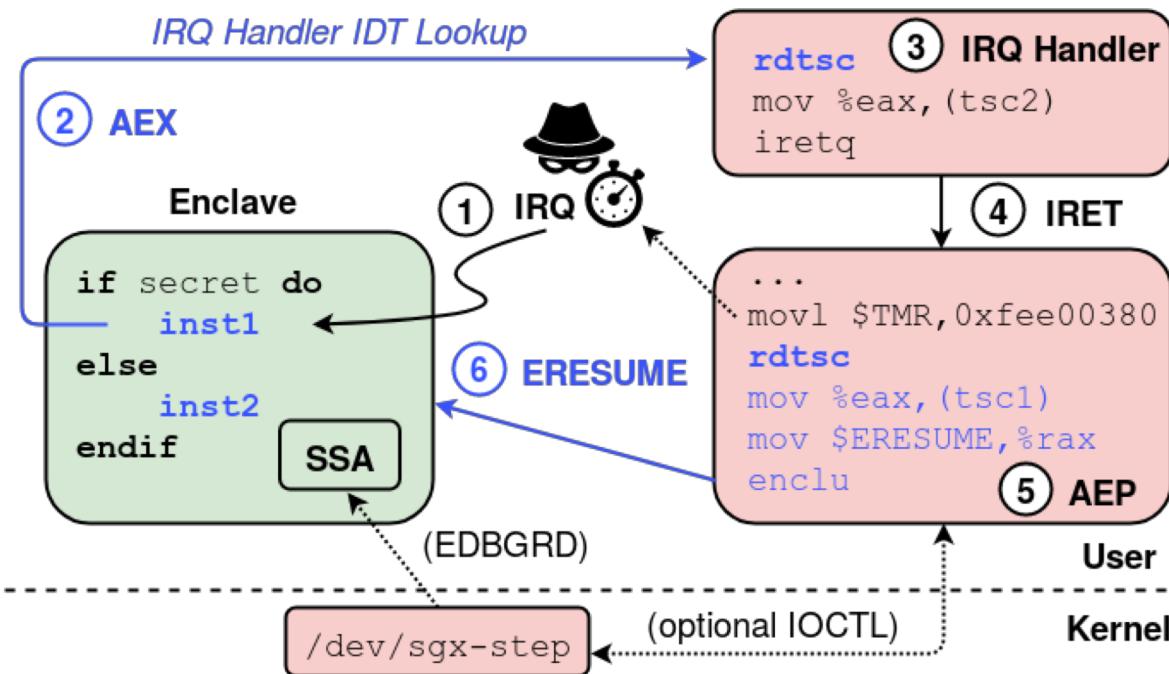
- If LBR reports:
  - **Predicted** → The target branch **has been taken**
  - **Mispredicted** → The target branch **has NOT been taken**

# Enabling Single Stepping!

- Check branch state as frequently as possible to overcome the capacity limit of BPU/BTB and LBR
  - e.g., BTB: 4,096 entries, LBR: 32 entries (Skylake)
- Increase timer interrupt frequency
  - Adjust the TSC value of the local APIC timer **~50 cycles**
- Disable the CPU cache
  - CD bit of the CR0 register **~5 cycles**

# SGX-Step: Open Source Framework

- Local APIC
- Userspace mapping for PTE



## SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control

Jo Van Bulck  
imec-DistriNet, KU Leuven  
jo.vanbulck@cs.kuleuven.be

Frank Piessens  
imec-DistriNet, KU Leuven  
frank.piessens@cs.kuleuven.be

Raoul Strackx  
imec-DistriNet, KU Leuven  
raoul.strackx@cs.kuleuven.be

**Abstract**  
Protected module architectures such as Intel SGX hold the promise of protecting sensitive computations from a potentially compromised operating system. Recent research convincingly demonstrated, however, that SGX's strengthened adversary model also gives rise to a new class of powerful, low-noise side-channel attacks leveraging first-rate control over hardware. These attacks commonly rely on frequent enclave preemptions to obtain fine-grained side-channel observations. A maximal temporal resolution is achieved via the `rdtsc` instruction, which is, however, not supported by state-of-the-art enclave execution control schemes, hence do not generally achieve such instruction-level granularity.

This paper presents SGX-Step, an open-source Linux kernel framework that allows an untrusted host process to configure APIC timer interrupts and track page table entries directly from user space. We contribute and evaluate concerns, the past years have seen a significant research effort [3, 6, 9] on Protected Module Architectures (PMAs) that support isolated execution of security-sensitive application components or *enclaves* with a minimal Trusted Computing Base (TCB). These proposals have in common that they enforce security primitives directly in hardware, or in a small hypervisor, so as to prevent the untrusted OS from access-

SysTEX'17

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

# Example: Attacking RSA Exp.

```
/* X = A^E mod N */
mbedtls_mpi_exp_mod(X, A, E, N, _RR) {
    ...
    while (1) {
        ...
        // i-th bit of exponent
        ei = (E->p[nblimbs] >> bufsize) & 1;

        if (ei == 0 && state == 0)
            continue;
        if (ei == 0 && state == 1)
            mpi_montmul(X, X, N, mm, &T);
        ...
    }
    ...
}
```

Sliding-window  
exponentiation of mbedtls

# Example: Attacking RSA Exp.

```
/* X = A^E mod N */  
mbedtls_mpi_exp_mod(X, A, E, N, _RR) {  
    ...  
    while (1) {  
        ...  
        // i-th bit of exponent  
        ei = (E->p[nblimbs] >> bufsize) & 1;  
  
        if (ei == 0 && state == 0)  
            continue;  
        if (ei == 0 && state == 1)  
            mpi_montmul(X, X, N, mm, &T);  
        ...  
    }  
    ...  
}
```

Sliding-window  
exponentiation of mbedtls

Taken only when ei is zero

# Example: Attacking RSA Exp.

```
/* X = A^E mod N */  
mbedtls_mpi_exp_mod(X, A, E, N, _RR) {  
    ...  
    while (1) {  
        ...  
        // i-th bit of exponent  
        ei = (E->p[nblimbs] >> bufsize) & 1;  
  
        if (ei == 0 && state == 0)  
            continue;  
        if (ei == 0 && state == 1)  
            mpi_montmul(X, X, N, mm, &T);  
    }  
}
```

Sliding-window  
exponentiation of mbedtls

Taken only when ei is zero

- The probability that the two branches return different results: **0.34** (error rates)
- The inference accuracy of the remaining bits: **0.998**
- ***We were able to recover 66% of an RSA private key bit from a single run.***
  - ***≤10 runs are enough to fully recover the key.***

# DEMO: Branch Shadowing Attack

# What Else?

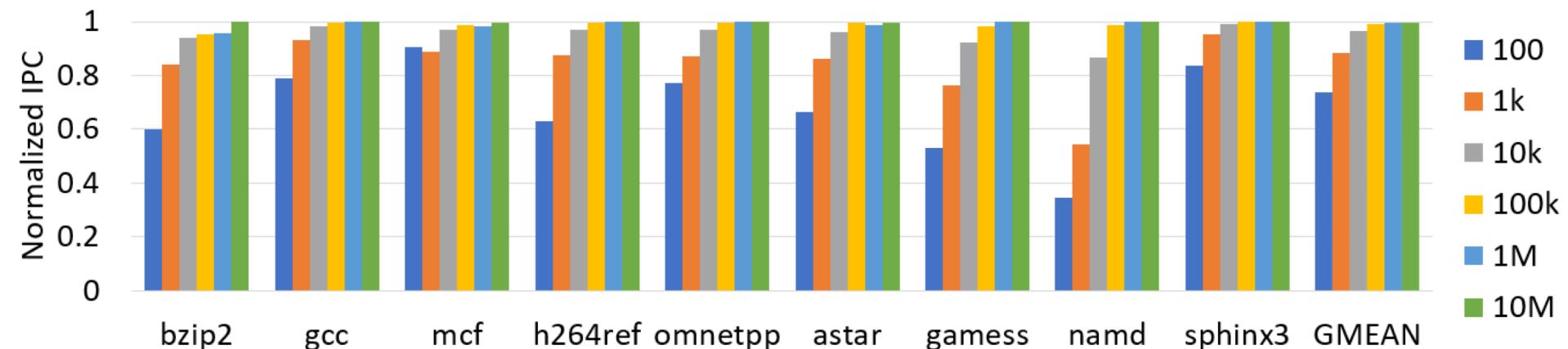
Program/Function	Description	Leakages
libc/strtol	Convert a string into an integer	The sign and length of an input Hexadecimal digits
libc/vfprintf	Print a formatted string	The input format string
LIBSVM/k_function	Evaluate a kernel function	The type of a kernel (e.g., linear, RBF) The number of features
Apache/lookup_builtin_method	Parse the method of an HTTP request	HTTP request method (e.g., GET, POST)

# Defense: Flushing Branch States (Hardware)

- Clear branch states during enclave mode switches

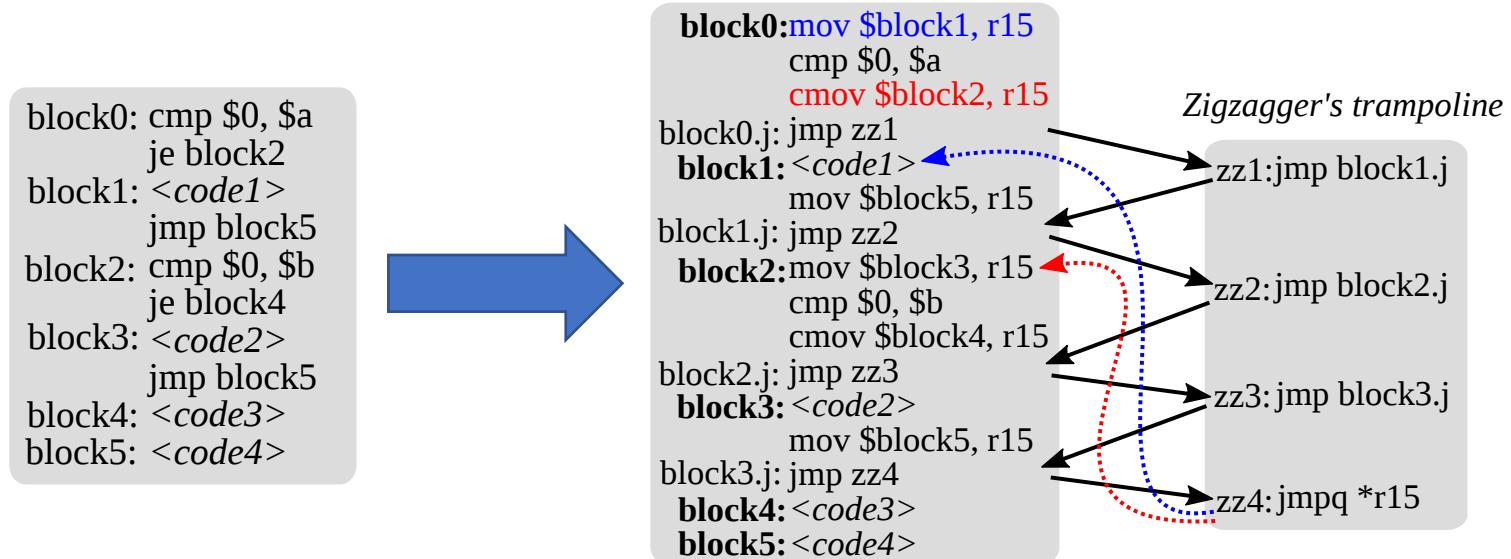
# Defense: Flushing Branch States (Hardware)

- Clear branch states during enclave mode switches
- How much overheads (depending on frequency)?
  - Simulation: Flushing per >10k cycles incurs negligible overheads



# Mitigation: Obfuscating Branch (Software/Compiler)

- Set of conditional/indirect branches → a single indirect branch + conditional move instructions
- The final indirect branch has a lot of targets such that it is difficult to infer its state.



# Example: Branch Obfuscation

```
L0: cmp $0, $a  
    je L2  
L1: ...  
L2: ...
```

transformation

```
L0: mov $L1, r15  
    cmp $0, $a  
    cmov $L2, r15  
    jmp Z1  
L1: ...  
L2: ...  
...  
Z1: jmpq *r15
```

Can identify whether L1 or L2 has been executed

Can identify whether Z1 has been executed but not its target

# Mitigation: Obfuscating Branch (Software/Compiler)

- LLVM-based implementation
- Overhead (nbench):  $\leq 1.5\times$
- Just mitigate the attack, don't solve it completely

# New Attack Vectors

- Page table attack
  - e.g., leaking image data
- Branch shadowing attack
  - e.g., breaking RSA
- Rowhammer against SGX
  - e.g., freezing machines
- L1 terminal fault against SGX (i.e., Foreshadow)
  - e.g., breaking SGX ecosystem (and more!)

# Controlling Bit Flipping in DRAM

- Reported random bit flippings happening in DRAM
- Rowhammer by Google Project Zero (2015)
- Further enhanced by many researchers

## Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

Yoongu Kim<sup>1</sup> Ross Daly\* Jeremie Kim<sup>1</sup> Chris Fallin\* Ji Hye Lee<sup>1</sup>  
Donghyuk Lee<sup>1</sup> Chris Wilkerson<sup>2</sup> Konrad Lai Onur Mutlu<sup>1</sup>

<sup>1</sup>Carnegie Mellon University    <sup>2</sup>Intel Labs

**Abstract.** Memory isolation is a key property of a reliable and secure computing system — an access to one memory address should not have unintended side effects on data stored in other addresses. However, as DRAM process technology scales down to smaller dimensions, it becomes more difficult to prevent DRAM cells from electrically interacting with each other. In this paper, we expose the vulnerability of commodity DRAM chips to disturbance errors. By reading from the same address in DRAM, we show that it is possible to corrupt data in nearby addresses. More specifically, activating the same row in DRAM corrupts data in nearby rows. We demonstrate this phenomenon on Intel and AMD systems using a malicious program that generates many DRAM accesses. We induce errors in most DRAM modules (110 out of 129) from three major DRAM manufacturers. From this we conclude that many deployed systems are likely to be at risk. We identify

disturbance errors, DRAM manufacturers have been employing a two-pronged approach: (i) improving inter-cell isolation through circuit-level techniques [22, 32, 49, 61, 73] and (ii) screening for disturbance errors during post-production testing [3, 4, 64]. We demonstrate that their efforts to contain disturbance errors have not always been successful, and that erroneous DRAM chips have been slipping into the field.<sup>1</sup>

In this paper, we expose the existence and the widespread nature of disturbance errors in commodity DRAM chips sold and used today. Among 129 DRAM modules we analyzed (comprising over 100 million DRAM cells), we found disturbance errors in 110 of them. These errors are not limited to one manufacturer; they are present in DRAM chips from three different manufacturers, which suggests that the problem is widespread. We also find errors in the first generation of DRAM chips, and we show that the problem continues to exist in more advanced generations of process technology. We show

ISCA'14

## Project Zero

News and updates from the Project Zero team at Google

Monday, March 9, 2015

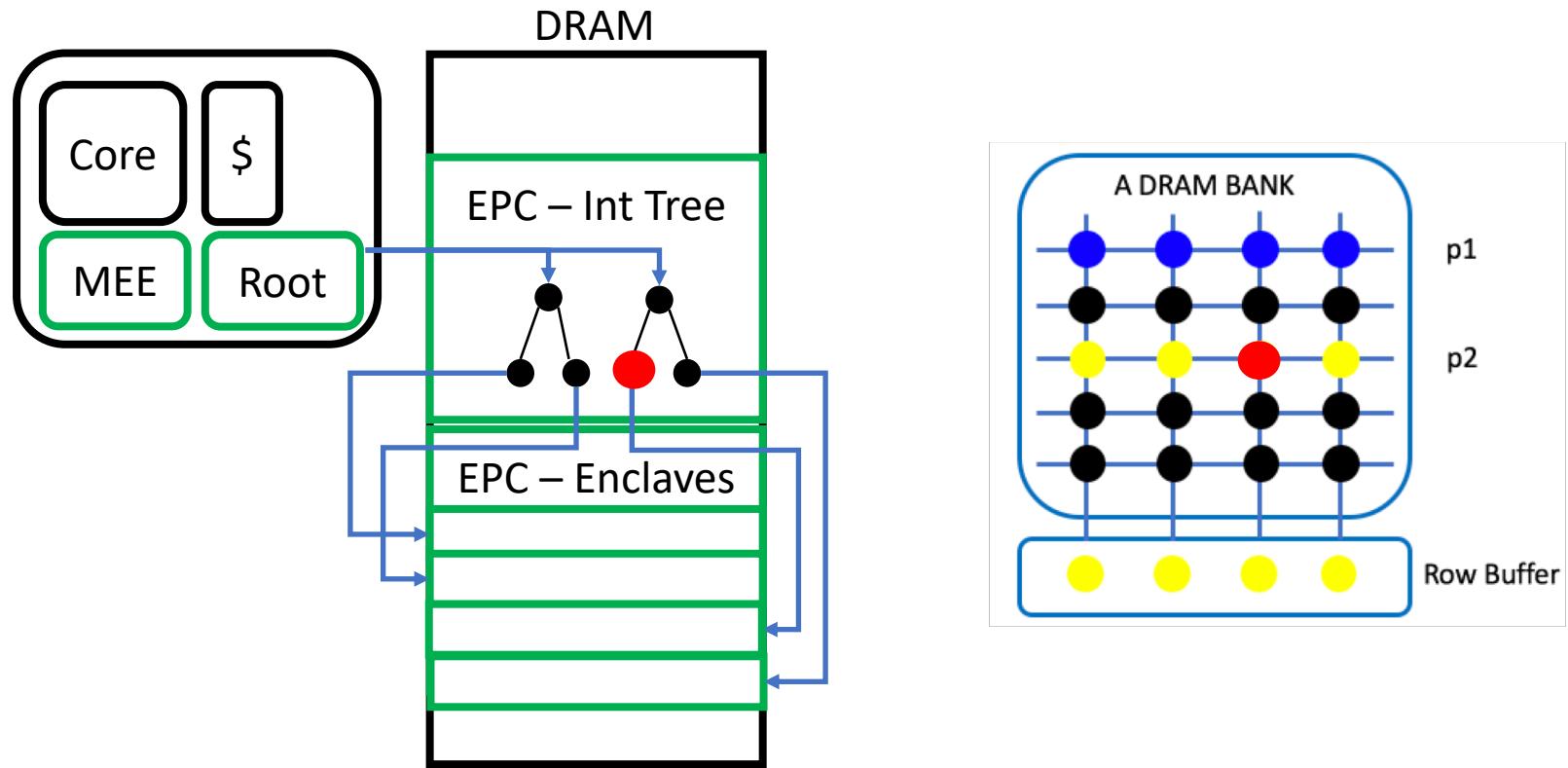
### Exploiting the DRAM rowhammer bug to gain kernel privileges

Posted by Mark Seaborn, sandbox builder and breaker, with contributions by Thomas Dullien, reverse engineer

[This guest post continues Project Zero's practice of promoting excellence in security research on the Project Zero blog]

# SGX-Bomb: Rowhammer Attack

- Integrity violation of EPC results in CPU lockdown
- Rowhammer (SW) can trigger the violation!



# SGX-Bomb: Rowhammer Attack

- Integrity violation of EPC results in CPU lockdown
- Rowhammer (SW) can trigger the violation!

```
void dbl_sided_rowhammer(uint64_t *p1, uint64_t *p2, uint64_t n_reads) {  
    while(n_reads-- > 0) {  
        // read memory p1 and p2  
        asm volatile("mov (%0), %%r10;" :: "r"(p1) : "memory");  
        asm volatile("mov (%0), %%r11;" :: "r"(p2) : "memory");  
        // flush p1 and p2 from the cache  
        asm volatile("clflushopt (%0);" :: "r"(p1));  
        asm volatile("clflushopt (%0);" :: "r"(p2));  
    }  
    chk_flip();  
}
```

## SGX-BOMB: Locking Down the Processor via Rowhammer Attack

Yeongjin Jang\*  
Oregon State University  
yeongjin.jang@oregonstate.edu

Jaehyuk Lee  
KAIST  
jaehyuk.lee@kaist.ac.kr

Sangho Lee  
Georgia Institute of Technology  
sangho@gatech.edu

Taesoo Kim  
Georgia Institute of Technology  
taesoo@gatech.edu

### Abstract

Intel Software Guard Extensions (SGX) provides a strongly isolated memory space, known as an *enclave*, for a user process, ensuring confidentiality and integrity against software and hardware attacks. Even the operating system and hypervisor cannot access the enclave because of the hardware-level isolation. Further, hardware attacks are neither able to disclose plaintext data from the enclave because its memory is always encrypted nor modify it because its integrity is always verified using an integrity tree. When the processor detects any integrity violation, it locks itself to prevent further damages; that is, a system reboot is necessary. The processor lock seems a reasonable solution against such a powerful hardware attacker; however, if a software attacker has a way to trigger integrity

**ACM Reference Format:**  
Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. 2017. SGX-BOMB: Locking Down the Processor via Rowhammer Attack. In *SysTEX'17: 2nd Workshop on System Software for Trusted Execution*, October 28, 2017, Shanghai, China. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3152701.3152709>

### 1 Introduction

Trusted Execution Environments (TEEs) enable secure computation of sensitive programs without relying on the underlying hardware as an operating system component. Intel's Software Guard Extensions (SGX) [18] is a commodity hardware-based TEE imple-

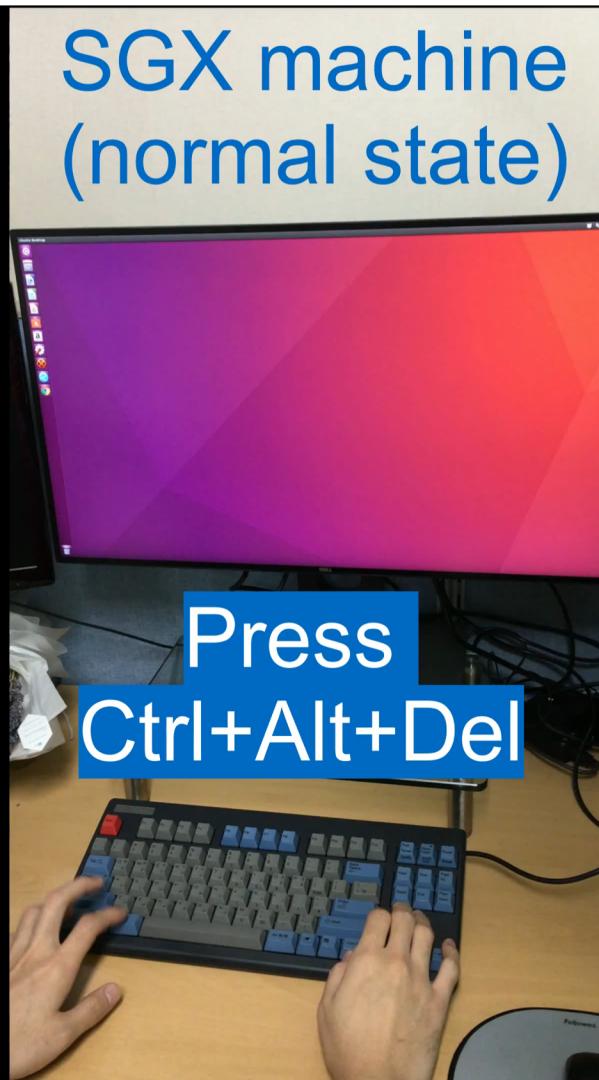
# About Integrity Violation

- SGX assumes HW/physical attackers
- Integrity violation → drop-and-lock policy
- Implications:
  - DoS: Freezing an entire machine (cloud provider)
  - Require power recycle (not via normal methods)

# SGX-Bomb Remarks

- Easier to trigger than normal rowhammer  
i.e., a single, arbitrary bit in EPC region (128MB)
- Harder to detect
  - Not notifiable in terms of resource usages
  - Popular defenses (e.g., in Linux) rely on PMU (e.g., cache misses) that is not possible for enclaves

# DEMO: SGX-Bomb



# Defenses against SGX-Bomb

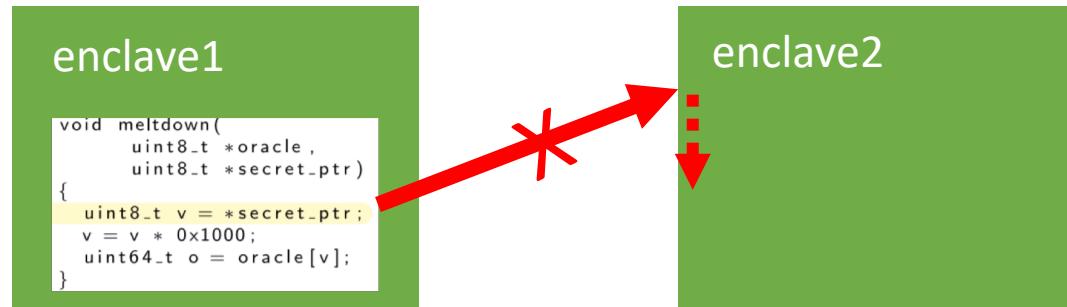
- Use non-faulty DRAM!
- Use LPDDR3 that has Pseudo-TRR (Target Row Refresh)
  - ECC can't completely block (easy to trigger multiple bits)
- Potential mitigations:
  - Higher refresh rate (2x)
  - Using Uncore PMU
  - Row-aware memory allocation for EPC regions

# New Attack Vectors

- Page table attack
  - e.g., leaking image data
- Branch shadowing attack
  - e.g., breaking RSA
- Rowhammer against SGX
  - e.g., freezing machines
- L1 terminal fault against SGX (i.e., Foreshadow)
  - e.g., breaking SGX ecosystem (and more!)

# L1TF: L1 Terminal Fault

*Not present & L1D*



## FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution

Jo Van Bulck<sup>1</sup>, Marina Minkin<sup>2</sup>, Ofir Weisse<sup>3</sup>, Daniel Genkin<sup>3</sup>, Baris Kasikci<sup>3</sup>, Frank Piessens<sup>1</sup>, Mark Silberstein<sup>2</sup>, Thomas F. Wenisch<sup>3</sup>, Yuval Yarom<sup>4</sup>, and Raoul Strackx<sup>1</sup>

<sup>1</sup>imec-DistriNet, KU Leuven, <sup>2</sup>Technion, <sup>3</sup>University of Michigan, <sup>4</sup>University of Adelaide and Data61

### Abstract

Trusted execution environments, and particularly the Software Guard eXtensions (SGX) included in recent Intel x86 processors, gained significant traction in recent years. A long track of research papers, and increasingly also real-world industry applications, take advantage of the strong hardware-enforced confidentiality and integrity guaran-

distrusting enclaves with a minimal Trusted Computing Base (TCB) that includes only the processor package and microcode. Enclaves mitigate CDEs and memory leaks is exclusively software-based. This makes SGX a potentially vulnerable target for attacks. Besides strong memory isolation, TEEs typically offer an

SEC'18

## Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution

Revision 1.0 (August 14, 2018)

Ofir Weisse<sup>3</sup>, Jo Van Bulck<sup>1</sup>, Marina Minkin<sup>2</sup>, Daniel Genkin<sup>3</sup>, Baris Kasikci<sup>3</sup>, Frank Piessens<sup>1</sup>, Mark Silberstein<sup>2</sup>, Raoul Strackx<sup>1</sup>, Thomas F. Wenisch<sup>3</sup>, and Yuval Yarom<sup>4</sup>

<sup>1</sup>imec-DistriNet, KU Leuven, <sup>2</sup>Technion, <sup>3</sup>University of Michigan, <sup>4</sup>University of Adelaide and Data61

### Abstract

In January 2018, we discovered the *Foreshadow* transient execution attack (USENIX Security'18) targeting Intel SGX technology. Intel's subsequent investigation of our attack uncovered two closely related variants, which we collectively call *Foreshadow-NG* and which Intel refers to as L1 Terminal Fault. Current analyses focus mostly on mitigation strategies, providing only limited insight into the attacks themselves and their consequences. The

tion requires different computational tasks belonging to separate security domains to be isolated from each other and prevented from reading each other's memory. In modern computer architectures this is typically achieved via hardware mechanisms such as memory protection, page tables, and address translation. These mechanisms are designed for the convenience of simulating a memory space much

ArXiv'18

# Impacts of L1TF on SGX

- Broken isolation guarantees
- Distrustful remote attestation, thus ecosystem
  - Leaking secrets from architectural enclaves (e.g., quoting/launching)
  - Emulator vs. SGX

# Defense: L1TF against SGX

- Immediate steps (via microcode update):
  - Flushing L1 on EEXIT/AEX
  - Disabling hyperthreading
- Q. What should we do to address this issue more fundamentally?
- Q. What's the right way to prevent further issues?

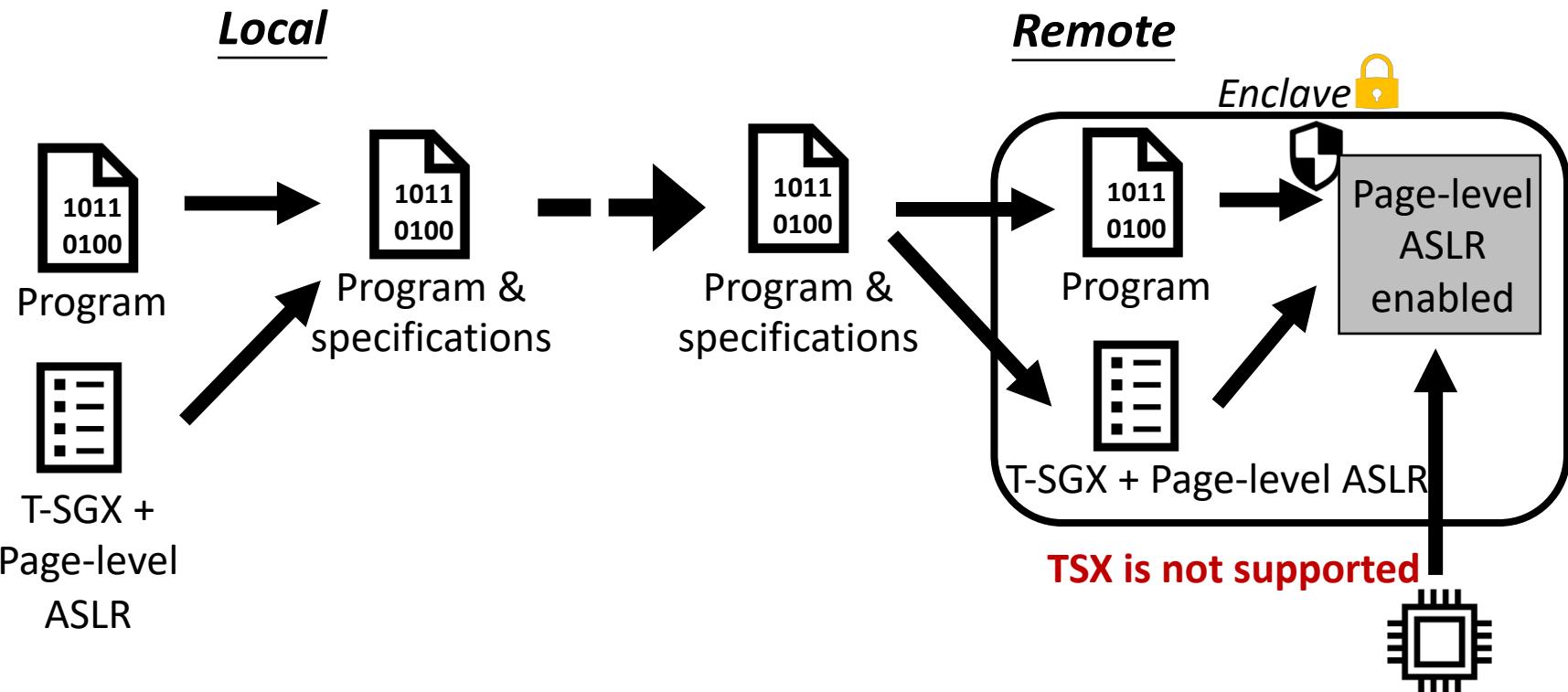
# Outline

- Threat model / assumption
- Traditional attack vectors
- New attack vectors
- On-going approaches
- Summary

# On-going Projects for Defenses (collaborating with MS and Intel)

- 1) Multifaceted side-channel attack (under review)
- 2) Hardware-based fault isolation (on-going)
  - Seeking a better HW abstraction to contain faults  
(i.e., ideal interface to replace ad-hoc TSX)
- 3) Loading-time synthesis (on-going)
  - Addressing side-channel at loading time, depending on  
the execution environment at end points  
(i.e., compositing SW-based schemes without conflicts)

# PRIDWIN: Load-time Synthesis



# PRIDWIN: Load-time Synthesis

## *Specifications & Constraints*

### T-SGX

**Target:** Page-table attacks

**Priority:** High

**Requirement:** TSX

**Instrumentation:**

- Insert XBEGIN, XEND at each block

### Page-level ASLR

**Target:** Page-table attacks

**Priority:** Low

**Requirement:** N/A

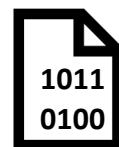
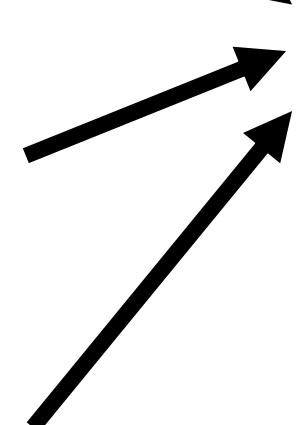
**Instrumentation:**

- Break program into 4-KB pages

### Hardware configuration

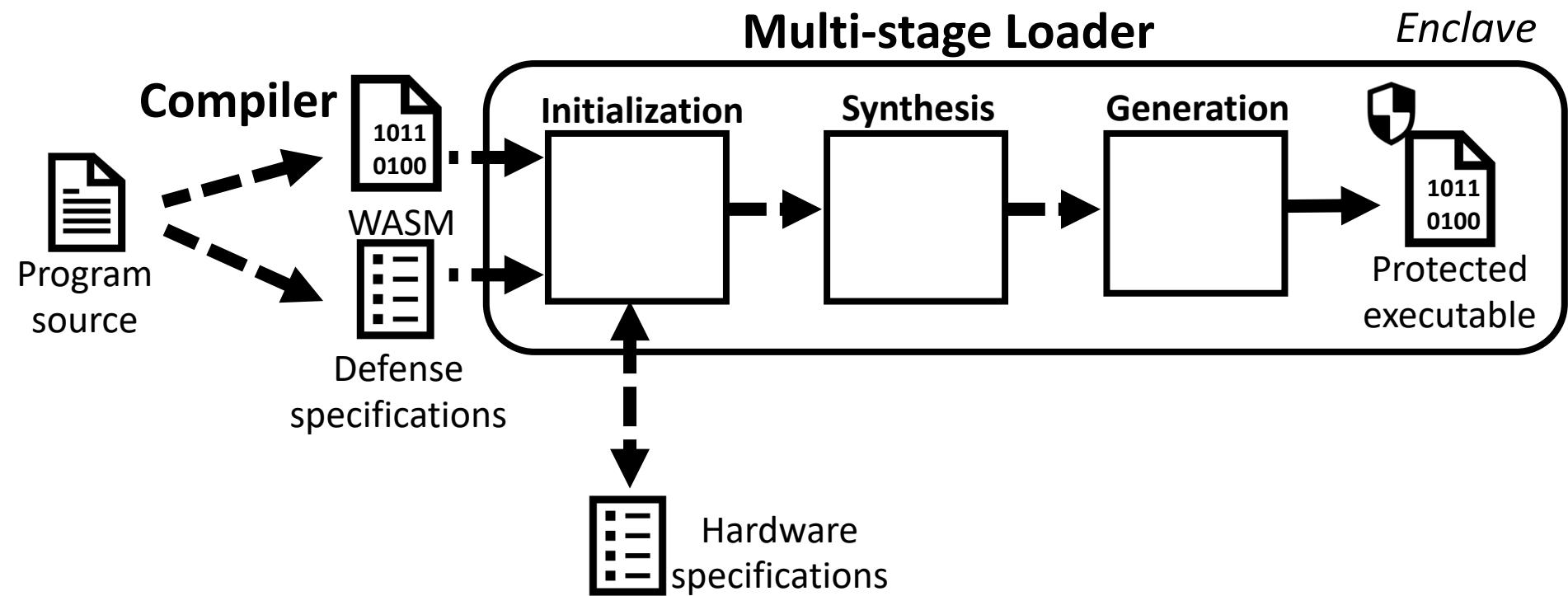
**TSX support:** No

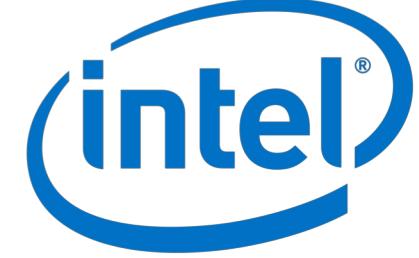
## *Load-time synthesis*



Program +  
Page-level  
ASLR

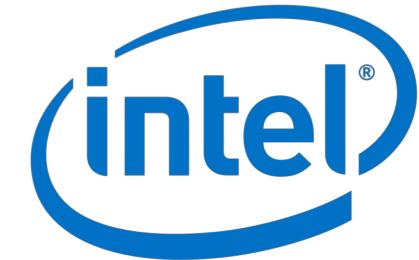
# PRIDWIN: Load-time Synthesis





# Summary

- Intel SGX is a practical, promising building block to write a secure program
- Intel SGX has unusually strong threat model, opening up unexpected attacks
- Today's Talk: Recent Attack/Defense of Intel SGX



# Summary

- It's not future technology; it's already everywhere!

**Microsoft Azure**

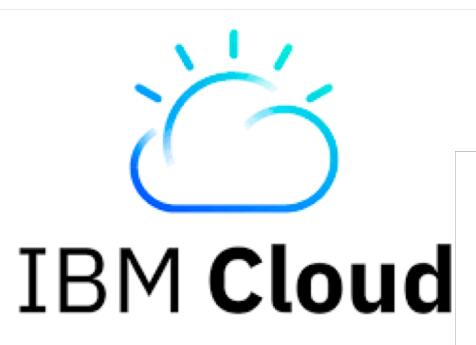
Why Azure Solutions Products Documentation Pricing Training Marketplace

Blog > Virtual Machines

## Introducing Azure confidential computing

Posted on September 14, 2017

 Mark Russinovich, CTO, Microsoft Azure



**OASIS LABS**

HOME DEVELOPERS TEAM  
BLOG PRESS WE'RE HIRING

## INTRODUCING OASIS LABS

Creating a privacy-first cloud computing platform on blockchain.

Join our private testnet

APPLY NOW

**Fortanix**

MEET THE TEAM

We put software and hardware security into billions of devices. 100+ security patents. 30+ papers in top conferences. USENIX, CCC, and Blackhat presenters.

Ready to test Fortanix SDKMS beta? REQUEST A DEMO

