GEORGIA TECH INFORMATION SECURITY CENTER

Safeguarding Digital Infomation Through Innovative Research and Education

Attacks and Defenses for Intel SGX

Taesoo Kim

Georgia Institute of Technology

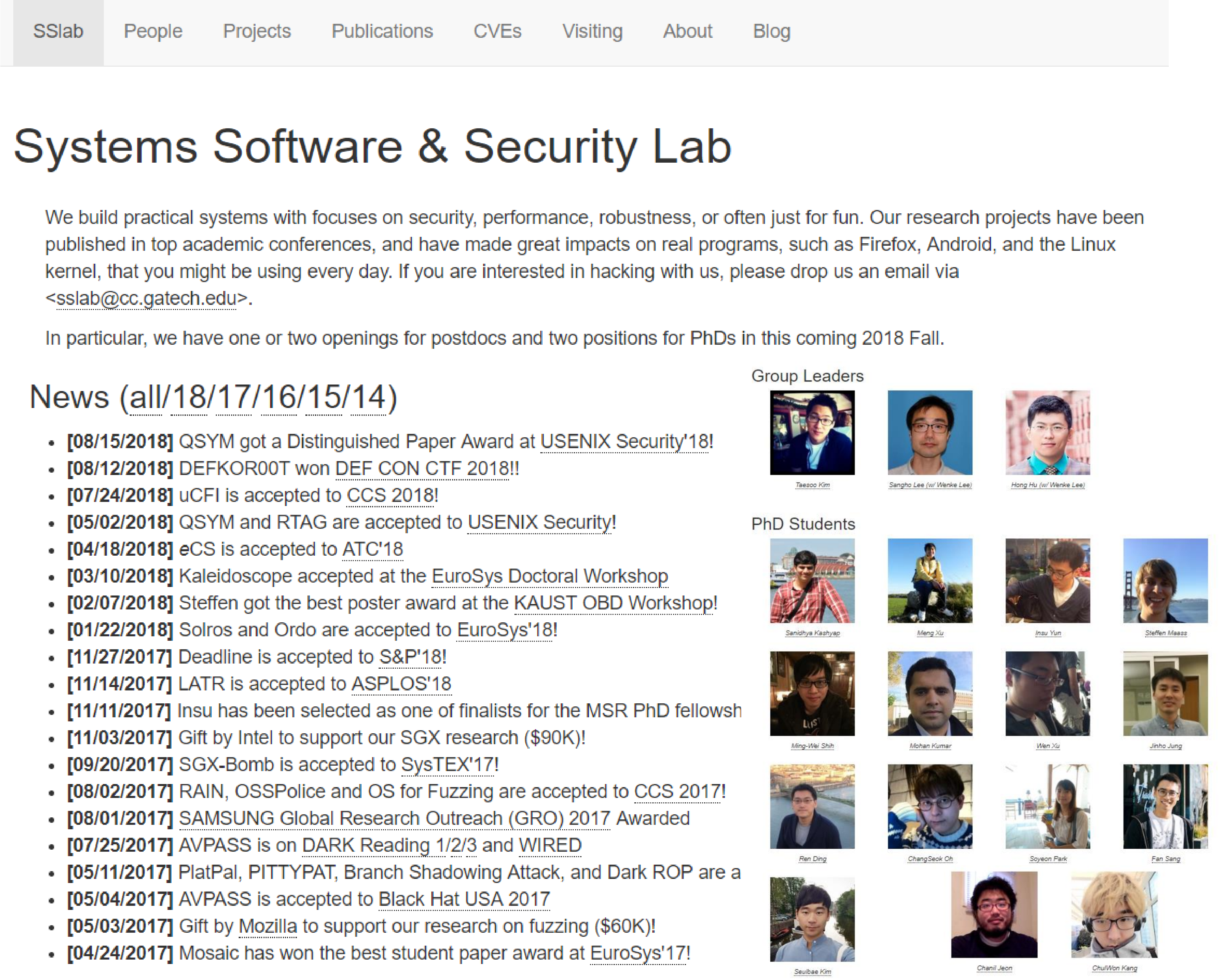
About Myself

- 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 <sslab@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]** DEFKOR00T 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 fellowsh
- **[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 a
- **[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

PhD Students

Our Group’s Research Interests

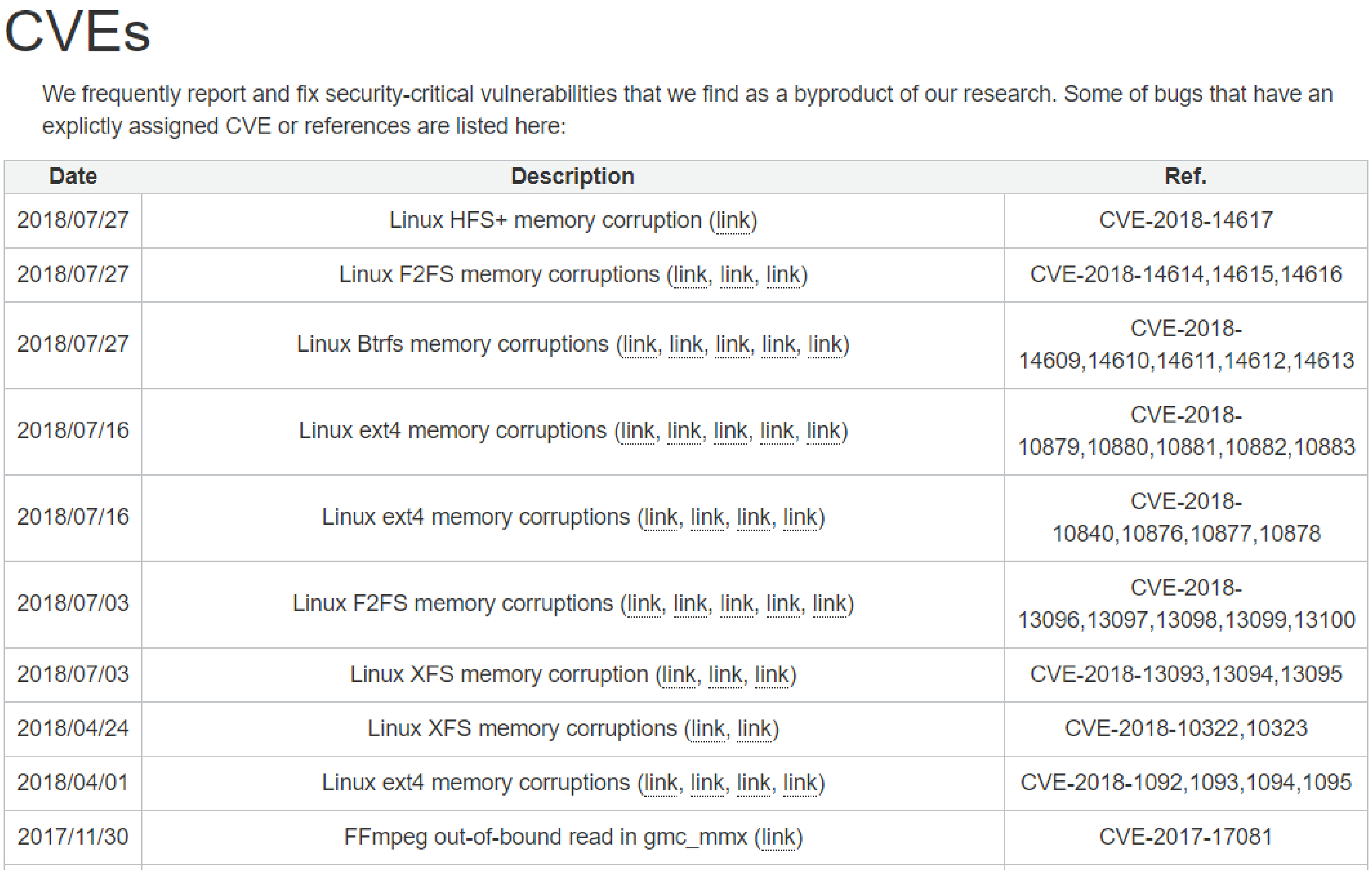
- **Bug finding:**
  e.g., static analysis, fuzzing, symbolic execution, etc.

- **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 explictly assigned CVE or references are listed here:

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

https://gts3.org/pages/cves.html

DEFKOR00T: Won DEF CON CTF'18 (DEFKOR + R00tmentary)

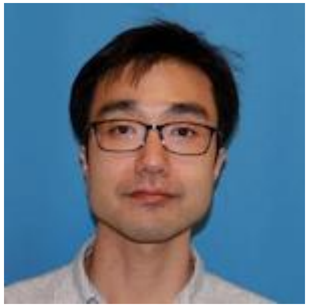
(R00tmentary)

Attacks and Defenses for Intel SGX

Taesoo Kim

Georgia Institute of Technology

The Team



GT

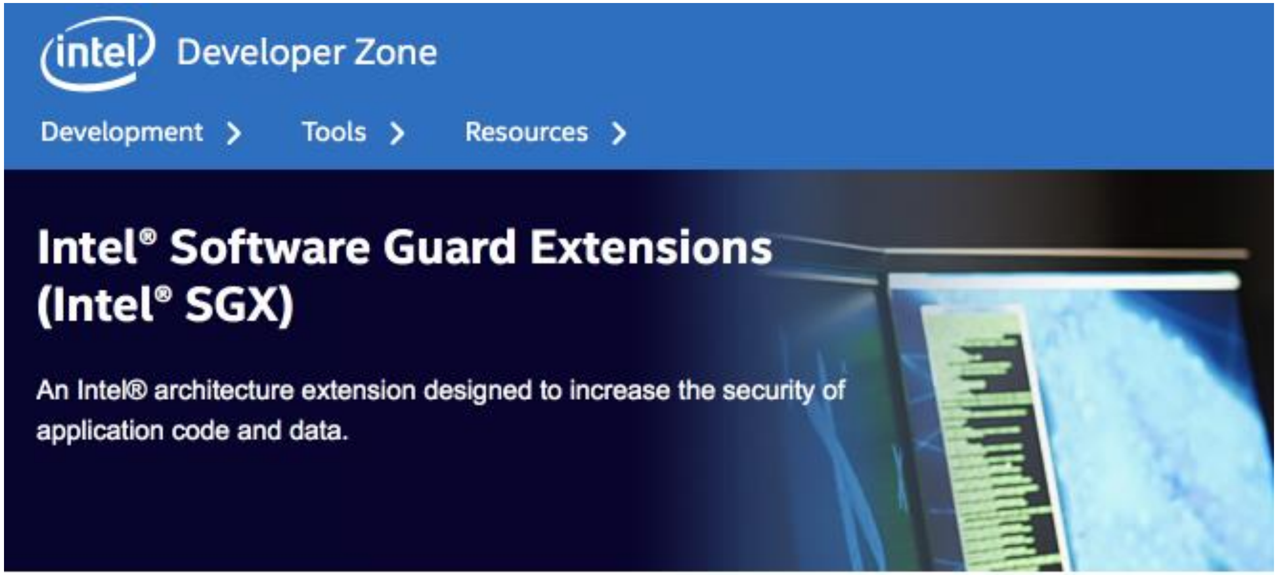
Oregon State University

Microsoft Research

KAIST

Disclaimer

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



Intel® Software Guard Extensions (Intel® SGX)

An Intel® architecture extension designed to increase the security of application code and data.

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

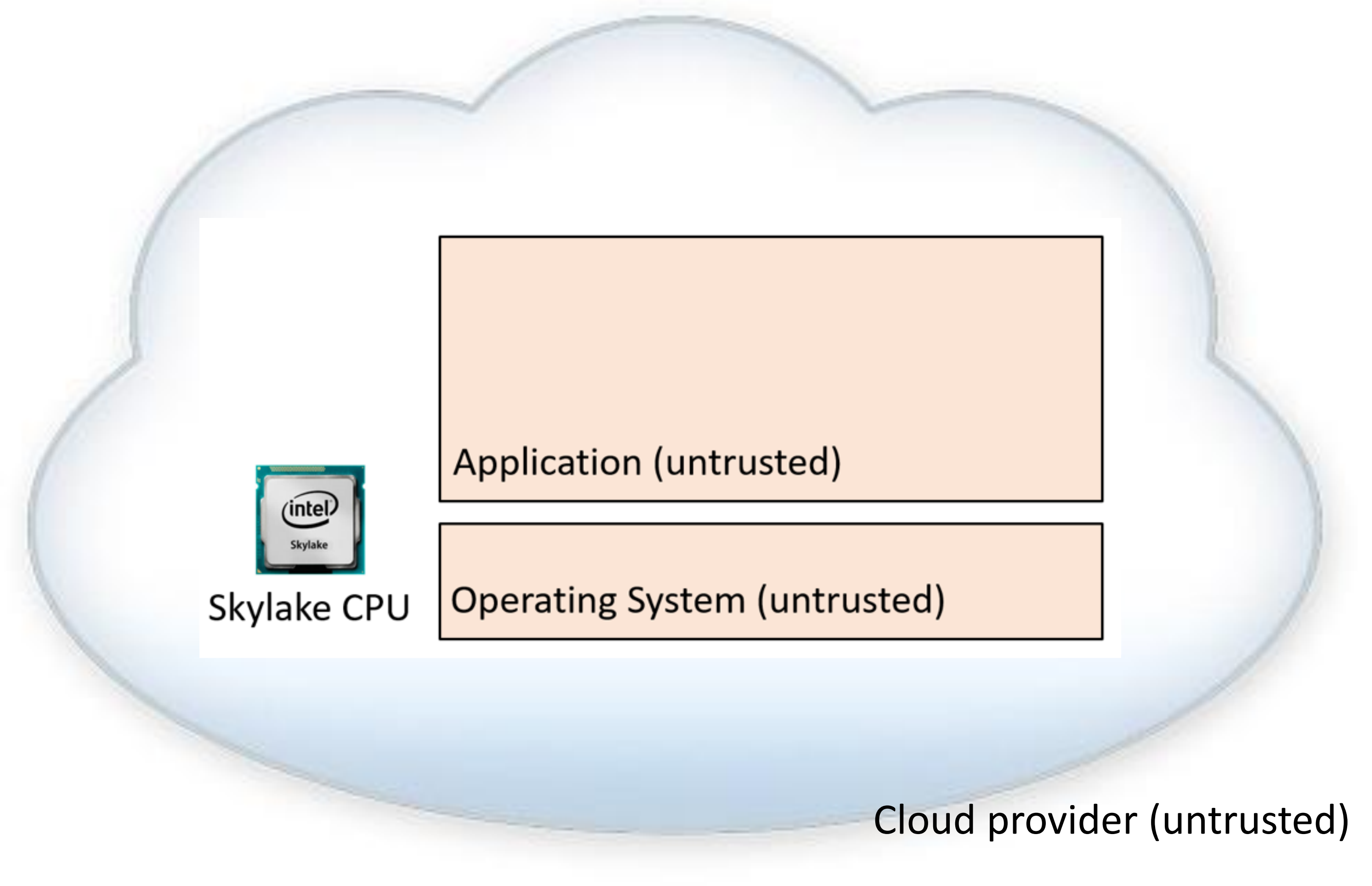
- **“Practical”** TEE implementation by Intel
- Extending x86 Instruction Set Architecture (ISA)
  - Native performance
  - Compatible to x86
  - Commodity (i.e., cheap)

Lenovo T560

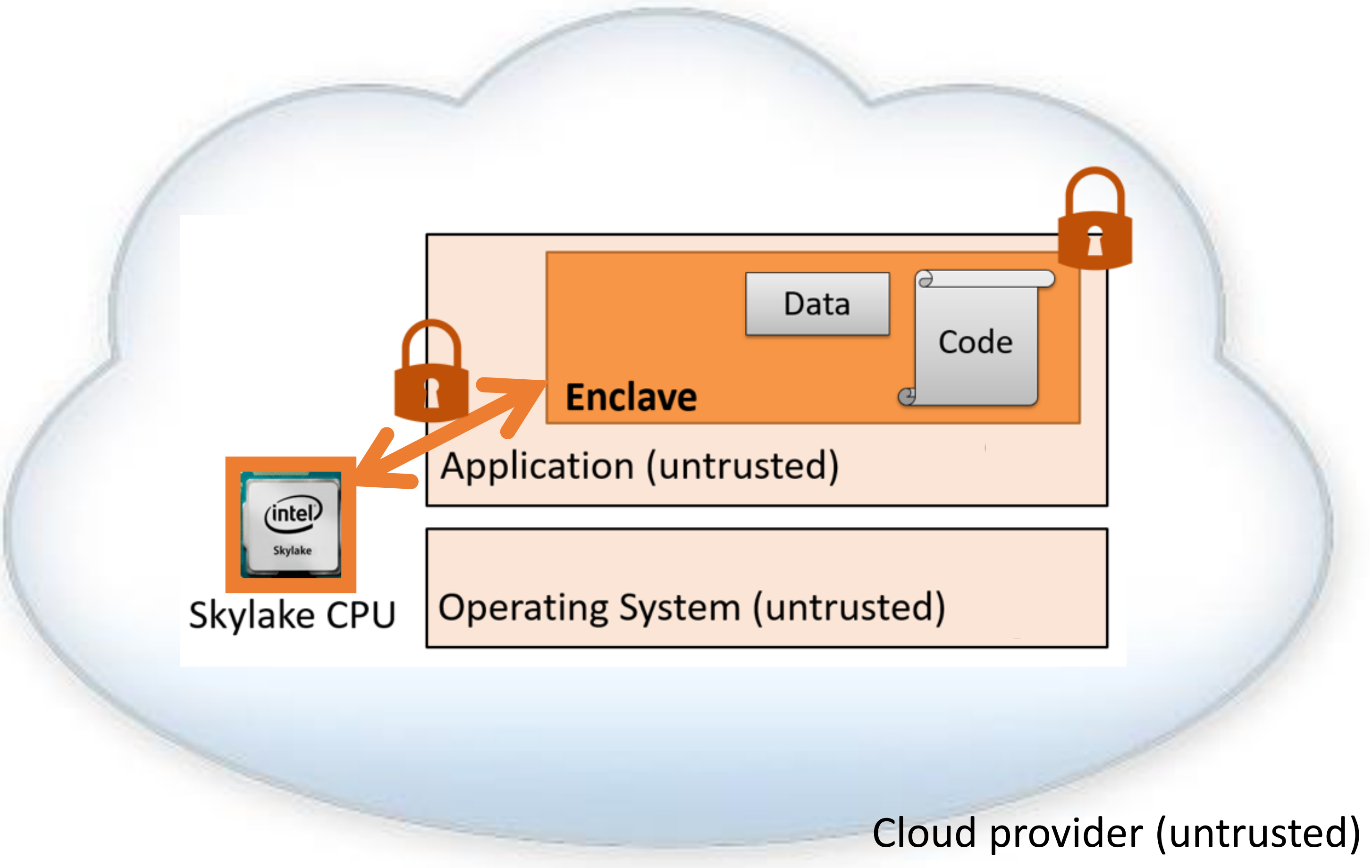
Dell OptiPlex 5040

Supermicro Server

Revisited: SGX for Cloud

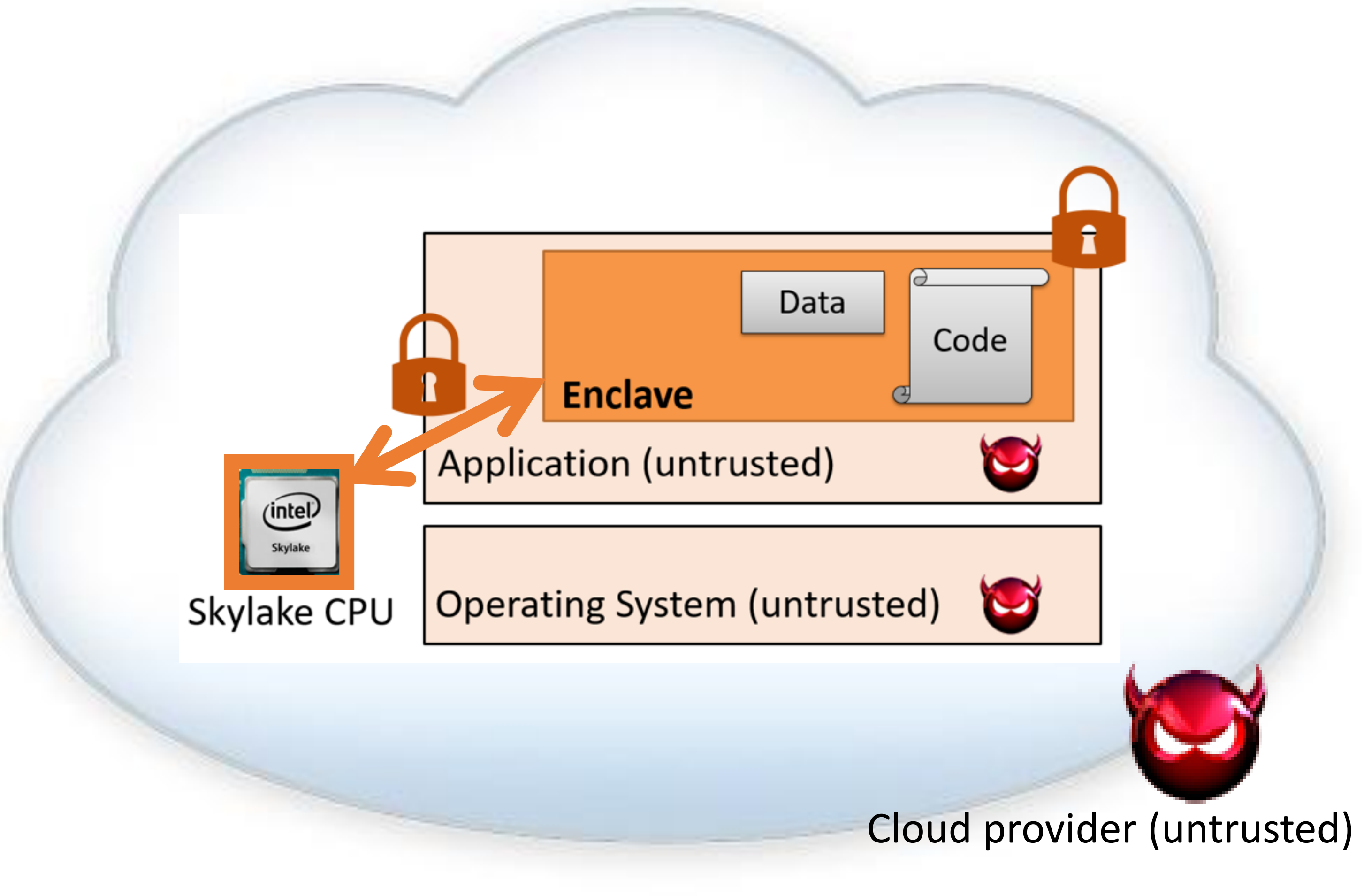


Revisited: SGX for Cloud

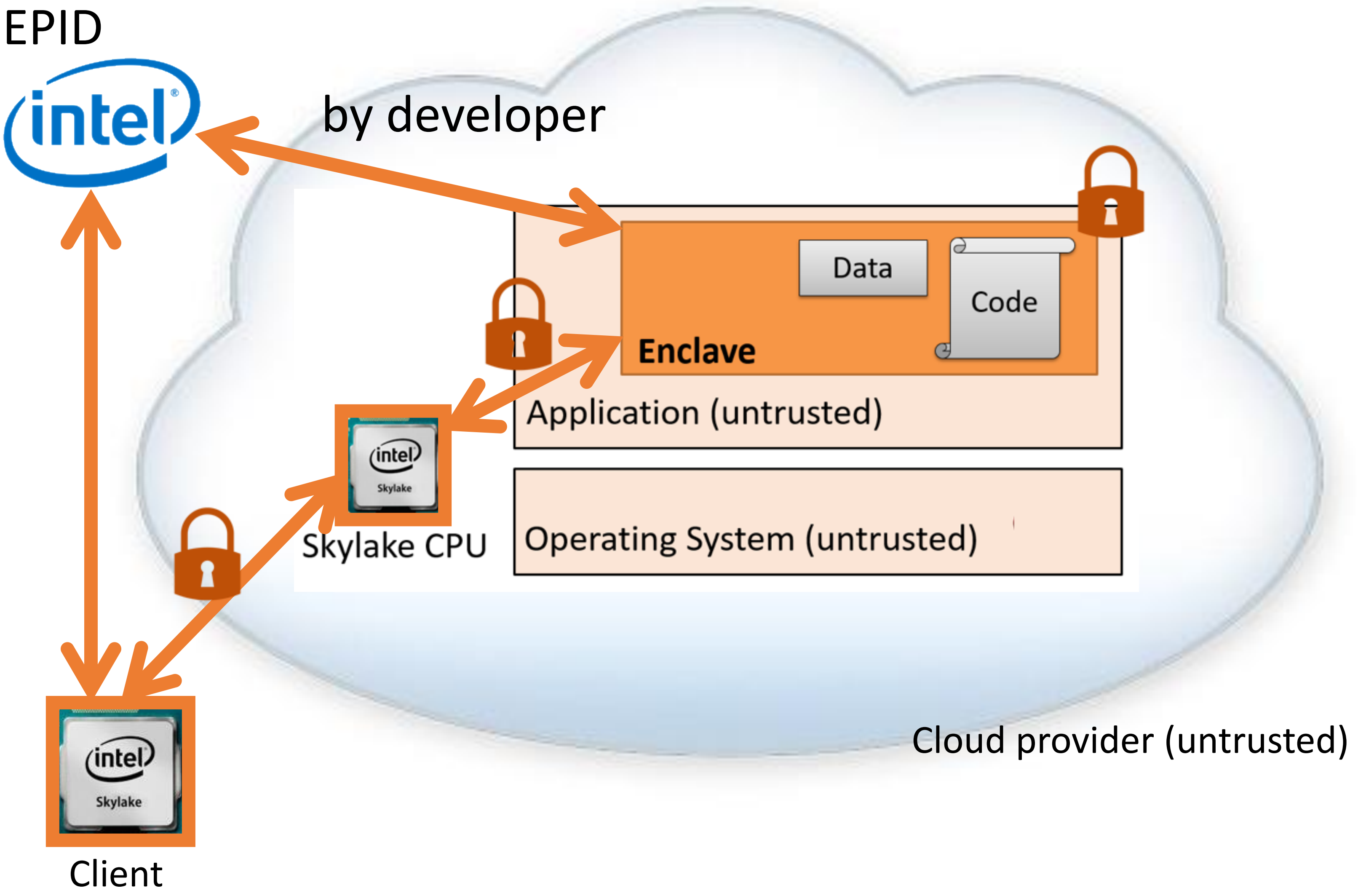


Cloud provider (untrusted)

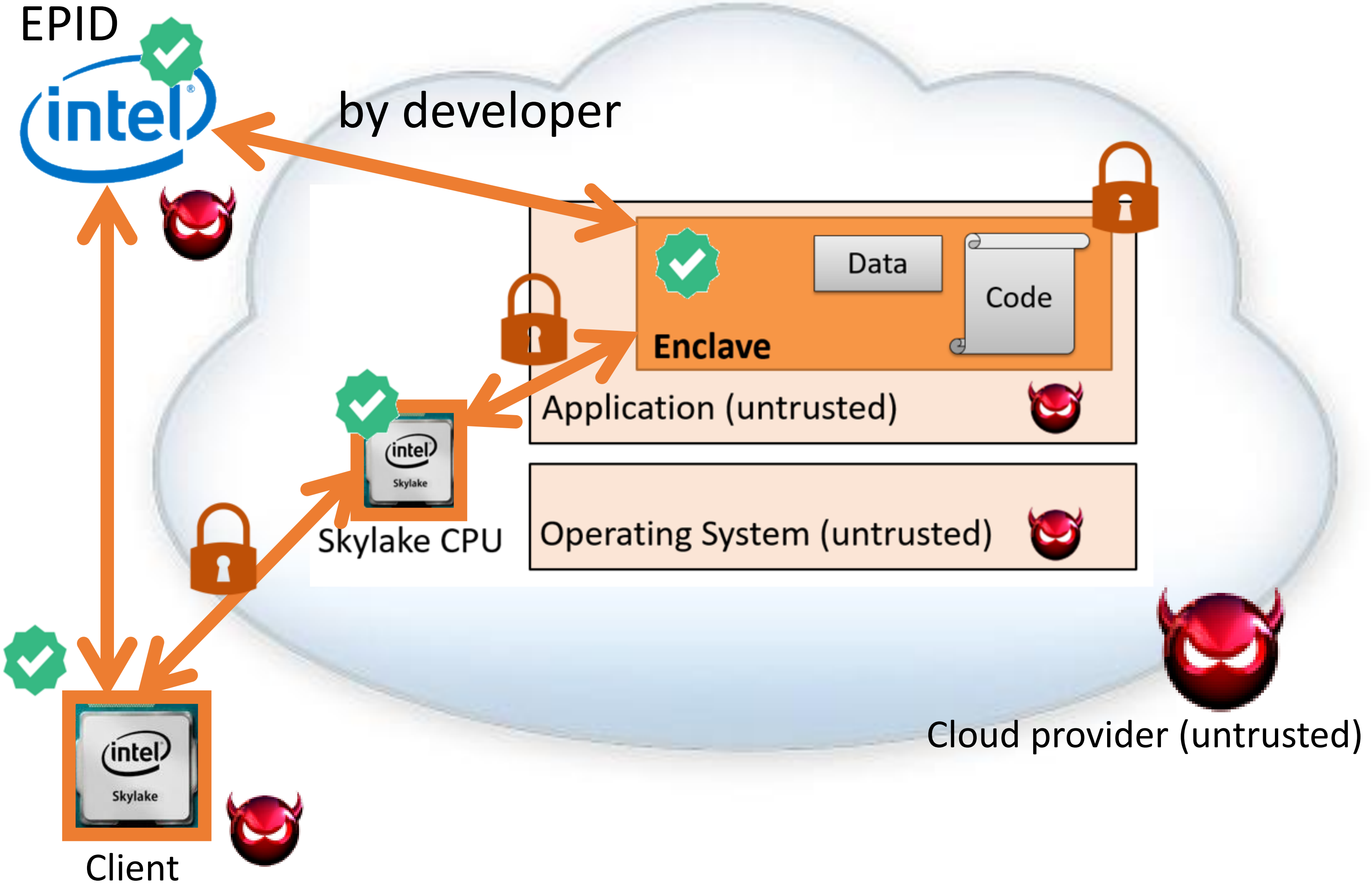
Revisited: SGX for Cloud (Isolation)



Revisited: SGX for Cloud (Remote attestation)



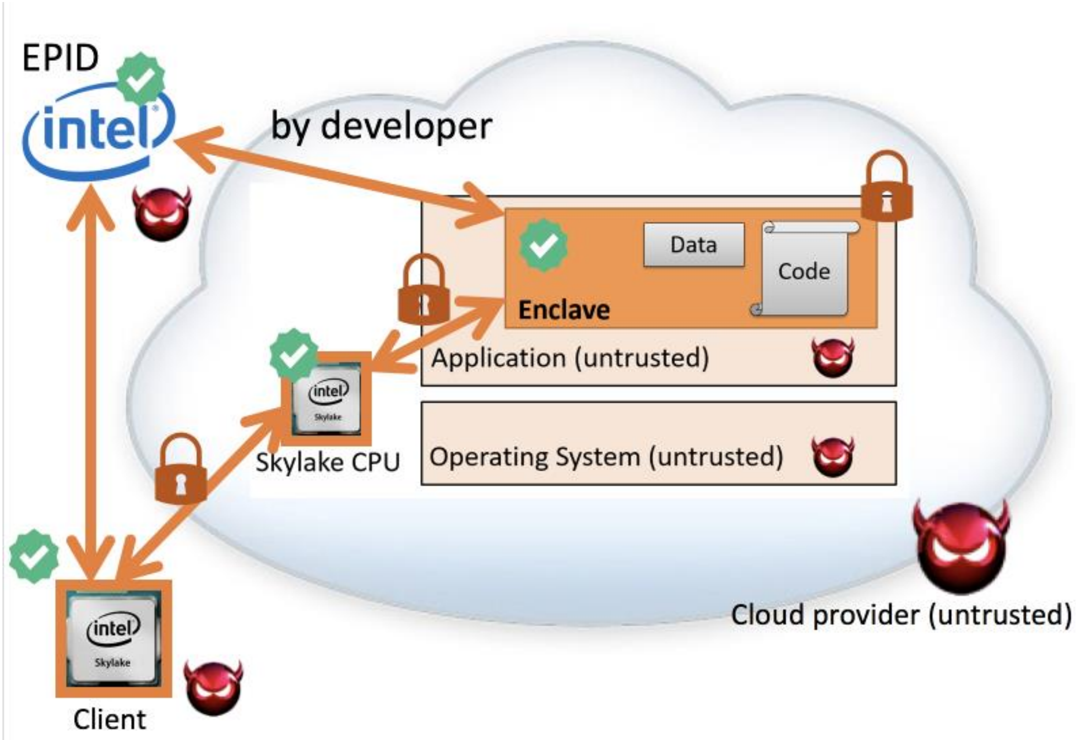
Revisited: SGX for Cloud (Remote attestation)



SGX Ecosystem for Attackers

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

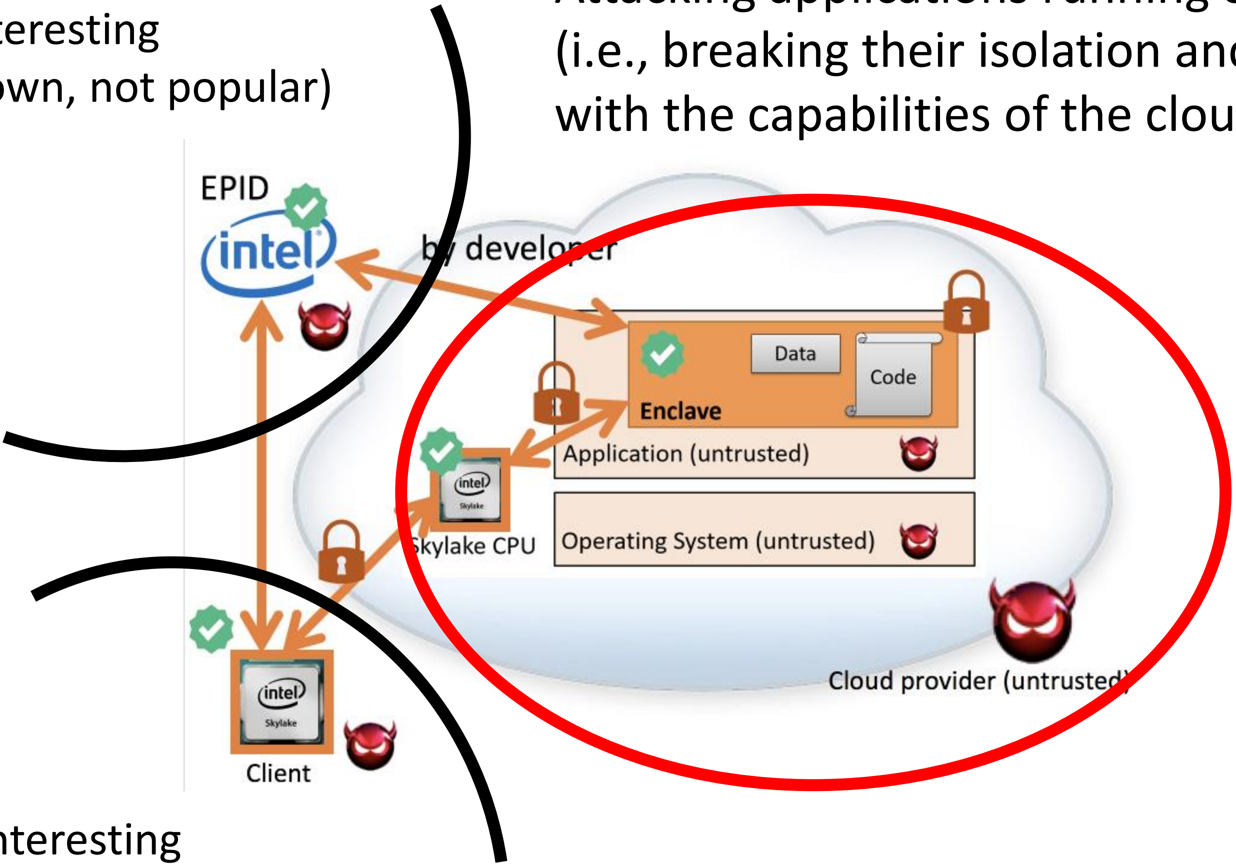
😈 : Attacker's capabilities (i.e., what attackers can do)



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)

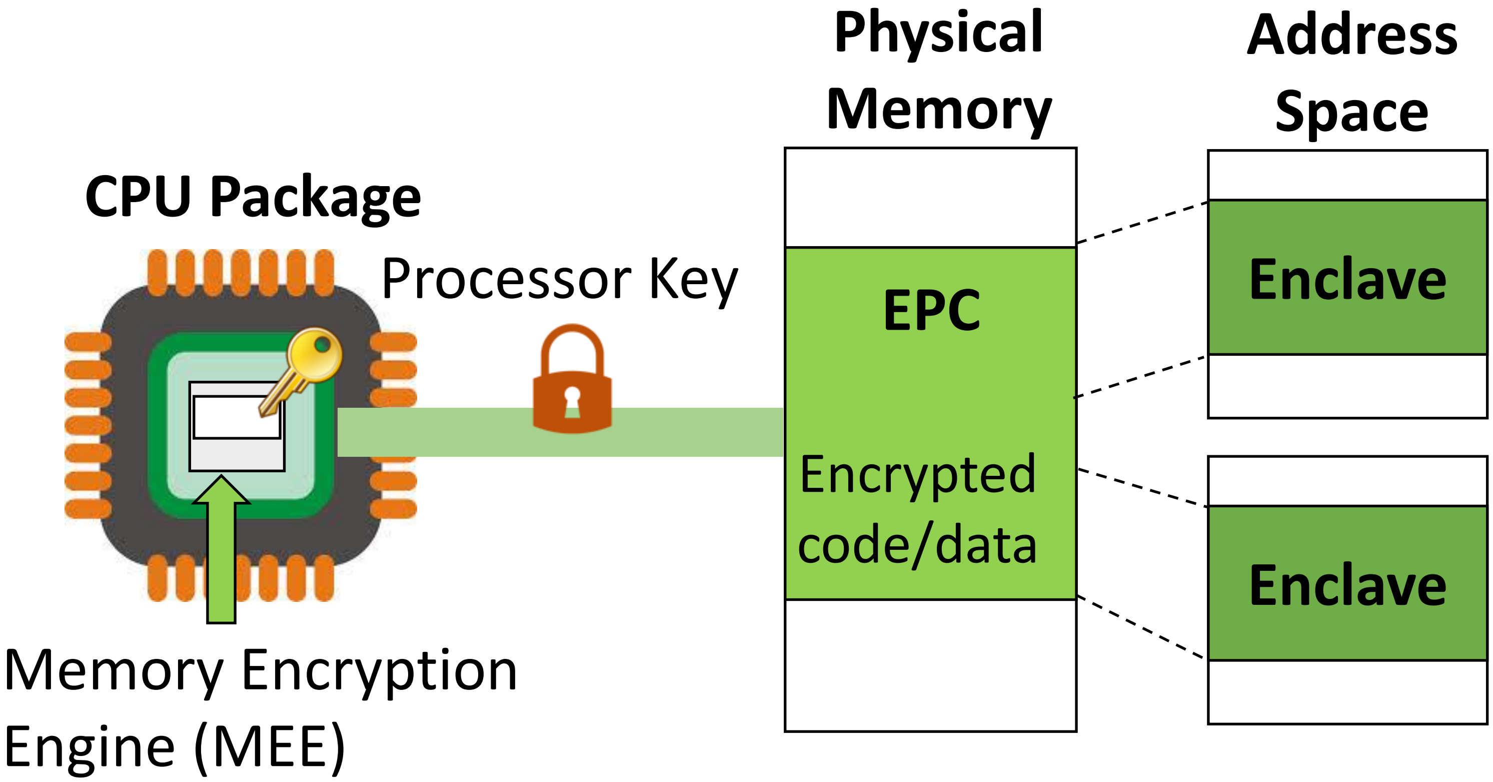
Summary: Intel SGX 101

- 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

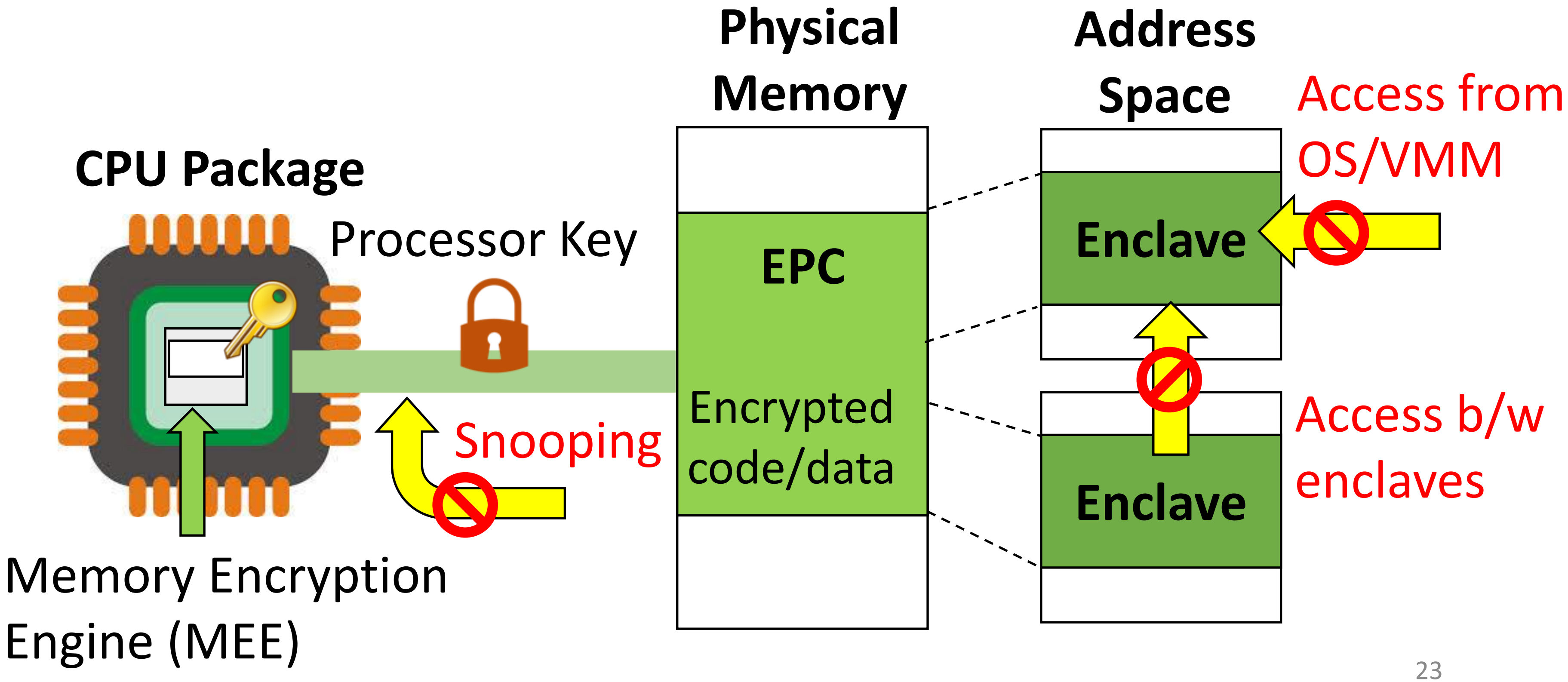
Isolated Execution

- 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)



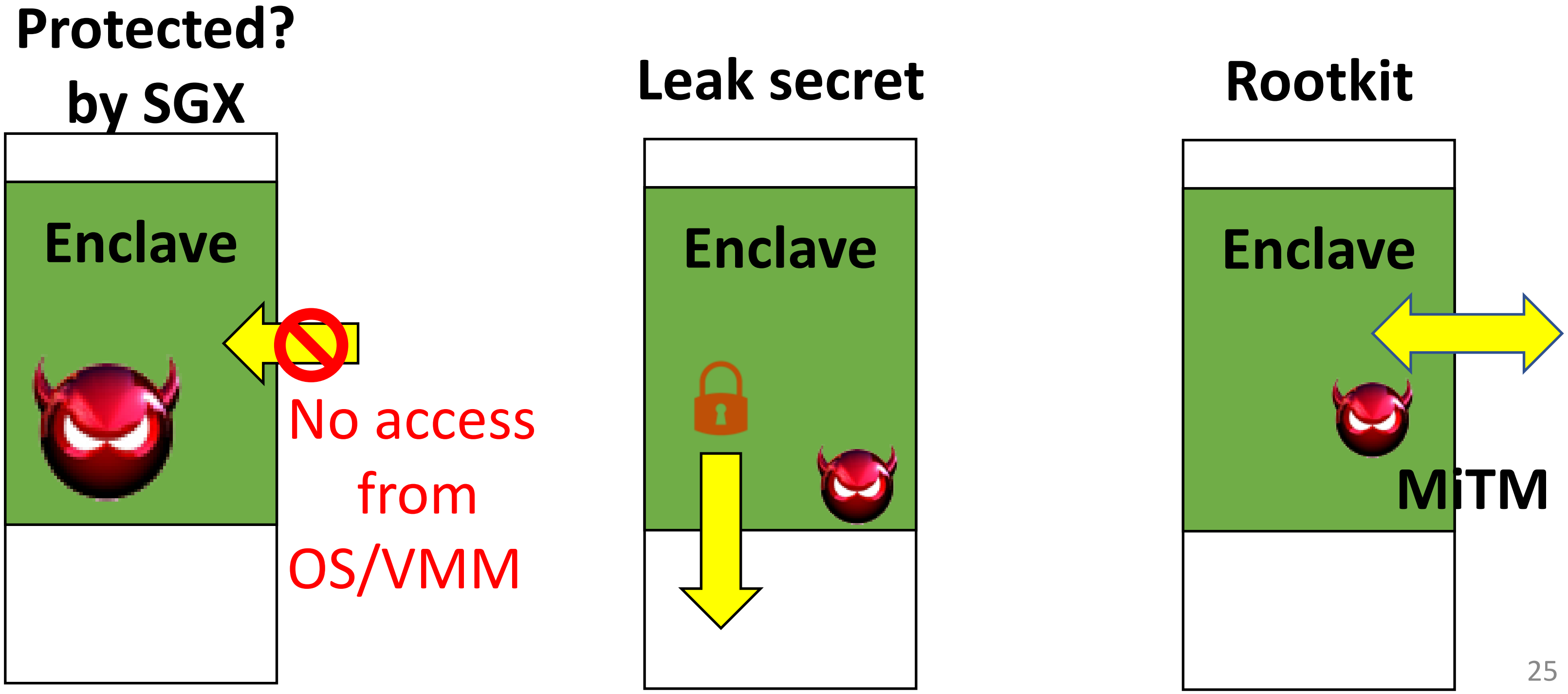
SGX’s Threat Model (very strong!)

- *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

Enclave

No access from OS/VMM

Leak secret

Enclave

Rootkit

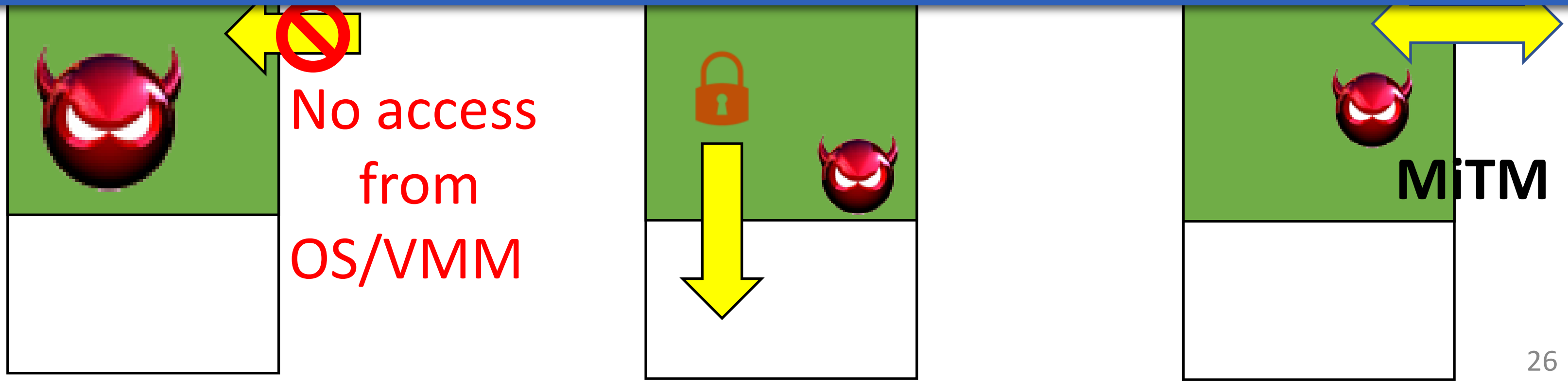
Enclave

MiTM

What if Enclave is Compromised?

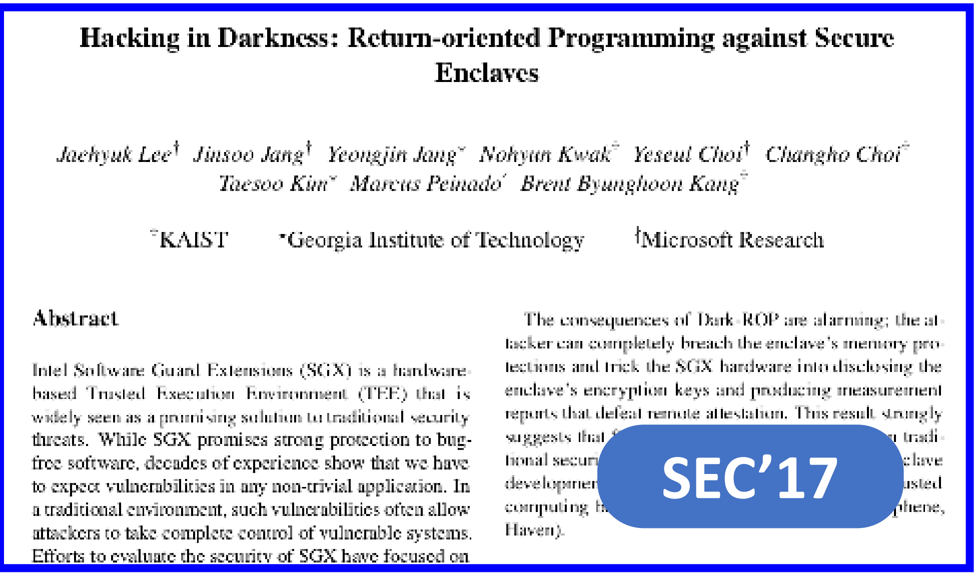
- Leak sensitive information

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[†] *Jinsoo Jang*[†] *Yeongjin Jang*[*] *Nohyun Kwak*[‡] *Yeseul Choi*[†] *Changho Choi*[‡] *Taesoo Kim*[*] *Marcus Peinado*[′] *Brent Byunghoon Kang*[‡]

[‡]KAIST [*]Georgia Institute of Technology [†]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 f … tradi-tional securi … clave developmen … usted computing h … phene, Haven).

SEC'17

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

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

[1]*imec-DistriNet, KU Leuven,* [2]*Technion,* [3]*University of Michigan,* [4]*University of Adelaide and Data61*

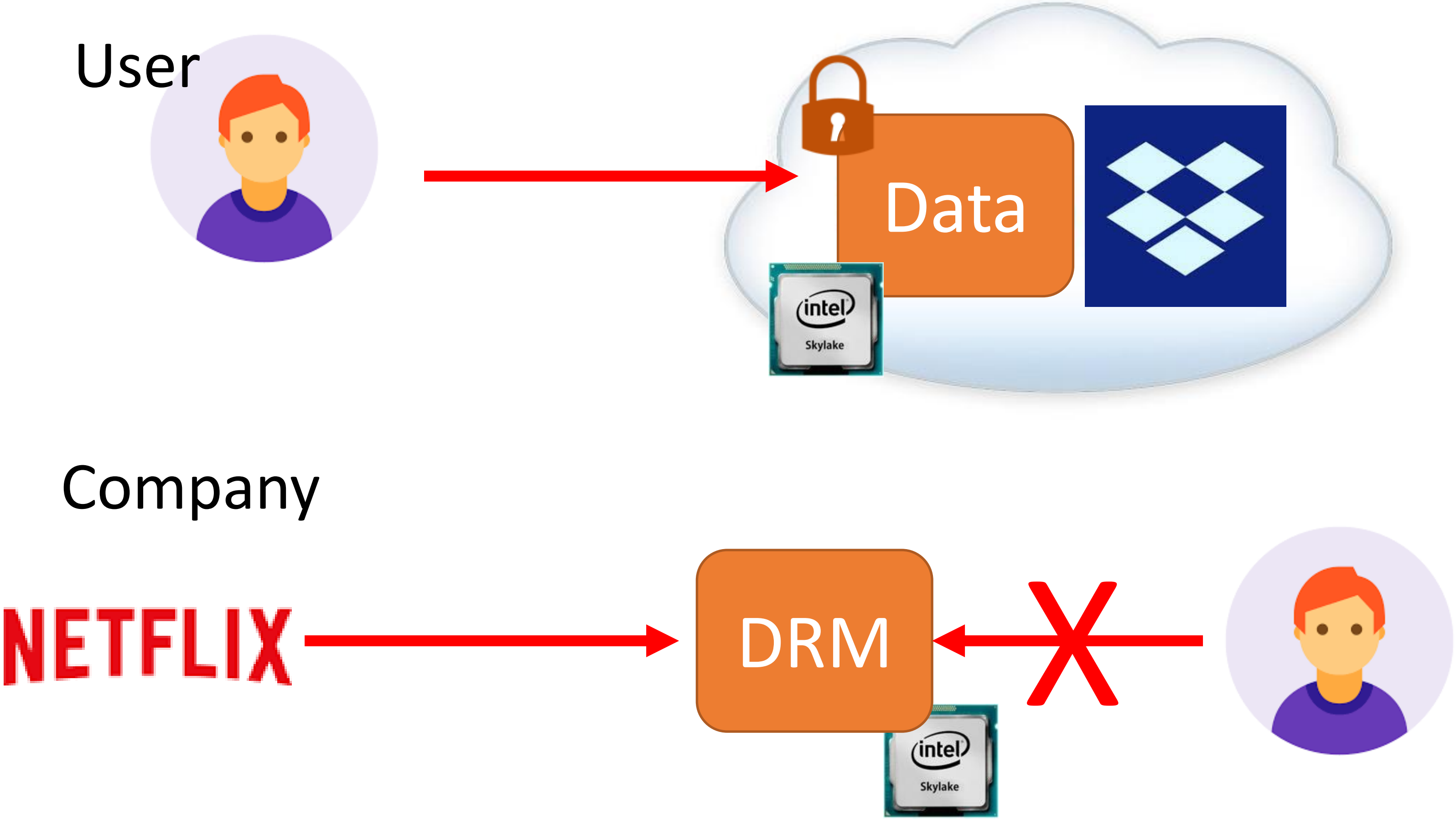
Abstract

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. 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 runni … po-tentially mali … or.

SEC'18

Thinking of SGX Usages



User

Data

Company

DRM

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

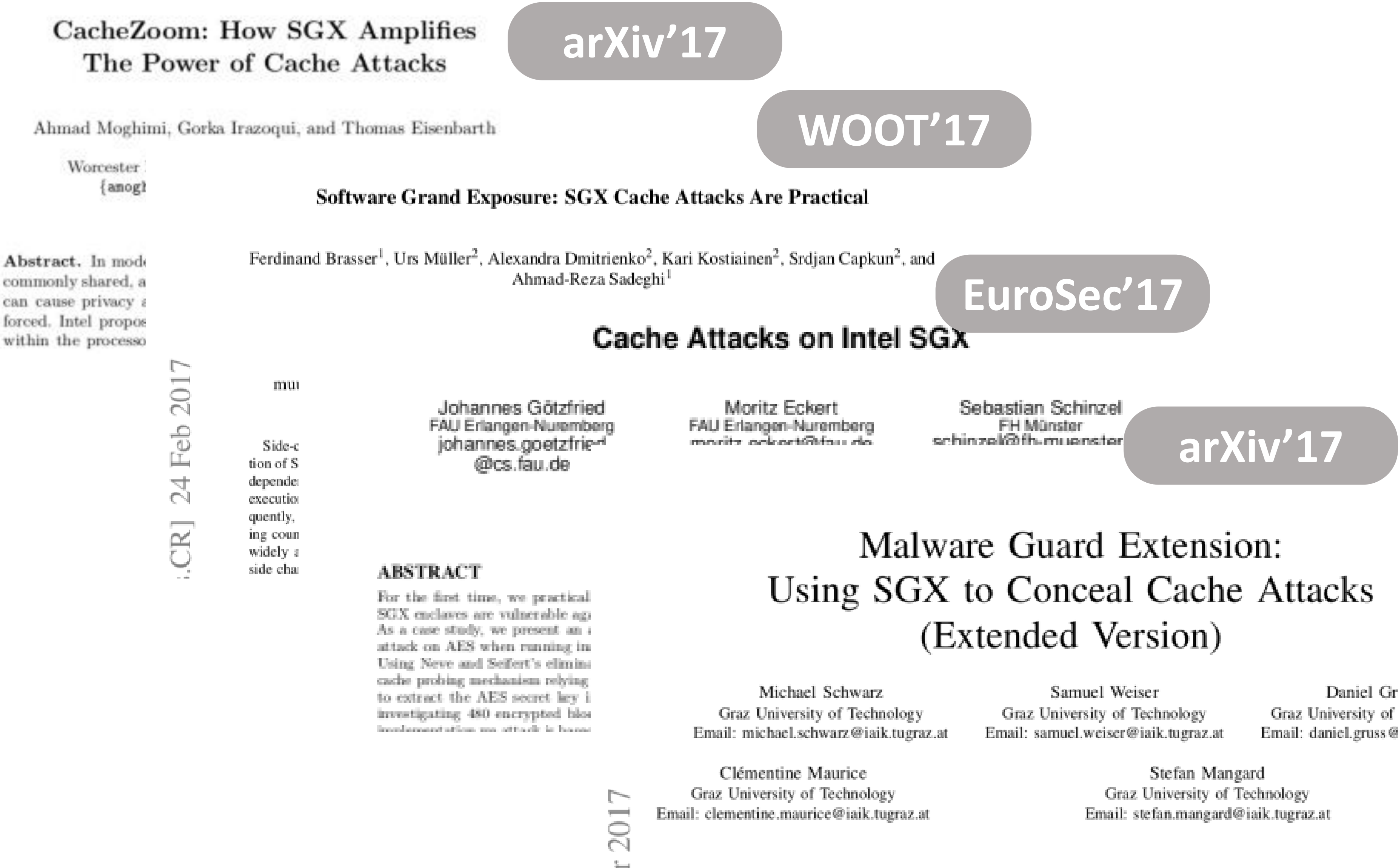
  → e.g., inferring a private key
- Memory safety

  → 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



arXiv'17

CacheZoom: How SGX Amplifies The Power of Cache Attacks

Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth

Worcester
{amogh

Abstract. In mod
commonly shared, a
can cause privacy a
forced. Intel propos
within the processo

:CR] 24 Feb 2017

WOOT'17

Software Grand Exposure: SGX Cache Attacks Are Practical

Ferdinand Brasser[1], Urs Müller[2], Alexandra Dmitrienko[2], Kari Kostiainen[2], Srdjan Capkun[2], and Ahmad-Reza Sadeghi[1]

mu

Side-c
tion of S
depende
executio
quently,
ing coun
widely a
side cha

EuroSec'17

Cache Attacks on Intel SGX

Johannes Götzfried
FAU Erlangen-Nuremberg
johannes.goetzfried
@cs.fau.de

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

Sebastian Schinzel
FH Münster
schinzel@fh-muenster

ABSTRACT
For the first time, we practical
SGX enclaves are vulnerable ag
As a case study, we present an a
attack on AES when running in
Using Neve and Seifert's elimina
cache probing mechanism relying
to extract the AES secret key i
investigating 480 encrypted bloc
implementation we attack is based

arXiv'17

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

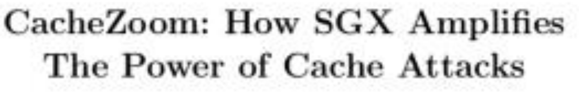
Daniel Gr
Graz University of
Email: daniel.gruss@

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

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

r 2017

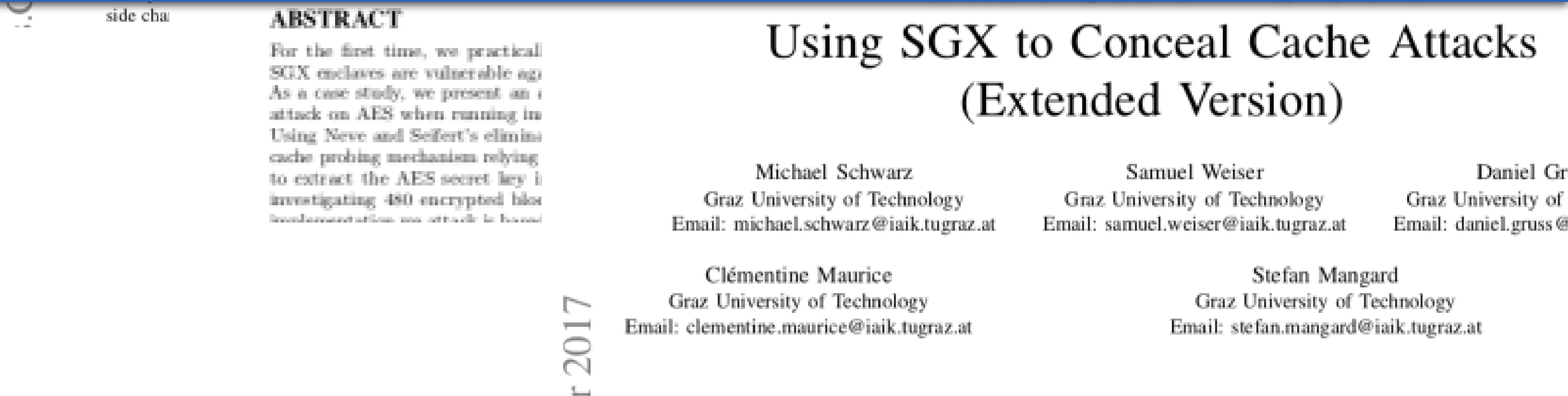
Cache-based Side-channel Attacks



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 ...)



side cha

ABSTRACT

For the first time, we practicall
SGX enclaves are vulnerable ag
As a case study, we present an
attack on AES when running in
Using Neve and Seifert's elimina
cache probing mechanism relying
to extract the AES secret key i
investigating 480 encrypted bloc

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

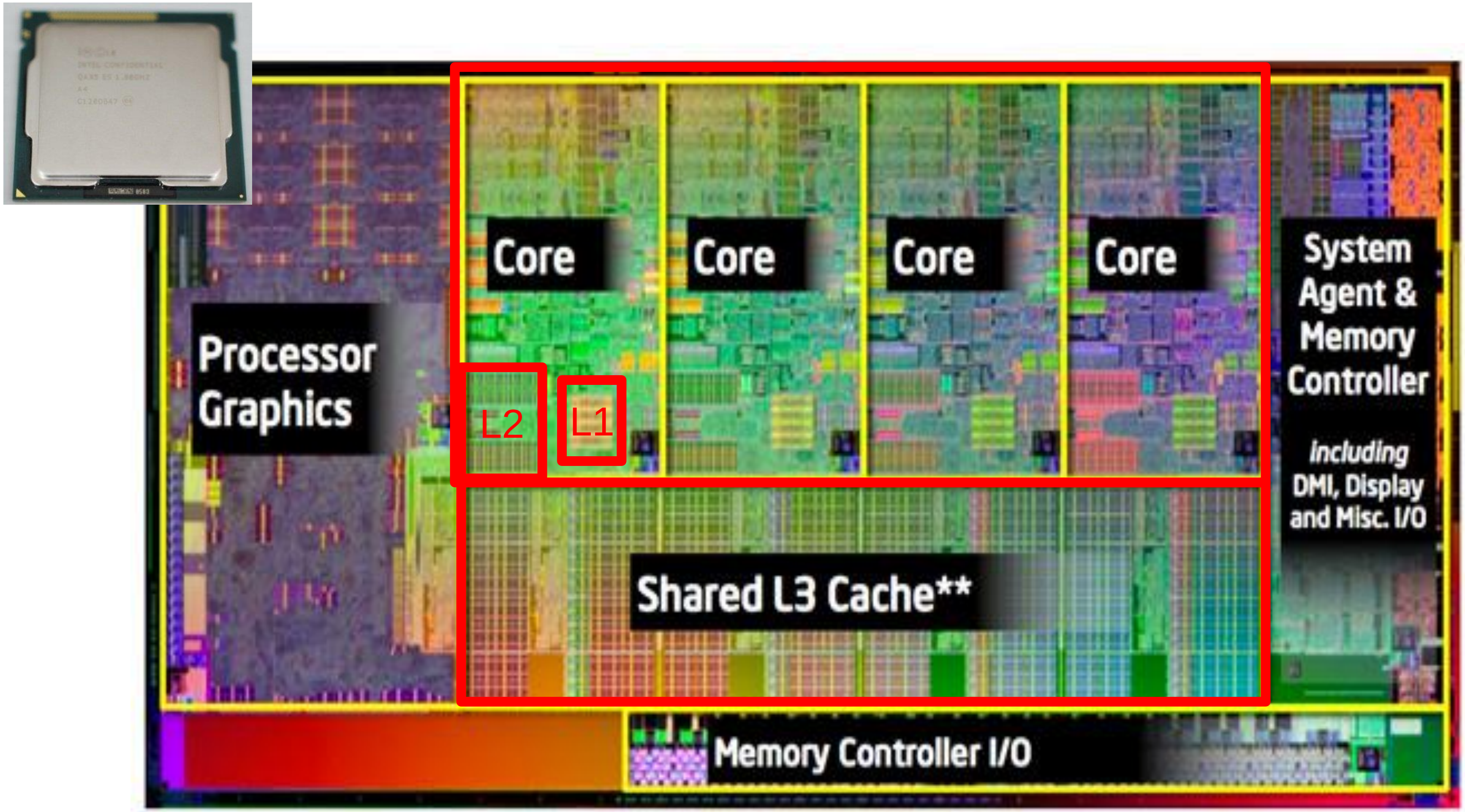
Daniel Gr
Graz University of
Email: daniel.gruss@

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

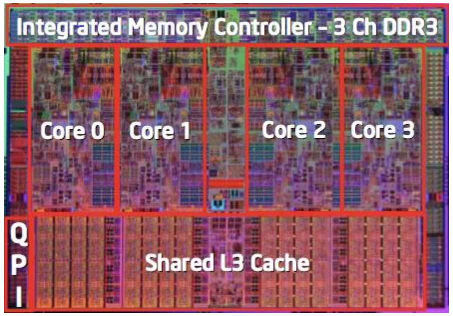
Stefan Mangard
Graz University of Technology
Email: stefan.mangard@iaik.tugraz.at

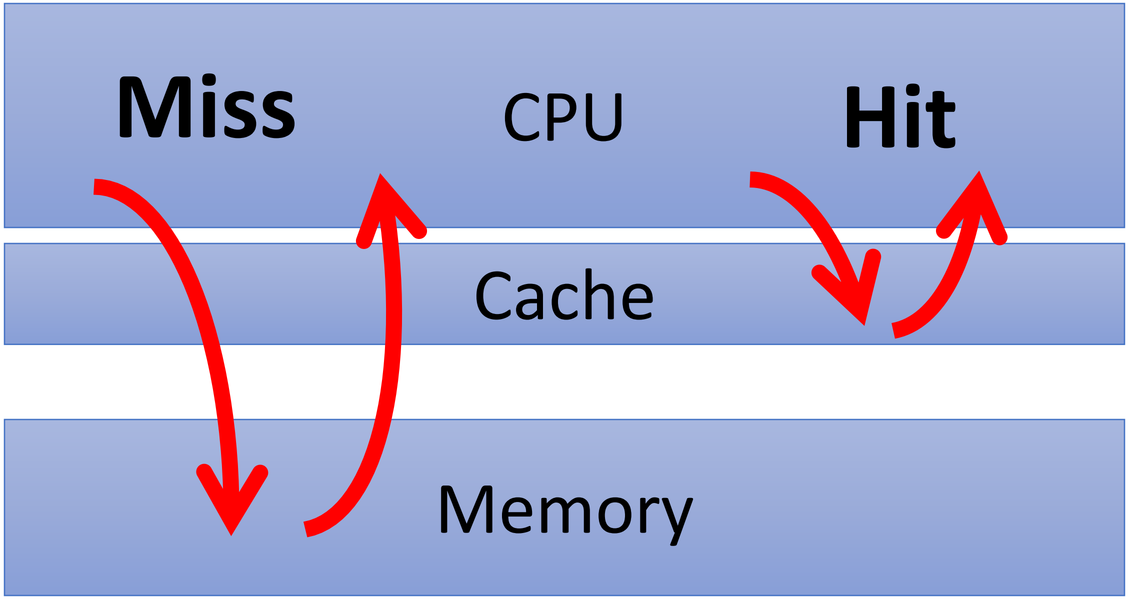
r 2017

CS101: Cache Structure

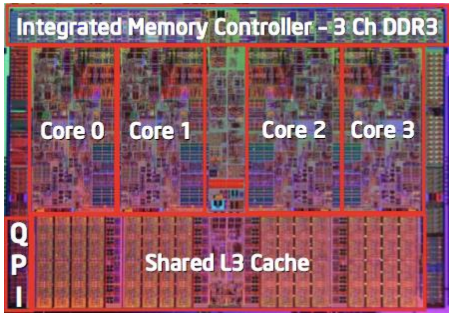


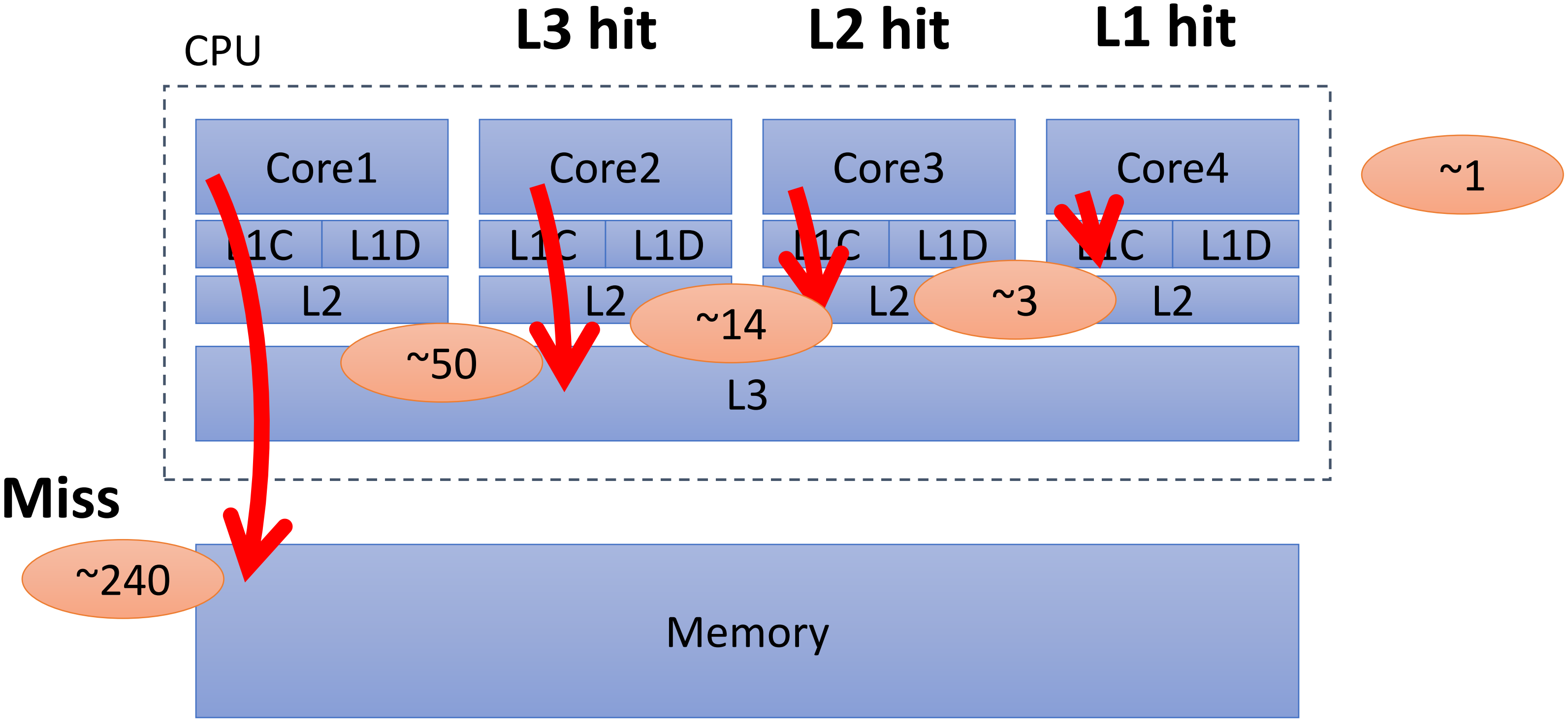
CS101: Cache


Integrated Memory Controller – 3 Ch DDR3
Core 0
Core 1
Core 2
Core 3
Q
P
I
Shared L3 Cache

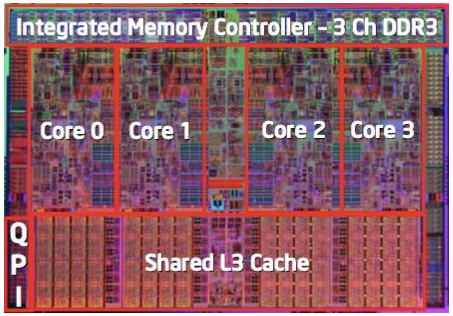

Miss
CPU
Hit
Cache
Memory

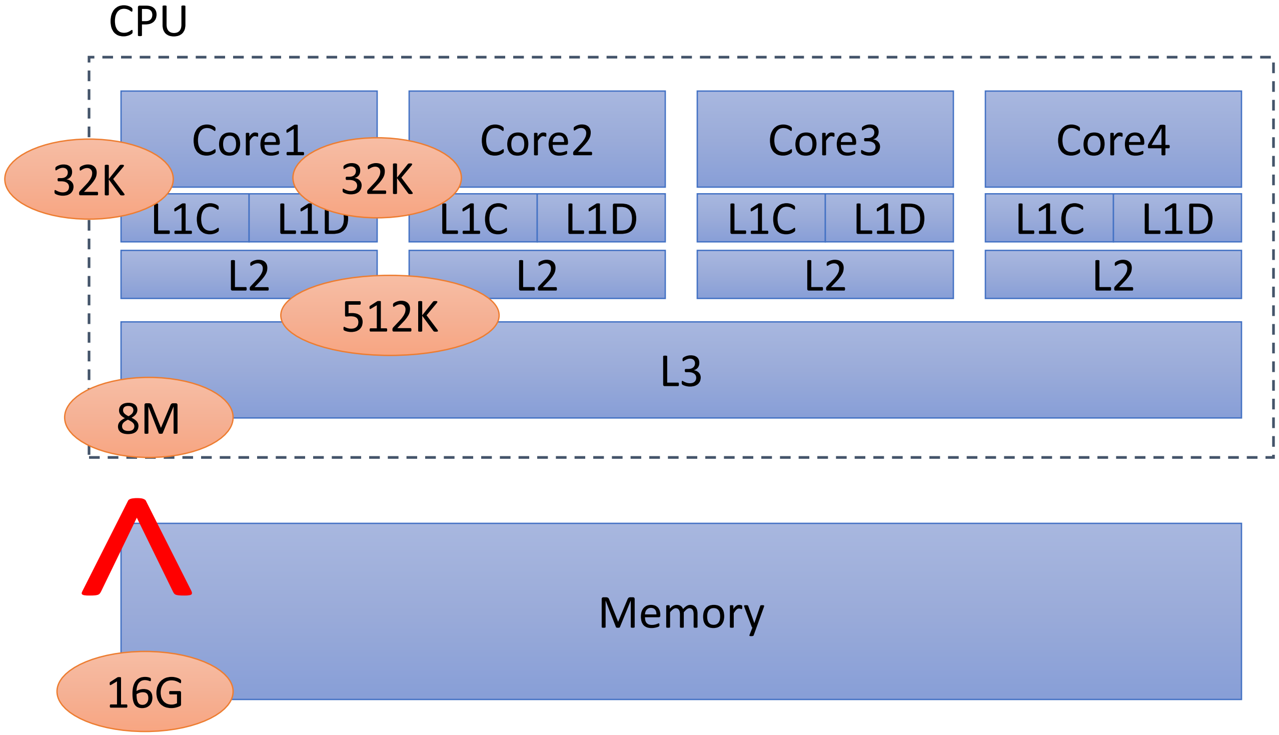
CS101: Cache


Integrated Memory Controller – 3 Ch DDR3
Core 0
Core 1
Core 2
Core 3
QPI
Shared L3 Cache


L3 hit
L2 hit
L1 hit
CPU
Core1
Core2
Core3
Core4
L1C
L1D
L1C
L1D
L1C
L1D
L1C
L1D
L2
L2
L2
L2
~1
~3
~14
~50
L3
Miss
~240
Memory

CS101: Cache

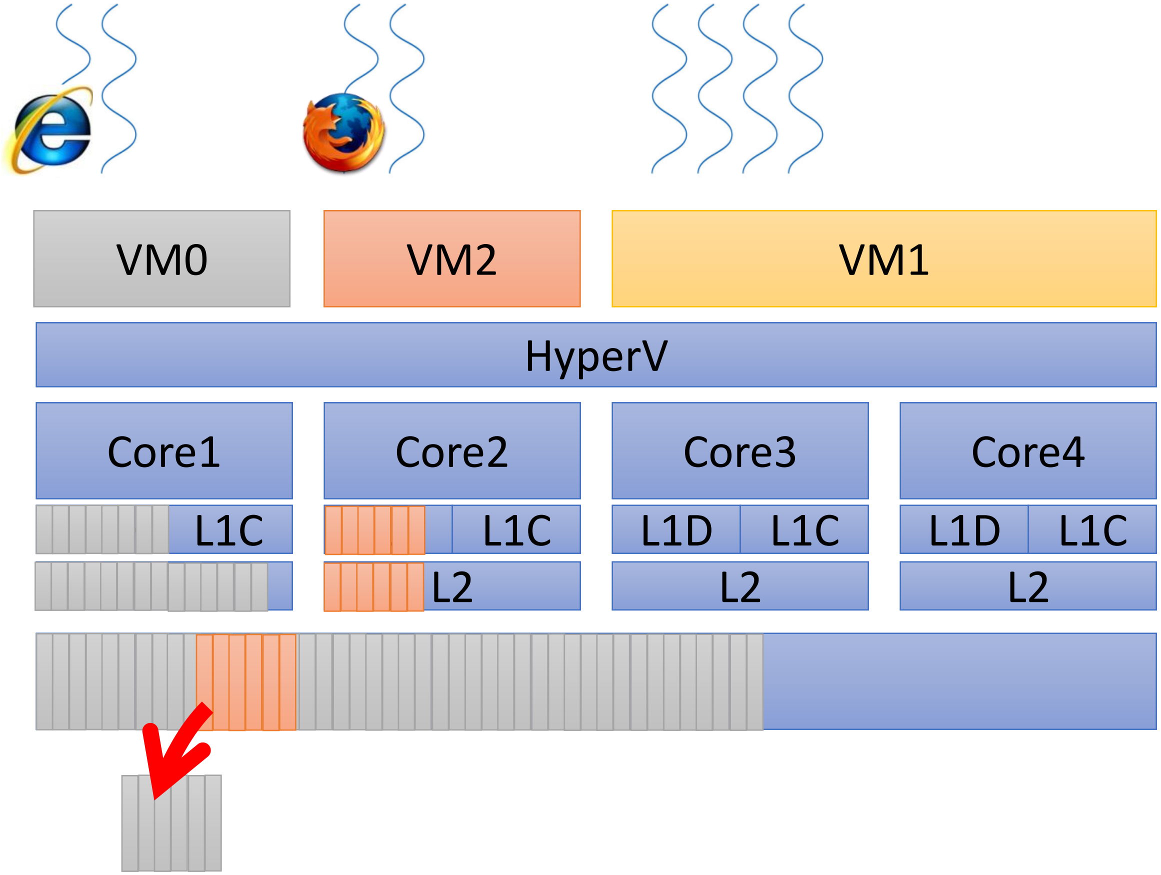




Which cacheline do we have to keep/evict (policy)?

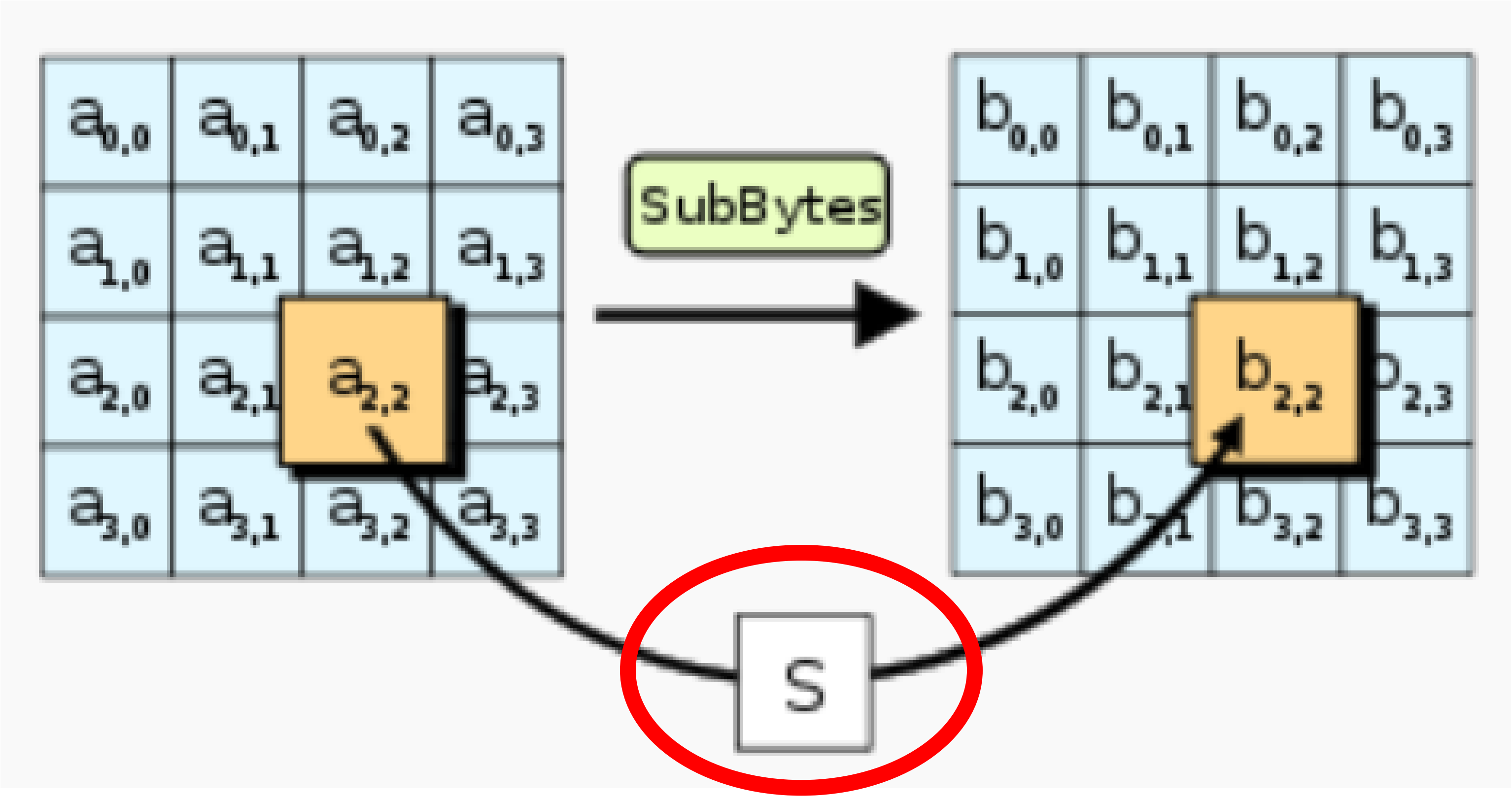
How to organize cacheline (structure)?

Basic Idea: Cache Side-channel



Real Attack: AES?

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

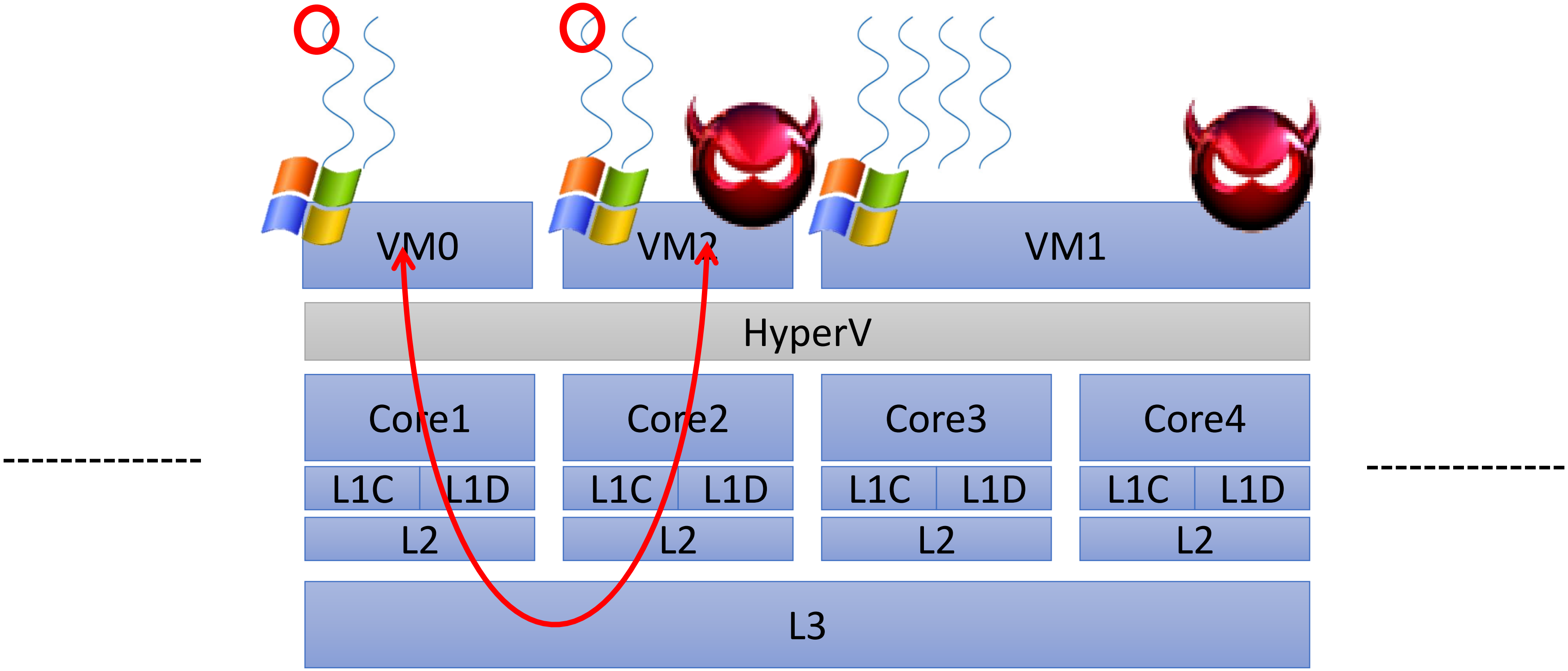

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
S

Monitoring cacheline access of Lookup Table!

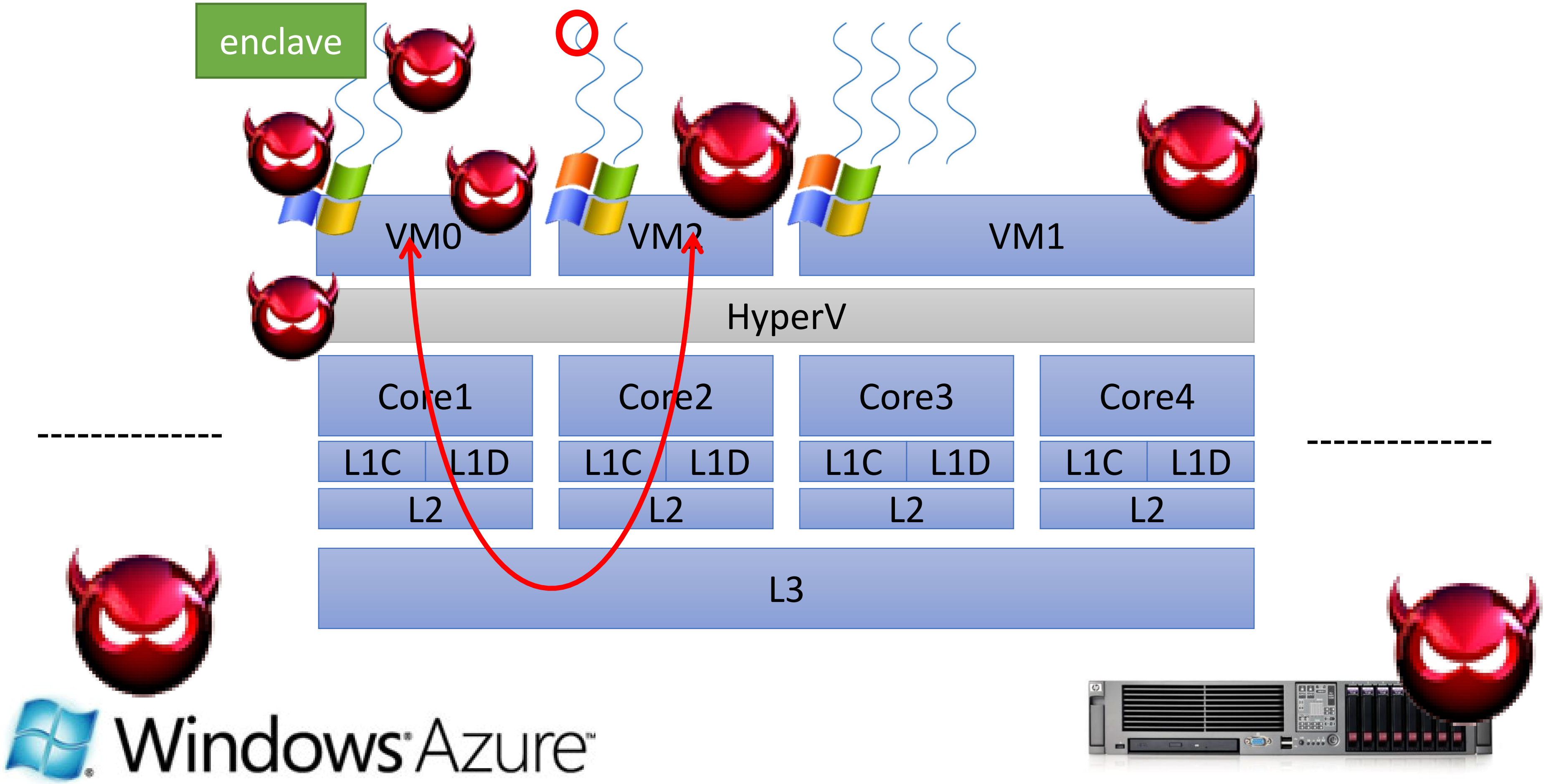
Known Attack Demonstrations

- 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)

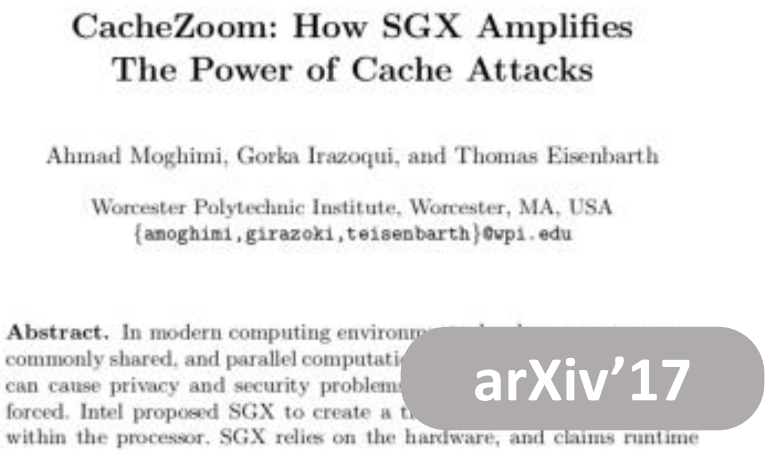


Cache Side-channel against SGX



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
{amoghimi,girazoki,teisenbarth}@wpi.edu

Abstract. In modern computing environm commonly shared, and parallel computati can cause privacy and security problem forced. Intel proposed SGX to create a t within the processor. SGX relies on the hardware, and claims runtime

arXiv'17

Software Grand Exposure: SGX Cache Attacks Are Practical

Ferdinand Brasser[1], Urs Müller[2], Alexandra Dmitrienko[2], Kari Kostiainen[2], Srdjan Capkun[2], and Ahmad-Reza Sadeghi[1]

[1]System Security Lab, Technische Universität Darmstadt, Germany
{ferdinand.brasser,ahmad.sadeghi}@trust.tu-darmstadt.de
[2]Institute of Information Security, ETH Zurich, Switzerland
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, ment of applica security. The cloud computin can be outsourced ture without having to fully trust the cloud provider and the entire software stack.[1]

WOOT'17

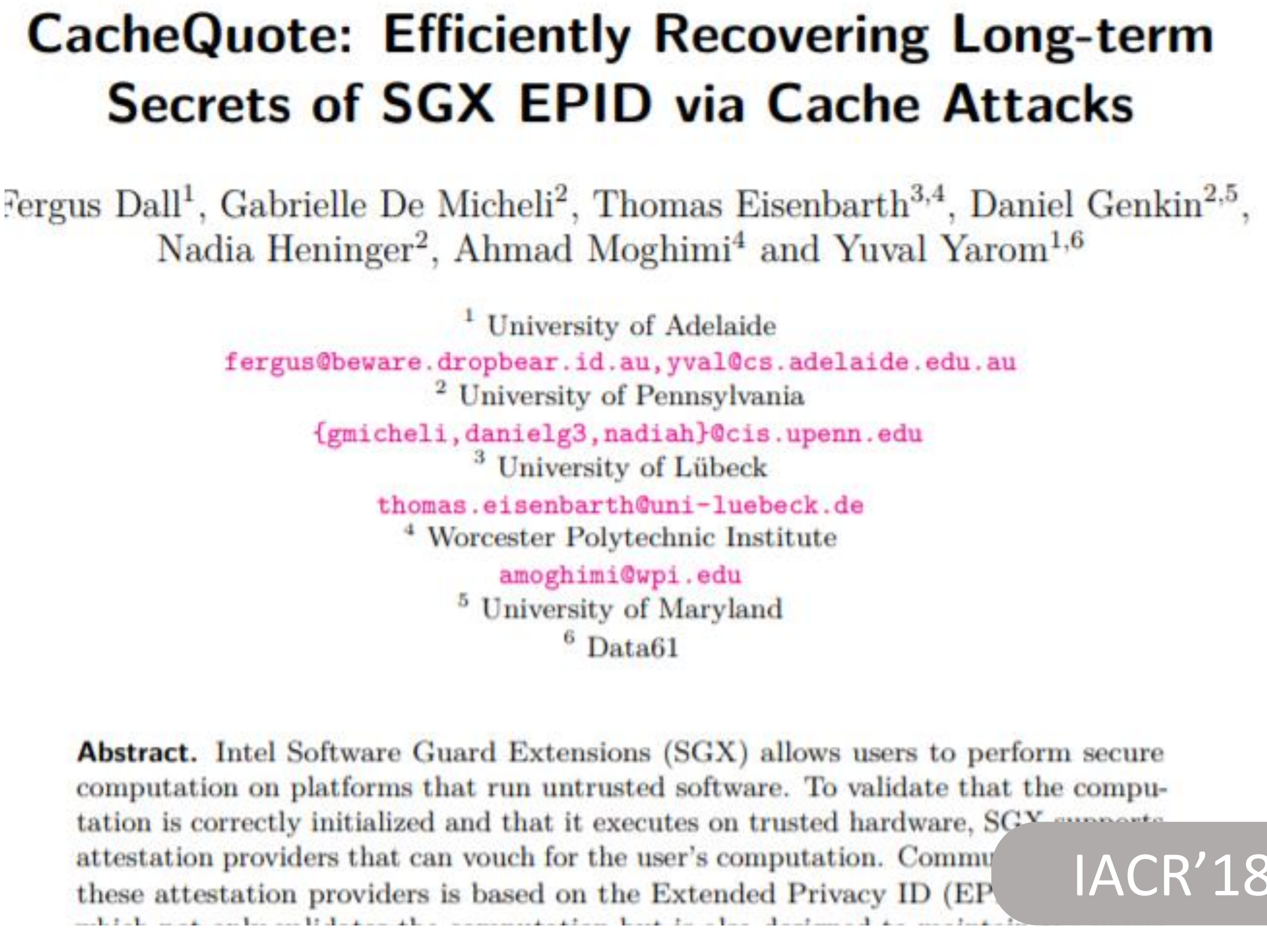
Cache Attack is Practical Concern?

- 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 *explictly* 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[1], Gabrielle De Micheli[2], Thomas Eisenbarth[3,4], Daniel Genkin[2,5], Nadia Heninger[2], Ahmad Moghimi[4] and Yuval Yarom[1,6]

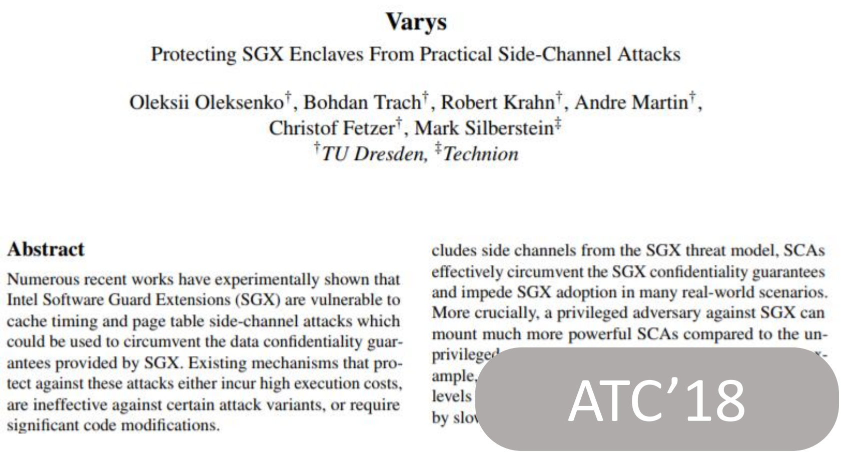
[1] University of Adelaide
fergus@beware.dropbear.id.au,yval@cs.adelaide.edu.au
[2] University of Pennsylvania
{gmicheli,danielg3,nadiah}@cis.upenn.edu
[3] University of Lübeck
thomas.eisenbarth@uni-luebeck.de
[4] Worcester Polytechnic Institute
amoghimi@wpi.edu
[5] University of Maryland
[6] Data61

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. Commu these attestation providers is based on the Extended Privacy ID (EP

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 Attacks

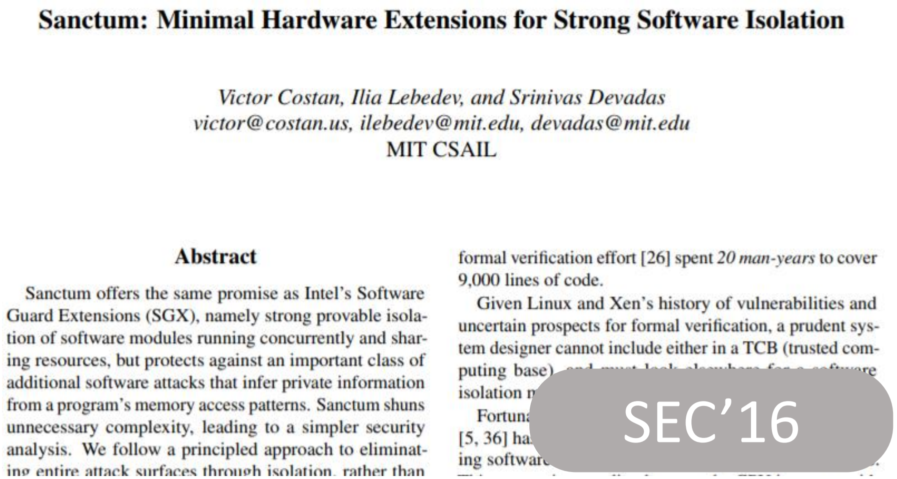
Oleksii Oleksenko[†], Bohdan Trach[†], Robert Krahn[†], Andre Martin[†], Christof Fetzer[†], Mark Silberstein[‡]
[†]*TU Dresden,* [‡]*Technion*

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 ample, levels by slov

ATC’18



Sanctum: Minimal Hardware Extensions for Strong Software Isolation

Victor Costan, Ilia Lebedev, and Srinivas Devadas
victor@costan.us, ilebedev@mit.edu, devadas@mit.edu
MIT CSAIL

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) isolation r

Fortuna [5, 36] ha ing software

SEC’16

Traditional Attack Vectors

- Cache-based side channel

  → e.g., inferring a private key
- Memory safety

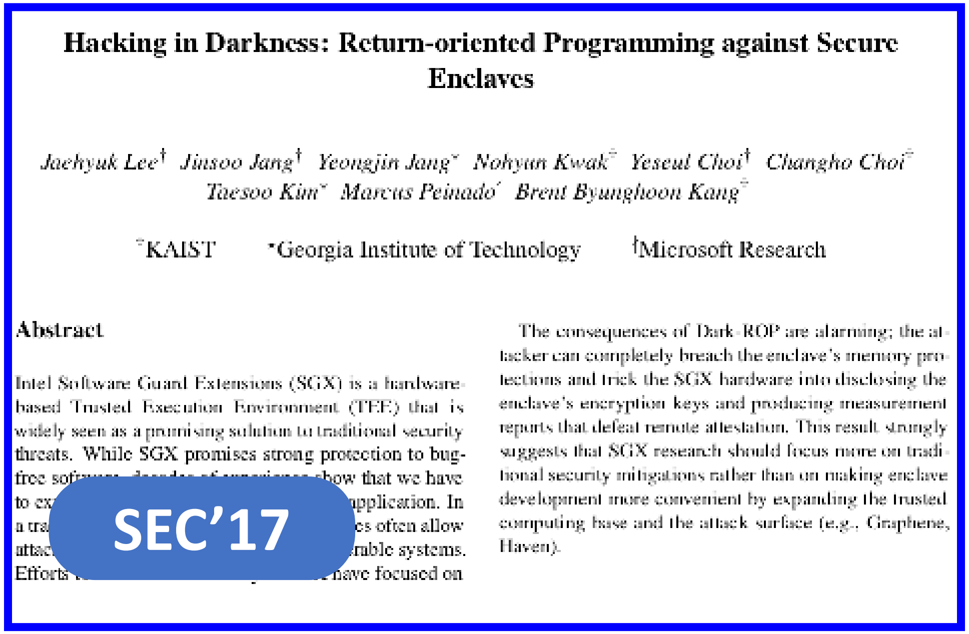
  → e.g., control flow hijacking
- Weak mitigation techniques

  → 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[†] *Jinsoo Jang*[†] *Yeongjin Jang*[*] *Nohyun Kwak*[‡] *Yeseul Choi*[†] *Changho Choi*[‡]
Taesoo Kim[*] *Marcus Peinado*[′] *Brent Byunghoon Kang*[‡]

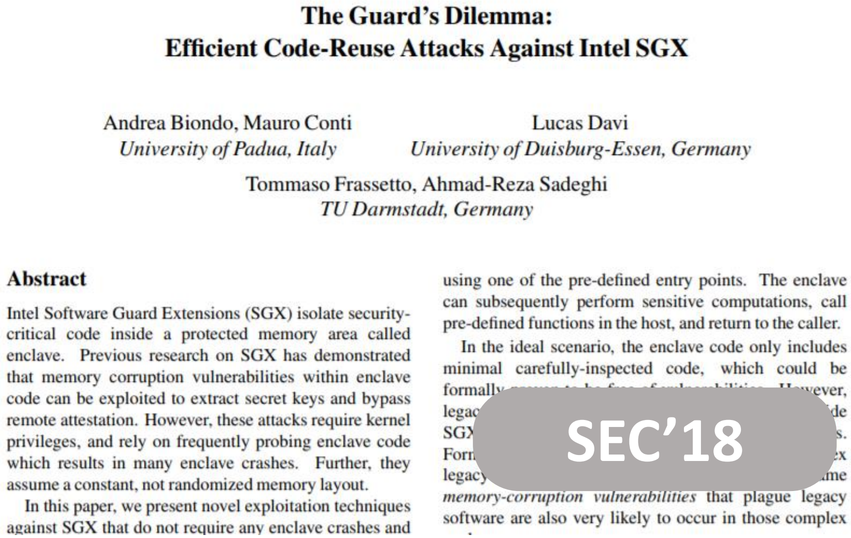
[‡]KAIST [*]Georgia Institute of Technology [†]Microsoft Research

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 sof
show that we have
to ex
pplication. In
a tra
es often allow
attac
rable systems.
Efforts
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

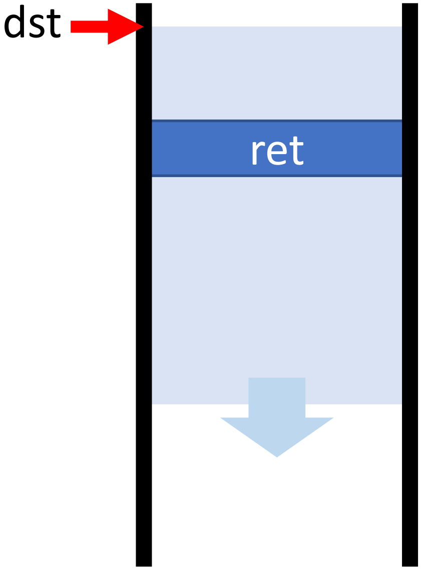
using one of the pre-defined entry points. The enclave can subsequently perform sensitive computations, call pre-defined functions in the host, and return to the caller.

In the ideal scenario, the enclave code only includes minimal carefully-inspected code, which could be formally
wever,
legac
de
SGX
s.
Forn
ex
legacy
me
memory-corruption vulnerabilities that plague legacy software are also very likely to occur in those complex

SEC'18

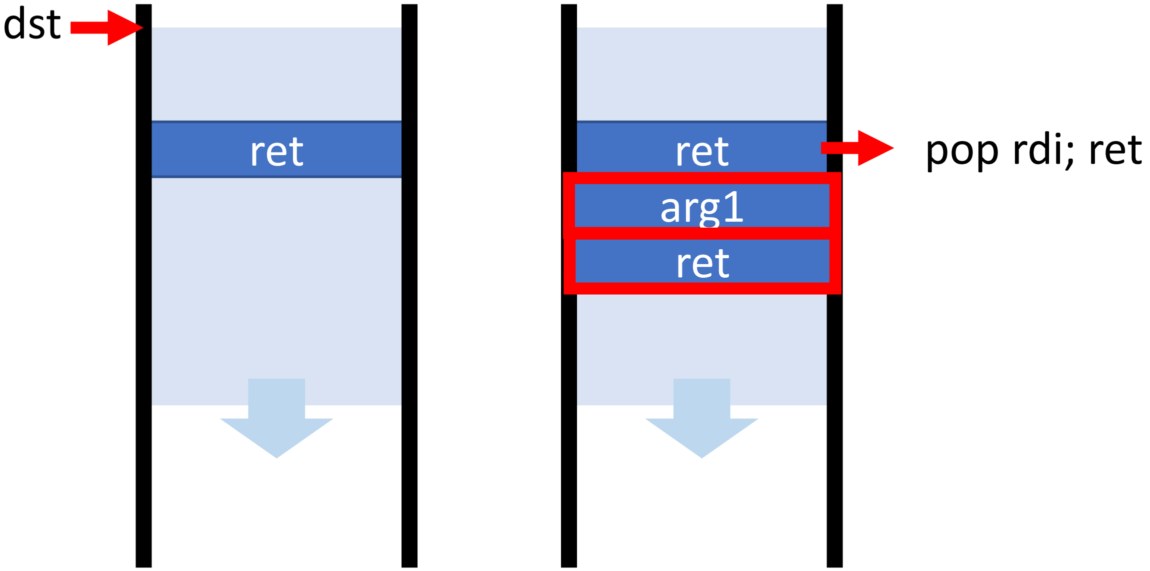
Return-oriented Programming (ROP)

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



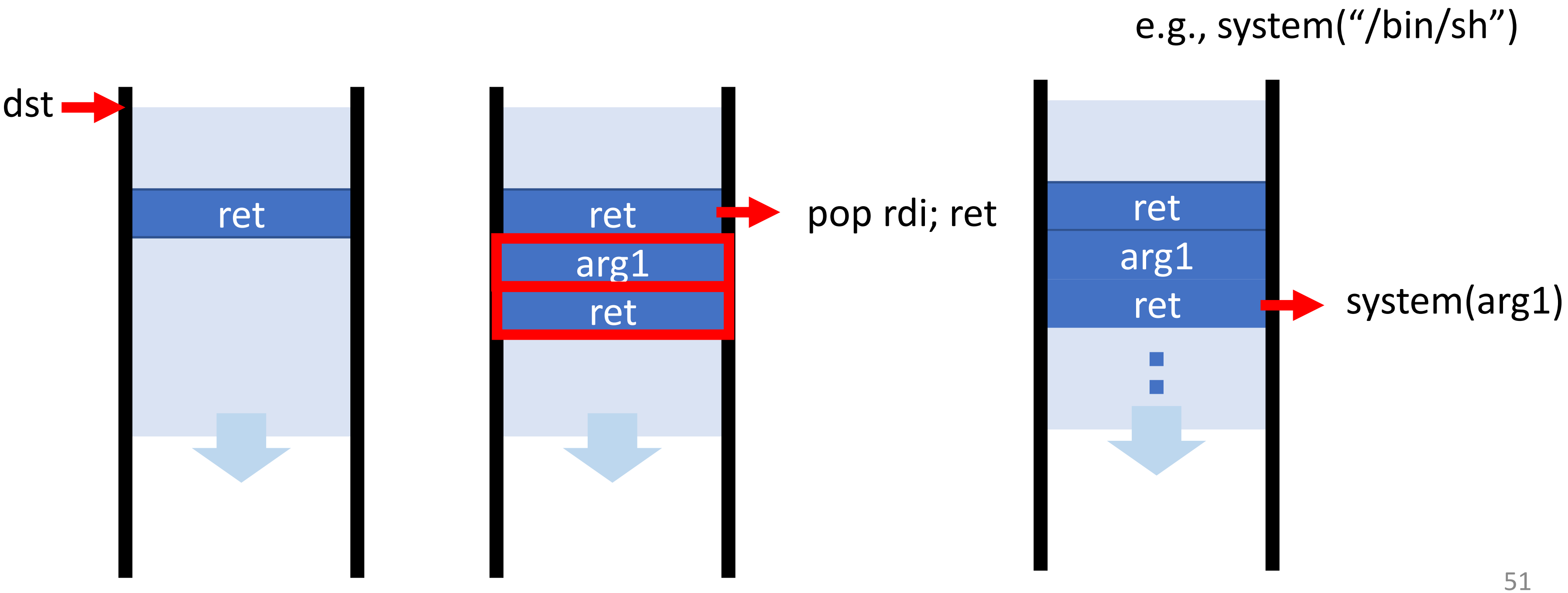
Return-oriented Programming (ROP)

```
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);
}
```



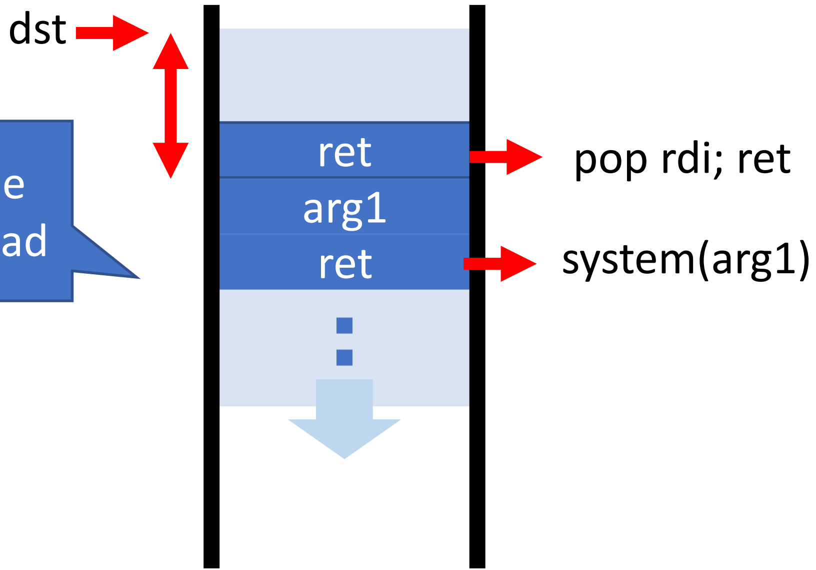
Typical Requirements for ROP

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

Code (via reverse engineering)

e.g., system(“/bin/sh”)

Need to determine the length of payload



ROP Inside an Enclave

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

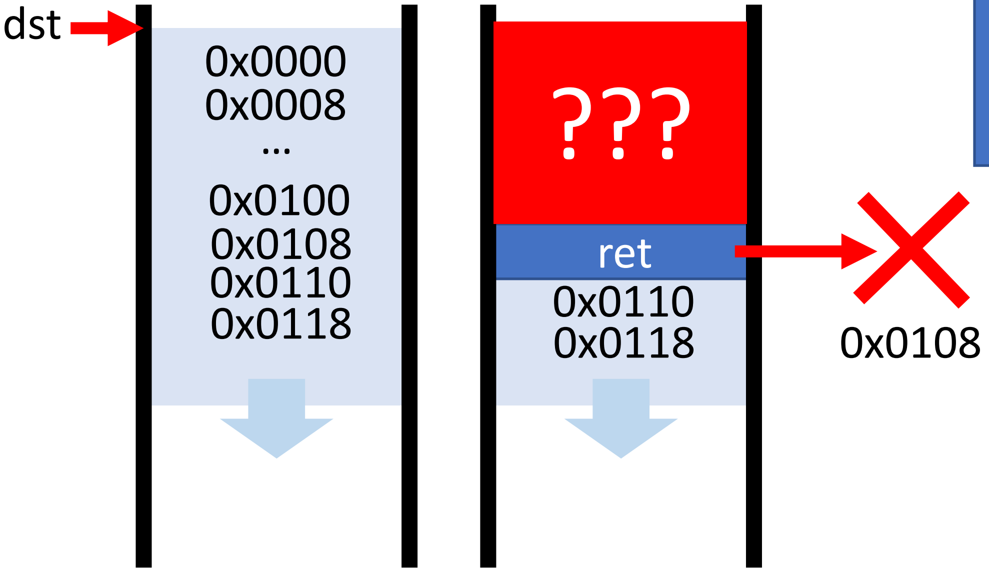
Code is not visible!

(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)



SGX doesn't report RIP directly but the corresponding page

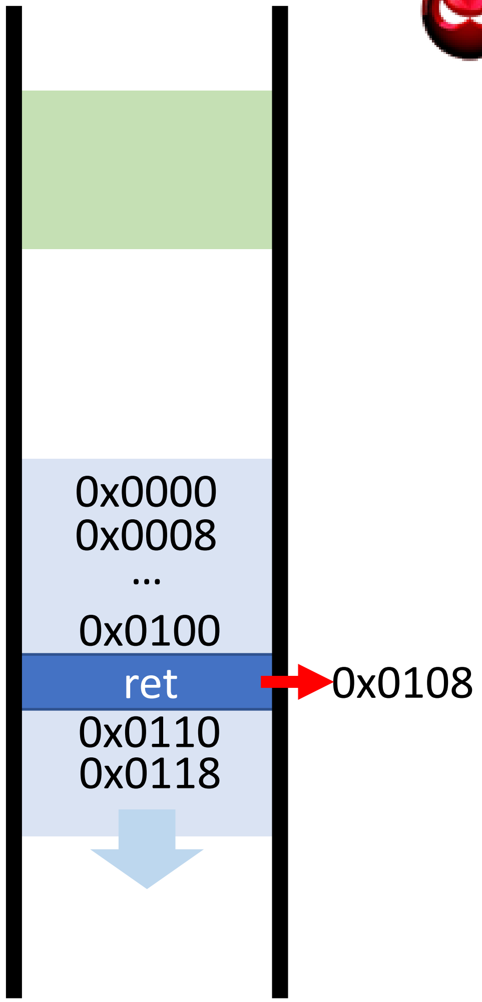
ROP in Darkness: Dark ROP

- 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

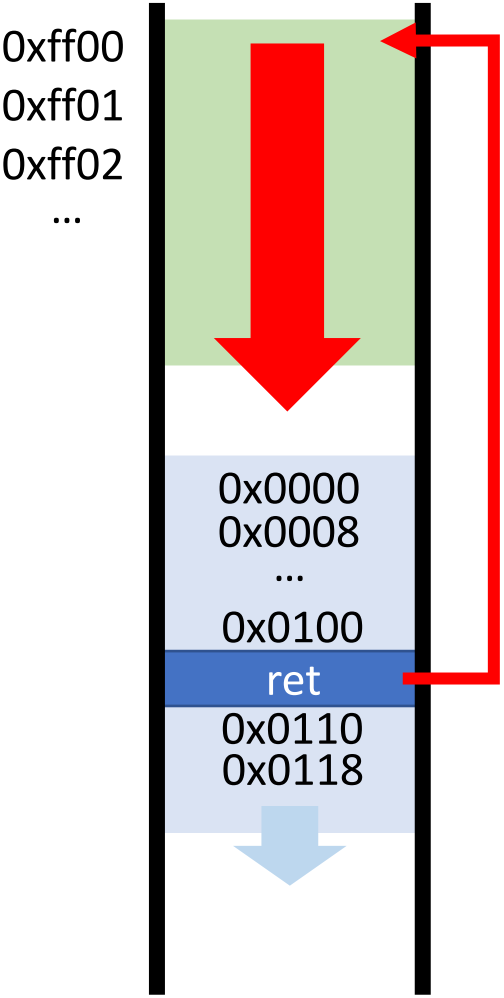
- Know existence of a buffer overflow (i.e., crash)
- Crashing the enclave arbitrarily times
- Built with standard libraries (e.g., SGX SDK)
- **Distributed in an encrypted** form (like VC3)

Step 1. Looking for pop Gadgets



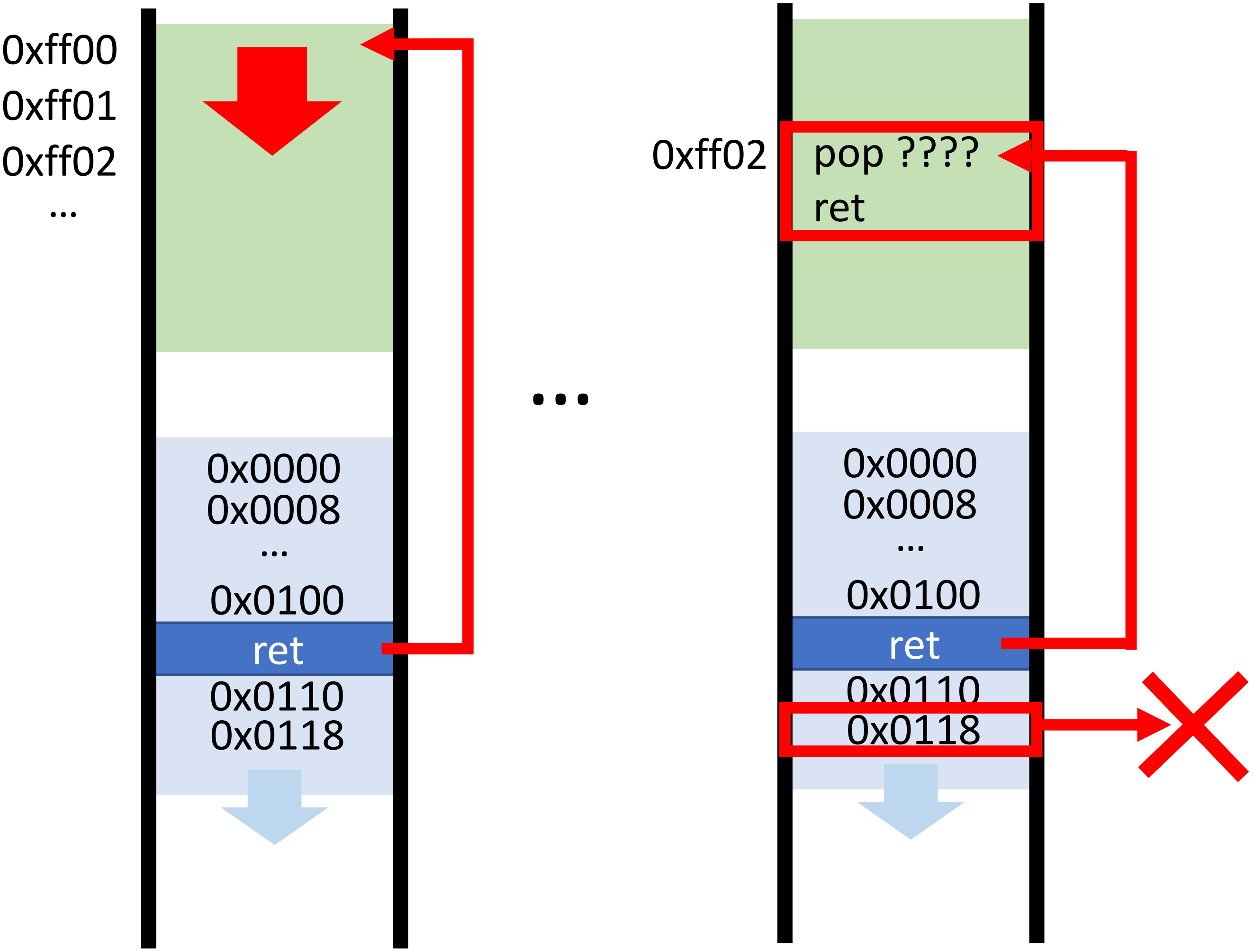
You have a full control over the layout of the enclave

Step 1. Looking for pop Gadgets



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

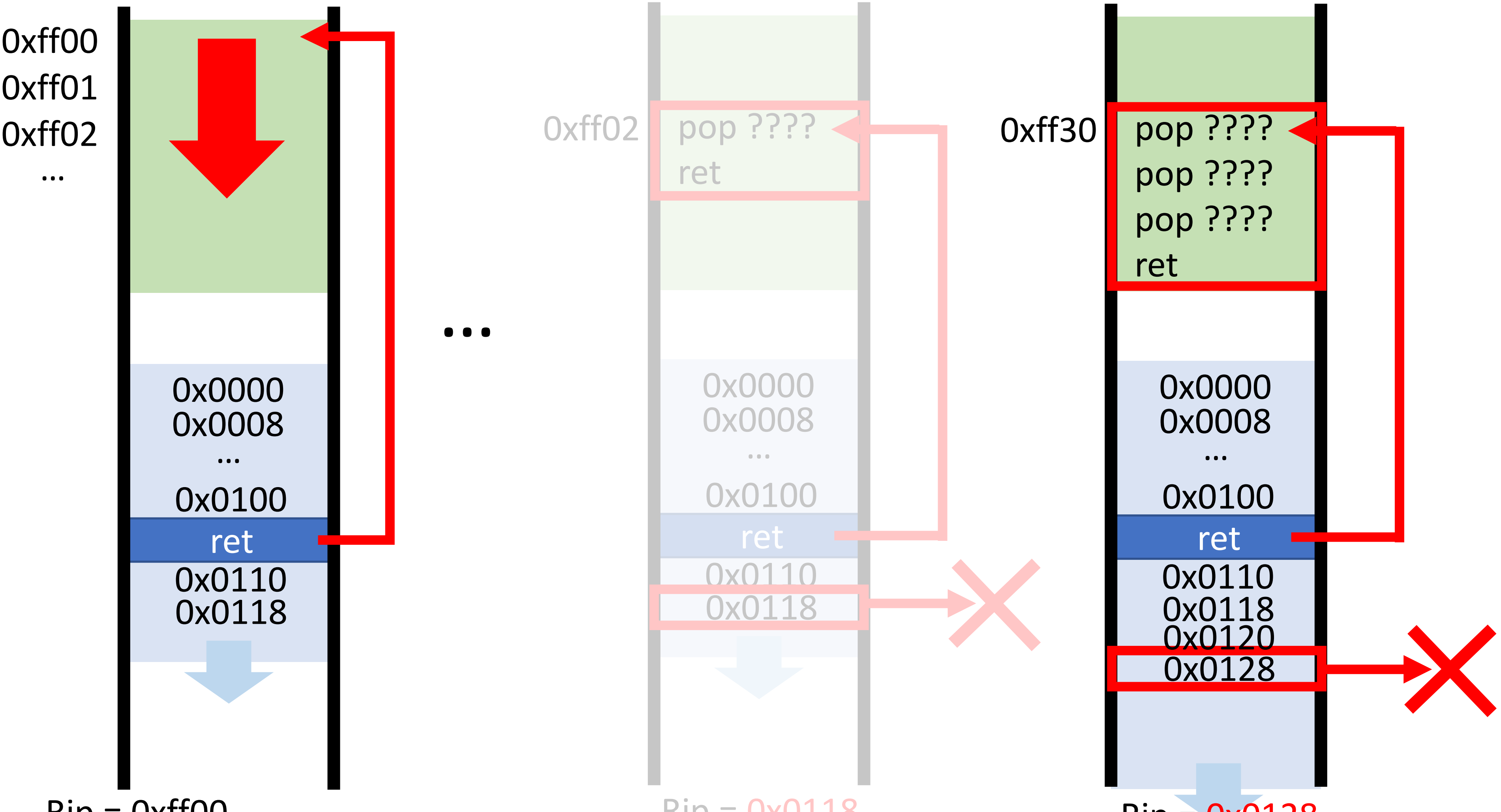
Step 1. Looking for pop Gadgets



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

Rip = 0x0118
(segfault)

Step 1. Looking for pop Gadgets



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

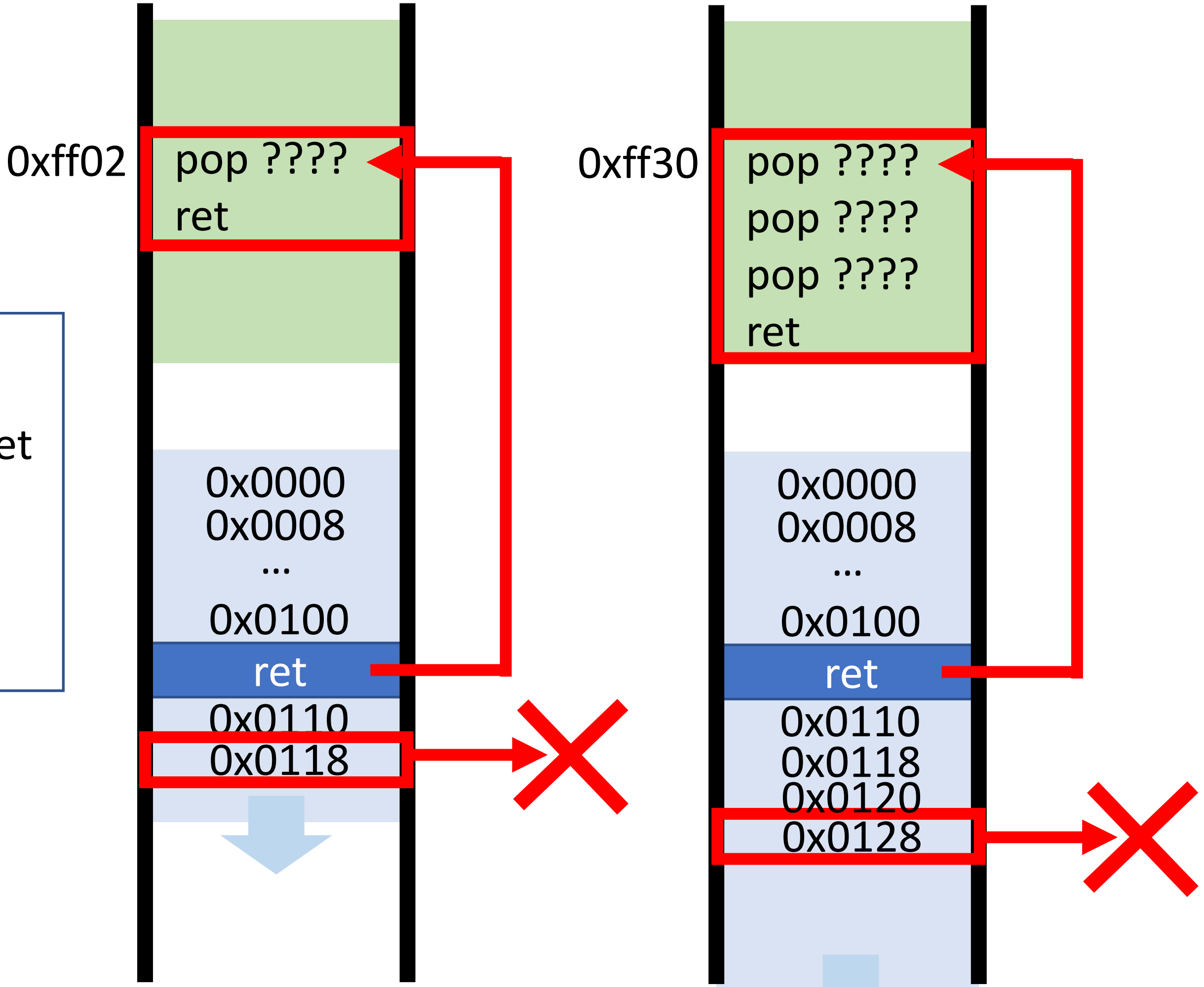
Rip = 0x0118
(segfault)

Rip = 0x0128
(segfault)

Step 1. Looking for pop Gadgets

Catalog of pop gadgets
(unknown args)

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



Rip = 0x0118
(segfault)

Rip = 0x0128
(segfault)

Step 2. Looking for ENCLU

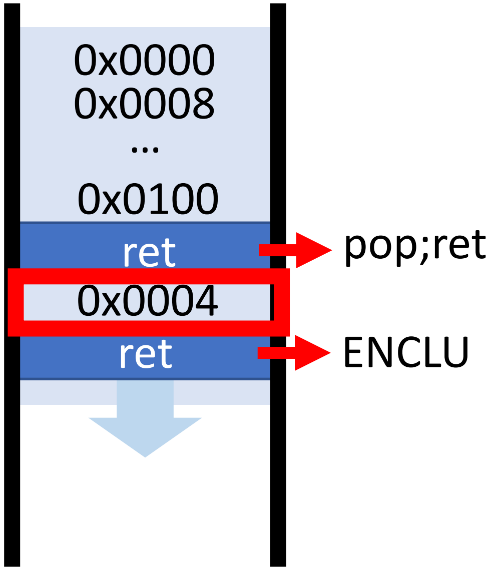
- ENCLU: an inst. dispatches to various leaf functions
  - rax = 0: EREPORT
  - rax = 1: EGETKEY
  - …
  - rax = 4: EEXIT

Step 2. Looking for ENCLU

- ENCLU: an inst. dispatches to various leaf functions
    - rax = 0: EREPORT
    - rax = 1: EGETKEY
    - …
    - rax = 4: EEXIT

→ Scan code for each "pop????;ret"

→ If gracefully exit, rip = ENCLU

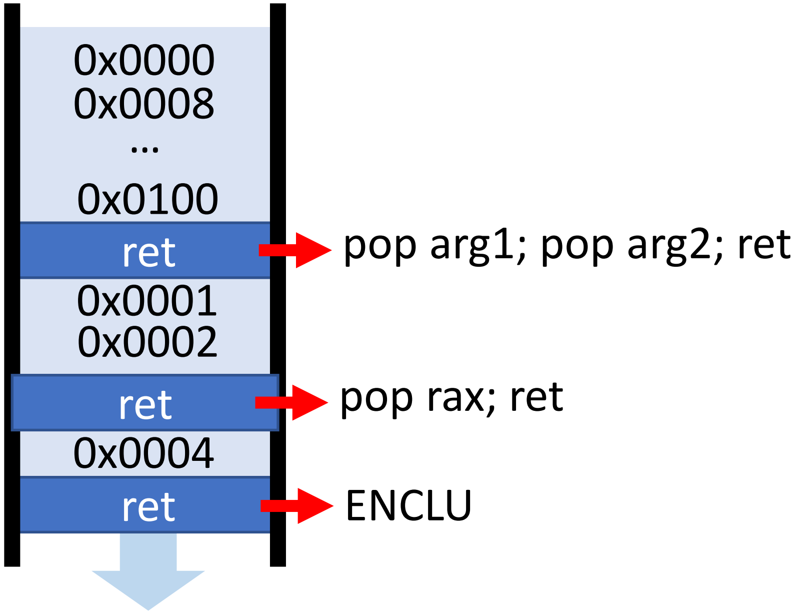


Step 3. Deciphering pop Gadgets

- EEXIT (ENCLU & rax=4) left a register file uncleaned

→ Scan code for all pop gadgets

→ Check arguments

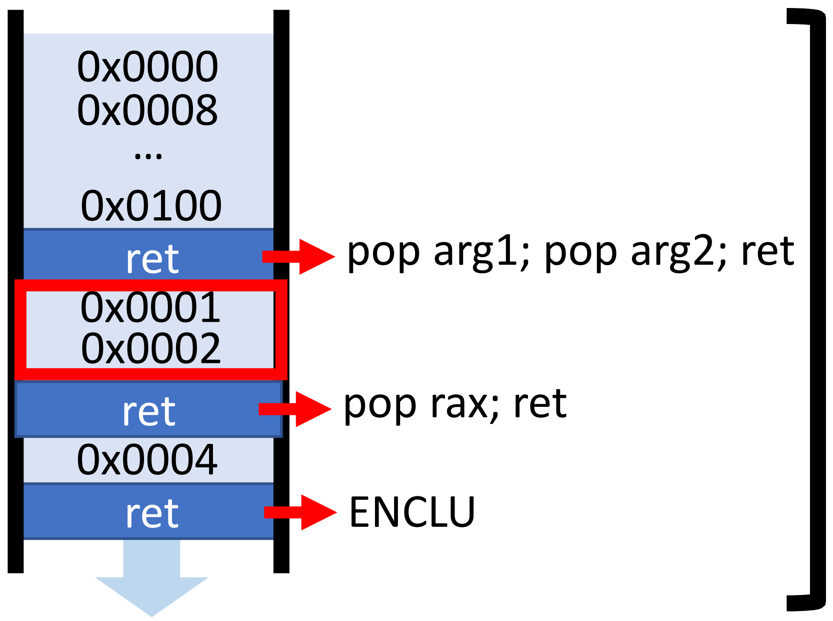


Step 3. Deciphering pop Gadgets

- EEXIT (ENCLU & rax=4) left a register file uncleaned

→ Scan code for all pop gadgets

→ Check arguments



Deciphering
pop? pop? gadget

arg1 = 0x0001
arg2 = 0x0002

+

Register file

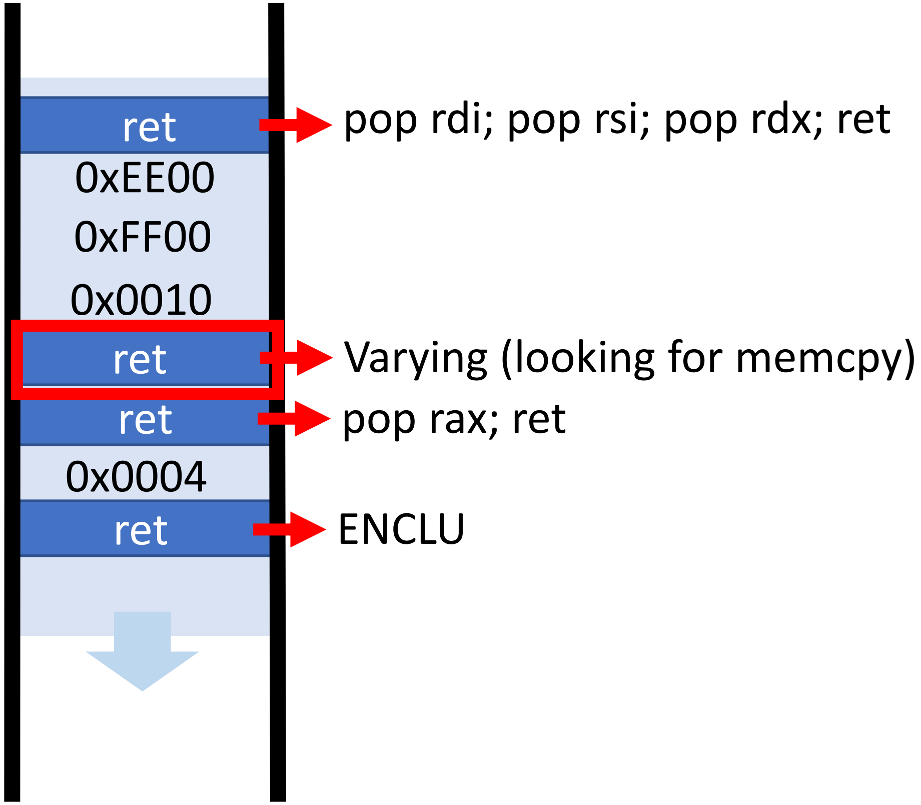
rax = 0x0004
rsi = 0x0001
rdi = 0x0002
…

=

pop rsi
pop rdi
ret

Step 4. Looking for memcpy()

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



pop rdi; pop rsi; pop rdx; ret

Varying (looking for memcpy)

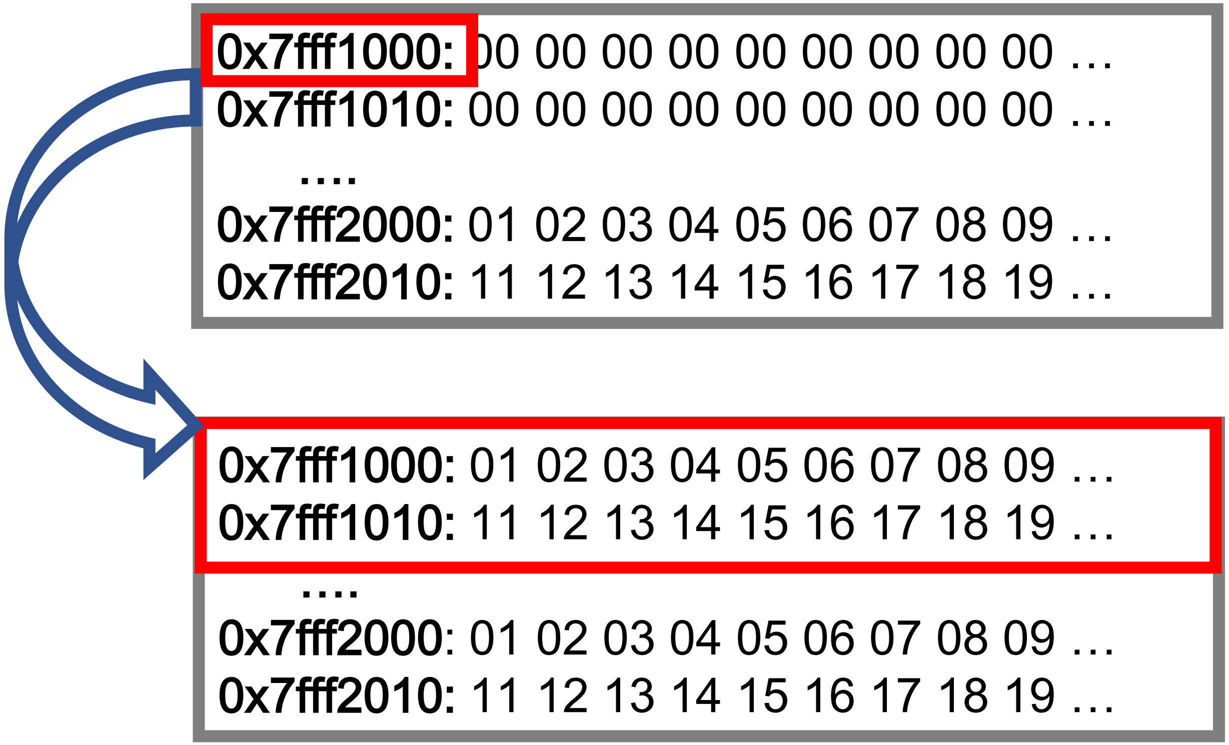
pop rax; ret

ENCLU

Step 4. Looking for memcpy()

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

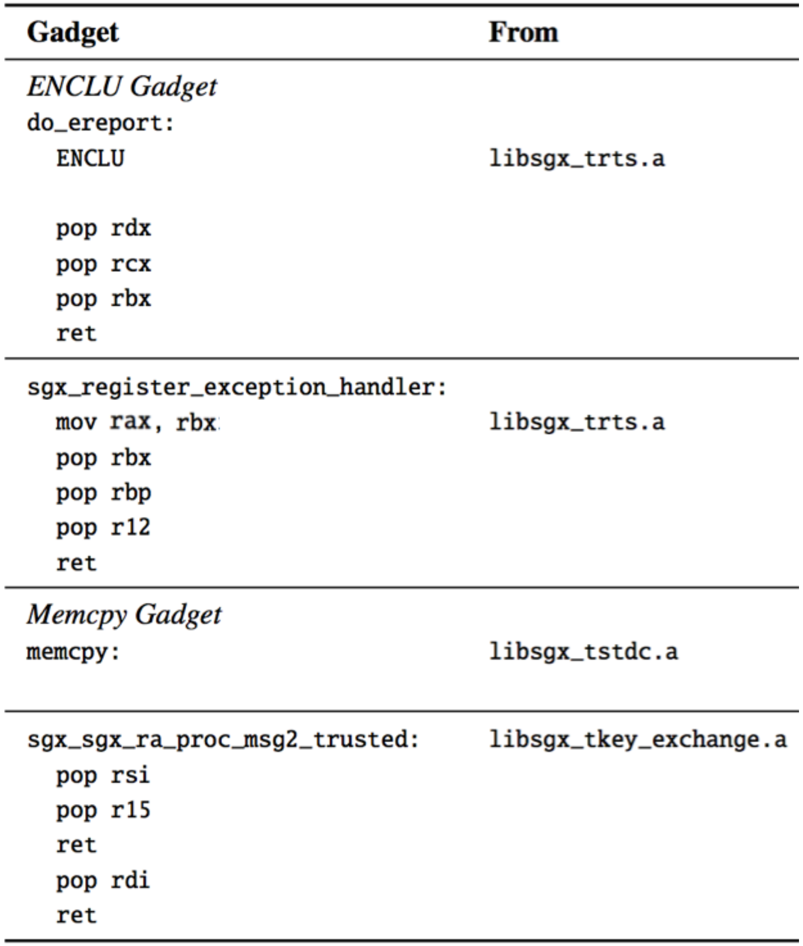
Untrusted application memory



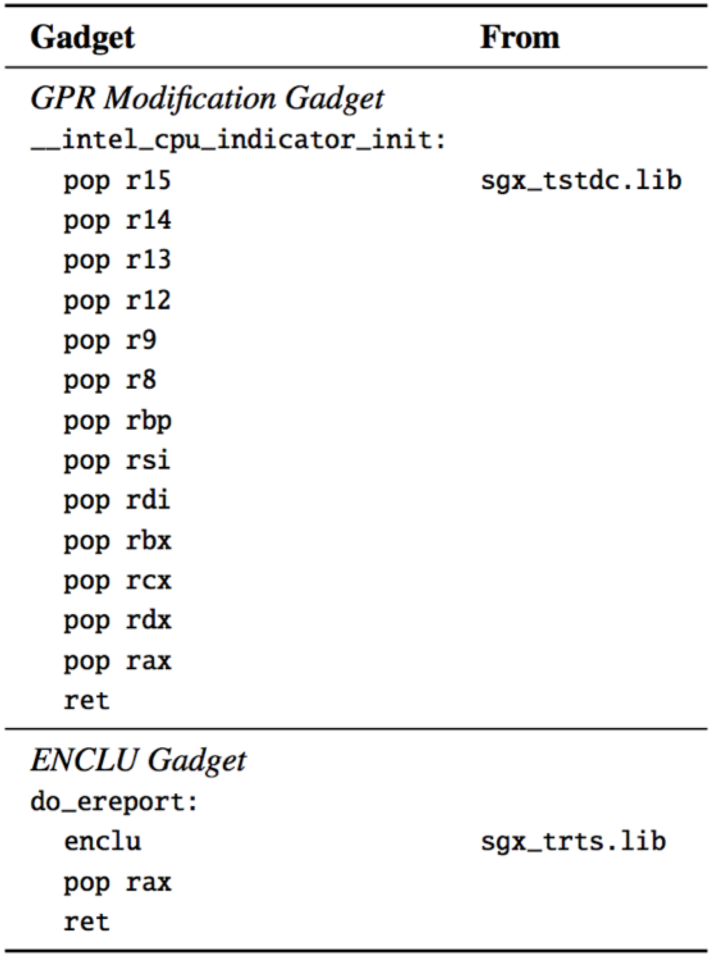
```
0x7fff1000: 00 00 00 00 00 00 00 00 00 …
0x7fff1010: 00 00 00 00 00 00 00 00 00 …
     ....
0x7fff2000: 01 02 03 04 05 06 07 08 09 …
0x7fff2010: 11 12 13 14 15 16 17 18 19 …
```

```
0x7fff1000: 01 02 03 04 05 06 07 08 09 …
0x7fff1010: 11 12 13 14 15 16 17 18 19 …
     ....
0x7fff2000: 01 02 03 04 05 06 07 08 09 …
0x7fff2010: 11 12 13 14 15 16 17 18 19 …
```

Gadgets Everywhere (e.g., SDK)



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

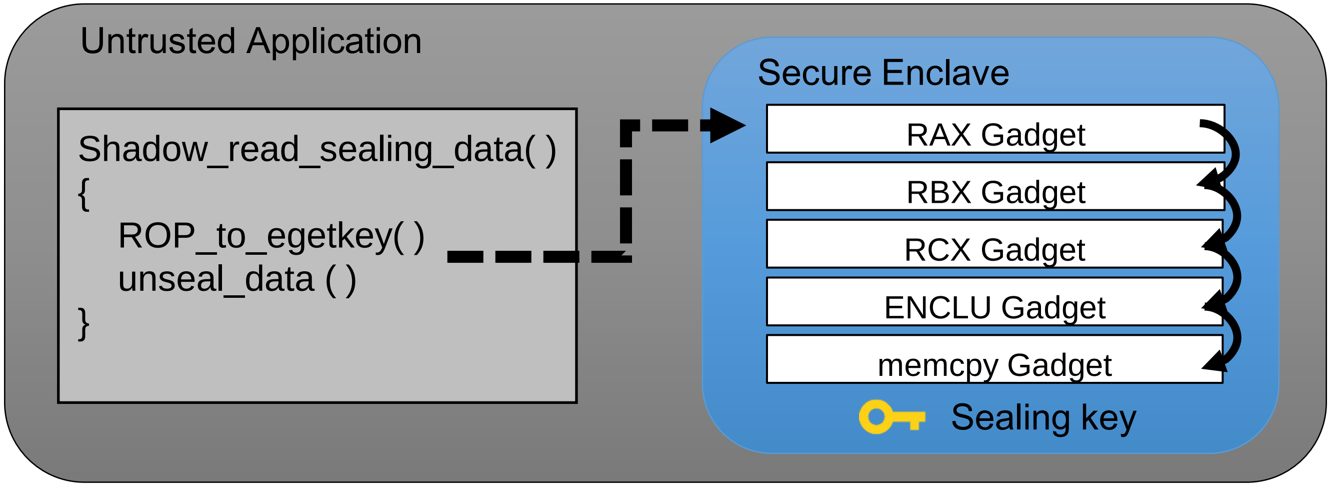


Gadget	From
GPR Modification Gadget	
__intel_cpu_indicator_init:	
pop r15	sgx_tstdc.lib
pop r14	
pop r13	
pop r12	
pop r9	
pop r8	
pop rbp	
pop rsi	
pop rdi	
pop rbx	
pop rcx	
pop rdx	
pop rax	
ret	
ENCLU Gadget	
do_ereport:	
enclu	sgx_trts.lib
pop rax	
ret	

DEMO: PoC Dark ROP

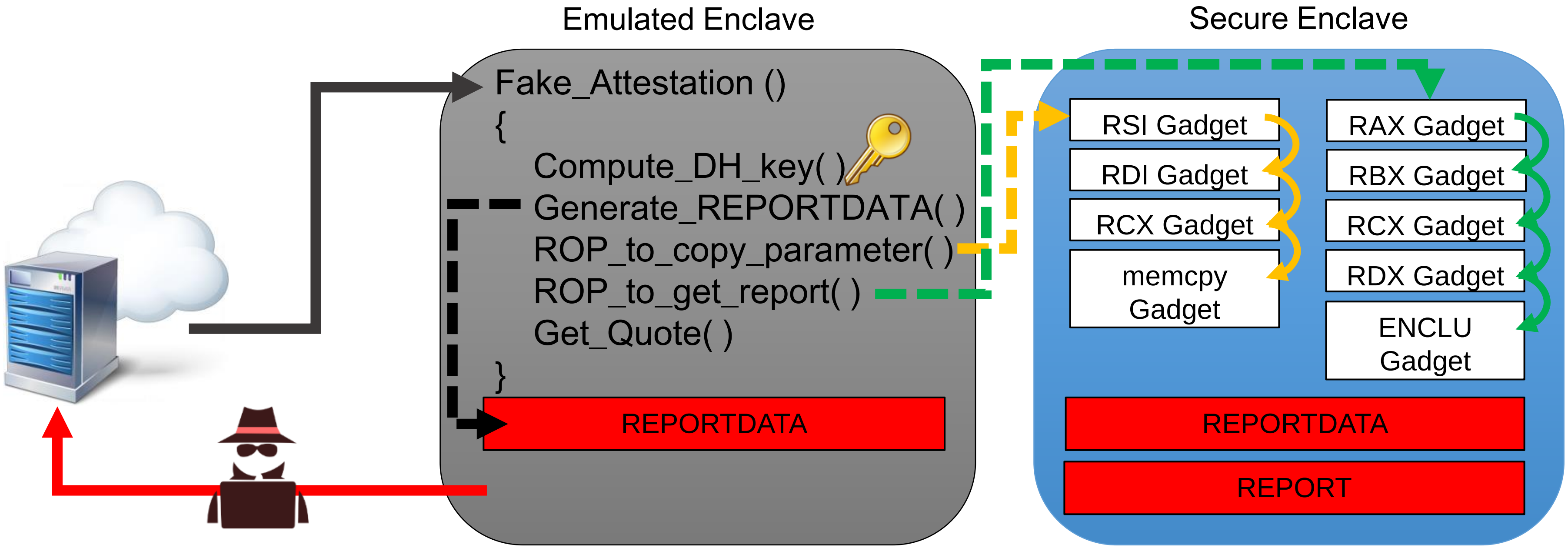
Step1. Looking for pop gadgets

Case Study 1: Unsealing Data



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

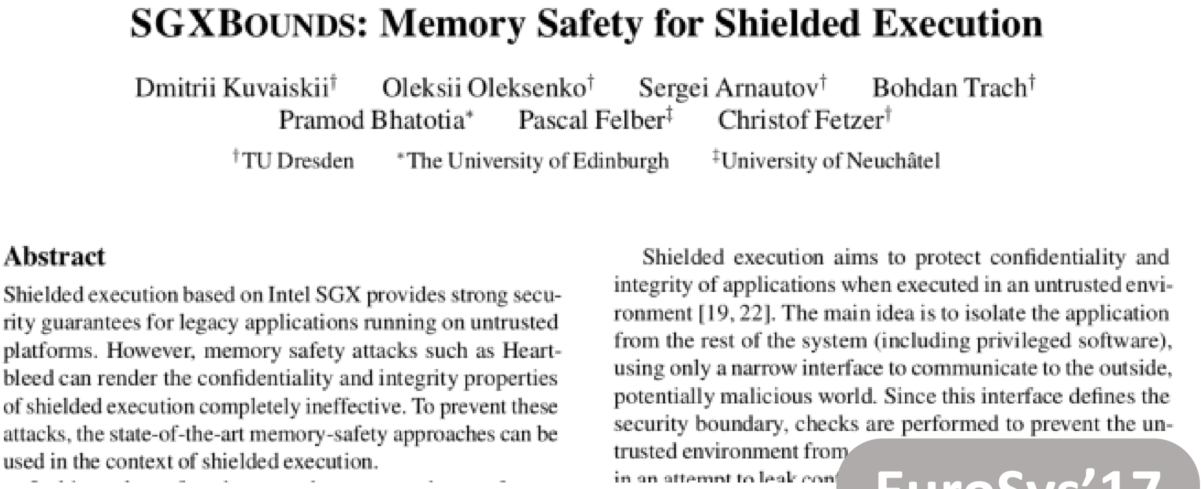
Case Study 2: Hijacking Remote Attestation



- 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[†] Oleksii Oleksenko[†] Sergei Arnautov[†] Bohdan Trach[†]
Pramod Bhatotia[*] Pascal Felber[‡] Christof Fetzer[†]
[†]TU Dresden [*]The University of Edinburgh [‡]University of Neuchâtel

Abstract

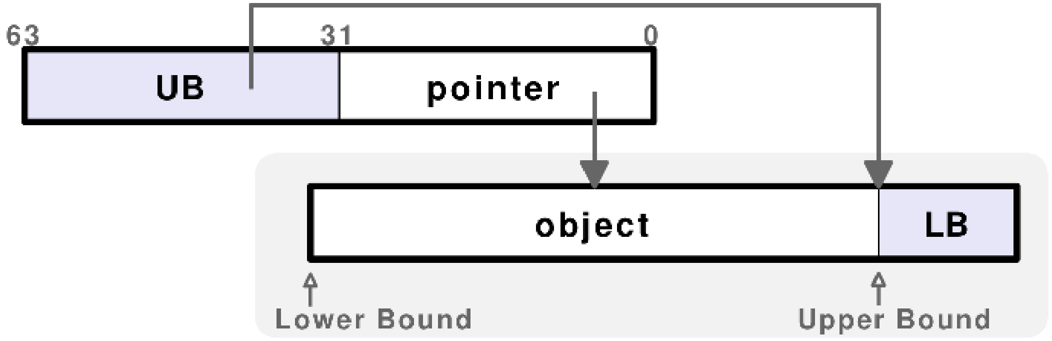
Shielded execution based on Intel SGX provides strong security guarantees for legacy applications running on untrusted platforms. However, memory safety attacks such as Heartbleed can render the confidentiality and integrity properties of shielded execution completely ineffective. To prevent these attacks,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

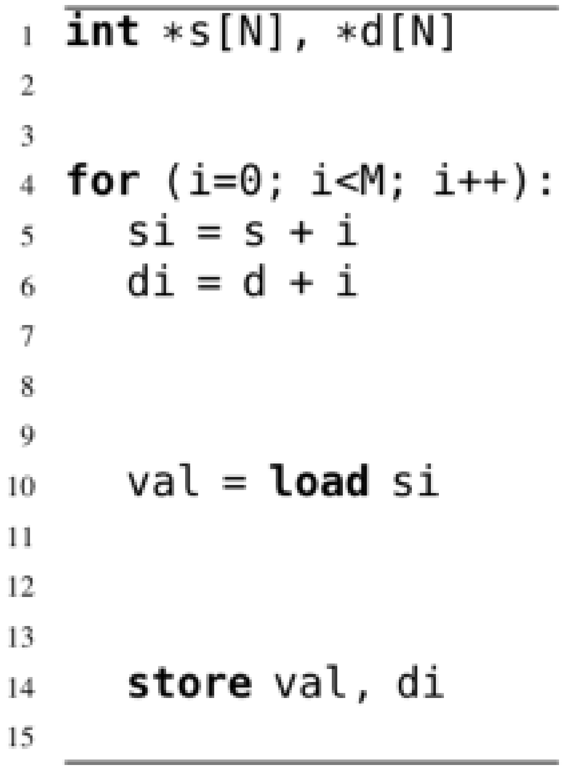
EuroSys'17

Defense: SGXBounds

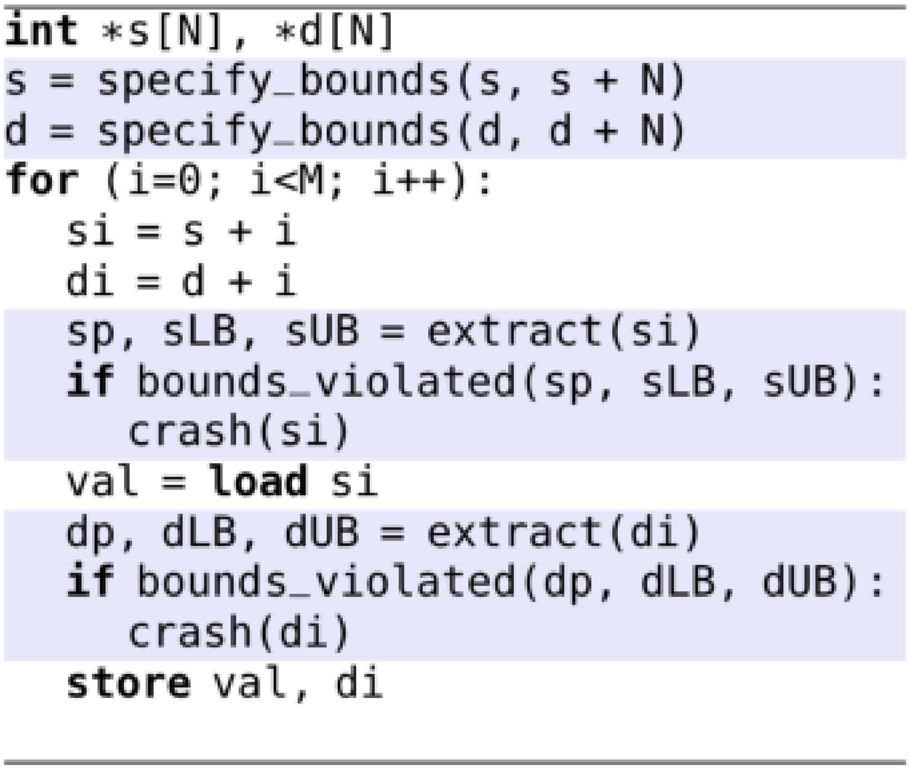
- Addressing *spatial memory* problems (bound chk)
- Key idea: an efficient tag representation thanks to smaller memory space!



Defense: SGXBounds



```
int *s[N], *d[N]


for (i=0; i<M; i++):
   si = s + i
   di = d + i



   val = load si



   store val, di

```



```
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

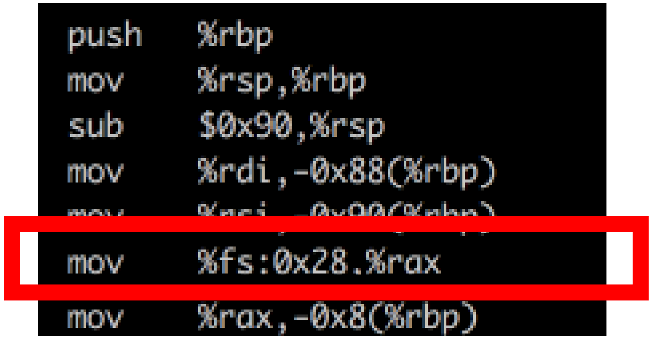
  DEP/NX

  ASLR/PIE

SGX Mitigation Checklist

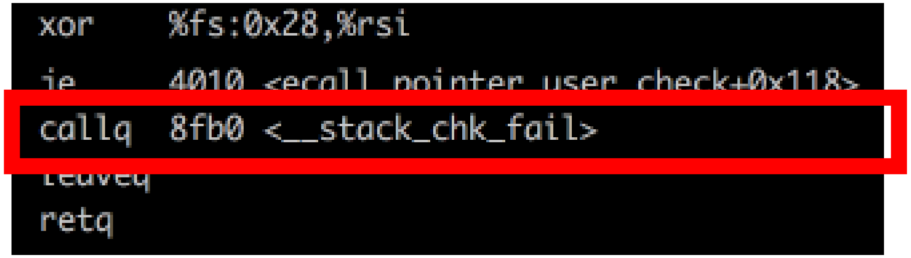
- Popular mitigation schemes:
  - ✓ Stack Canary
  - ✓ RELRO
  - DEP/NX
  - ASLR/PIE

ecall_pointer_user_check():



```
push    %rbp
mov     %rsp,%rbp
sub     $0x90,%rsp
mov     %rdi,-0x88(%rbp)
mov     %rsi,-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
```

epillogue

SGX Mitigation Checklist

- Popular mitigation schemes:

✓ Stack Canary

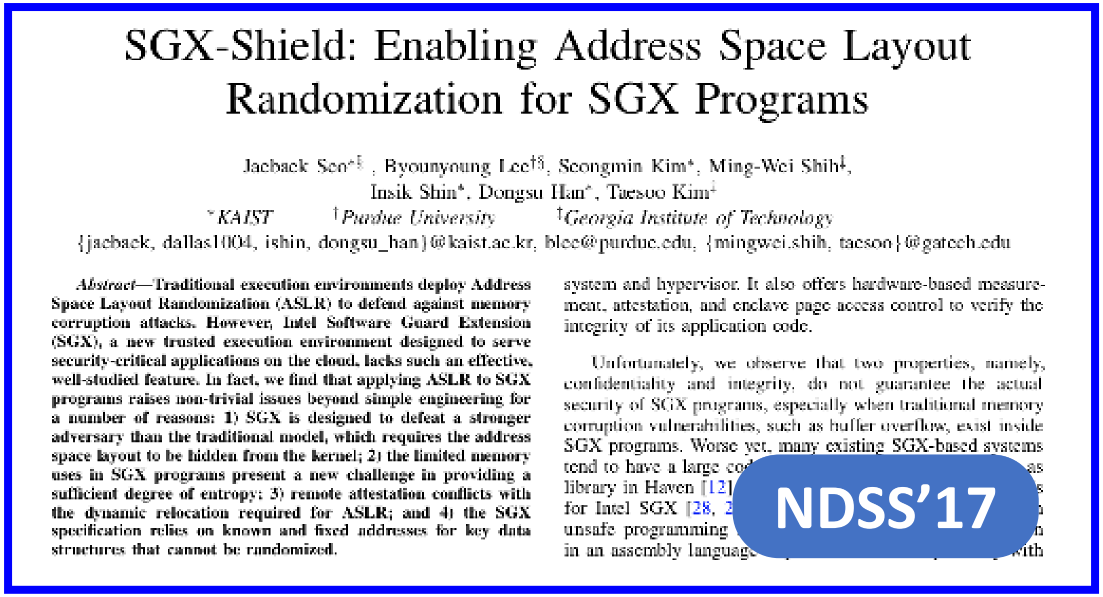
✓ RELRO

✗ 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

Jaeback Seo*§, Byounyoung Lee†§, Seongmin Kim*, Ming-Wei Shih‡,
Insik Shin*, Dongsu Han*, Taesoo Kim‡
*KAIST †Purdue University ‡Georgia Institute of Technology
{jaeback, dallas1004, ishin, dongsu_han}@kaist.ac.kr, blee@purdue.edu, {mingwei.shih, taesoo}@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 cod as library in Haven [12] s for Intel SGX [28, 2 unsafe programming n in an assembly language with

NDSS'17

Challenges for Mitigation Schemes

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

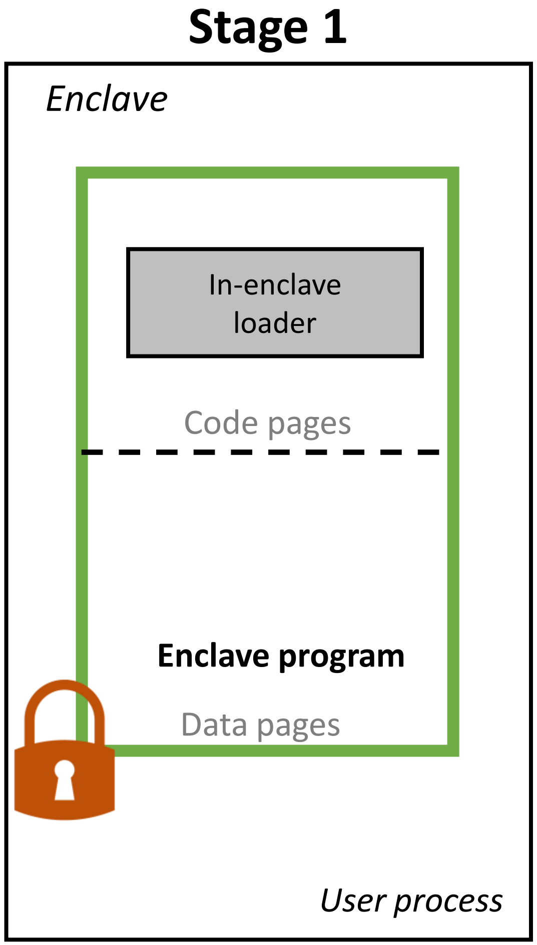
- Visible memory layout
- Small randomization entropy
- No runtime page permission change

Challenges for Mitigation Schemes

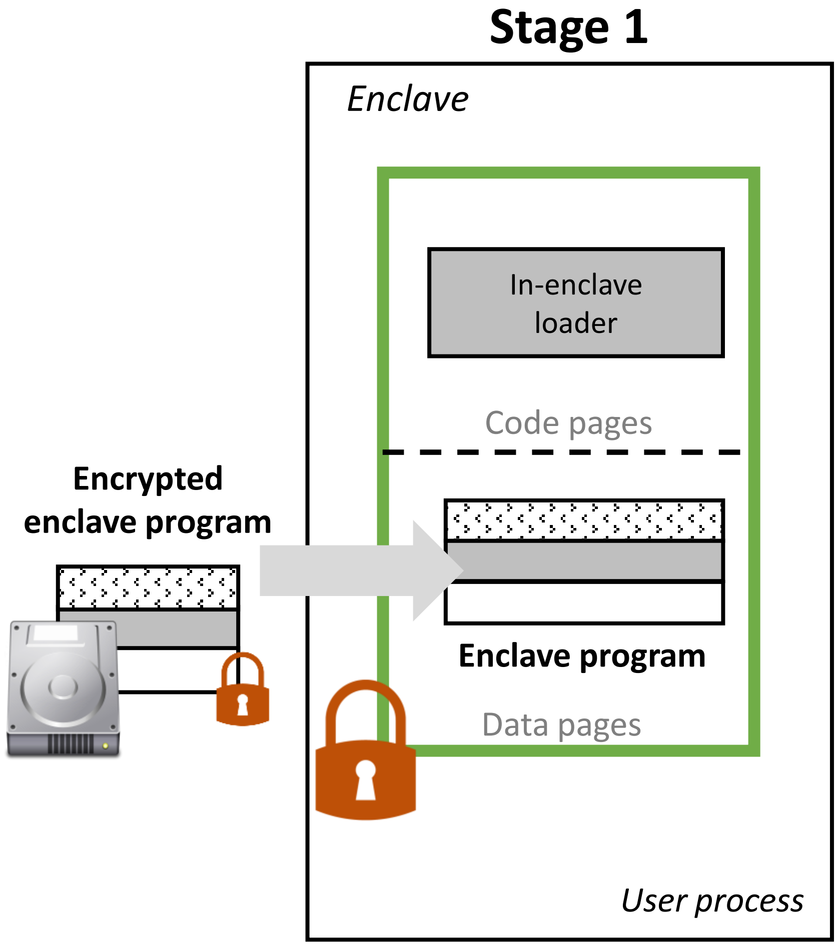
It is non-trivial when an attacker is the kernel:

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

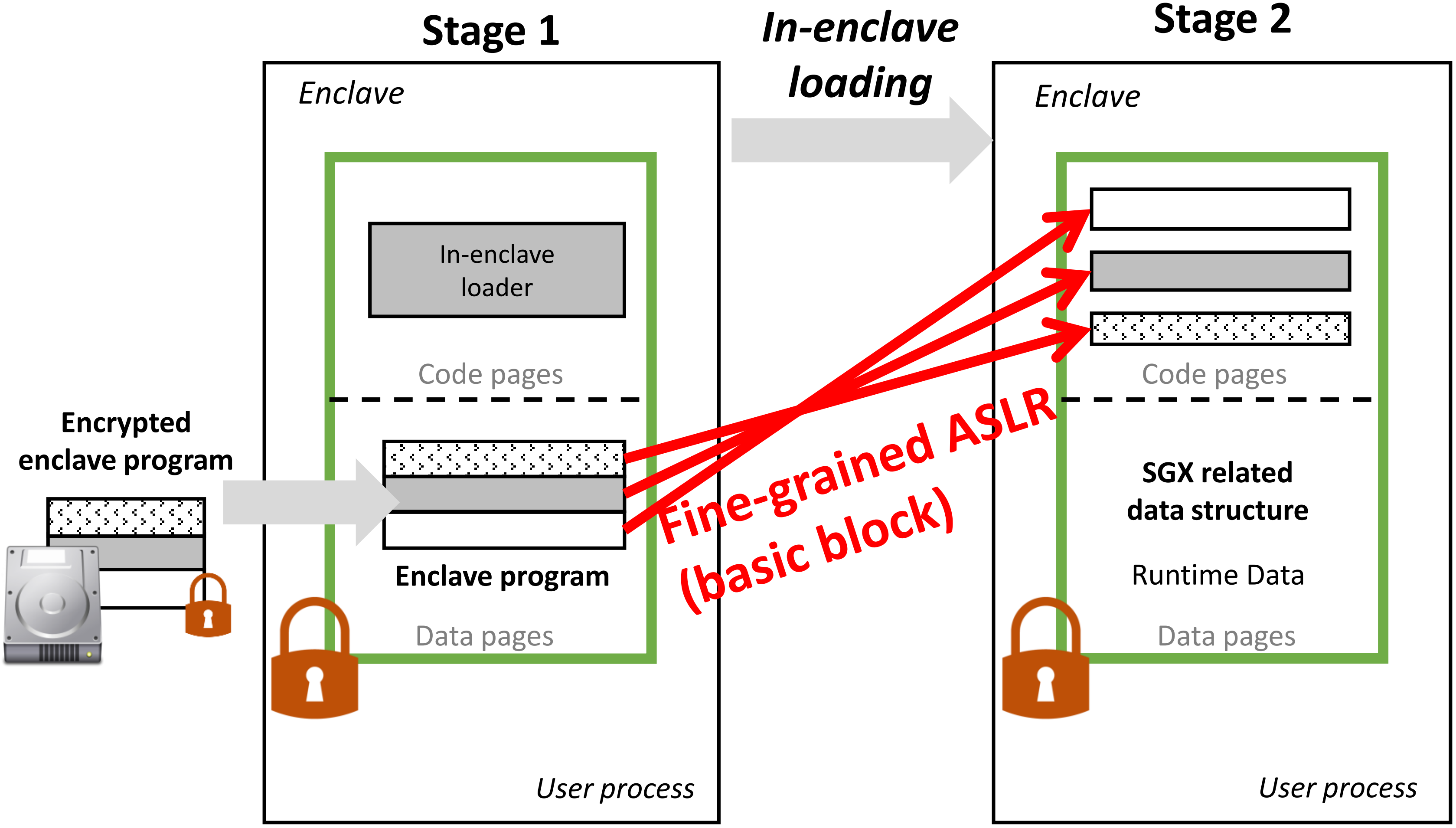
SGX-Shield's Approach: In-enclave Loading



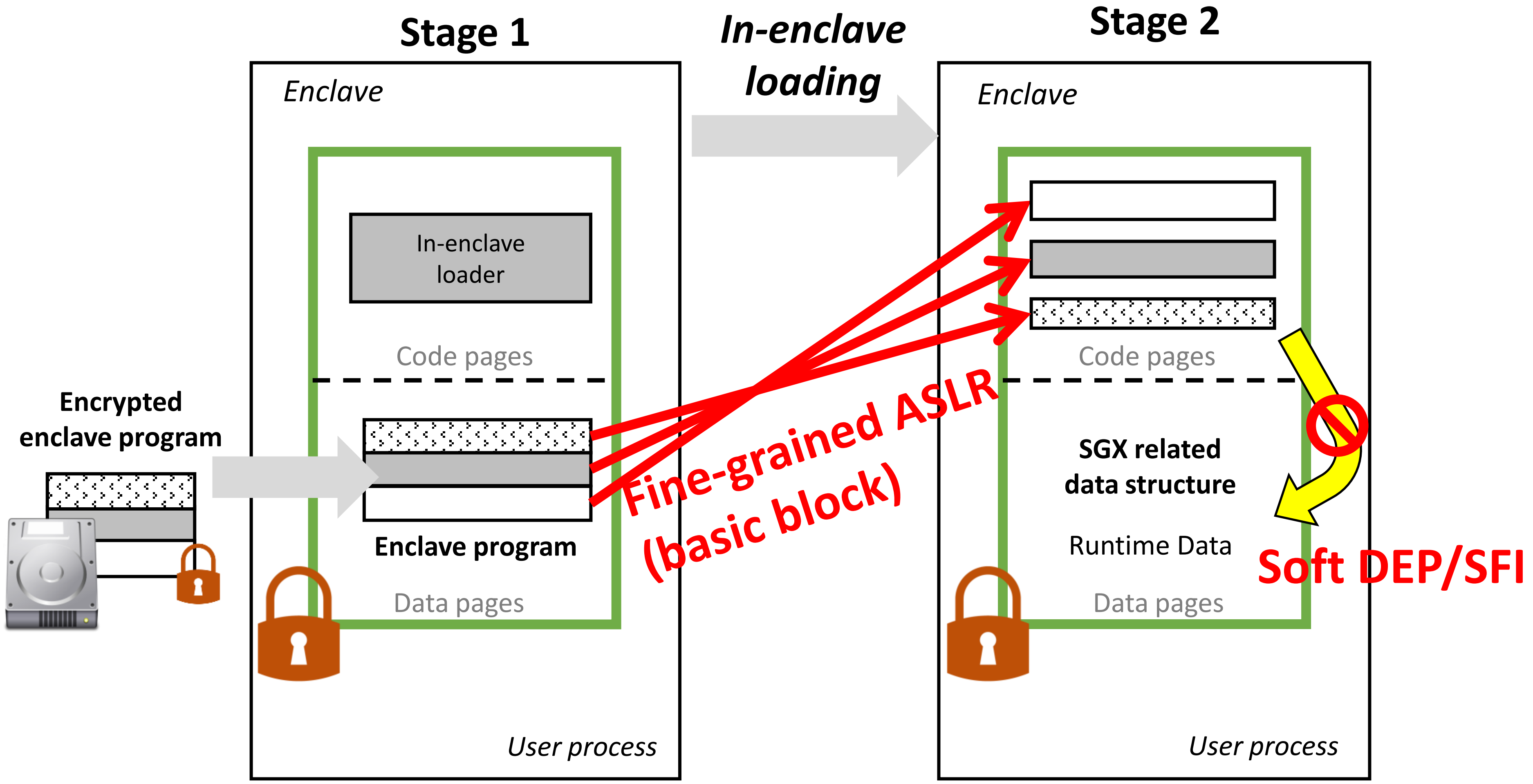
SGX-Shield's Approach: In-enclave Loading



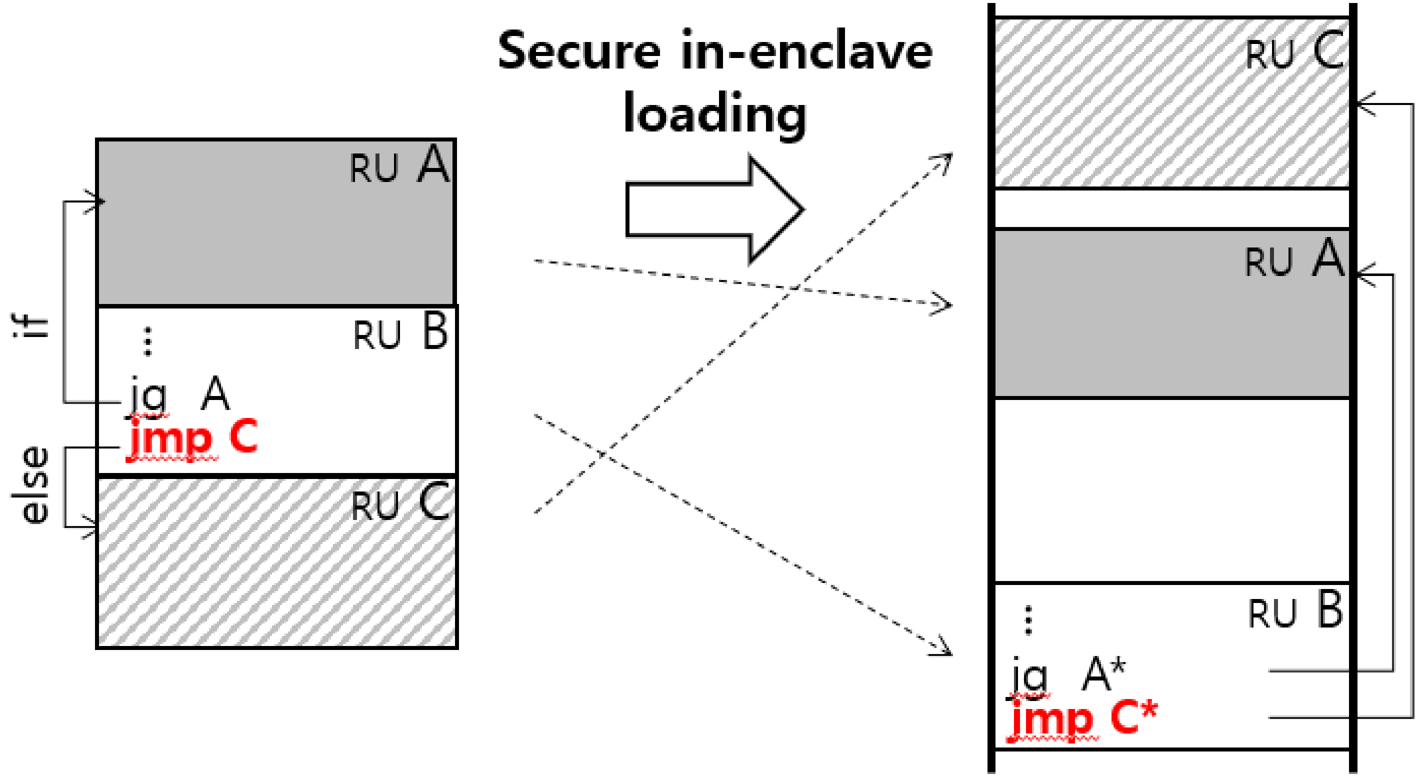
SGX-Shield's Approach: In-enclave Loading



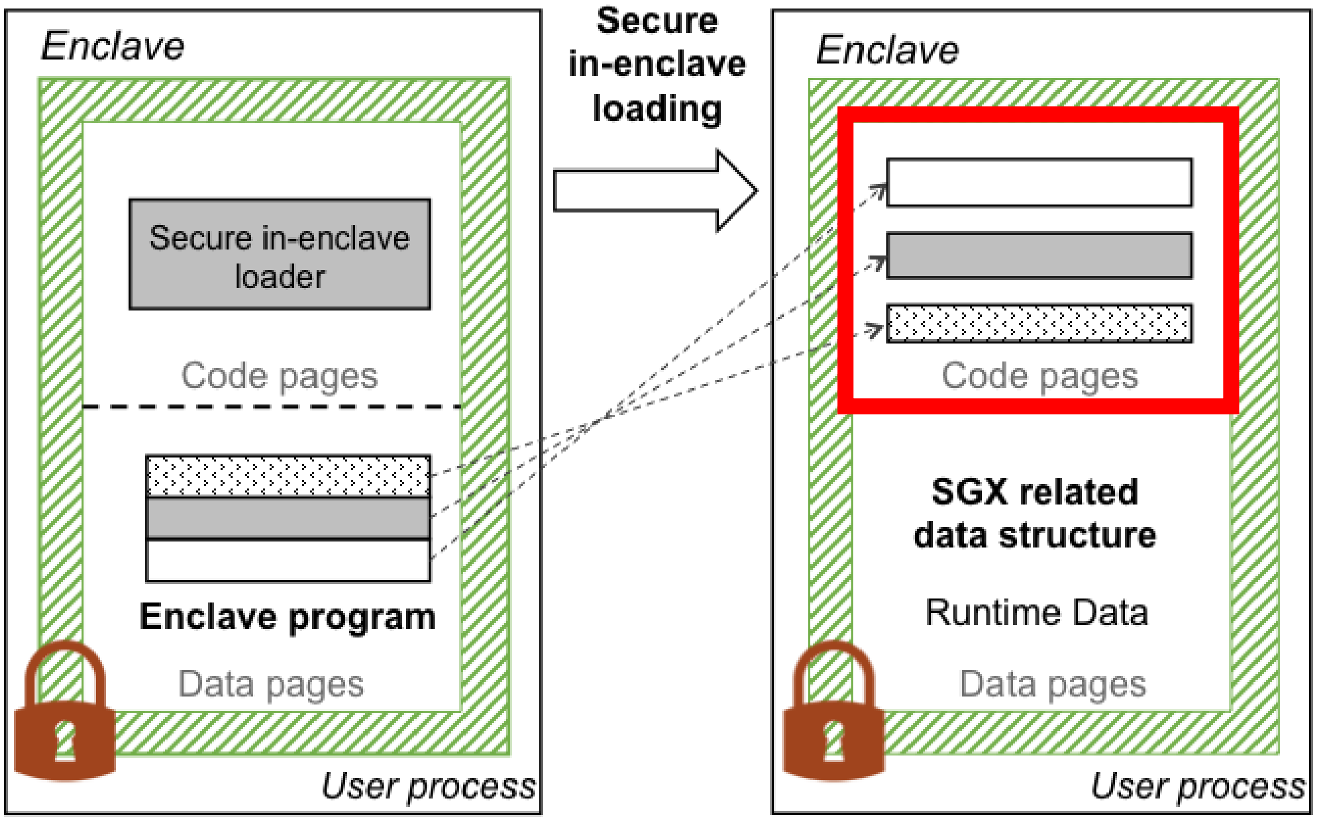
SGX-Shield's Approach: In-enclave Loading



SGX-Shield’s Fine-grained ASLR



No Runtime Permission Change



RWX

SW-based Permission Enforcement (via SFI like Nacl)

	Hardware-based permission		Software+Hardware permission
Out of enclave ↑			
Code of loader	RWX	**Loading**	*No Permission*
Code	RWX	→	X
Data of loader	RW		*No Permission*
Data	RW		RW
Out of enclave ↓			

Virtual address space of an enclave

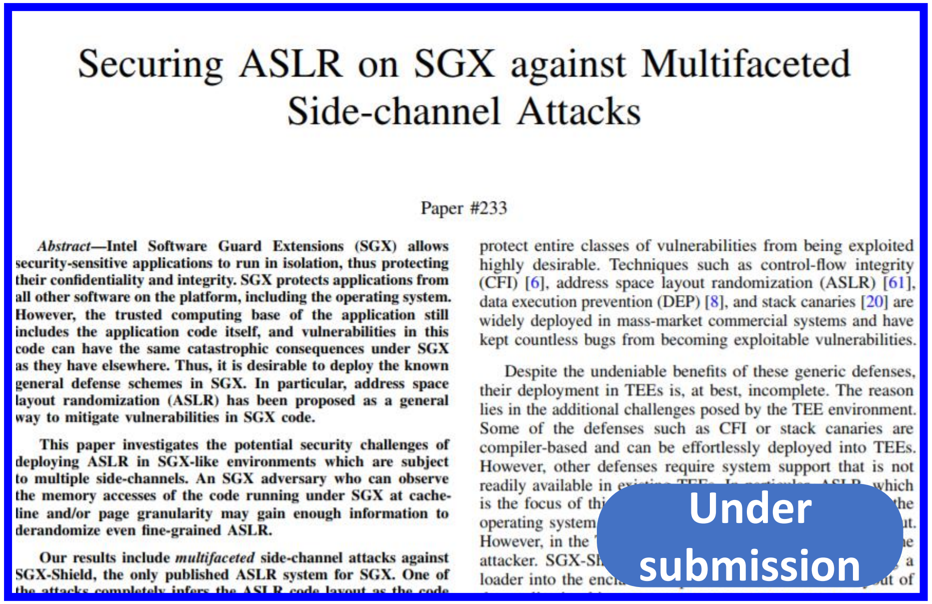
DEMO: SGX-Shield

```
mingwei@sgx3:~/workspace/sgx-attack/SGX-Shield/program$
```

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

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.**

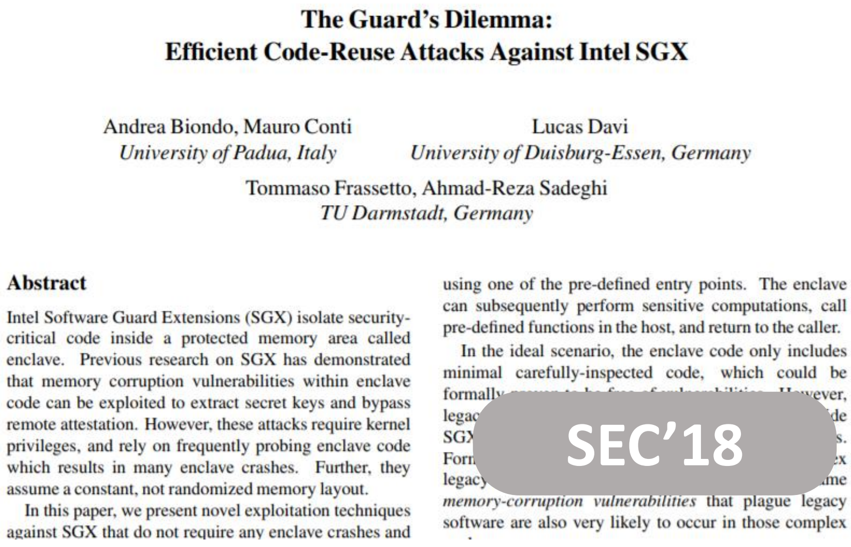
This paper investigates the potential security challenges of deploying ASLR in SGX-like environments which are subject to multiple side-channels. An SGX adversary who can observe the memory accesses of the code running under SGX at cache-line and/or page granularity may gain enough information to derandomize 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 e... which
is the focus of thi... the
operating system... ut.
However, in the ... he
attacker. SGX-Sh... a
loader into the encl... ut of

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.

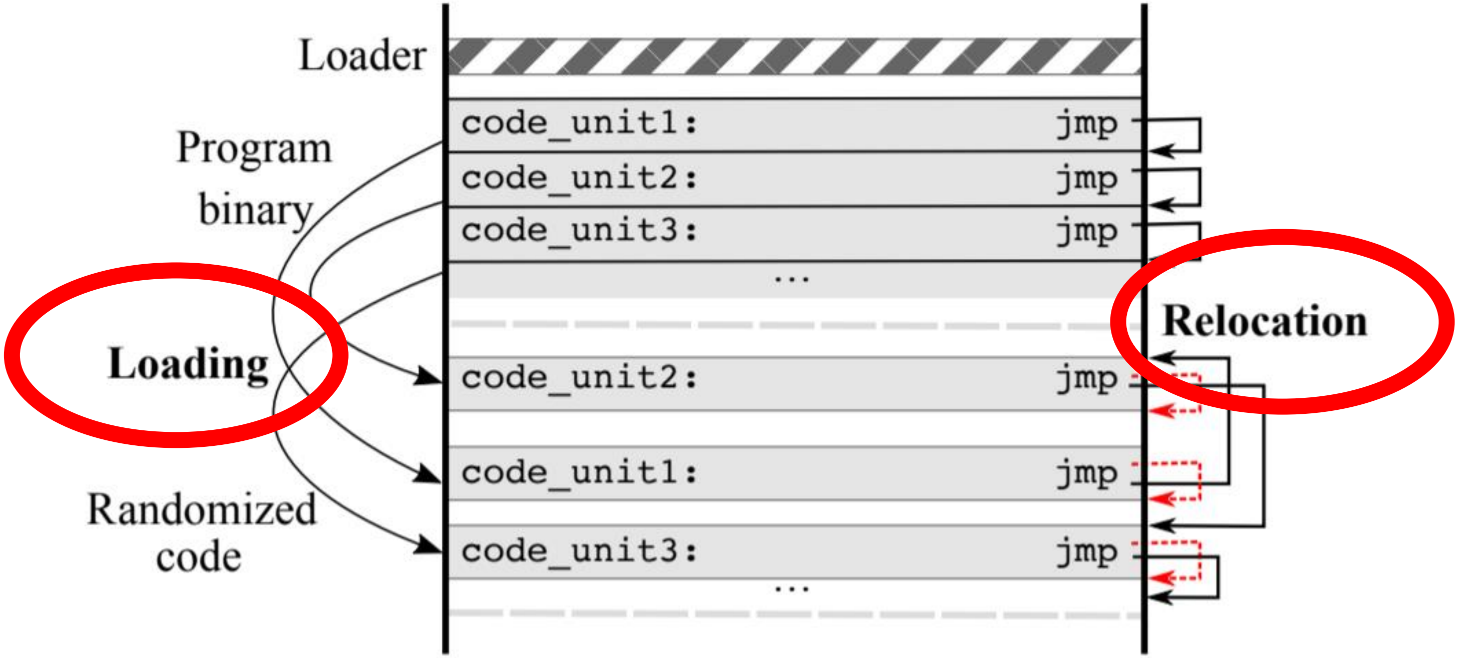
In this paper, we present novel exploitation techniques against SGX that do not require any enclave crashes and

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 ... However,
legac... de
SG... s.
Forn... ex
legacy... me
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

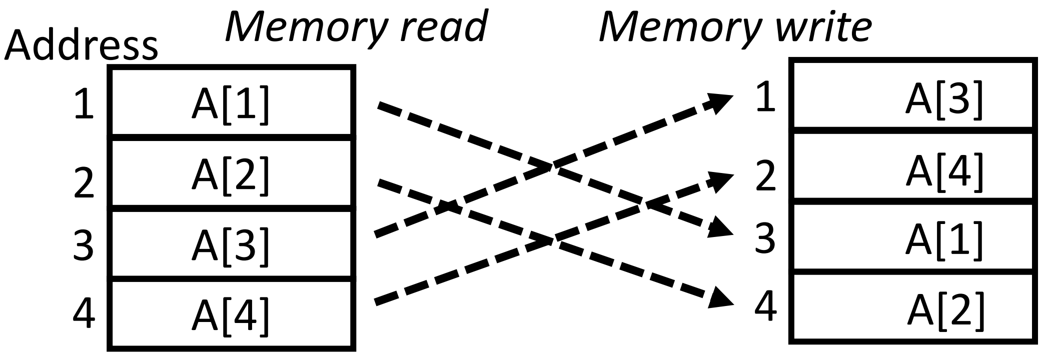
Side-channel observations

1 → 3 (A[1])

2 → 4 (A[2])

3 → 1 (A[3])

4 → 2 (A[4])



1 → A[1]
2 → A[2]
3 → A[3]
4 → A[4]

1 → A[3]
2 → A[4]
3 → A[1]
4 → A[2]

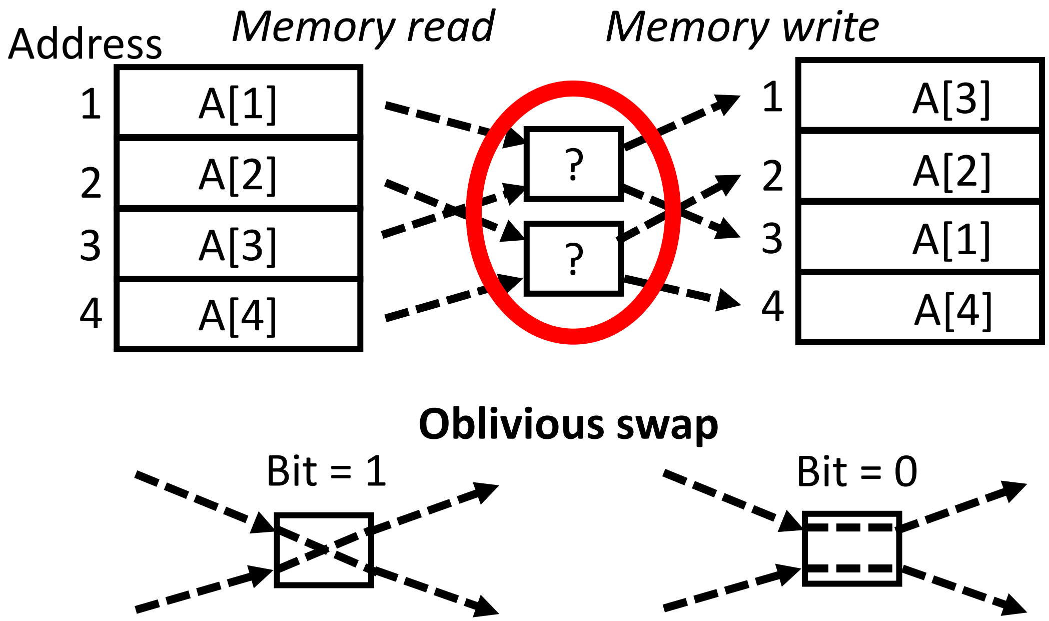
SGX-Armor: Obfuscating Randomization via Oblivious Swap

Side-channel observations

1 → 3 → 1 → 3 (Swap)

2 → 4 → 2 → 4 (No swap)

Swap or not reveals the same patterns



Address

Memory read

Memory write

1 A[1]

2 A[2]

3 A[3]

4 A[4]

?

?

1 A[3]

2 A[2]

3 A[1]

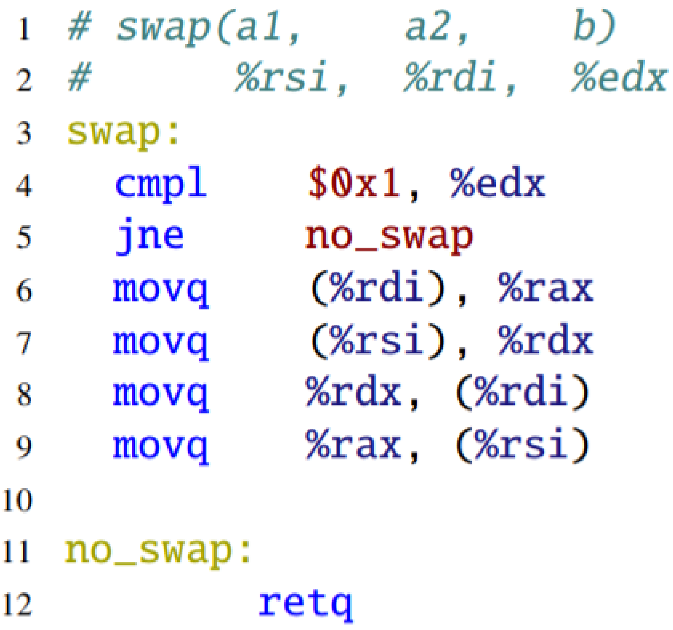
4 A[4]

Oblivious swap

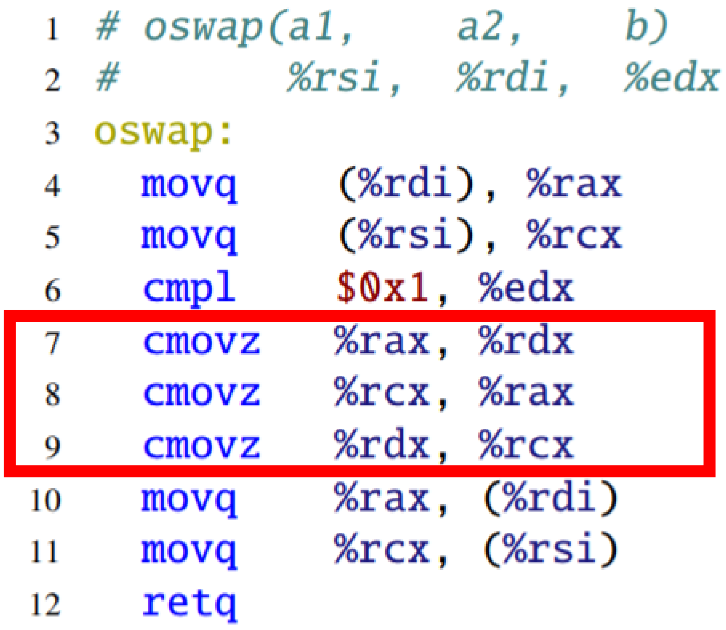
Bit = 1

Bit = 0

Oblivious Swap Primitive



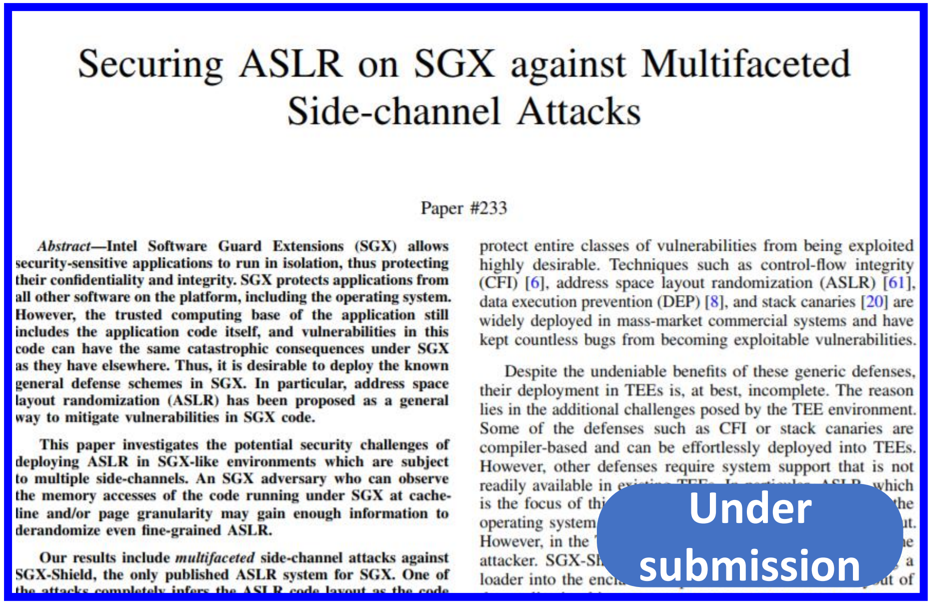
```
# swap(a1,     a2,     b)
#        %rsi,  %rdi,  %edx
swap:
   cmpl     $0x1, %edx
   jne      no_swap
   movq     (%rdi), %rax
   movq     (%rsi), %rdx
   movq     %rdx, (%rdi)
   movq     %rax, (%rsi)

no_swap:
            retq
```



```
# oswap(a1,     a2,     b)
#         %rsi,  %rdi,  %edx
oswap:
   movq     (%rdi), %rax
   movq     (%rsi), %rcx
   cmpl     $0x1, %edx
   cmovz    %rax, %rdx
   cmovz    %rcx, %rax
   cmovz    %rdx, %rcx
   movq     %rax, (%rdi)
   movq     %rcx, (%rsi)
   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.**

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

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
which
is the focus of thi
the
operating system
ut.
However, in the
he
attacker. SGX-Sh
a
loader into the encl
ut of

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.

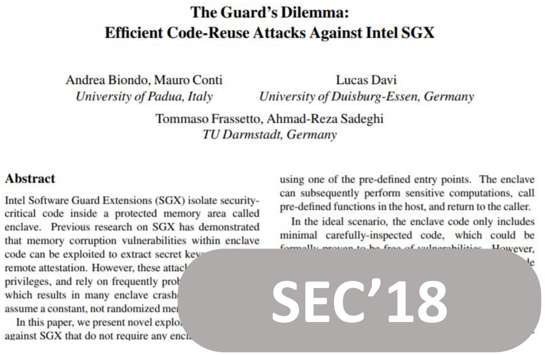
In this paper, we present novel exploitation techniques against SGX that do not require any enclave crashes and

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
However,
legac
de
SG
s.
Forr
x
legacy
me
memory-corruption vulnerabilities that plague legacy software are also very likely to occur in those complex

SEC'18

Another ROP



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 key
remote attestation. However, these attac
privileges, and rely on frequently prob
which results in many enclave crash
assume a constant, not randomized me

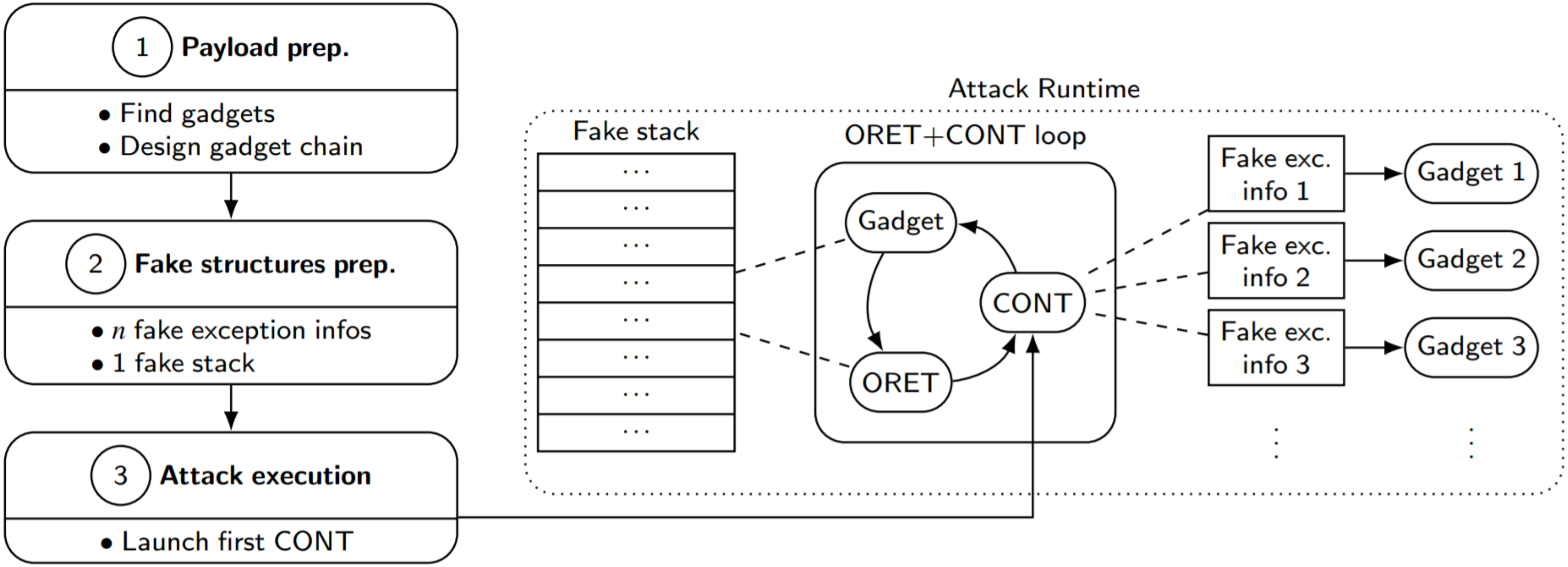
In this paper, we present novel explo
against SGX that do not require any encl

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 proven to be free of vulnerabilities. However,

SEC'18

- 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;
};
```

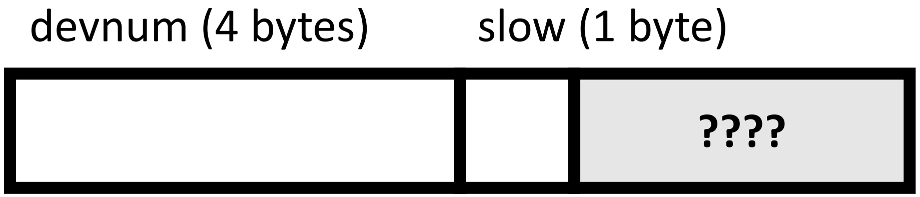
Uninitialized Padding Problem

```
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;
};
```



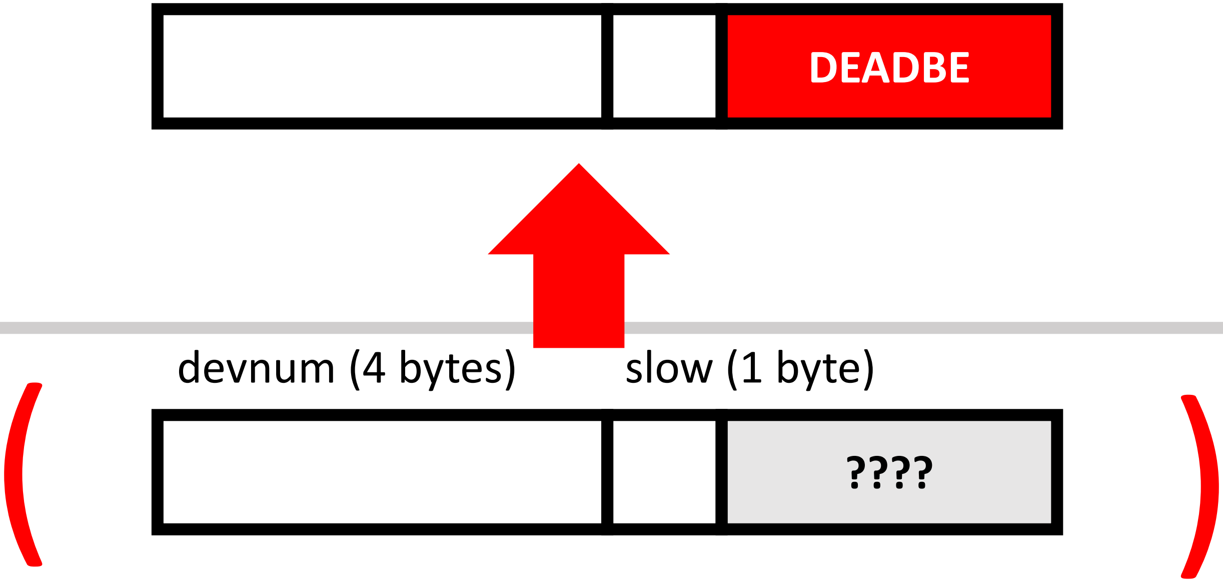
devnum (4 bytes)

slow (1 byte)

????

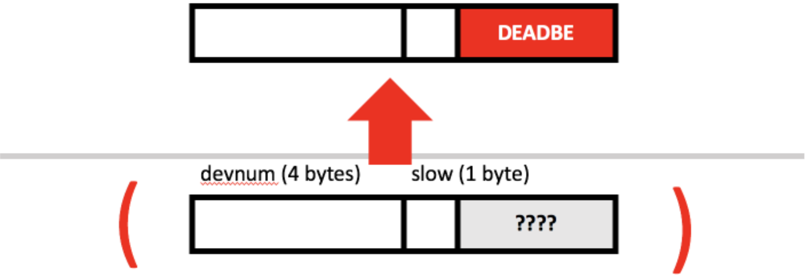
```
struct usbdevfs_connectinfo {
    .devnum = 1,
    .slow = 0,
};
```

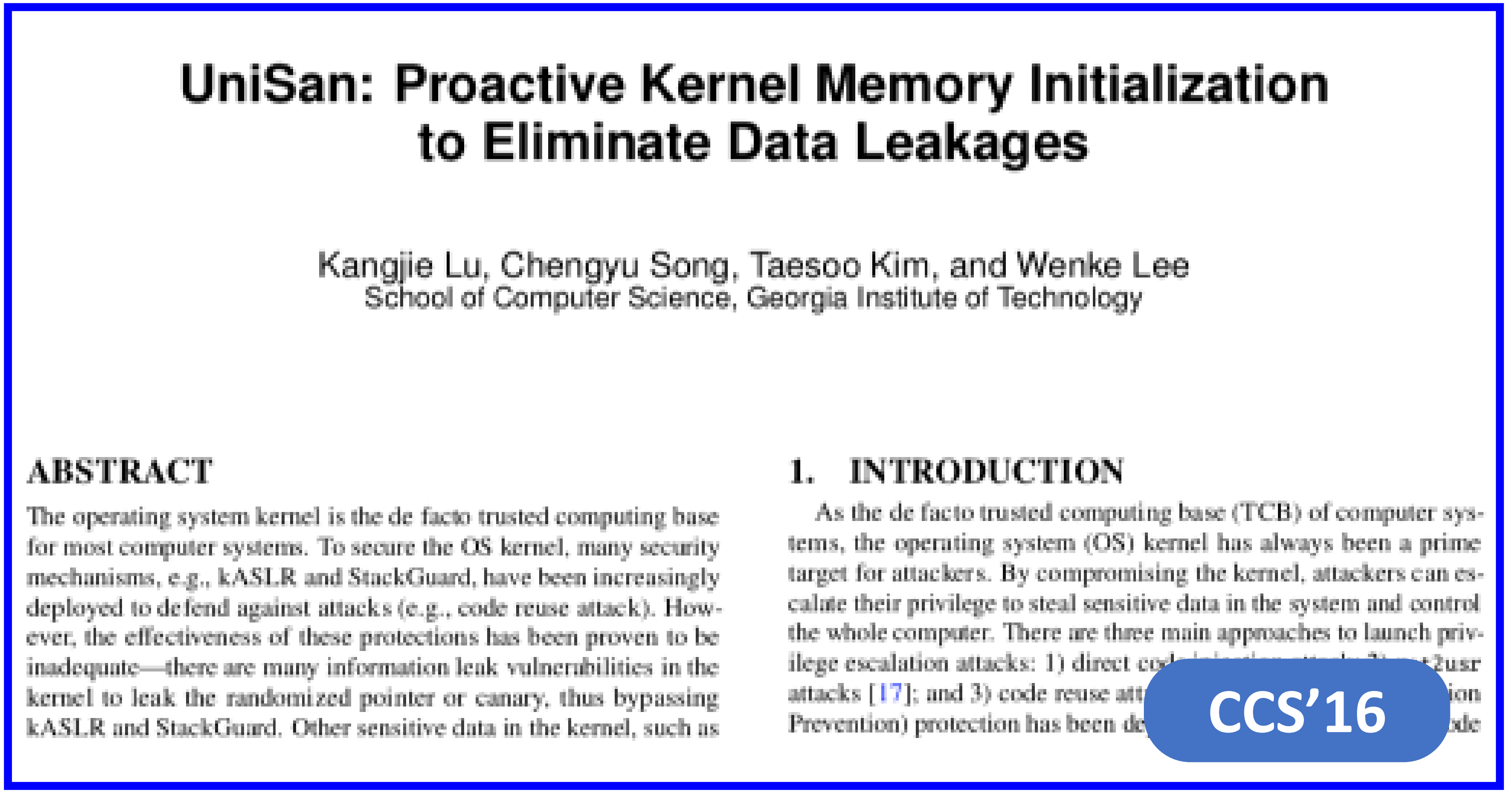
Uninitialized Padding Problem



```
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 co 2usr attacks [17]; and 3) code reuse att ion Prevention) protection has been de ode

CCS'16

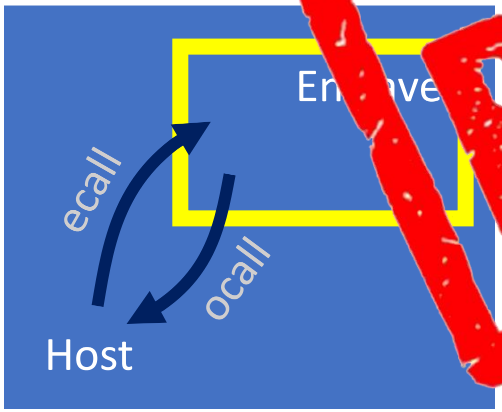
Ecall/Ocall: EDL Interface for SGX

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

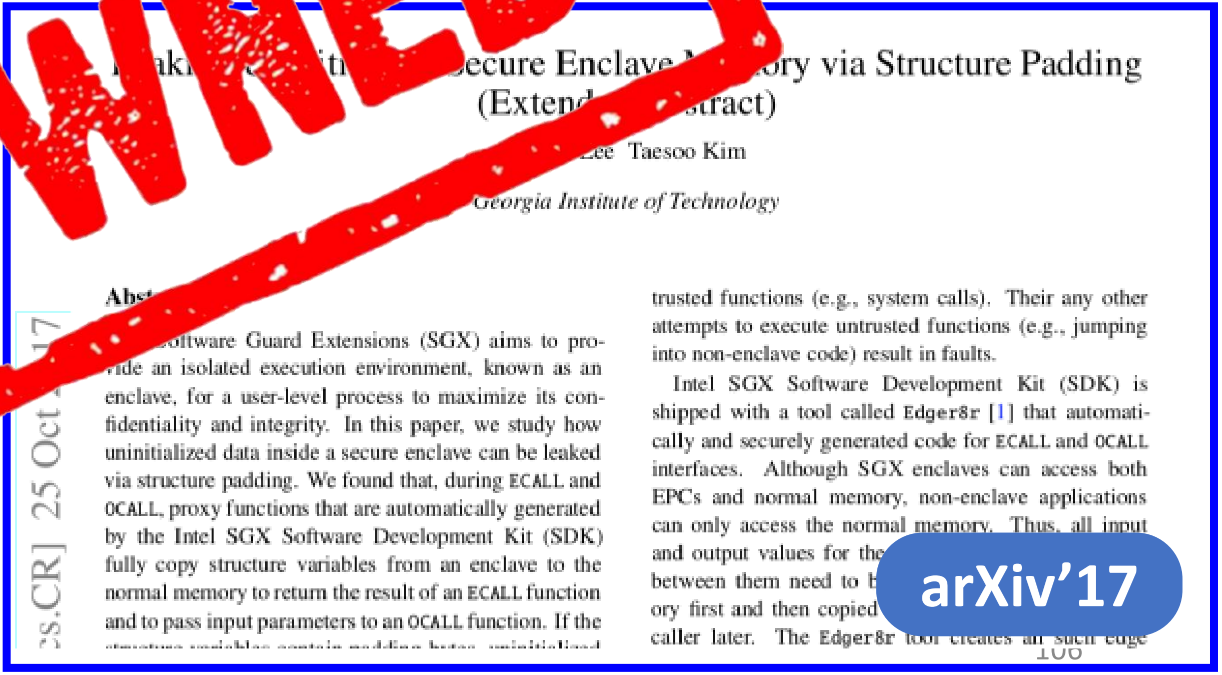
If there is a padding issue in test_struct, it leaks (or inject) potentially sensitive data (e.g., *a private key like HeartBleed*)

Ecall/Ocall: EDL Interface for SGX

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



PWNED!



Leaking [illegible] Secure Enclave Memory via Structure Padding (Extended Abstract)

[illegible] Lee Taesoo Kim

Georgia Institute of Technology

Abs[illegible][illegible] 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

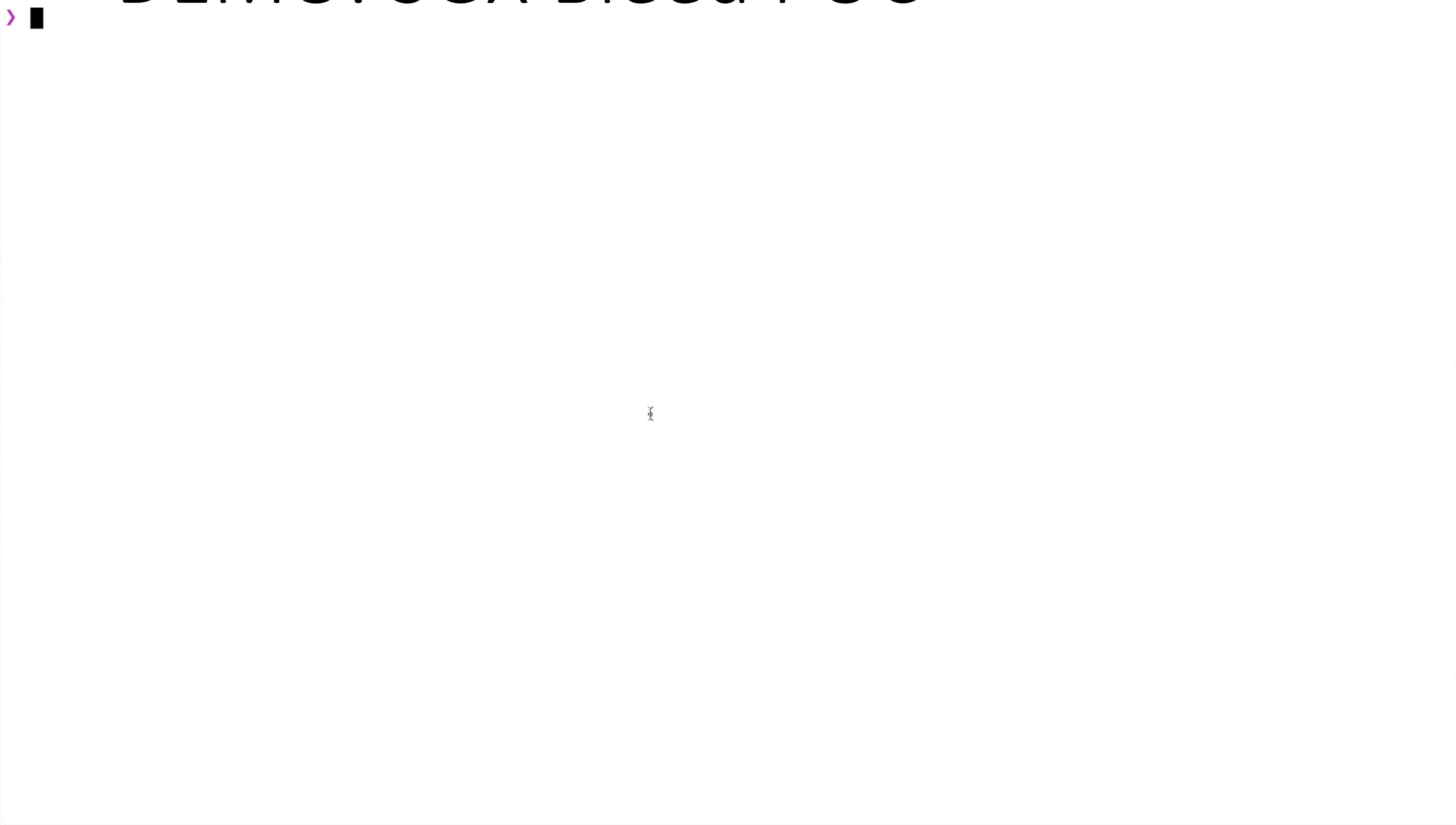
trusted functions (e.g., system calls). Their any other attempts to execute untrusted functions (e.g., jumping into non-enclave code) result in faults.

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 [illegible] between them need to b[illegible] ory first and then copied [illegible] caller later. The Edger8r [illegible]

[cs.CR] 25 Oct [illegible]17

arXiv'17

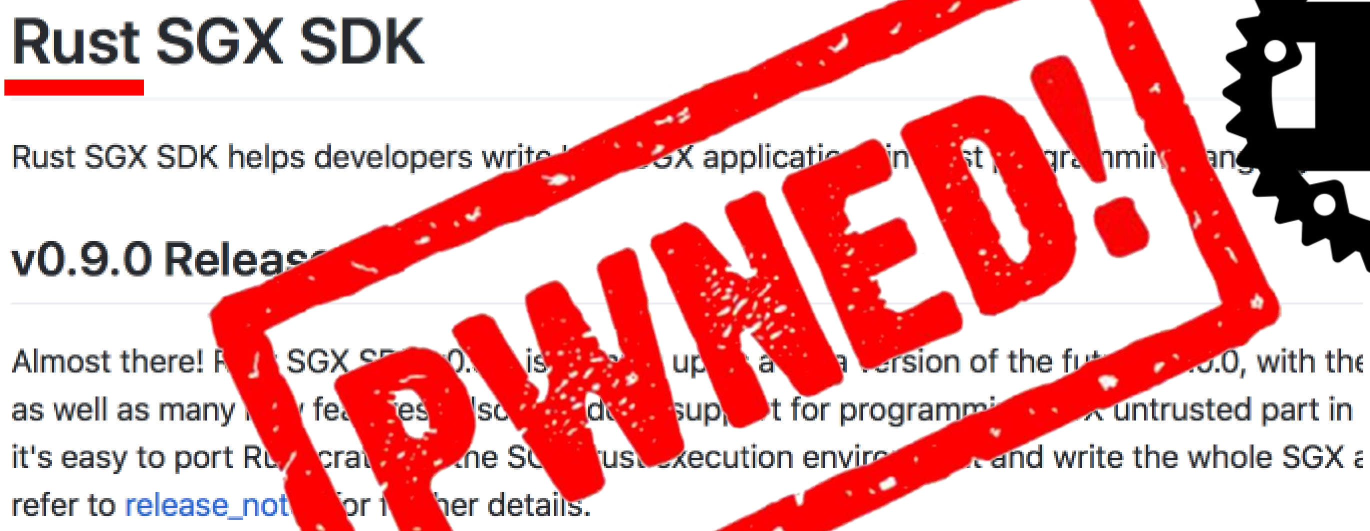
DEMO: SGX Bleed POC



```
❯
```

https://github.com/sslab-gatech/unisan

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



Rust SGX SDK

Rust SGX SDK helps developers write [illegible] SGX applicatio[illegible] in [illegible]st [illegible] gra[illegible]mmi[illegible] ang[illegible]

v0.9.0 Releas[illegible]

Almost there! R[illegible] SGX S[illegible]0.[illegible] is [illegible] up[illegible] a[illegible] [illegible]ersion of the fu[illegible].0.0, with the as well as many [illegible] fea[illegible]es [illegible]lso [illegible]d[illegible] sup[illegible]t for programm[illegible] untrusted part in it's easy to port Ru[illegible] crat[illegible] the S[illegible] rust execution envir[illegible] and write the whole SGX a refer to release_not[illegible] for f[illegible]her details.

PWNED!

Implication 2: Using certified C compilers doesn’t help neither

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

> §6.2.6.1/6
>
> When a value is stored in an object of structure (...), the bytes of the object representation that correspond to any padding bytes take unspecified values

PWNED!

New Attack Vectors

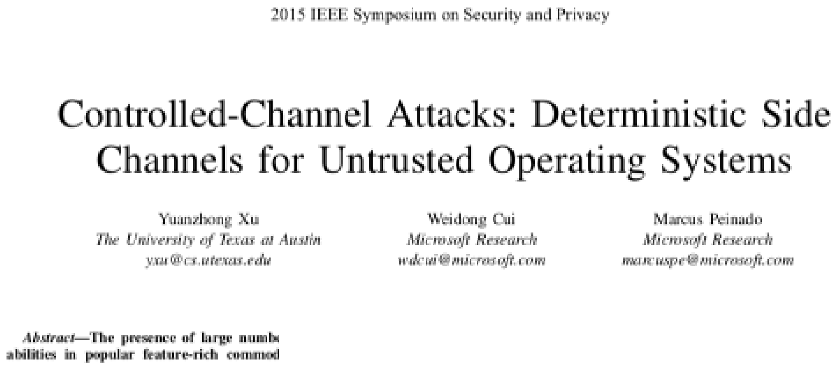
- Page table attack
- Branch shadowing attack
- Rowhammer against SGX
- L1 terminal fault against SGX (i.e., Foreshadow)

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

Weidong Cui
Microsoft Research
wdcui@microsoft.com

Marcus Peinado
Microsoft Research
marcuspe@microsoft.com

Abstract—The presence of large numb abilities in popular feature-rich commod has inspired a long line of work on excl

SP'15

Telling Your Secrets Without Page Faul Stealthy Page Table-Based Attacks on Enclaved

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

Nico Weichbrodt
IBR DS, TU Braunschweig
weichbr@ibr.cs.tu-bs.de

IBR
ka

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

Raoul Strac
imec-DistriNet, K
raoul.strackx@cs.kuleuven.be

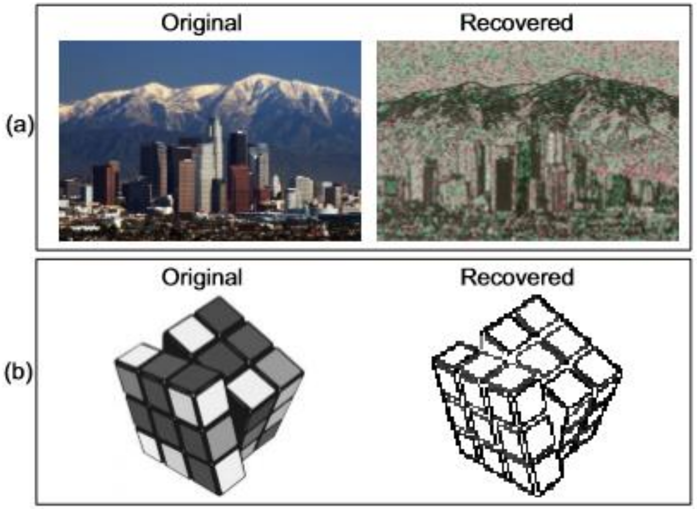
Abstract

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

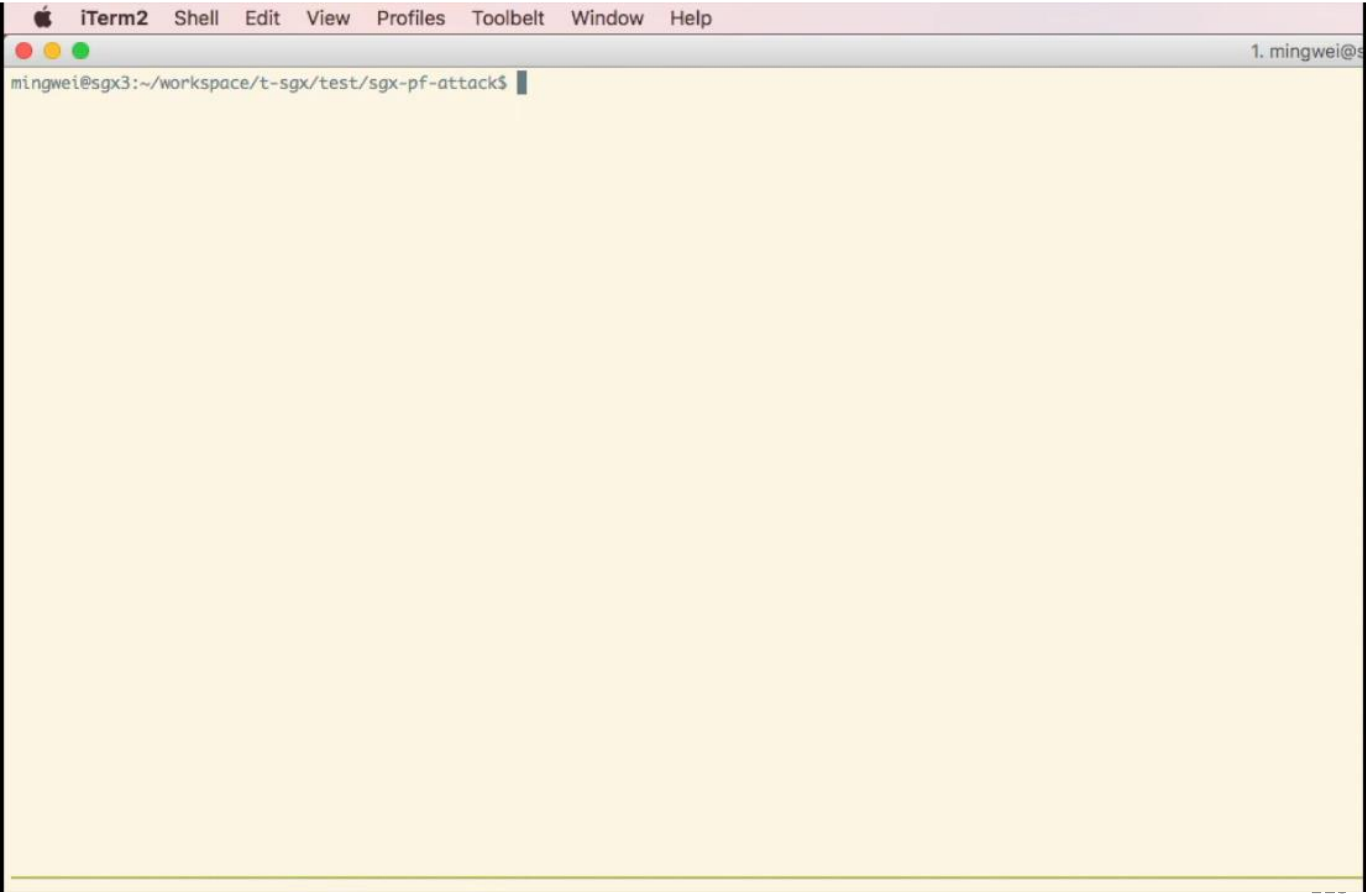
ware to make it rel legacy applications wit

An essential hardware preve ing or writing a

Sec'17



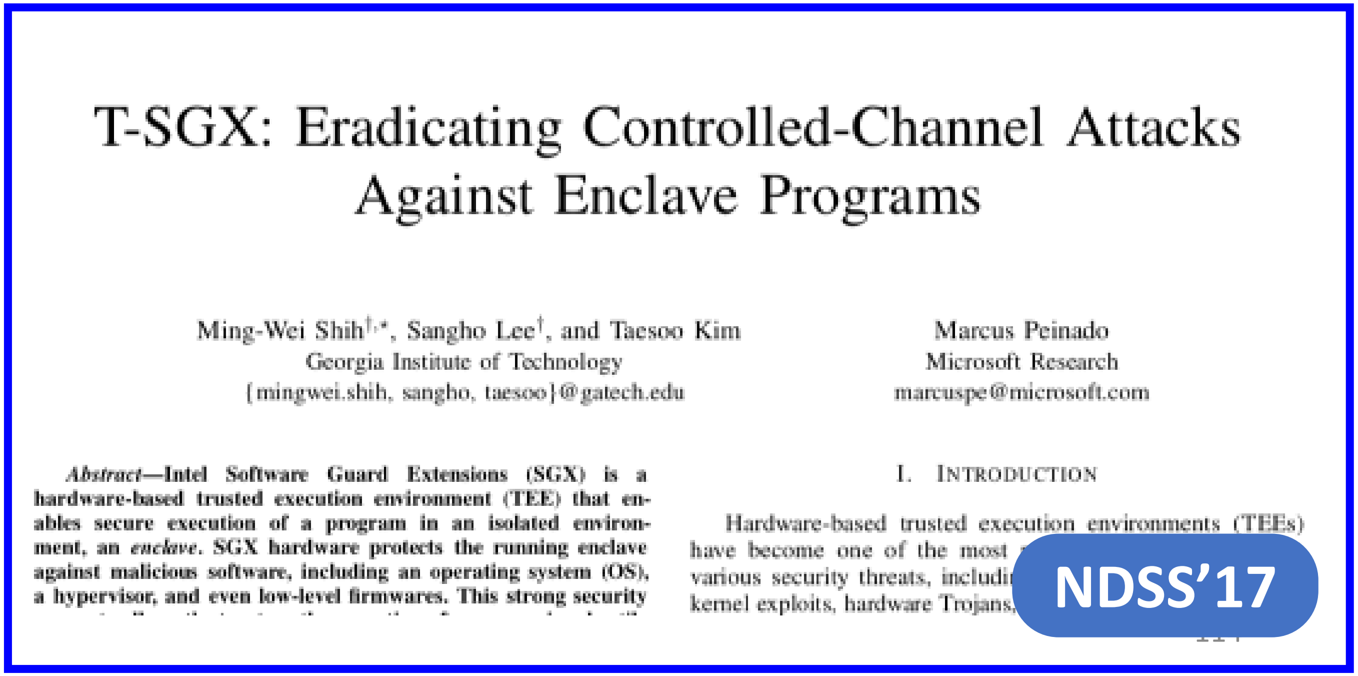
DEMO: Page Fault Attack



```
mingwei@sgx3:~/workspace/t-sgx/test/sgx-pf-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[†,*], Sangho Lee[†], and Taesoo Kim
Georgia Institute of Technology
{mingwei.shih, sangho, taesoo}@gatech.edu

Marcus Peinado
Microsoft Research
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

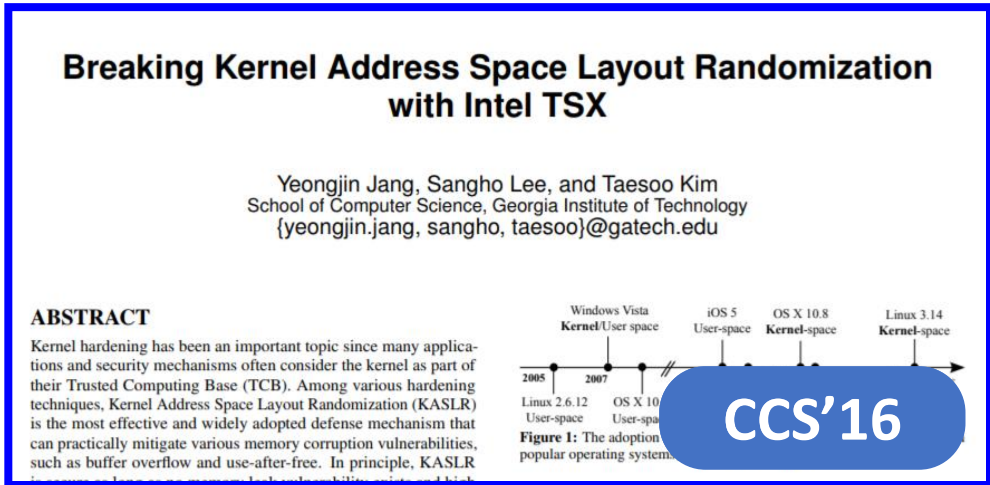
Hardware-based trusted execution environments (TEEs) have become one of the most various security threats, includi kernel exploits, hardware Trojans,

NDSS'17

Key Idea: TSX Isolates Faults!

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

```
unsigned status;

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



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

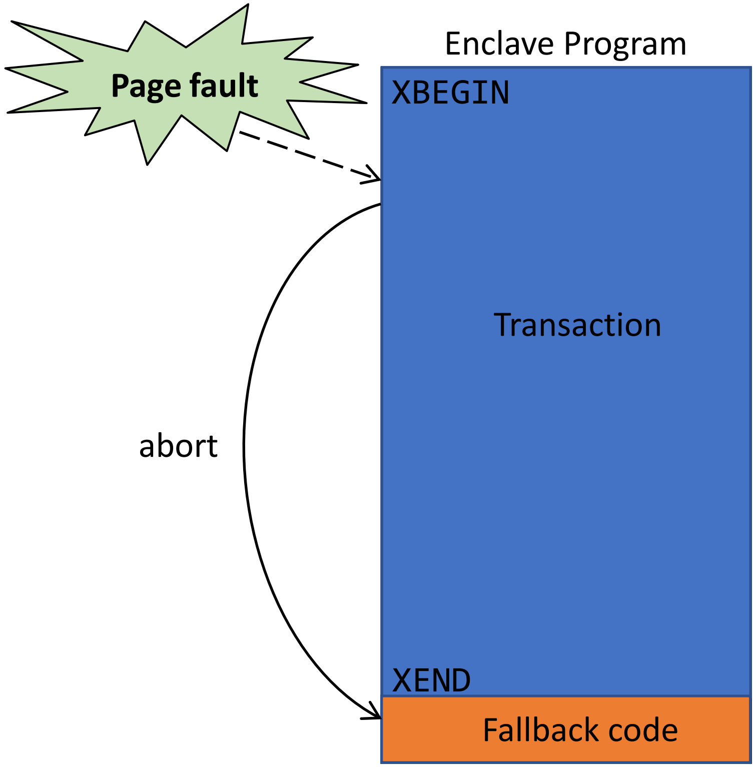
Kernel hardening has been an important topic since many applications and security mechanisms often consider the kernel as part of their Trusted Computing Base (TCB). Among various hardening techniques, Kernel Address Space Layout Randomization (KASLR) is the most effective and widely adopted defense mechanism that can practically mitigate various memory corruption vulnerabilities, such as buffer overflow and use-after-free. In principle, KASLR

Figure 1: The adoption
popular operating system

CCS'16

A Strawman Solution

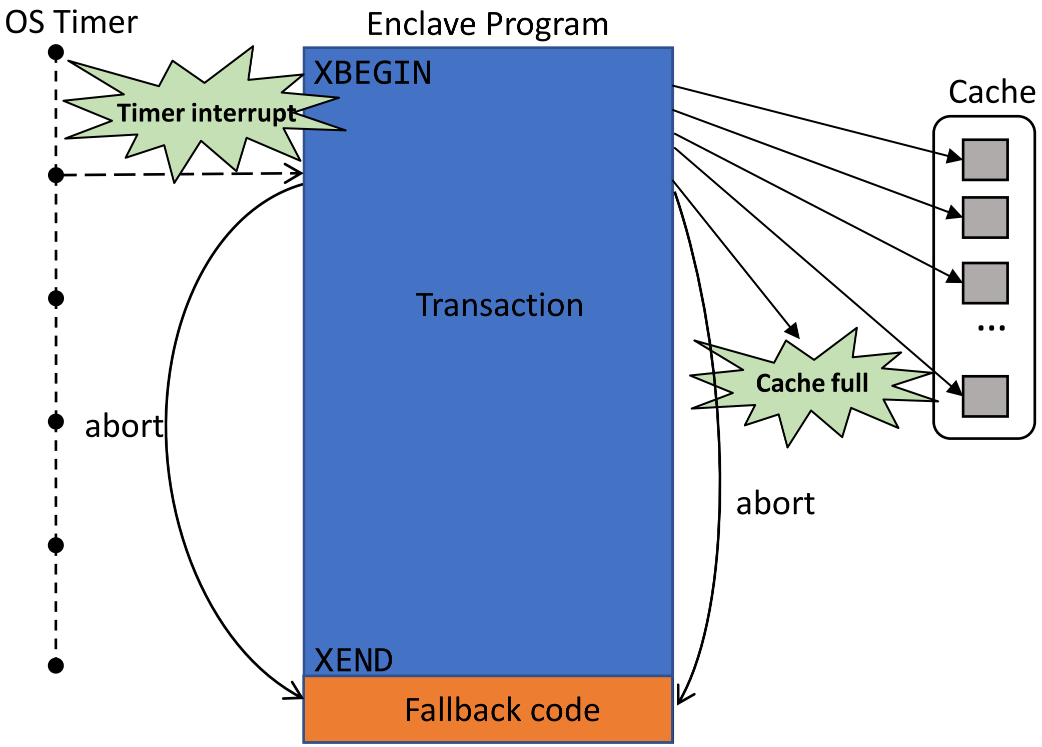
- Protect the entire program with TSX!



Challenge: Not Progressing!

1) Timer interrupt (i.e., external faults)

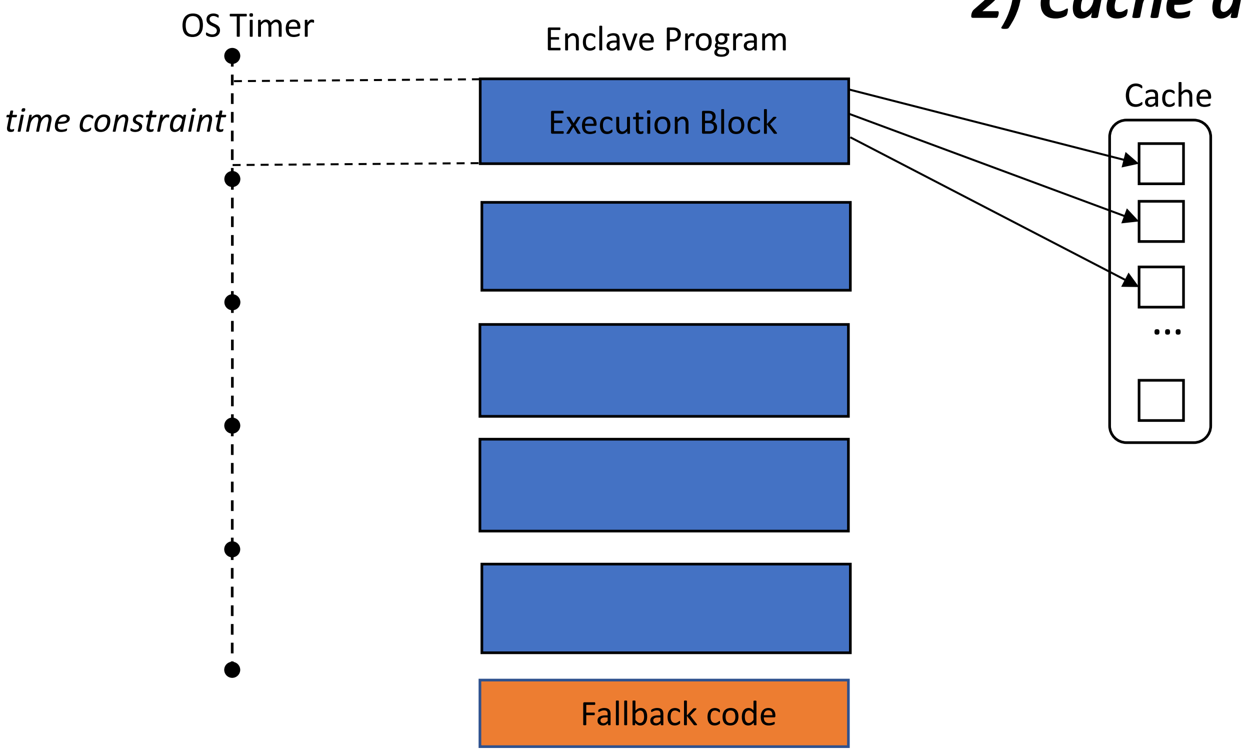
2) False TSX aborts (e.g., capacity)



Approach: Smaller Execution Units

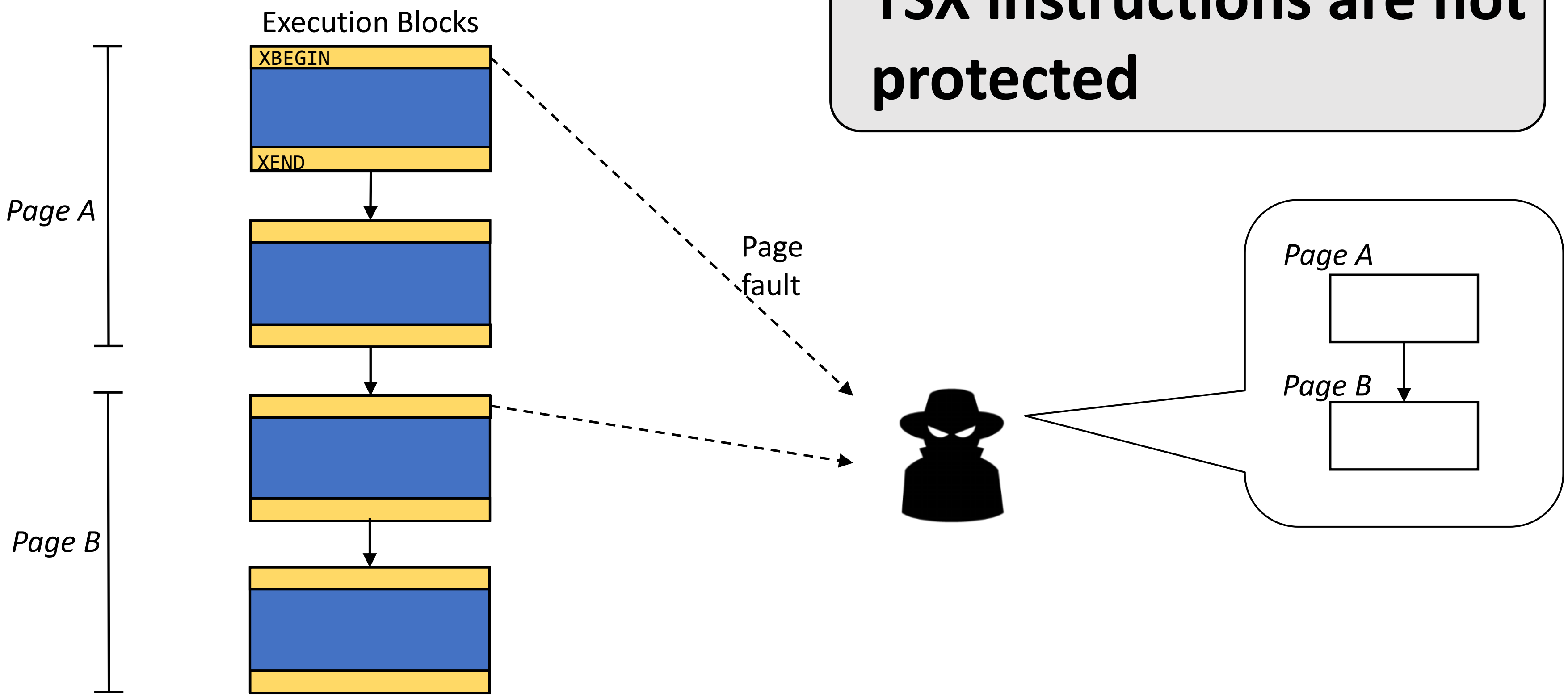
1) Execution time analysis

2) Cache analysis



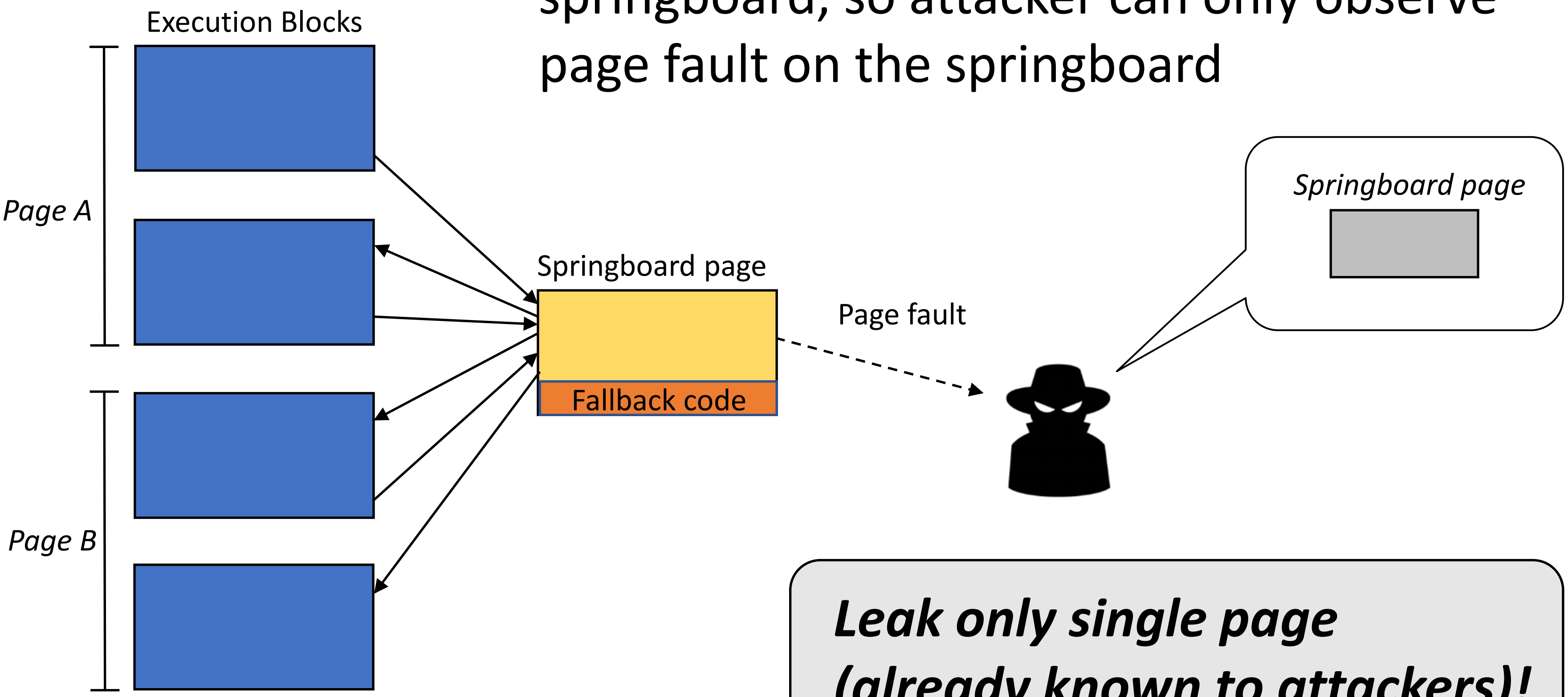
This design still leaks information

TSX instructions are not protected



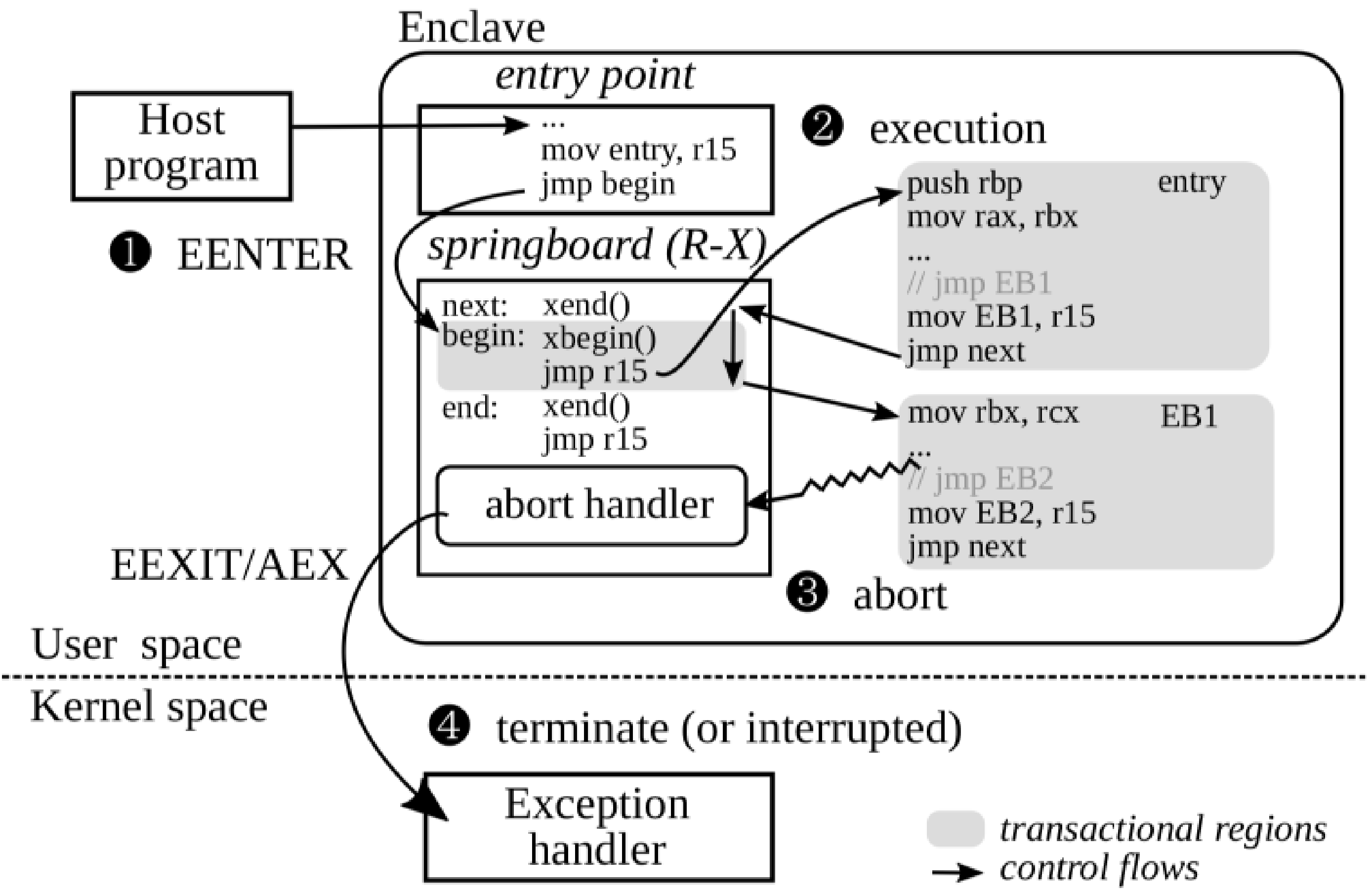
Solution: Springboard

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



Leak only single page (already known to attackers)!

Design of T-SGX (Compiler)



```
...
mov entry, r15
jmp begin
```

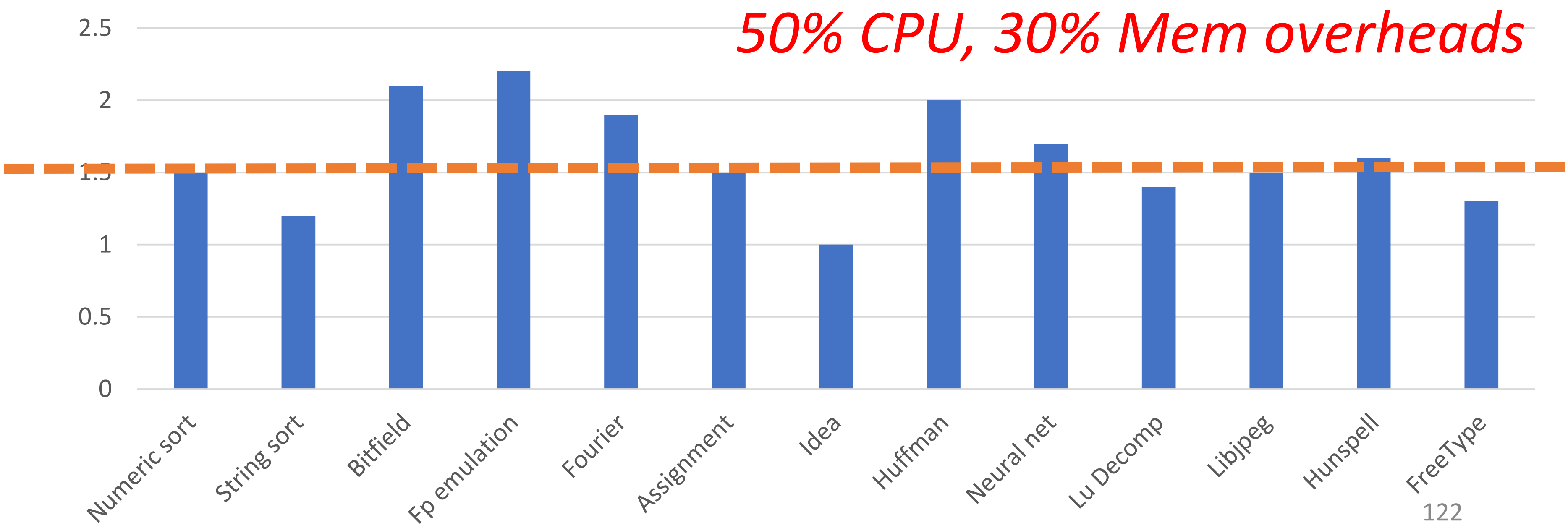
```
next:   xend()
begin:  xbegin()
        jmp r15
end:    xend()
        jmp r15
```

```
push rbp
mov rax, rbx
...
// jmp EB1
mov EB1, r15
jmp next
```

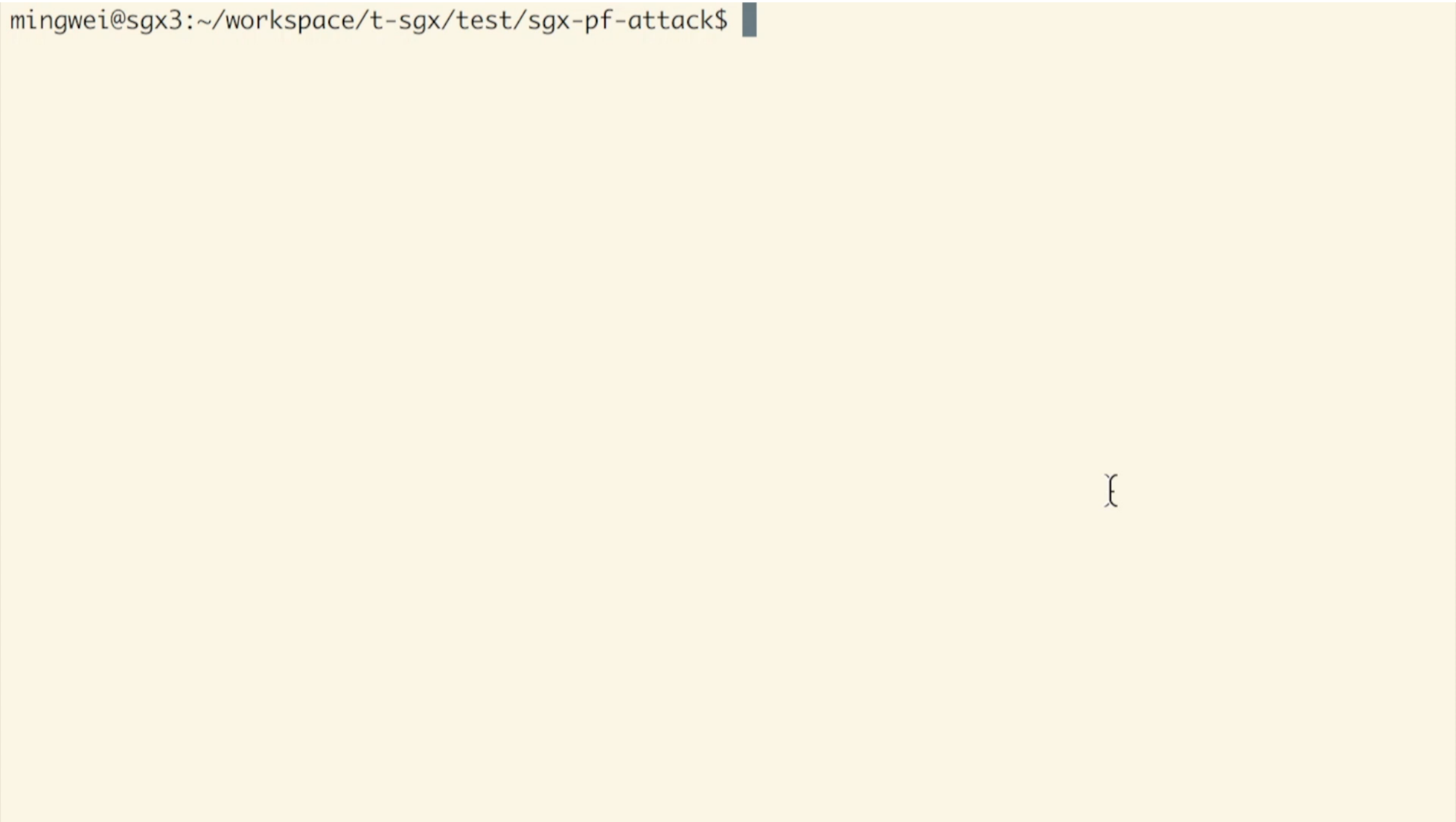
```
mov rbx, rcx
...
// jmp EB2
mov EB2, r15
jmp next
```

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$
```

https://github.com/sslab-gatech/t-sgx

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

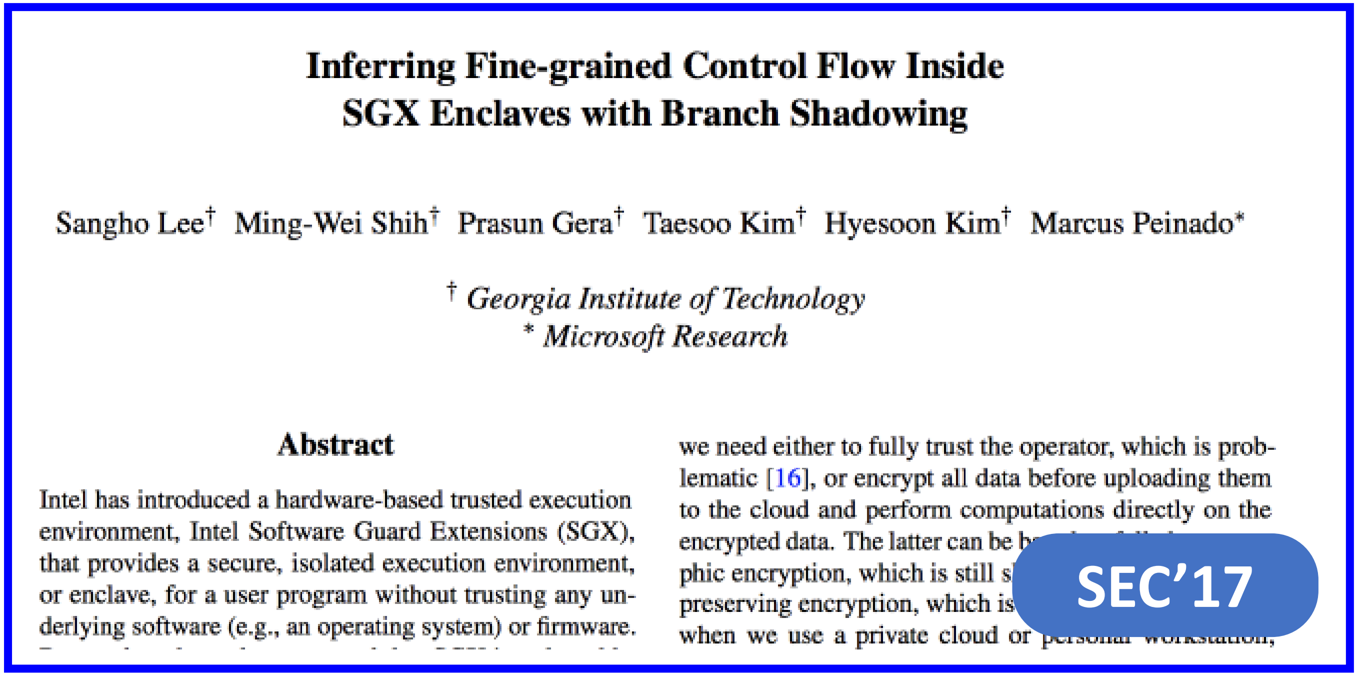
- Finer-grained, yet noise-free!
  (unlike page faults / cache attacks, respectively)

- 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[†] Ming-Wei Shih[†] Prasun Gera[†] Taesoo Kim[†] Hyesoon Kim[†] Marcus Peinado[*]

[†] *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 b
phic encryption, which is still s
preserving encryption, which is
when we use a private cloud or

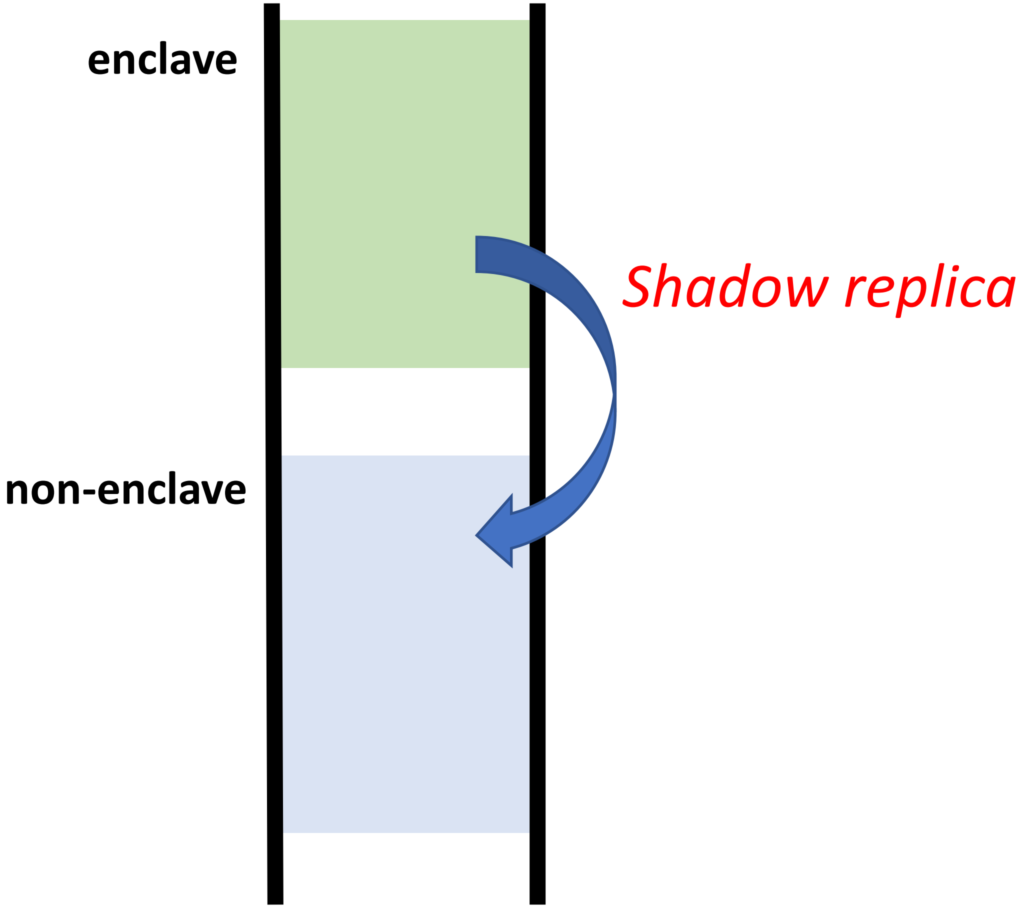
SEC'17

Idea: Exploiting New HW Features

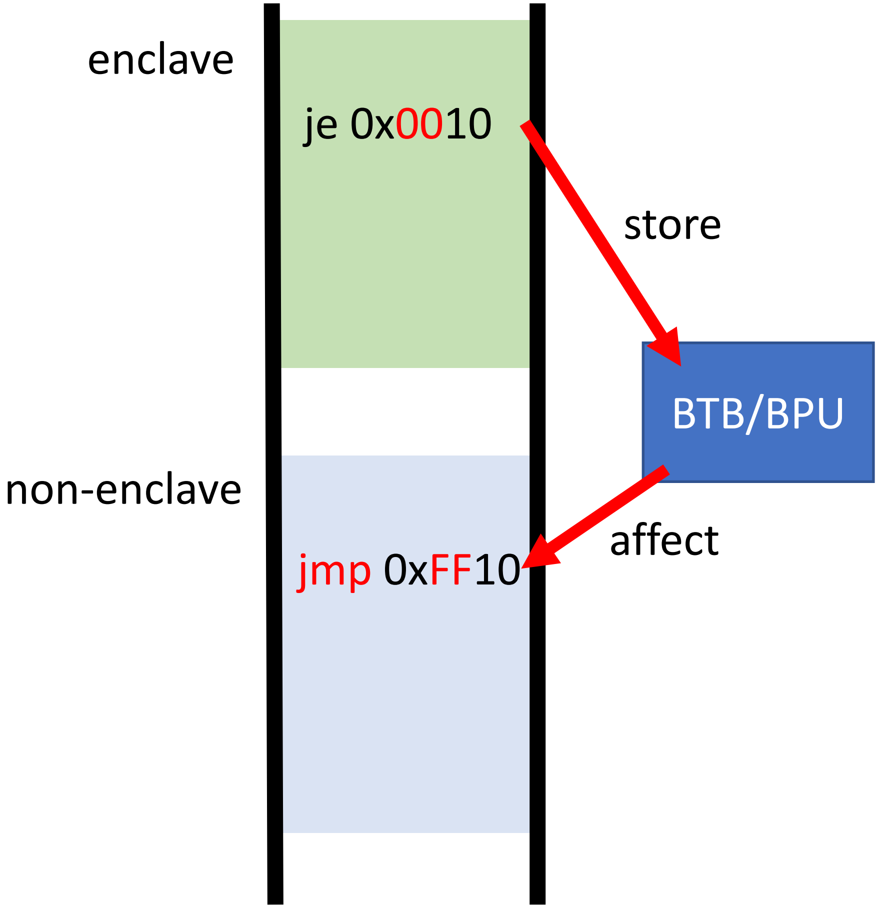
- Intel Skylake (and Broadwell) introduced two new debugging features that report prediction results
  - Last Branch Record (LBR)
  - Intel Processor Trace (PT)

→ But only for ***non-enclave*** programs
(or enclave on a debug mode)

Our Approach: Branch Shadowing

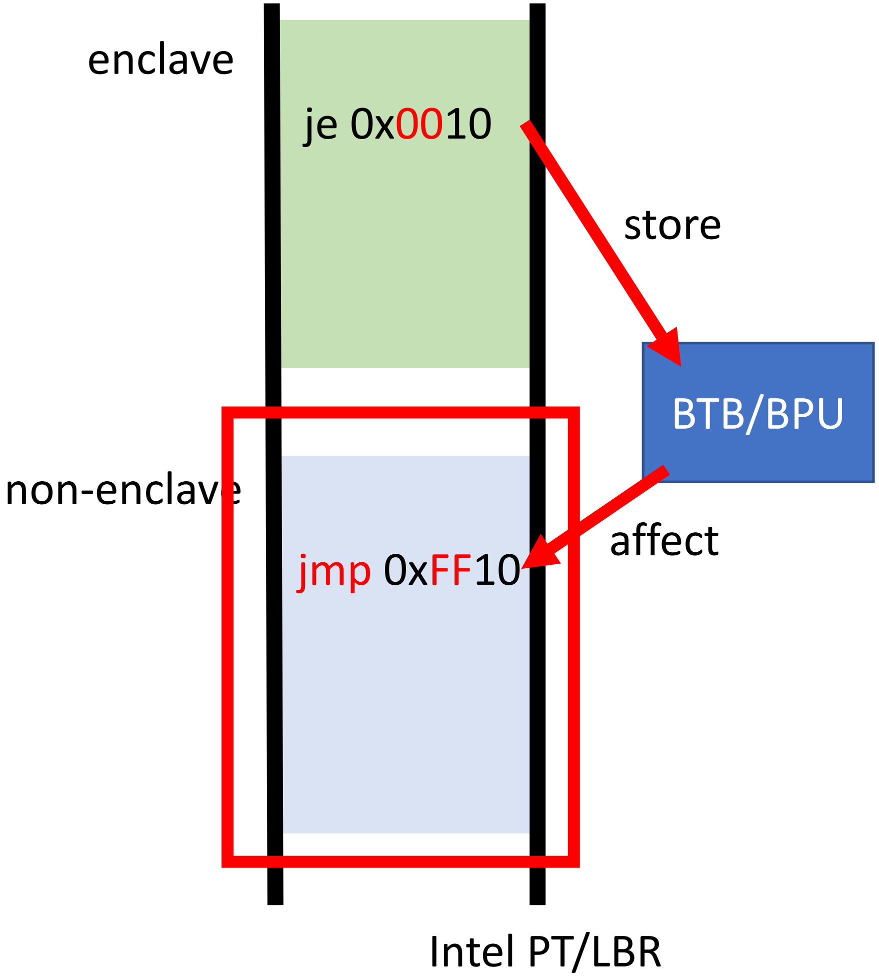


Our Approach: Branch Shadowing



- 🟦 🟩 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)

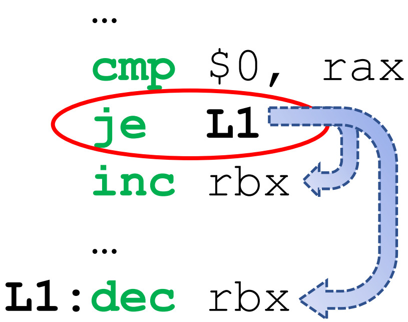
Our Approach: Branch Shadowing



- 🟦 🟩 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

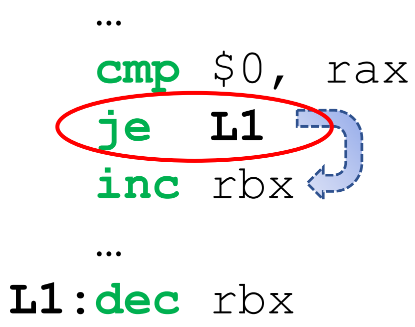


```
    …
    cmp $0, rax
    je  L1
    inc rbx
    …
L1:dec rbx
```

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



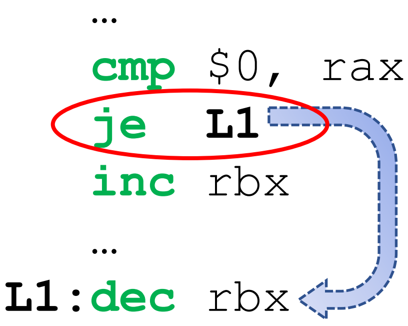
```
    …
    cmp $0, rax
    je  L1
    inc rbx
    …
L1: dec rbx
```

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



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

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

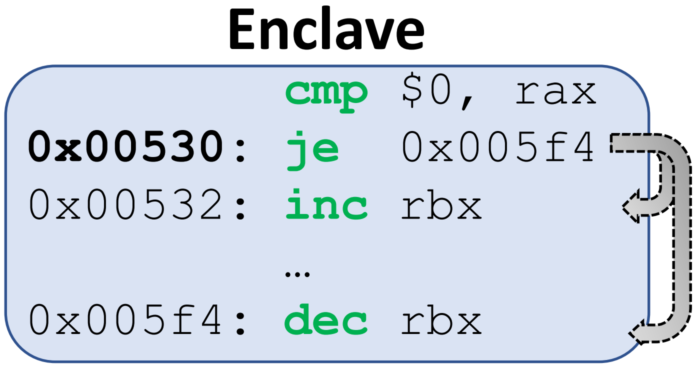
→ **Observable difference but high measurement noise**

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

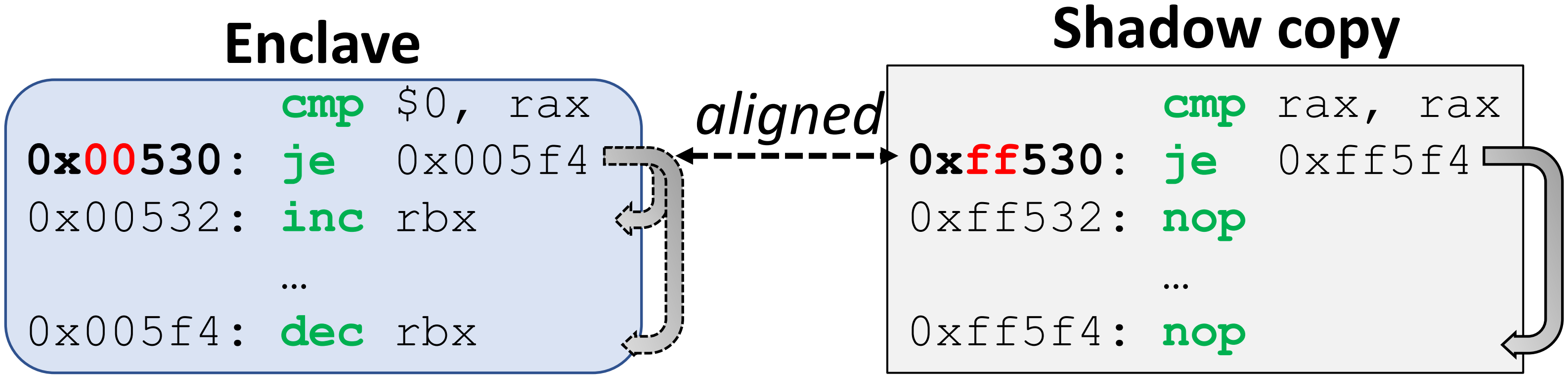
Example: Inferring Cond. Branch



Enclave

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

Example: Inferring Cond. Branch



Enclave

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

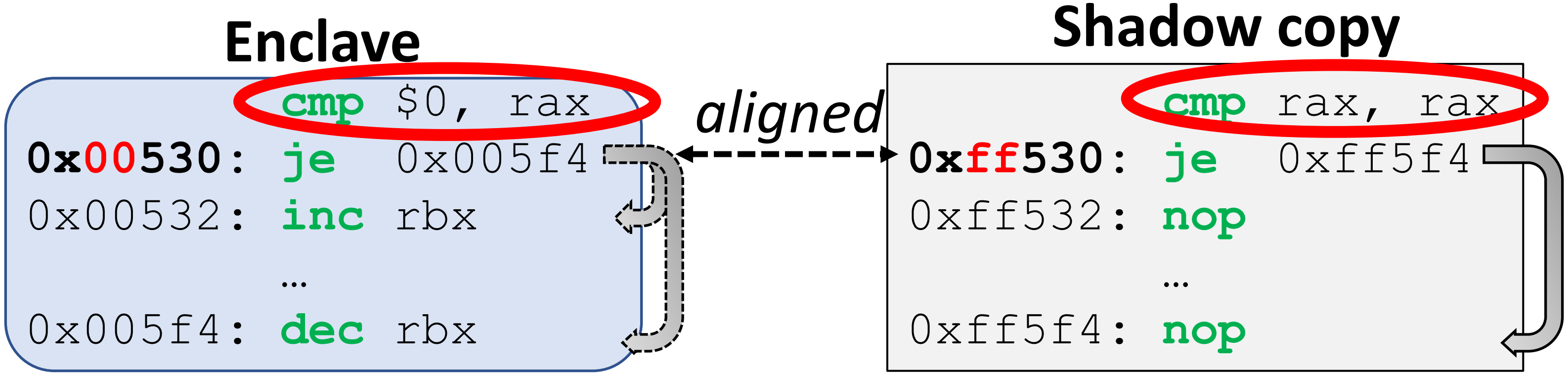
aligned

Shadow copy

```
          cmp rax, rax
0xff530:  je  0xff5f4
0xff532:  nop
          …
0xff5f4:  nop
```

- Prepare a shadow copy w/
    - Colliding conditional branches

Example: Inferring Cond. Branch



Enclave

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

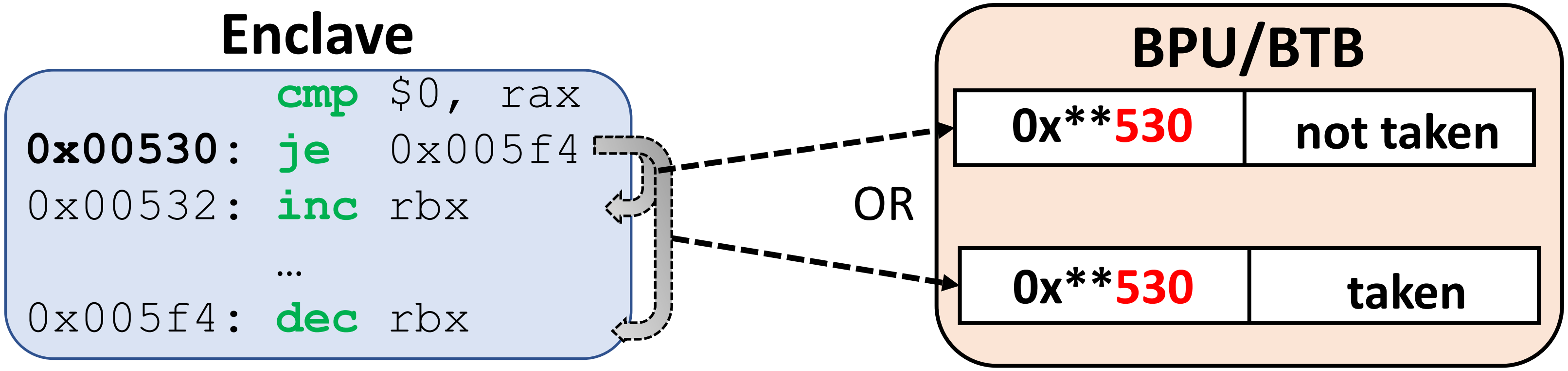
aligned

Shadow copy

```
           cmp rax, rax
0xff530:   je  0xff5f4
0xff532:   nop
           …
0xff5f4:   nop
```

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

Example: Inferring Cond. Branch



Enclave

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

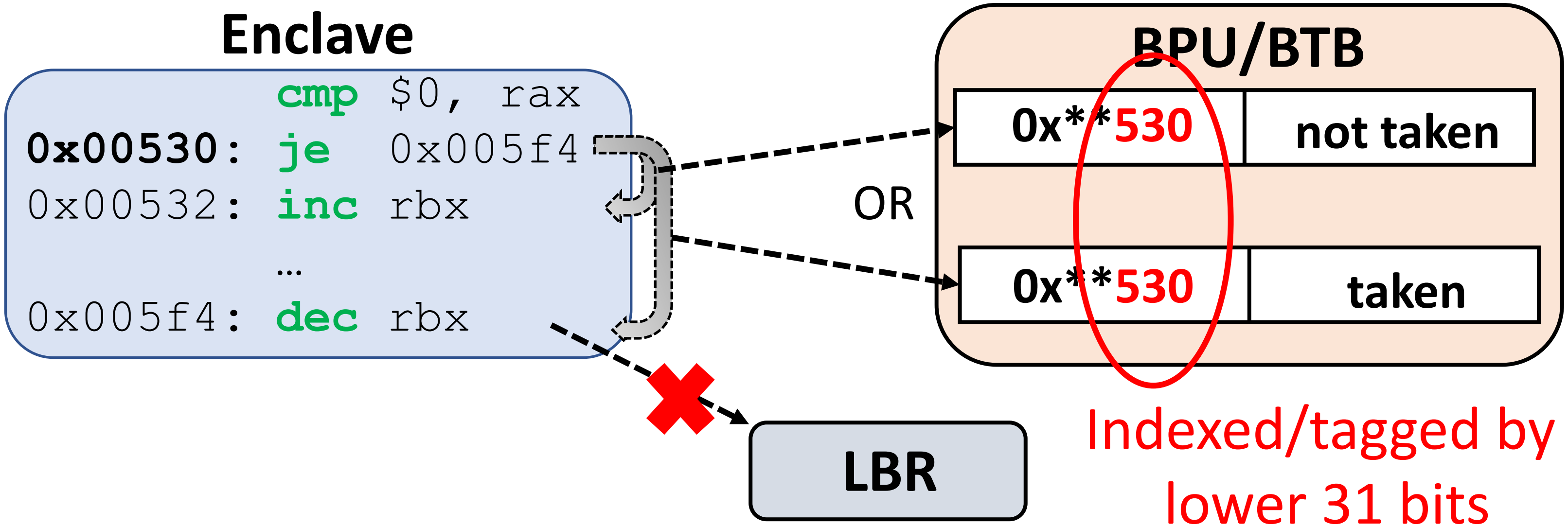
BPU/BTB

0x**530	not taken

OR

0x**530	taken

Example: Inferring Cond. Branch

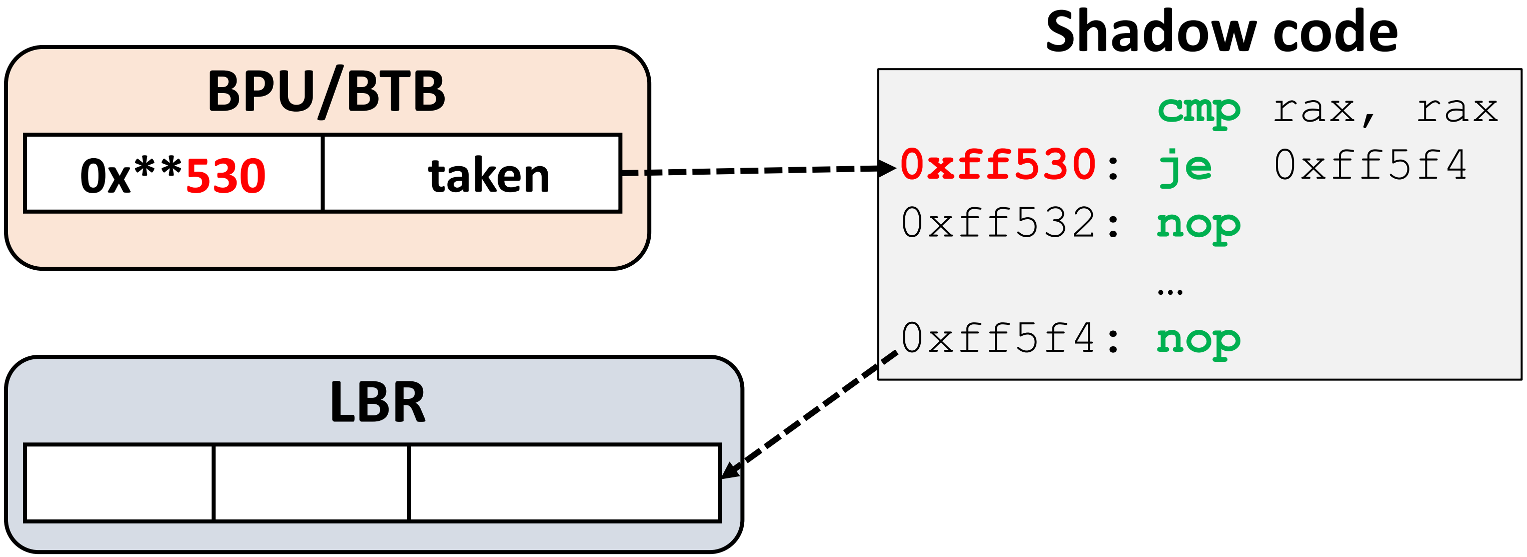


Enclave

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

Indexed/tagged by lower 31 bits

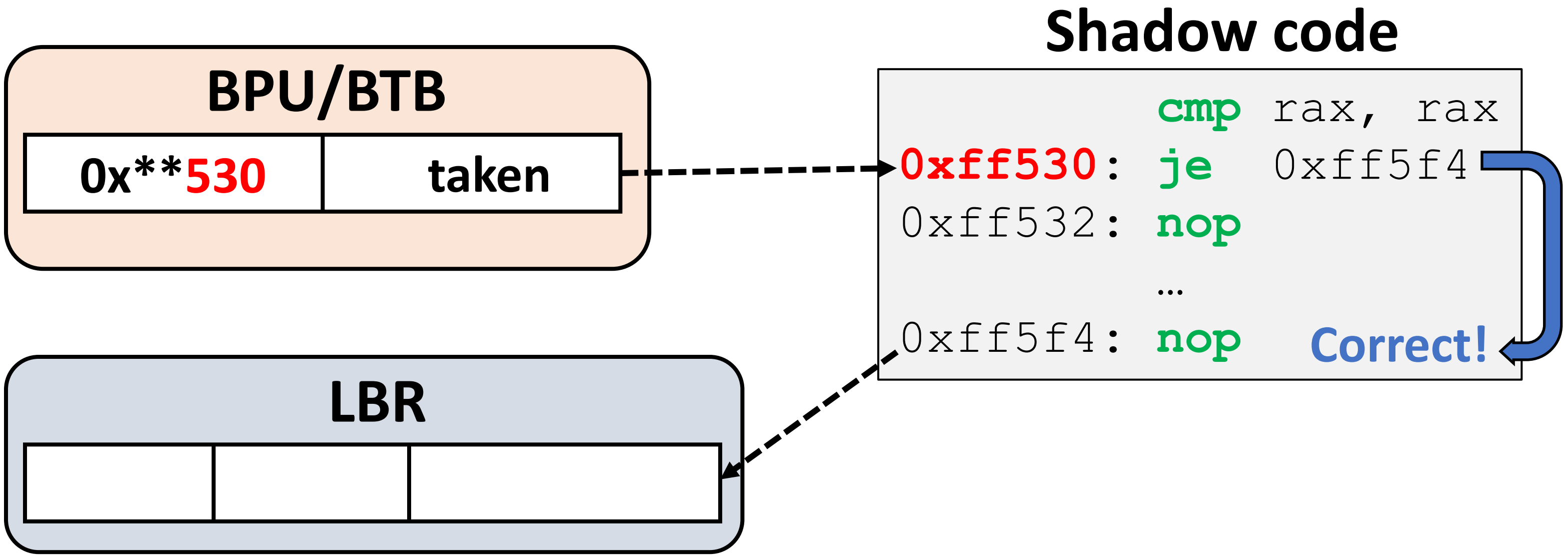
Example: Inferring Taken Branch



Shadow code

```
         cmp  rax, rax
0xff530: je   0xff5f4
0xff532: nop
         …
0xff5f4: nop
```

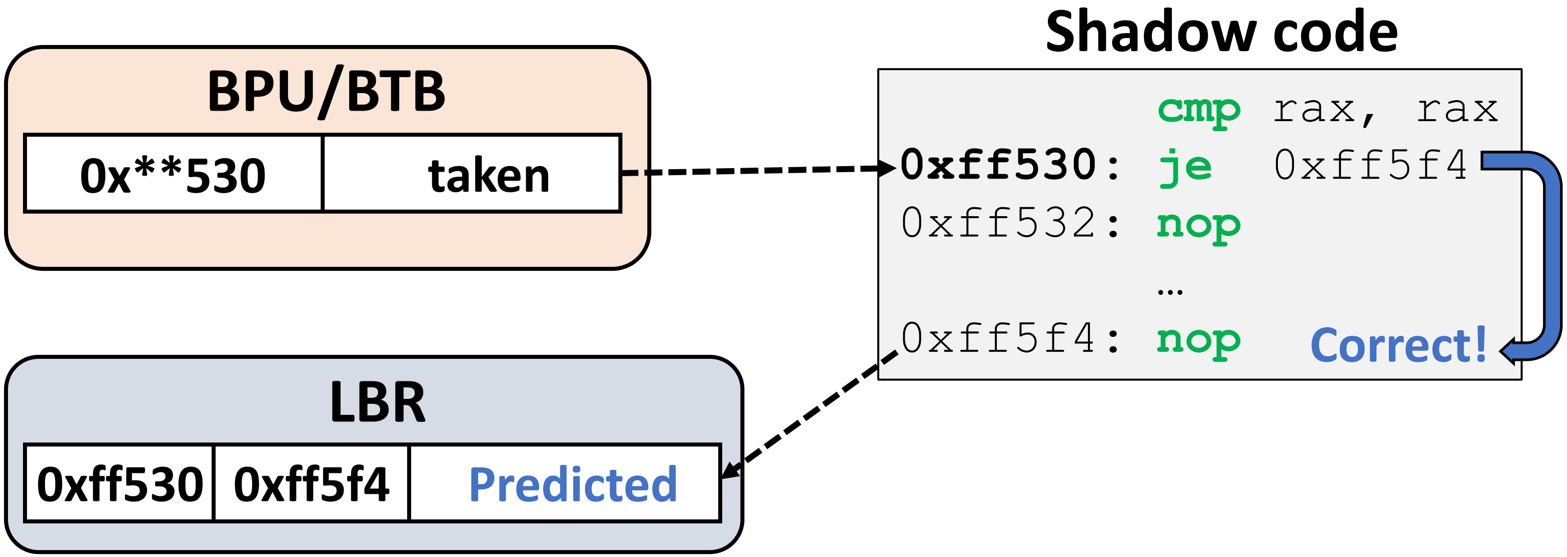
Example: Inferring Taken Branch



```
          cmp  rax, rax
0xff530:  je   0xff5f4
0xff532:  nop
          …
0xff5f4:  nop
```

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

Example: Inferring Taken Branch



BPU/BTB

0x**530	taken

Shadow code

```
         cmp  rax, rax
0xff530: je   0xff5f4
0xff532: nop
         …
0xff5f4: nop
```

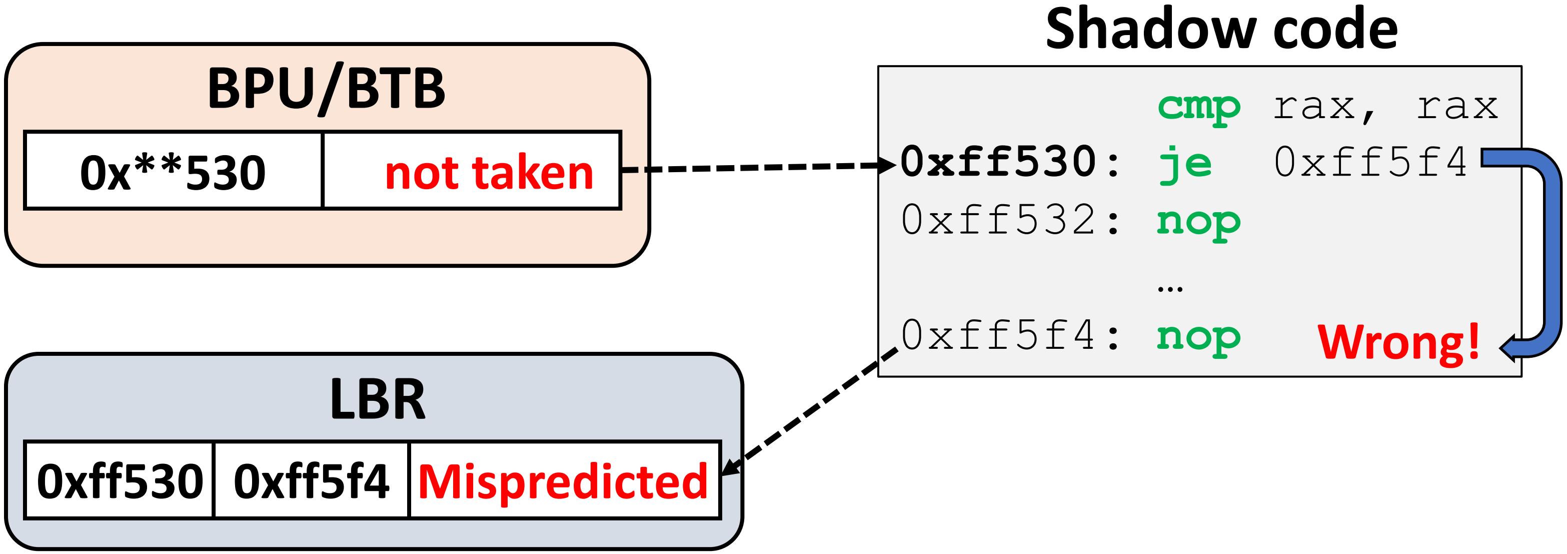
Correct!

LBR

0xff530	0xff5f4	Predicted

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

Example: Inferring Not-taken Branch



BPU/BTB

0x**530	not taken

Shadow code

```
         cmp rax, rax
0xff530: je  0xff5f4
0xff532: nop
         …
0xff5f4: nop
```

LBR

0xff530	0xff5f4	Mispredicted

- 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

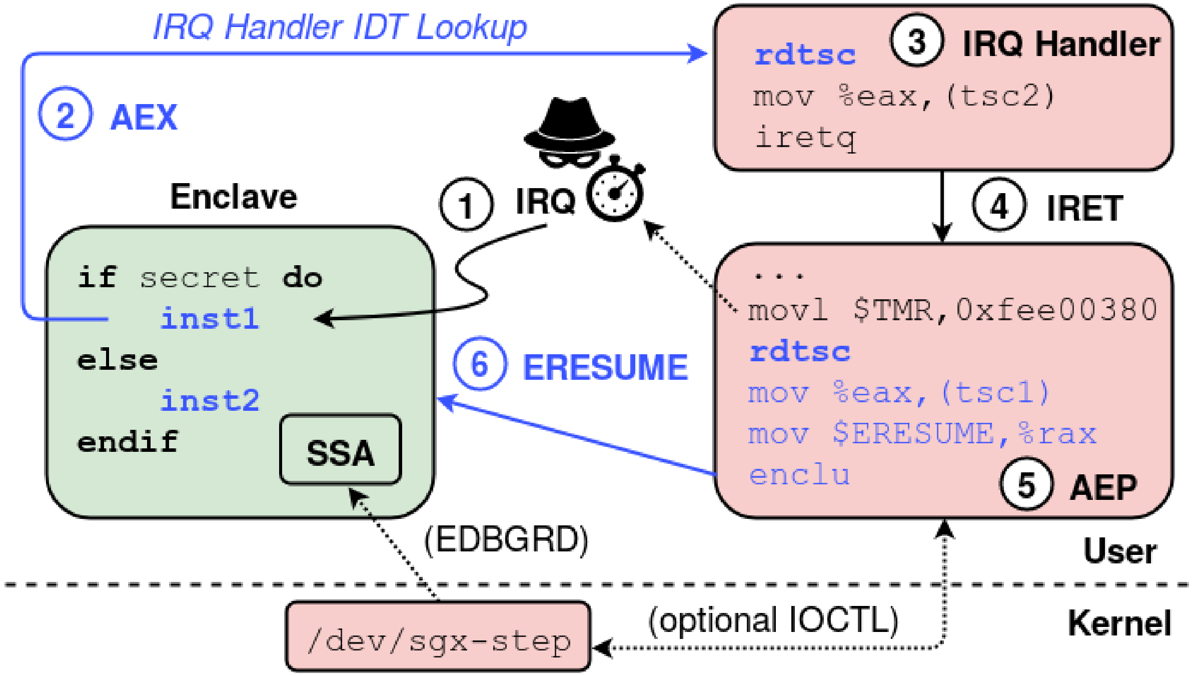
~50 cycles

- Disable the CPU cache
  - CD bit of the CR0 register

~5 cycles

SGX-Step: Open Source Framework

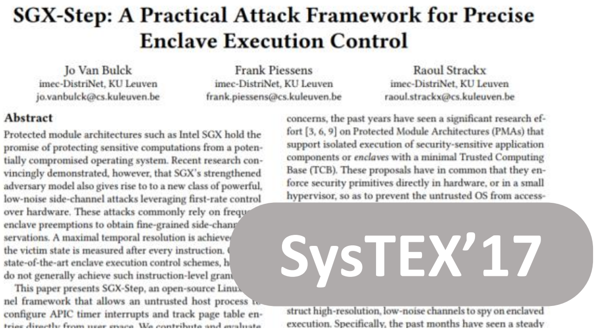
- Local APIC
- Userspace mapping for PTE



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

```
rdtsc
mov %eax,(tsc2)
iretq
```

```
...
movl $TMR,0xfee00380
rdtsc
mov %eax,(tsc1)
mov $ERESUME,%rax
enclu
```



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 to a new class of powerful, low-noise side-channel attacks leveraging first-rate control over hardware. These attacks commonly rely on frequ enclave preemptions to obtain fine-grained side-chann servations. A maximal temporal resolution is achieve the victim state is measured after every instruction. state-of-the-art enclave execution control schemes, h do not generally achieve such instruction-level gran

This paper presents SGX-Step, an open-source Linux nel framework that allows an untrusted host process 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-

struct high-resolution, low-noise channels to spy on enclaved execution. Specifically, the past months have seen a steady

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.

Sliding-window exponentiation of mbedTLS

```
/* 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);
    …
  }
  …
}
```

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?

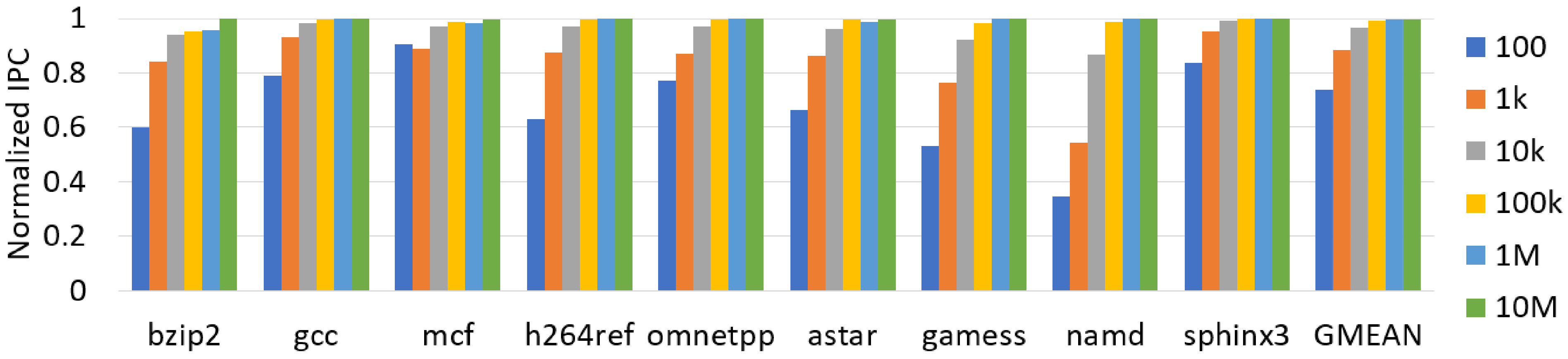
Program/Function	Description	Leakages
libc/strtol	Convert a string into an integer	The sign and length of an input Hexadecimal digits
libc/vfprintf	Print a formatted string	The input format string
LIBSVM/k_function	Evaluate a kernel function	The type of a kernel (e.g., linear, RBF) The number of features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

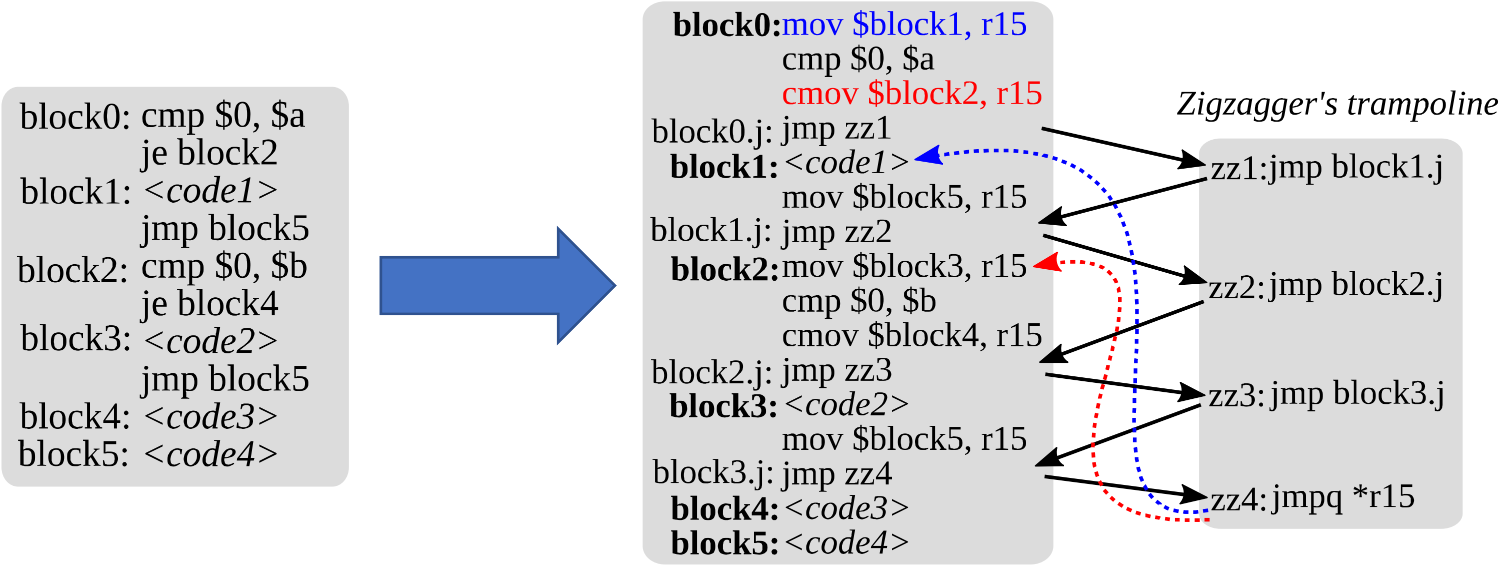
Defense: Flushing Branch States (Hardware)

- 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.



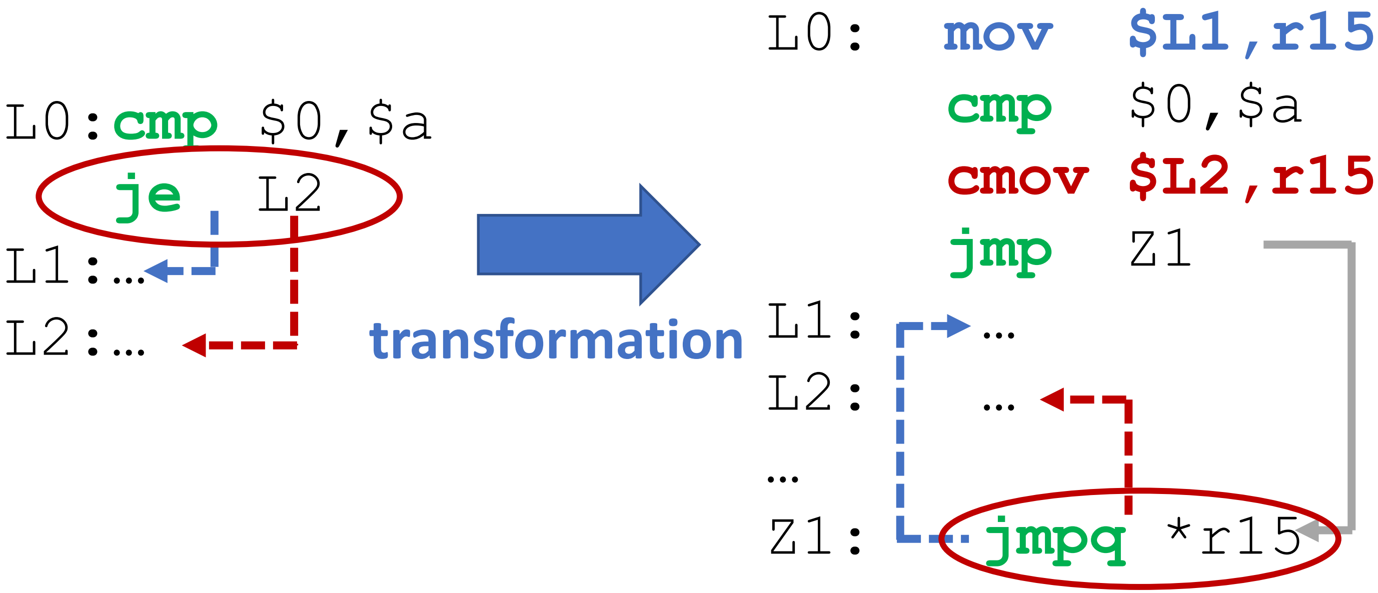
```
block0: cmp $0, $a
        je block2
block1: <code1>
        jmp block5
block2: cmp $0, $b
        je block4
block3: <code2>
        jmp block5
block4: <code3>
block5: <code4>
```

```
  block0:mov $block1, r15
         cmp $0, $a
         cmov $block2, r15
block0.j: jmp zz1
  block1:<code1>
         mov $block5, r15
block1.j: jmp zz2
  block2:mov $block3, r15
         cmp $0, $b
         cmov $block4, r15
block2.j: jmp zz3
  block3:<code2>
         mov $block5, r15
block3.j: jmp zz4
  block4:<code3>
  block5:<code4>
```

Zigzagger's trampoline

```
zz1:jmp block1.j
zz2:jmp block2.j
zz3:jmp block3.j
zz4:jmpq *r15
```

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)

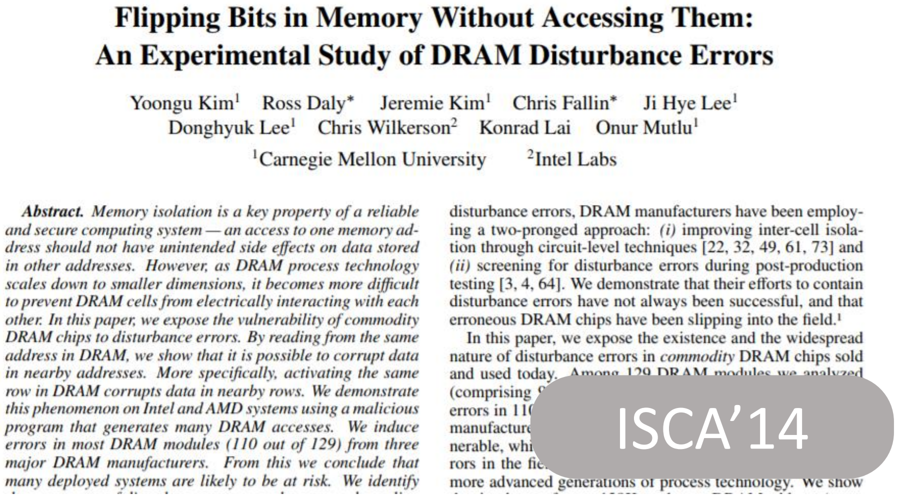
- LLVM-based implementation
- Overhead (nbench): $\leq 1.5\times$
- Just mitigate the attack, don’t solve it completely

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

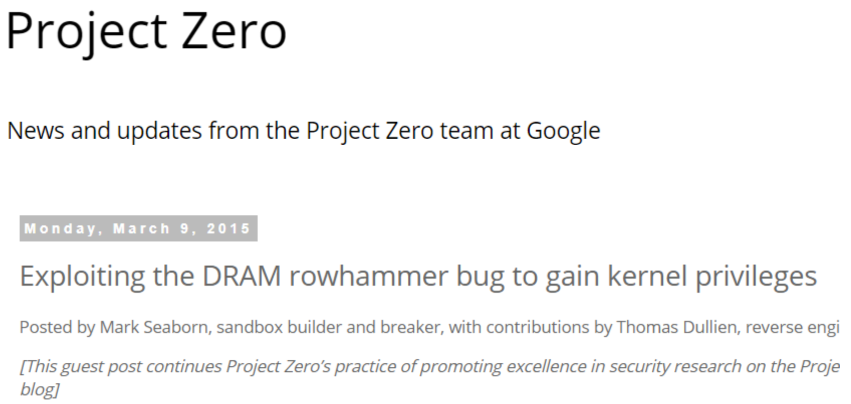
Yoongu Kim[1] Ross Daly* Jeremie Kim[1] Chris Fallin* Ji Hye Lee[1]
Donghyuk Lee[1] Chris Wilkerson[2] Konrad Lai Onur Mutlu[1]
[1]Carnegie Mellon University [2]Intel Labs

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.[1]

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
errors in 110
manufacture
nerable, whi
rors in the fie
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

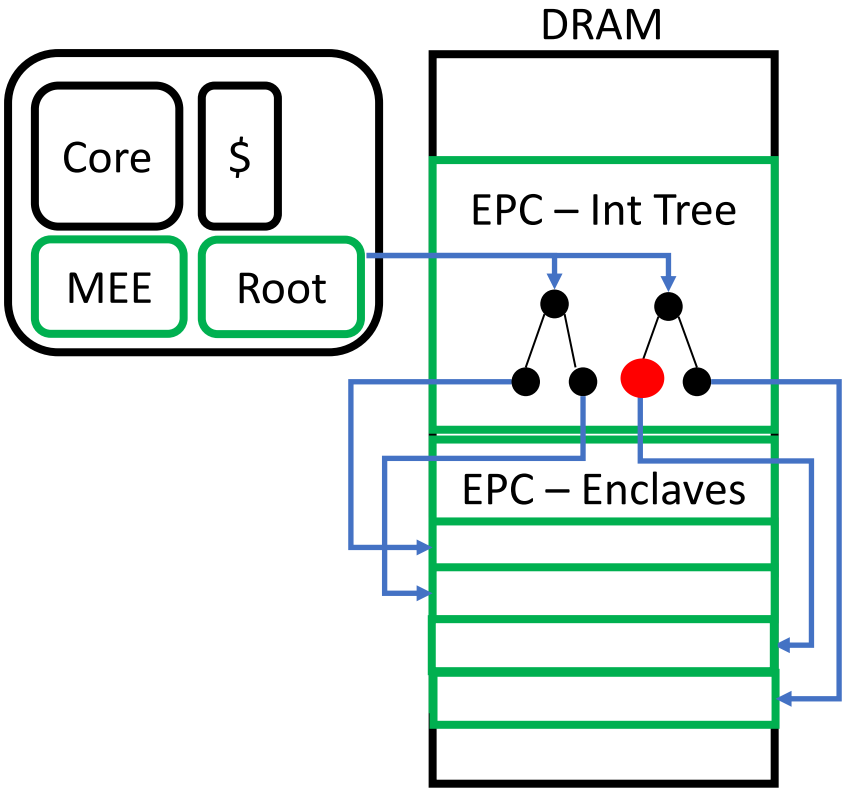
Exploiting the DRAM rowhammer bug to gain kernel privileges

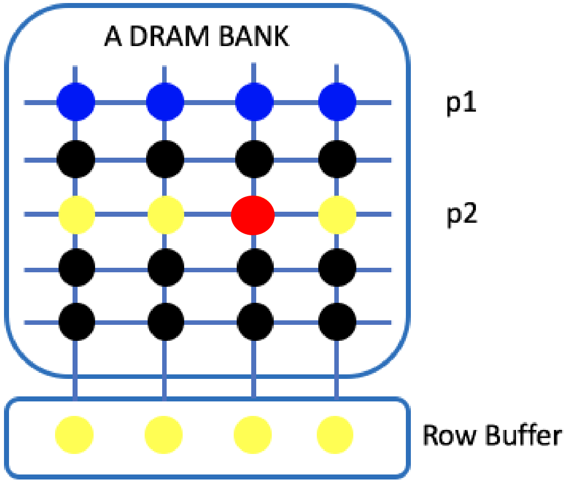
Posted by Mark Seaborn, sandbox builder and breaker, with contributions by Thomas Dullien, reverse engi

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

SGX-Bomb: Rowhammer Attack

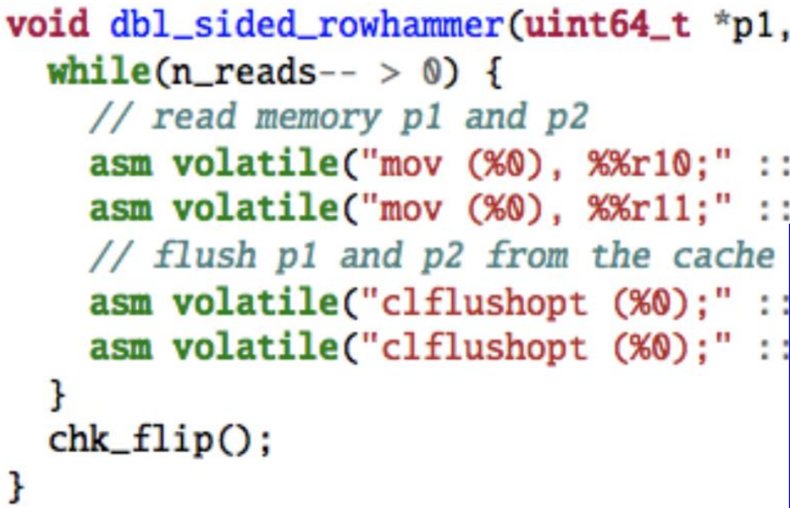
- Integrity violation of EPC results in CPU lockdown
- Rowhammer (SW) can trigger the violation!



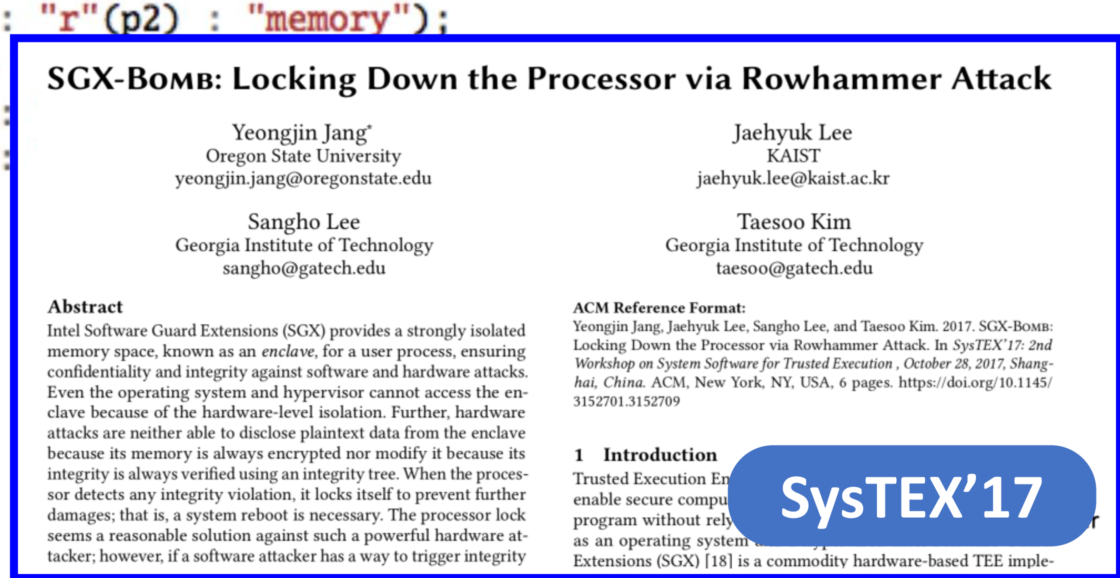


SGX-Bomb: Rowhammer Attack

- 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);" ::
    asm volatile("clflushopt (%0);" ::
  }
  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 En
enable secure compu
program without rely
as an operating system
Extensions (SGX) [18] is a commodity hardware-based TEE imple-

SysTEX'17

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)

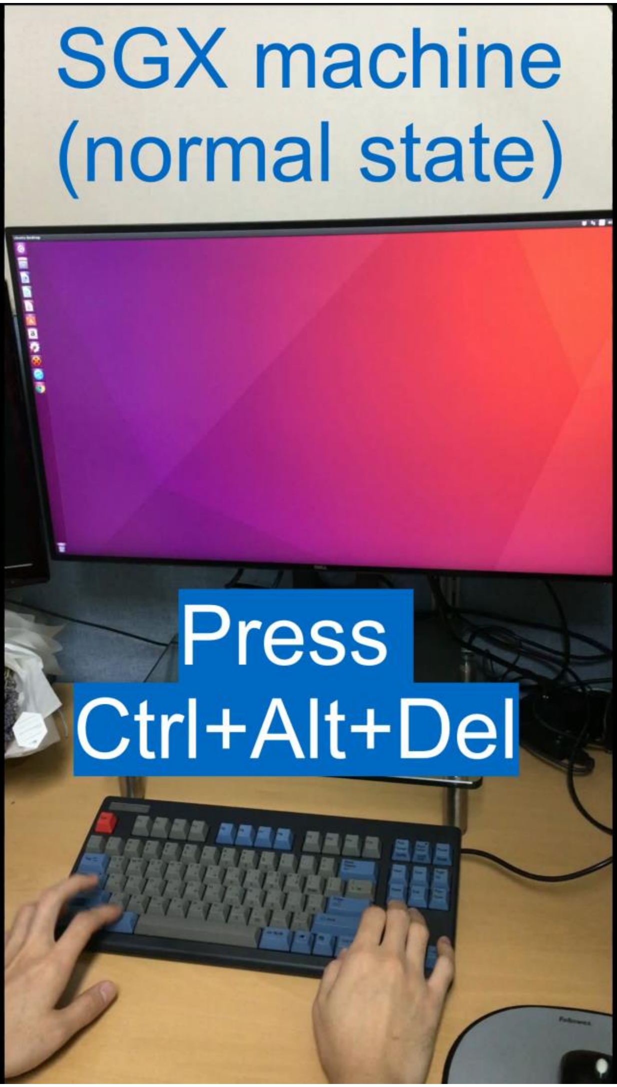
SGX-Bomb Remarks

- 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



https://github.com/sslab-gatech/sgx-bomb

Defenses against SGX-Bomb

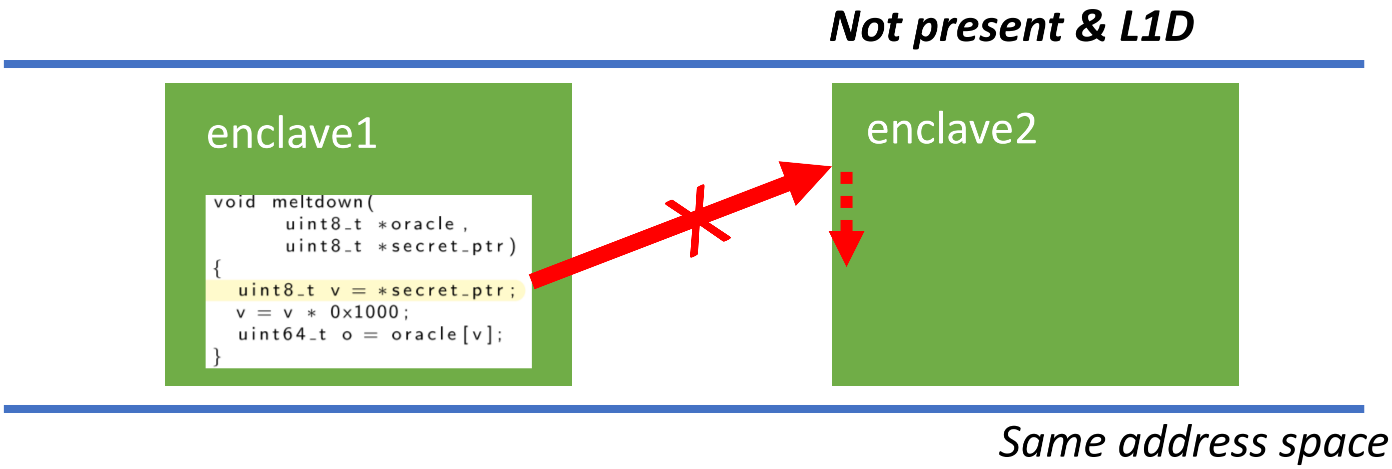
- Use non-faulty DRAM!
- Use LPDDR3 that has Pseudo-TRR (Target Row Refresh)
  - ECC can’t completely block (easy to trigger multiple bits)

- Potential mitigations:
  - Higher refresh rate (2x)
  - Using Uncore PMU
  - Row-aware memory allocation for EPC regions

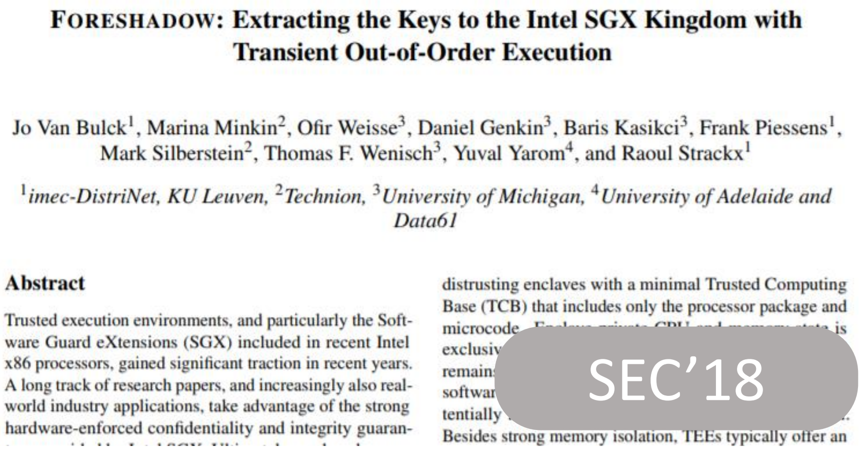
New Attack Vectors

- 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



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



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

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

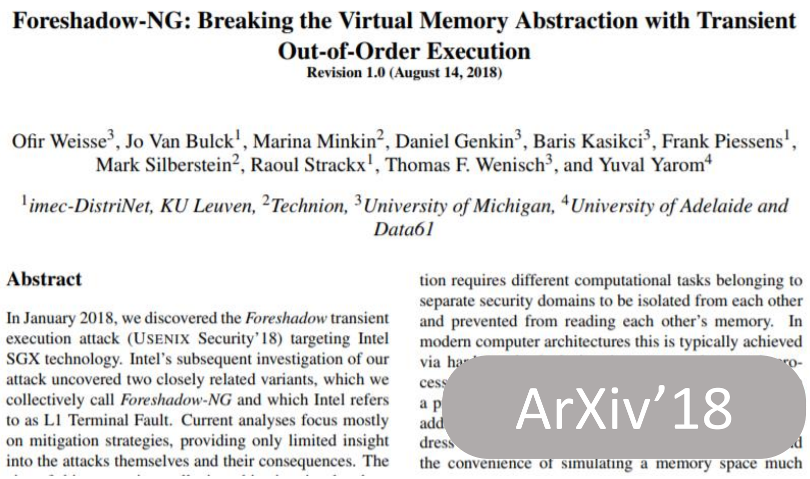
[1]*imec-DistriNet, KU Leuven,* [2]*Technion,* [3]*University of Michigan,* [4]*University of Adelaide and Data61*

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. Enclave-private CPU and memory state is exclusiv remain softwar tentially 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[3], Jo Van Bulck[1], Marina Minkin[2], Daniel Genkin[3], Baris Kasikci[3], Frank Piessens[1], Mark Silberstein[2], Raoul Strackx[1], Thomas F. Wenisch[3], and Yuval Yarom[4]

[1]*imec-DistriNet, KU Leuven,* [2]*Technion,* [3]*University of Michigan,* [4]*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 ha ro- cess a p add dress d 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?

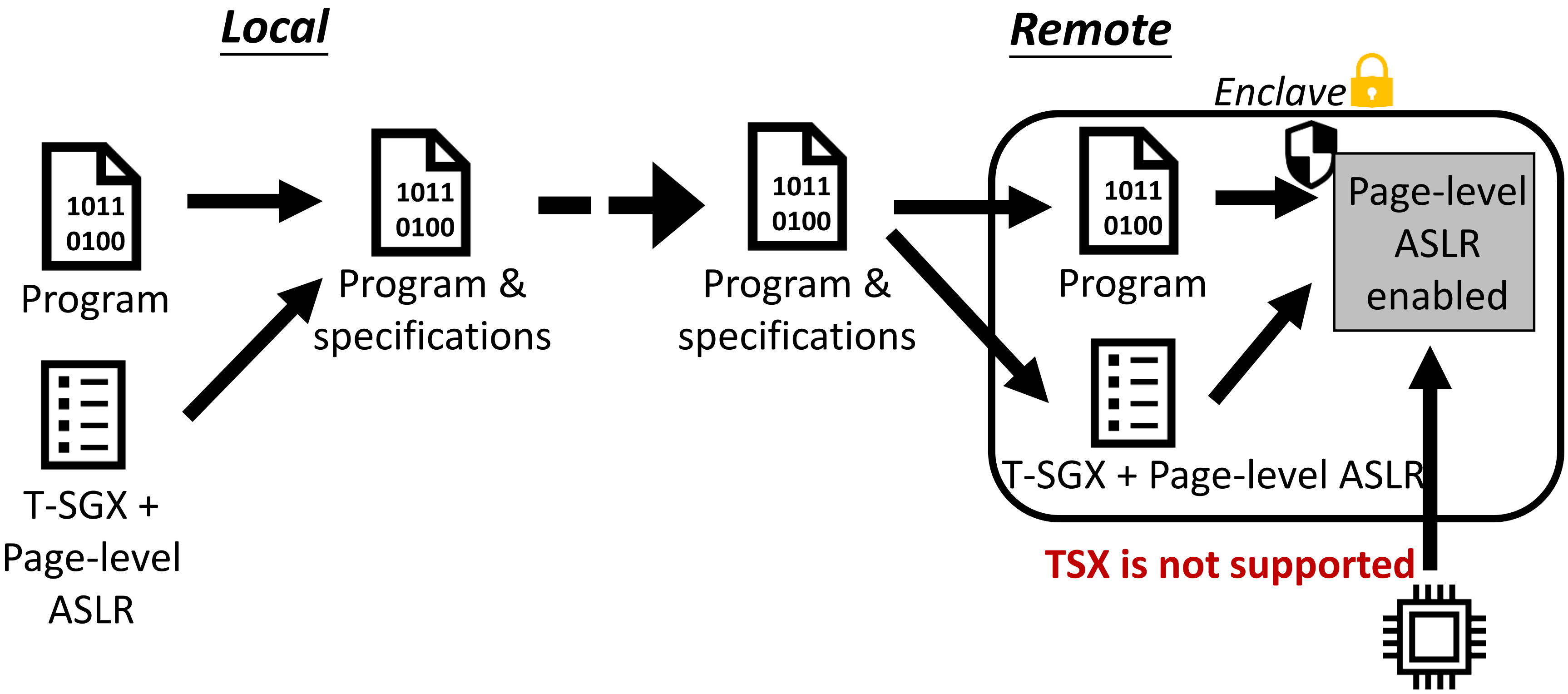
Outline

- 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



Local

Remote

Enclave

1011
0100
Program

T-SGX +
Page-level
ASLR

1011
0100
Program &
specifications

1011
0100
Program &
specifications

1011
0100
Program

T-SGX + Page-level ASLR

Page-level
ASLR
enabled

TSX is not supported

PRIDWIN: Load-time Synthesis

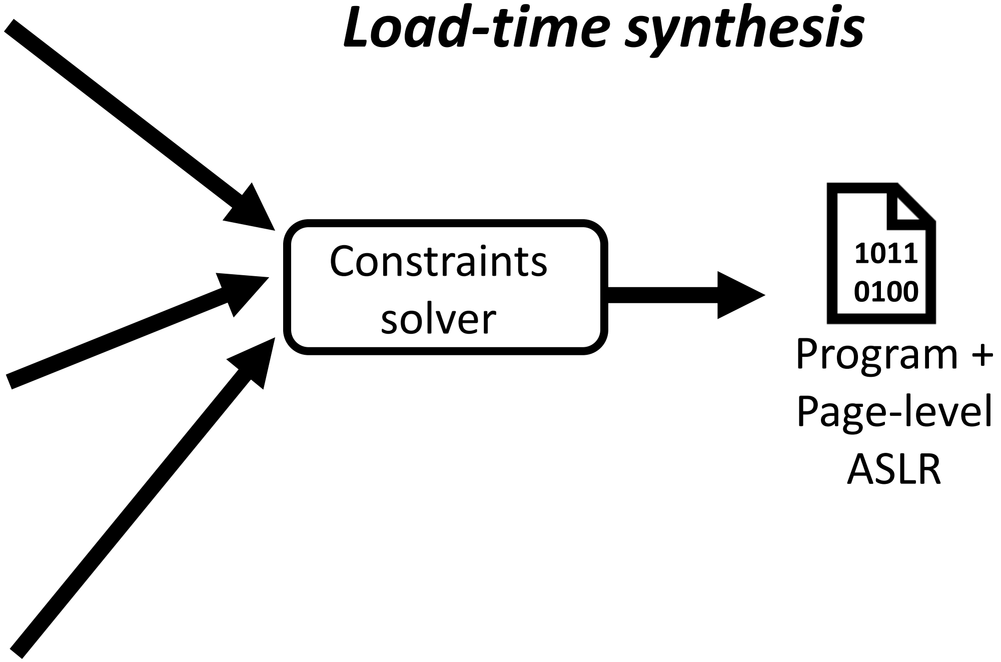
Specifications & Constraints

T-SGX
Target: Page-table attacks
Priority: High
Requirement: TSX
Instrumentation:
- Insert XBEGIN, XEDN 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**

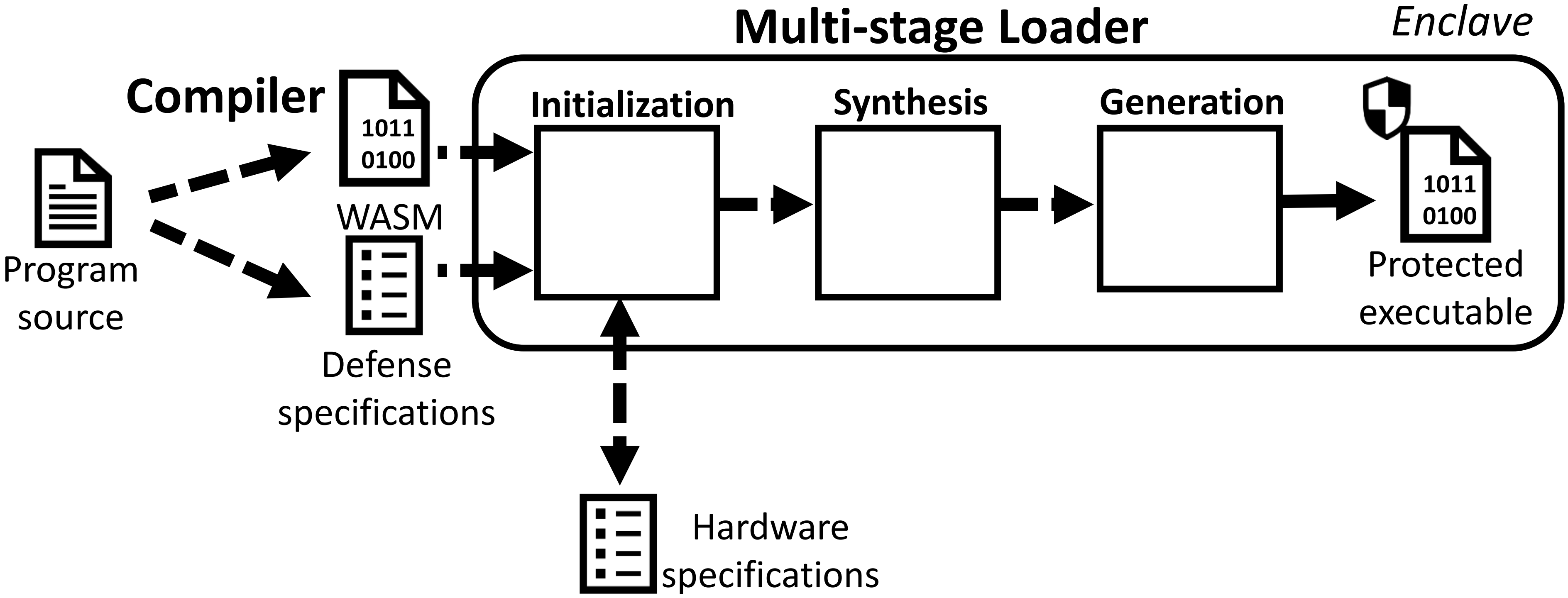
Load-time synthesis



Constraints solver

Program + Page-level ASLR

PRIDWIN: Load-time Synthesis

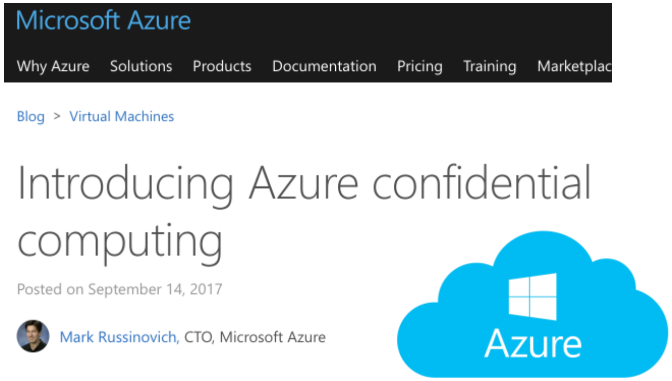


Summary

- 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 Marketplac

Blog > Virtual Machines

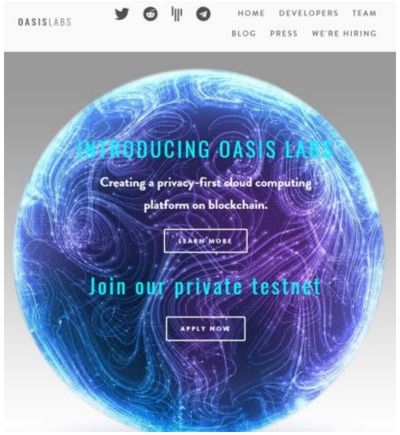
Introducing Azure confidential computing

Posted on September 14, 2017

Mark Russinovich, CTO, Microsoft Azure

Azure

IBM Cloud



OASISLABS

HOME DEVELOPERS TEAM

BLOG PRESS WE'RE HIRING

INTRODUCING OASIS LABS

Creating a privacy-first cloud computing platform on blockchain.

LEARN MORE

Join our private testnet

APPLY NOW

Fortanix

info@fortanix.com +1 (628) 400-2043

RESOURCES | BLOG

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