

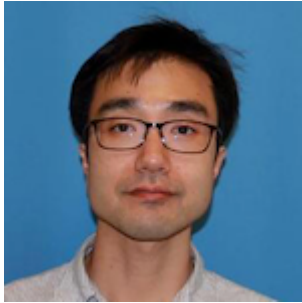
Security Issues on Intel SGX

(offensive and defensive techniques)

Taesoo Kim



The Team



Outline

- Threat model / assumption
- Traditional attack vectors
- New attack vectors
- 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
- Summary

Outline

- Threat model / assumption
- Traditional attack vectors

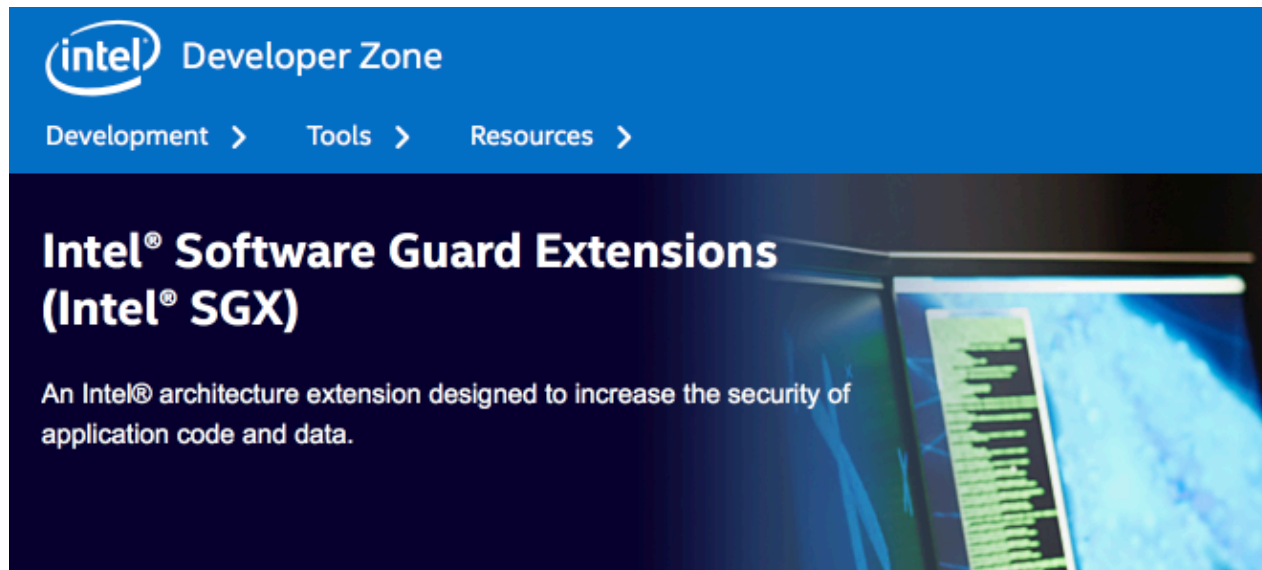
- **New attack vectors**

- Page table attack
- Branch shadowing attack
- ★ • Rowhammer against SGX

- Summary

Disclaimer

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



Academic Research

Run Unmodified Applications in Enclaves

- Graphene SGX: A Practical Library Operating System for Unmodified Applications
- SGXKernel: A Library Operating System Optimized for Intel SGX

Revisited: Intel SGX 101

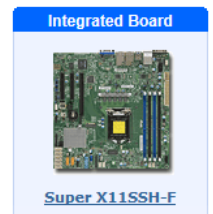
- **“Practical”** TEE implementation by Intel
- Extending x86 Instruction Set Architecture (ISA)
 - Native performance
 - Compatible to x86
 - Commodity (i.e., cheap)



Lenovo T560

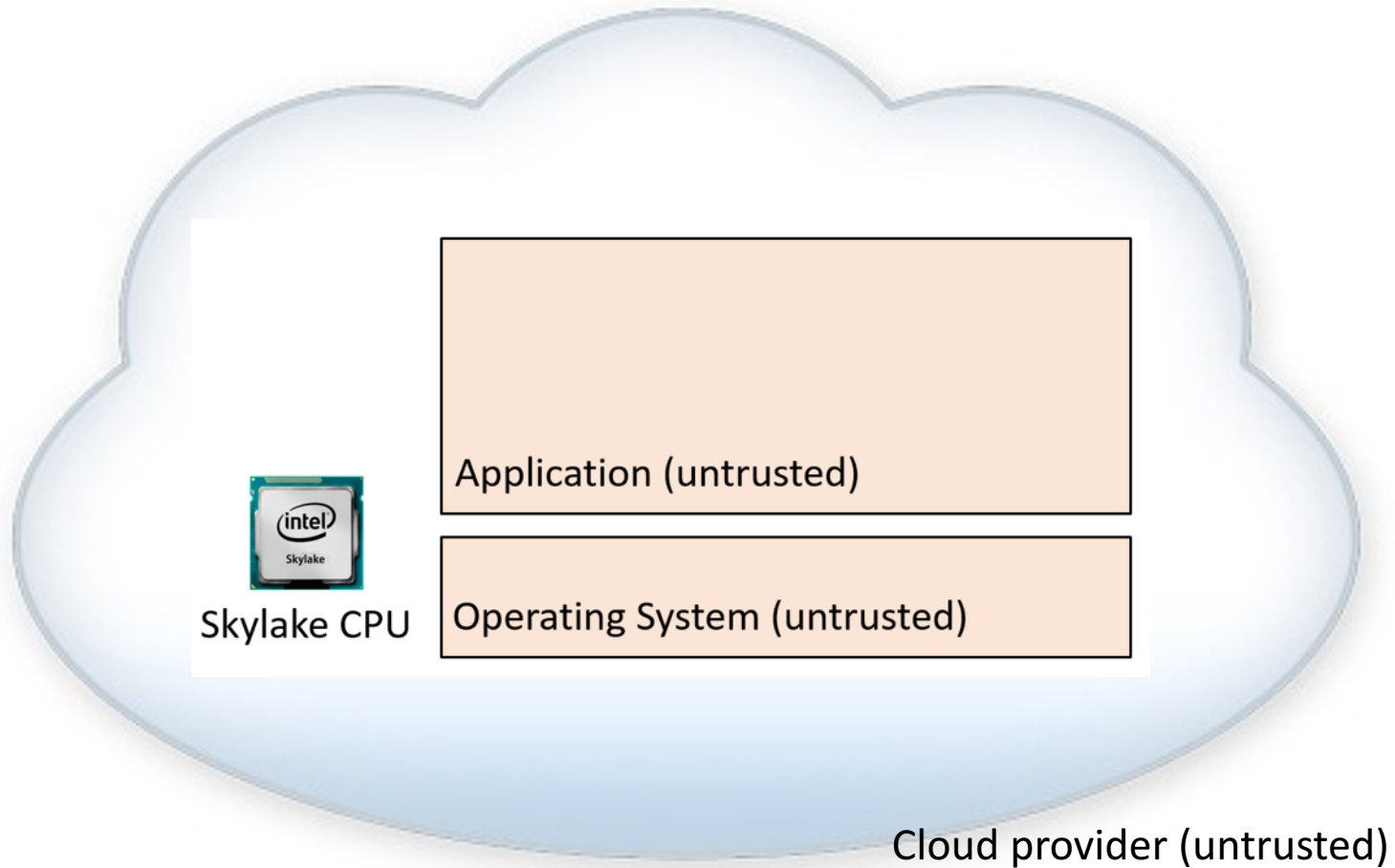


Dell OptiPlex 5040

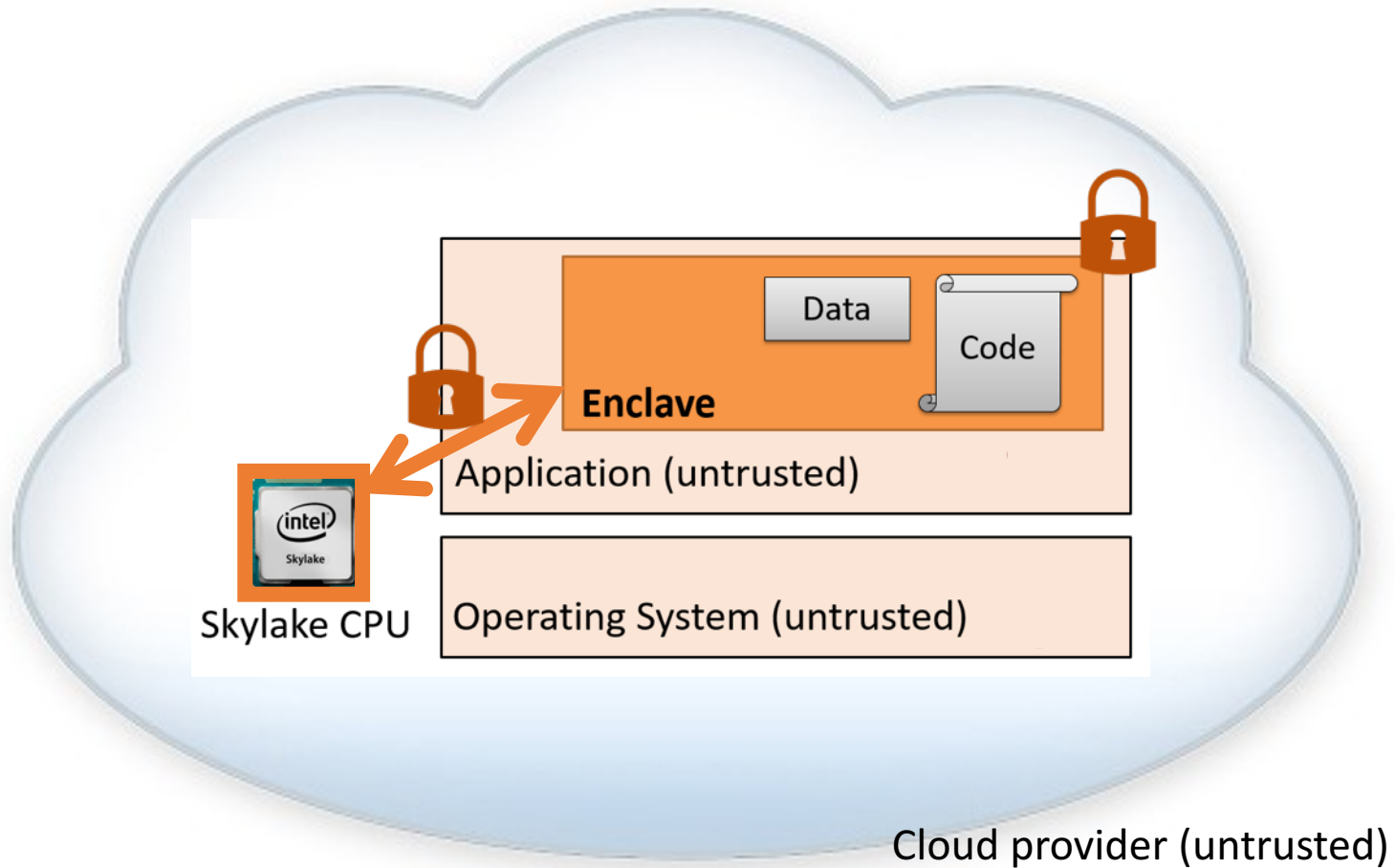


Supermicro Server

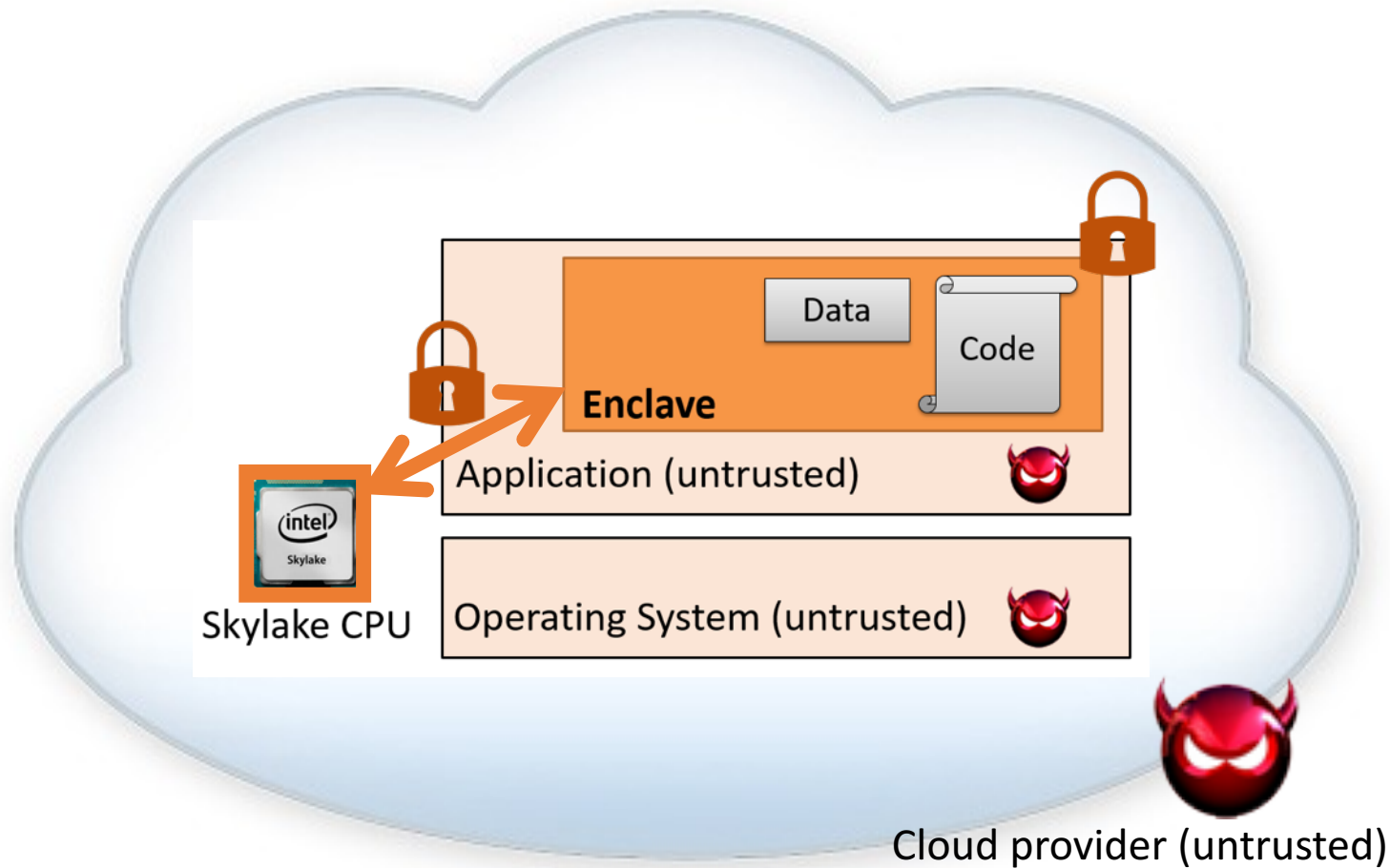
Revisited: SGX for Cloud



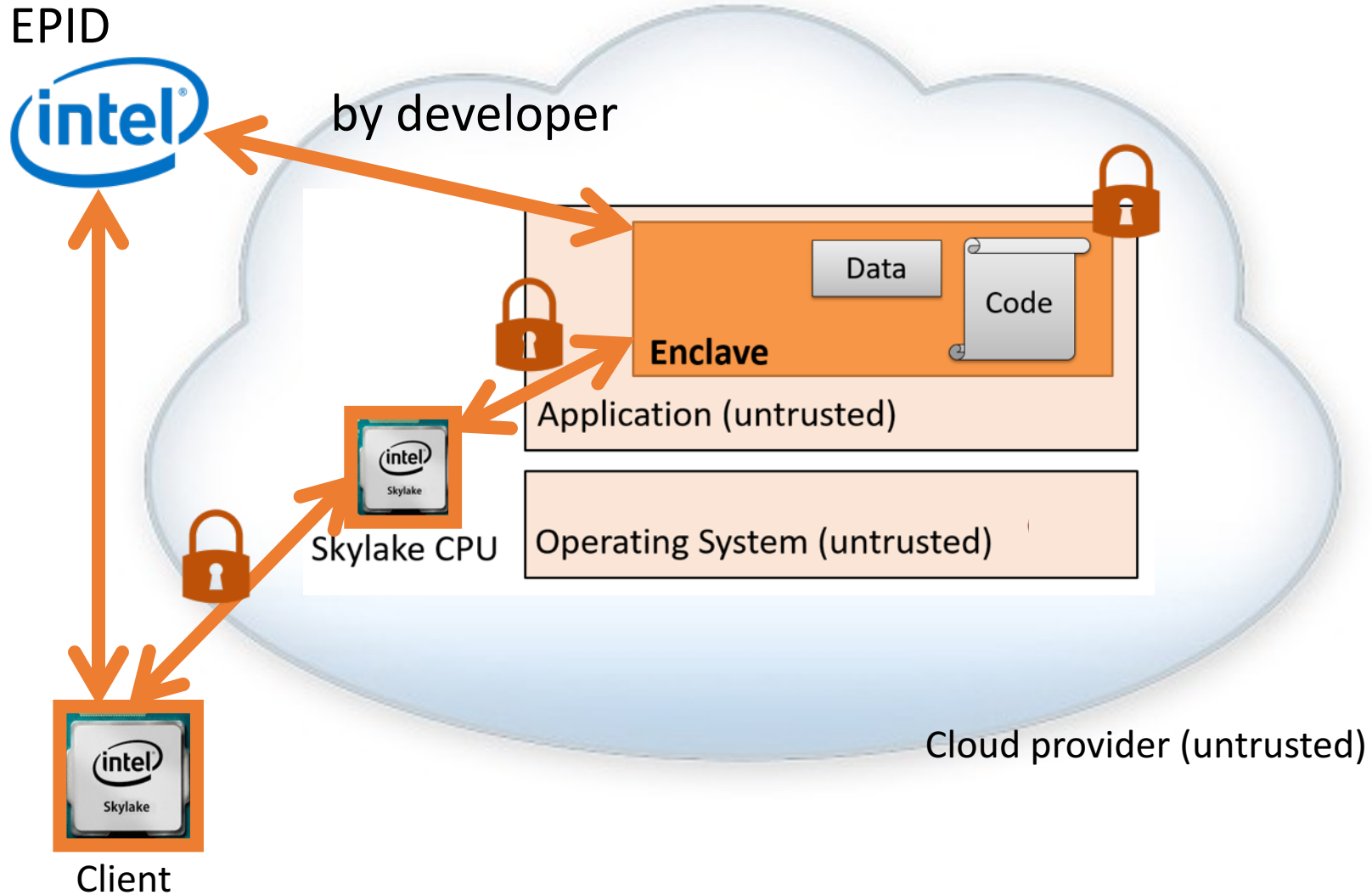
Revisited: SGX for Cloud



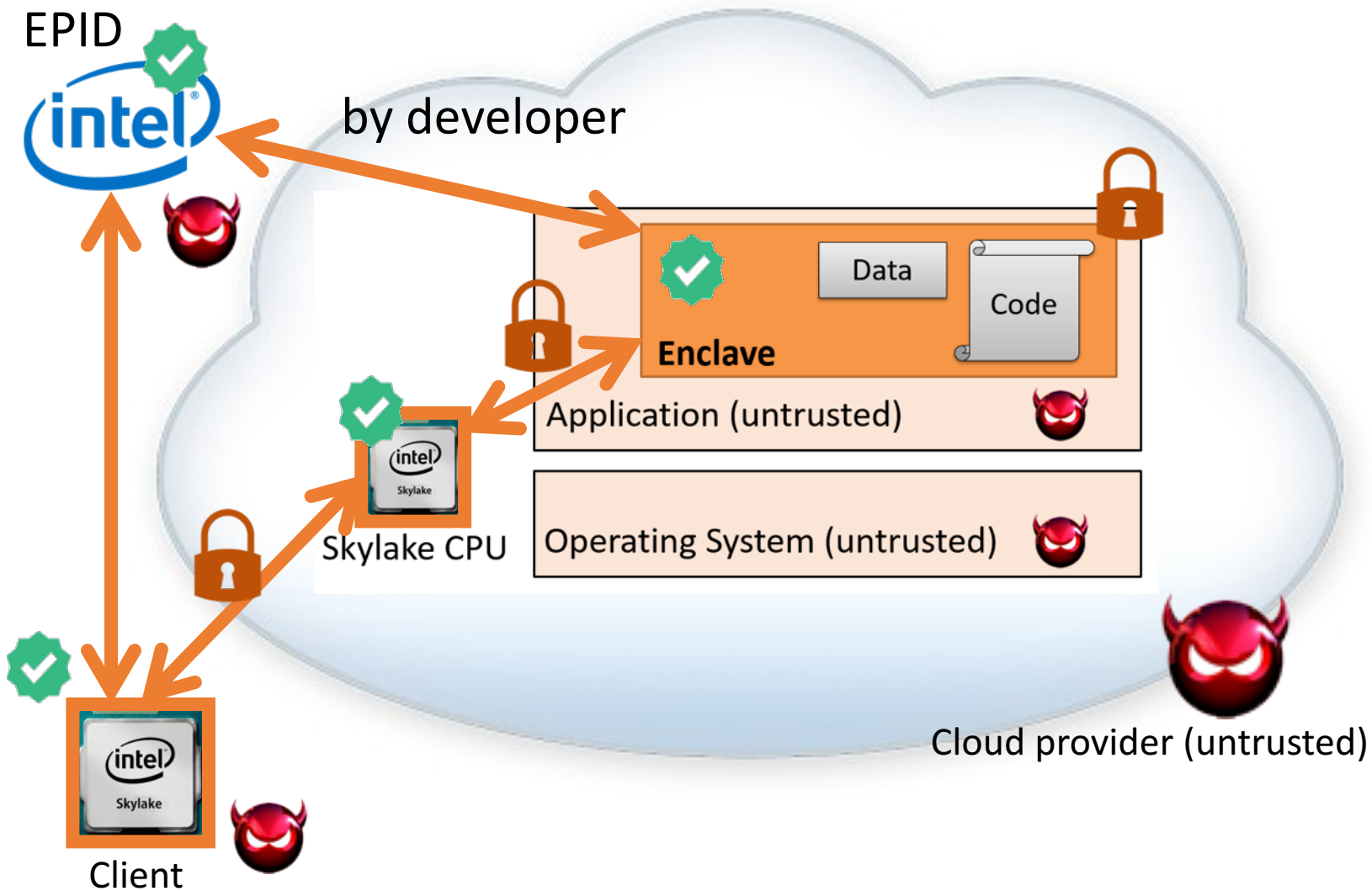
Revisited: SGX for Cloud (Isolation)



Revisited: SGX for Cloud (Remote attestation)



Revisited: SGX for Cloud (Remote attestation)



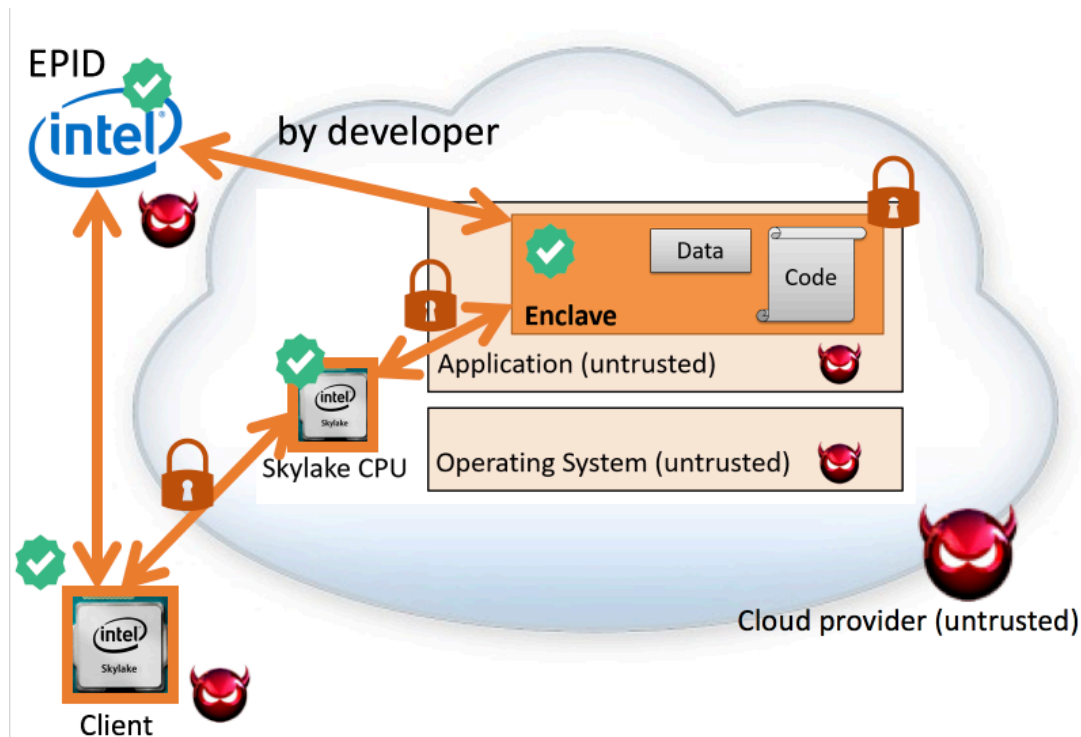
SGX Ecosystem for Attackers



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



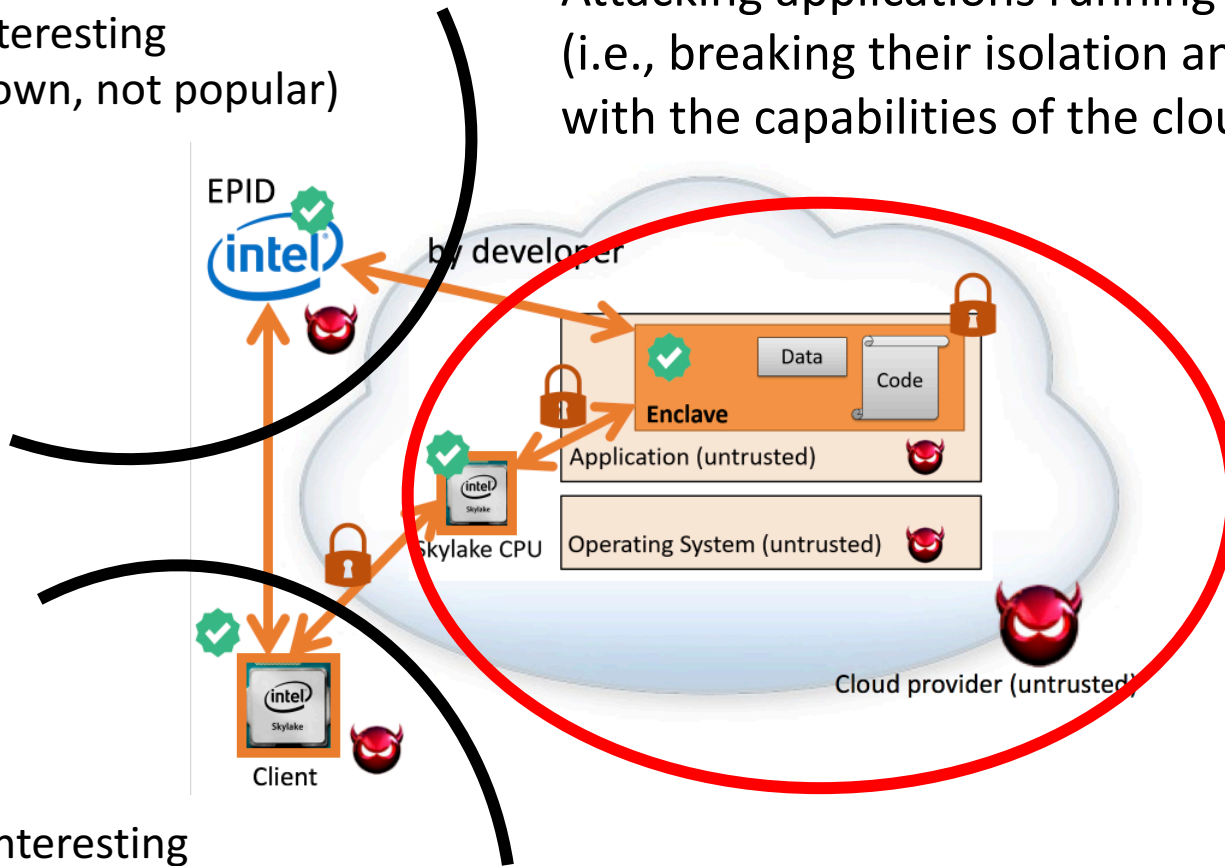
: Attacker's capabilities (i.e., what attackers can do)



Our Initial Interests as Attacker

Not interesting
(unknown, not popular)

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



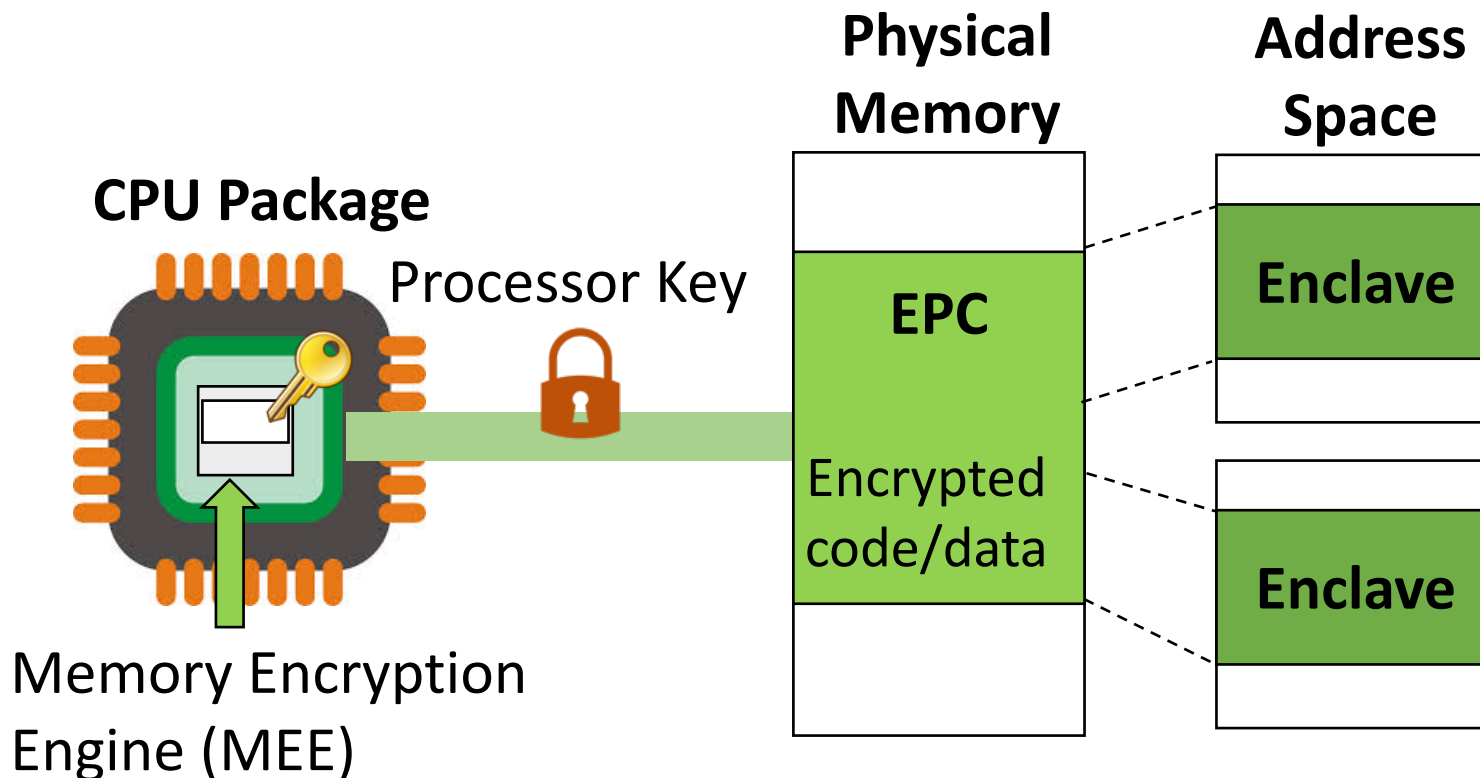
Not interesting
(non technical issues)

Summary: Intel SGX 101

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

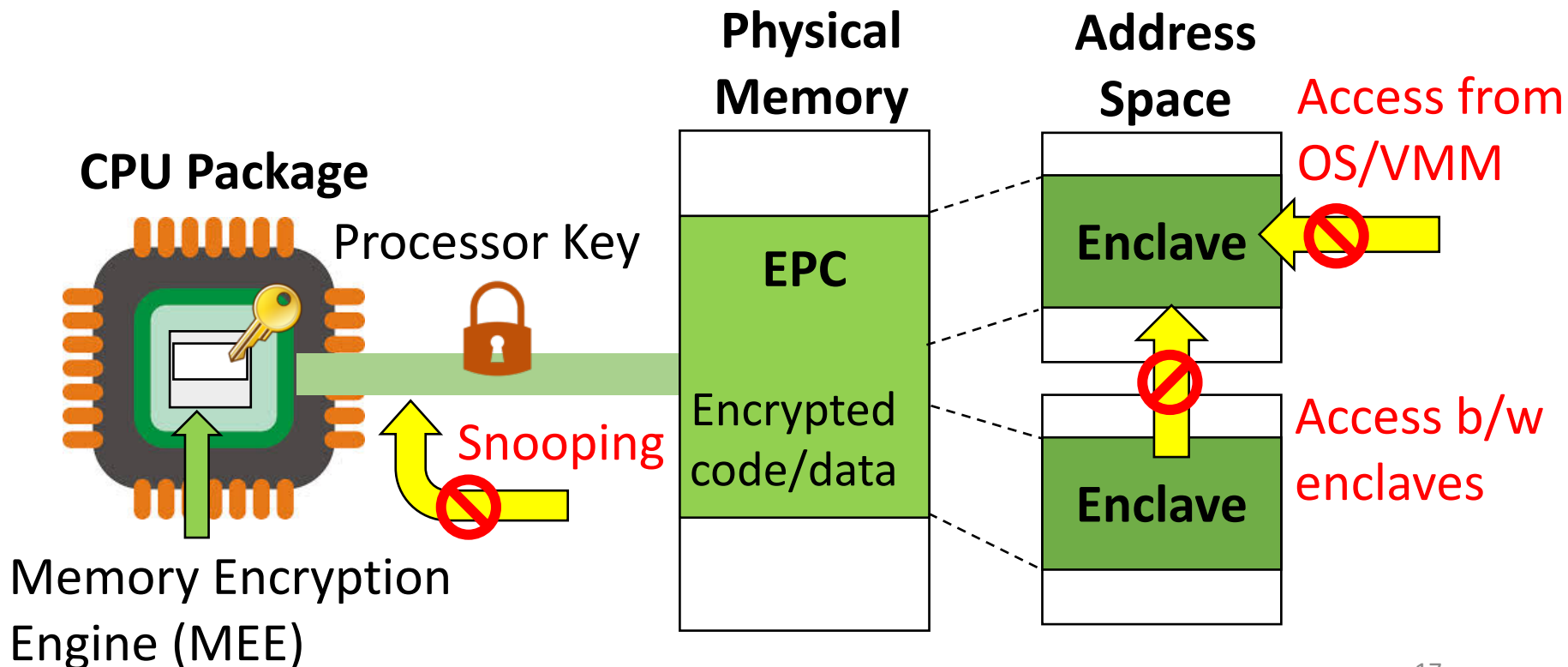
Intel SGX 101: Isolated Execution

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



Intel SGX 101: Isolated Execution

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



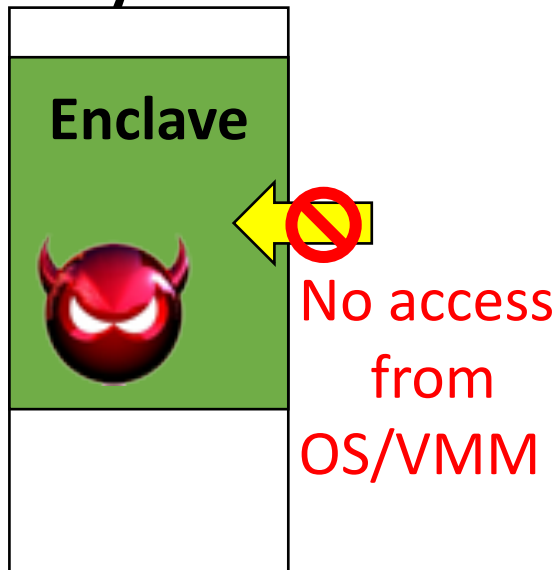
SGX's Threat Model (very strong!)

- *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)

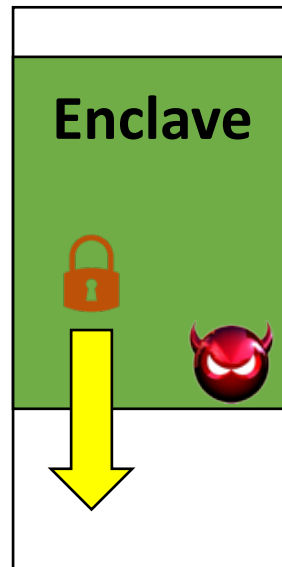
What if Enclave is Compromised?

- SGX protects attackers from auditing/analysis
- Leak sensitive information
- Permanently parasite to the enclave program

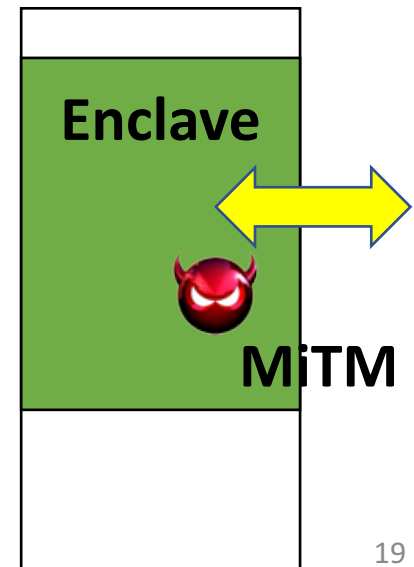
**Protected?
by SGX**



Leak secret



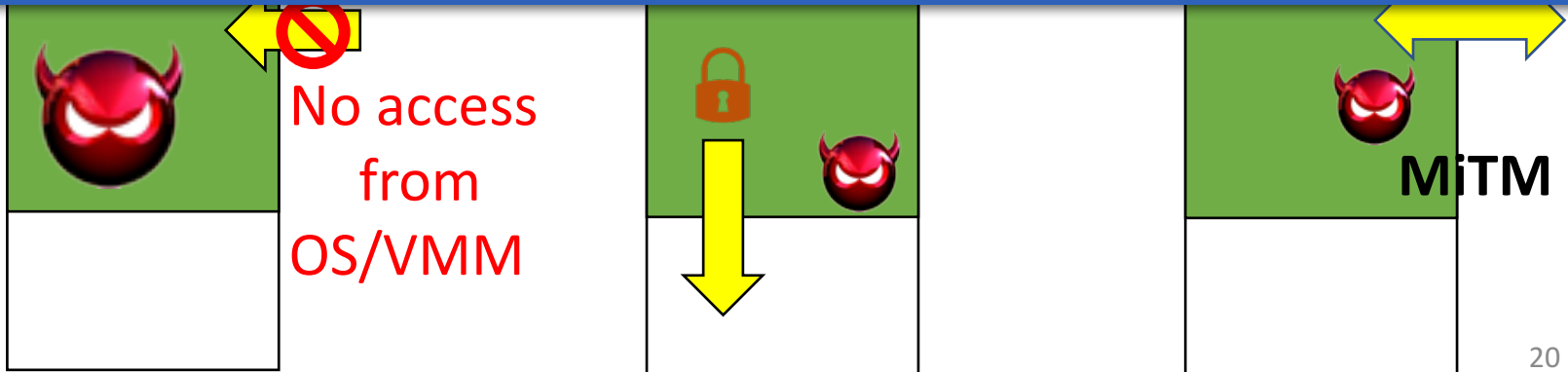
Rootkit



What if Enclave is Compromised?

- SGX protects attackers from auditing/analysis

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



Potential Post Exploitation

- Dumping confidential data
 - i.e., memcpy(non-enclave region, enclave, size)
- Permanent parasite
 - i.e., MiTM on the remote attestation

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 SGX is not a silver bullet against traditional security threats. Our work shows that on enclave development, we can use Dark-ROP to bypass trusted computing hardware (e.g., Intel SGX, ARM TrustZone, Haven).

SEC'17

Traditional Attack Vectors

- Cache-based side channel
- Memory safety
- Weak mitigation techniques (e.g., ASLR)
- ★ • Uninitialized padding in EDL

Cache-based Side-channel Attacks

CacheZoom: How SGX Amplifies
The Power of Cache Attacks

arXiv'17

Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth

Worcester
{amog}

SEC'17

Software Grand Exposure: SGX Cache Attacks Are Practical

Ferdinand Brasser¹, Urs Müller², Alexandra Dmitrienko², Kari Kostinen², Srdjan Capkun², and
Ahmad-Reza Sadeghi¹

EuroSec'17

Abstract. In modern
commonly shared, a
can cause privacy to
forced. Intel proposes
within the processor

Cache Attacks on Intel SGX

CR] 24 Feb 2017

Side-channel
dependence
execution
quently,
ing coun
widely
side cha

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.de

arXiv'17

ABSTRACT

For the first time, we practical
SGX enclaves are vulnerable again.
As a case study, we present an
attack on AES when running in
Using Neve and Seifert's elimin
cache probing mechanism relying
to extract the AES secret key in
investigating 480 encrypted bloc
Side-channel attacks on SGX are based

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

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

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

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

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

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

CR] 24 Feb 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)

→ Numerous defenses (e.g., coloring ...)

side cha

ABSTRACT

For the first time, we practical
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
Implementation: see attack in base

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

Cache Attack is Practical Concern?

- It depends on context/applications!
- Performance (= cache) vs. potential risks
- SGX makes the cache attack:
 - Easier: by allowing privileged features (e.g., PMU)
 - Harder: by leveraging isolation / randomization (security by obscurity practical)

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

Traditional Attack Vectors

- Cache-based side channel
- Memory safety
- Weak mitigation techniques (e.g., ASLR)
- ★ • Uninitialized padding in EDL

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 Chu
Taesoo Kim^{*} Marcus Peinado[†] Brent Byunghoon Kan[†]

[‡]KAIST ^{*}Georgia Institute of Technology [†]Microsoft

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 adversarial attacks on any application. In a traditional system, attackers often allow themselves to be detected by the operating system. In SGX, however, attackers can often avoid detection by using return-oriented programming (ROP) to hijack the execution flow of the application. In this paper, we show that we can use ROP to break the confidentiality and integrity of SGX enclaves. Our attack, called Hacking in Darkness, is a return-oriented programming attack that can be used to break the confidentiality and integrity of SGX enclaves. We show that our attack can be used to break the confidentiality and integrity of SGX enclaves. We show that our attack can be used to break the confidentiality and integrity of SGX enclaves.

SEC'17

Attack

The consequences of DarkLocker can completely breach the confidentiality and integrity of SGX enclaves. Our attack, called Hacking in Darkness, is a return-oriented programming attack that can be used to break the confidentiality and integrity of SGX enclaves. We show that our attack can be used to break the confidentiality and integrity of SGX enclaves. We show that our attack can be used to break the confidentiality and integrity of SGX enclaves.

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.

In this work, we first showcase that two prominent software- and hardware-based defenses, AddressSanitizer and Intel MPX respectively, are impractical for shielded execution due to their reliance on memory safety.

Defense

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

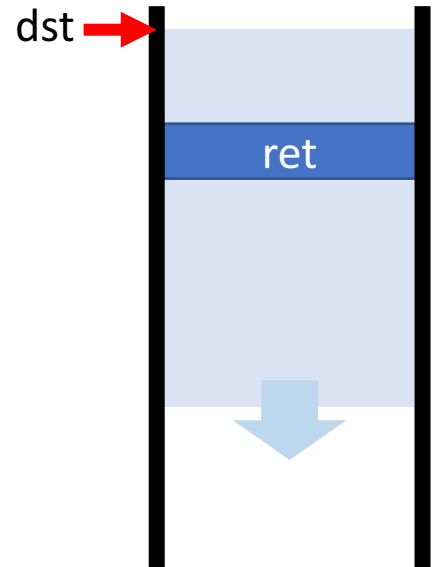
Shielded execution aims to protect confidentiality and integrity of applications when executed in an untrusted environment [19, 22]. The main idea is to isolate the application from the rest of the system (including privileged software), using only a narrow interface to communicate to the outside, potentially malicious world. Since this interface defines the security boundary, checks are performed to prevent the untrusted environment from accessing the application's memory. In this paper, we show that these checks are often ineffective. We show that our attack can be used to break the confidentiality and integrity of SGX enclaves. We show that our attack can be used to break the confidentiality and integrity of SGX enclaves.

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 accessing the application's memory. In this paper, we show that these checks are often ineffective. We show that our attack can be used to break the confidentiality and integrity of SGX enclaves. We show that our attack can be used to break the confidentiality and integrity of SGX enclaves.

EuroSys'17

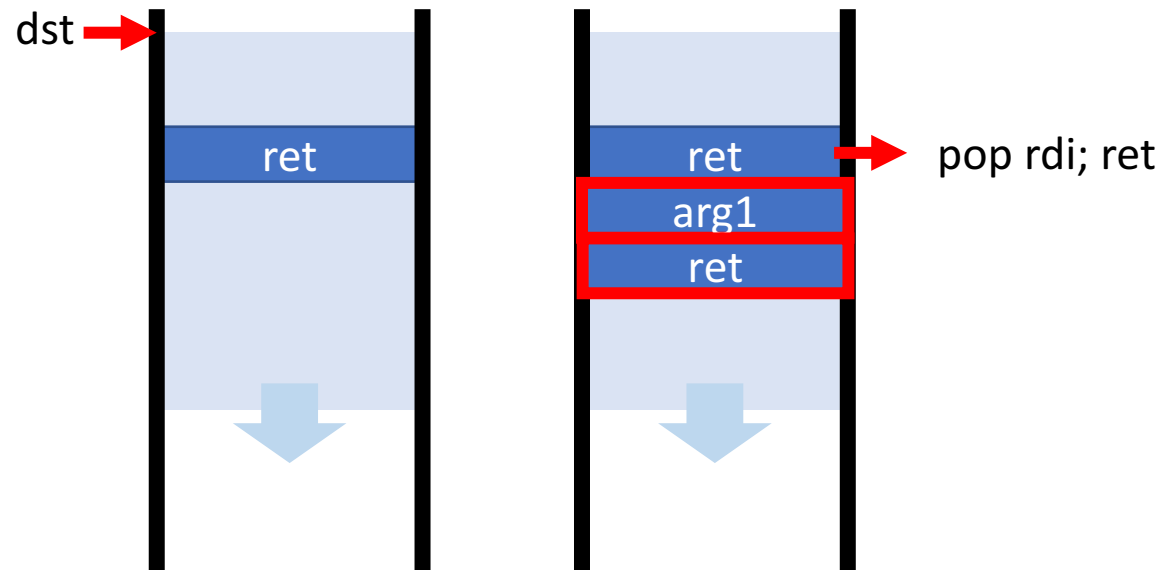
Return-oriented Programming (ROP)

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



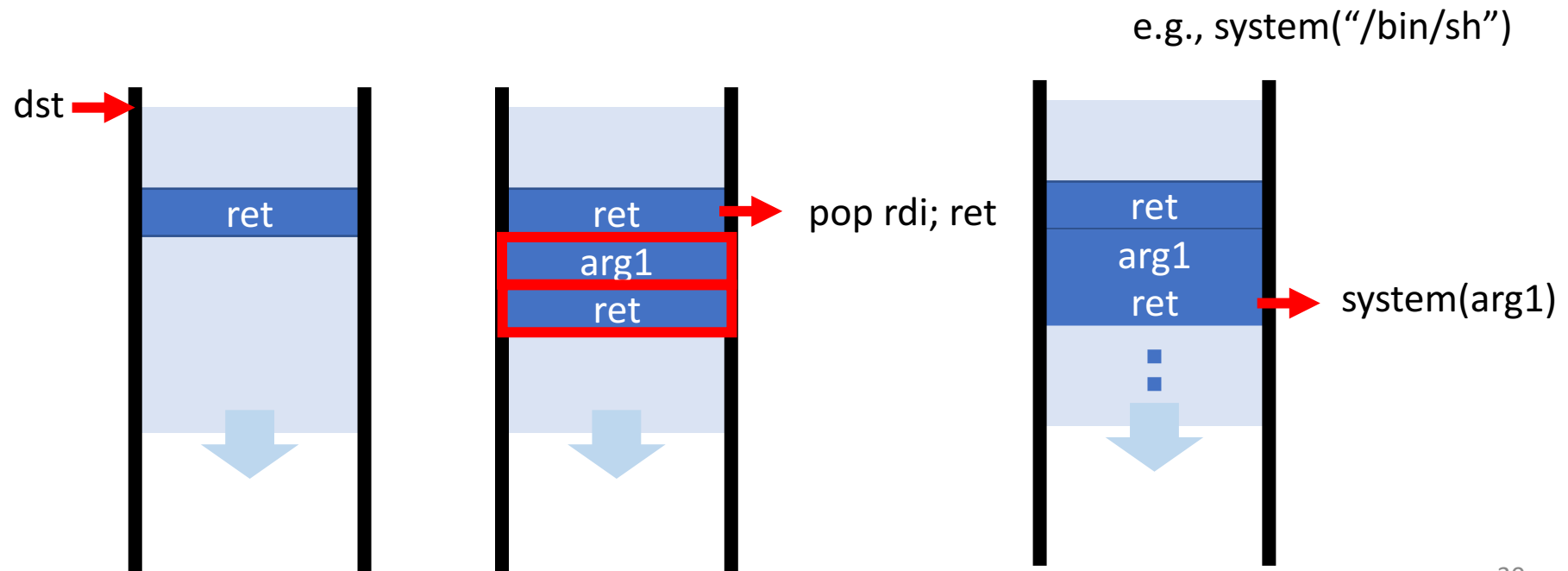
Return-oriented Programming (ROP)

```
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);  
}
```



ROP Inside an Enclave

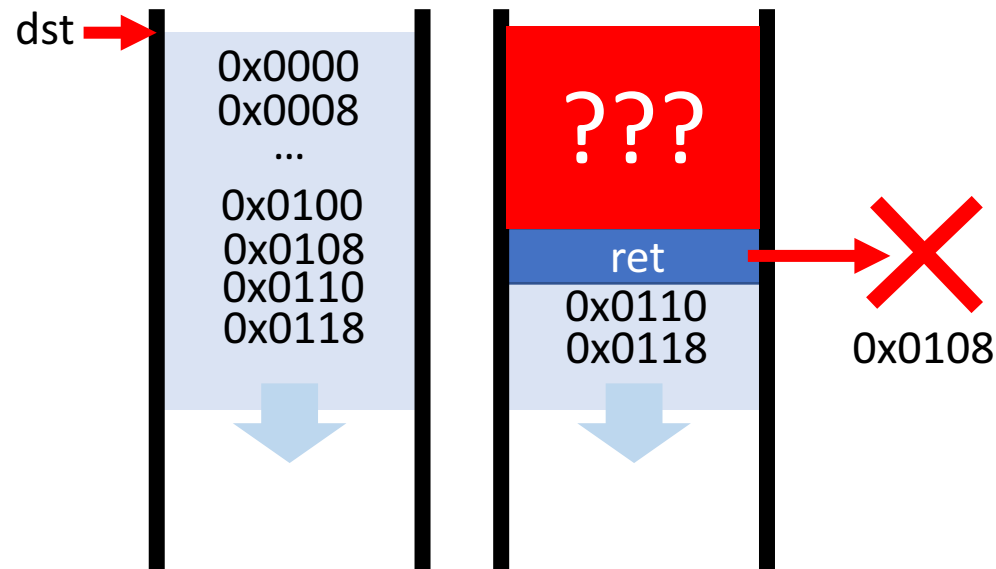
```
void vuln(char *input) {  
    char dst[ ??? ];  
    memcpy(dst, input, ??? );  
}
```

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

ROP Inside an Enclave

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

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



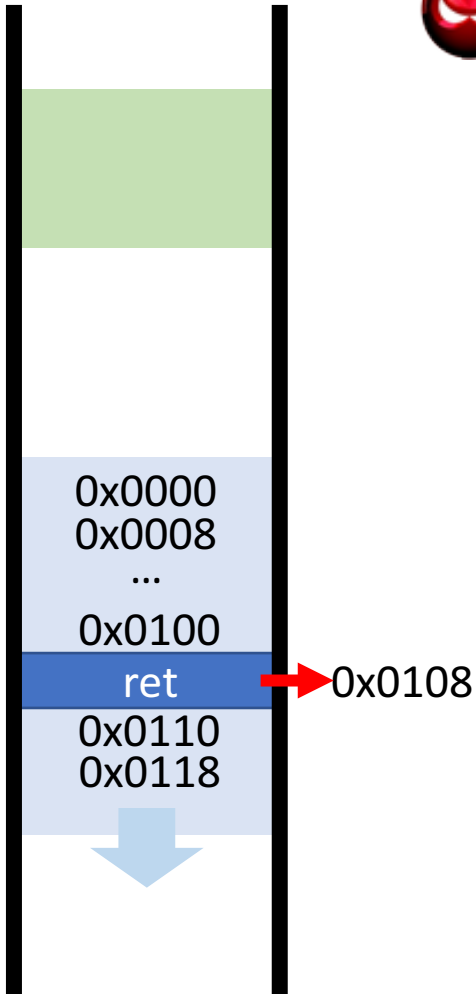
ROP in Darkness: Dark ROP

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

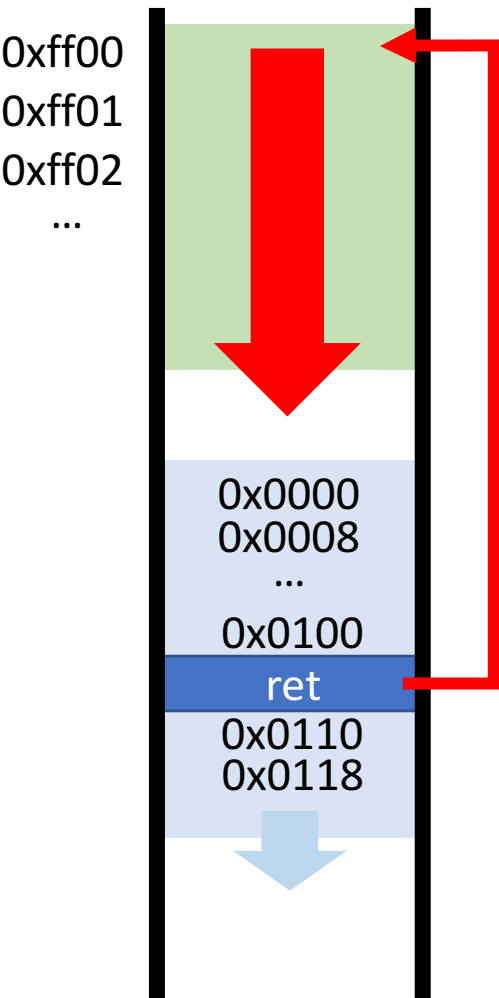
Step 1. Looking for pop Gadgets



You have a full control over the layout of the enclave

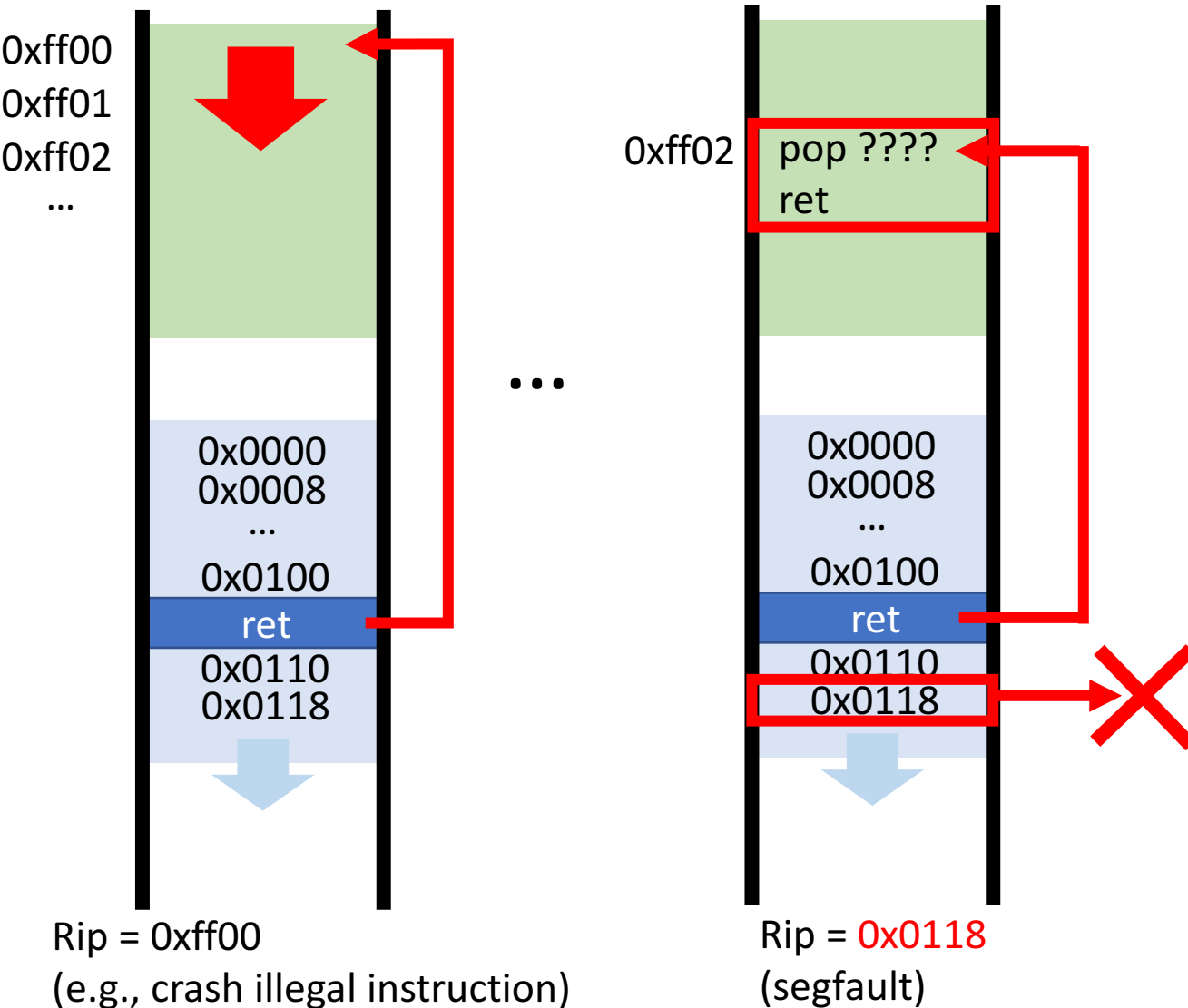


Step 1. Looking for pop Gadgets

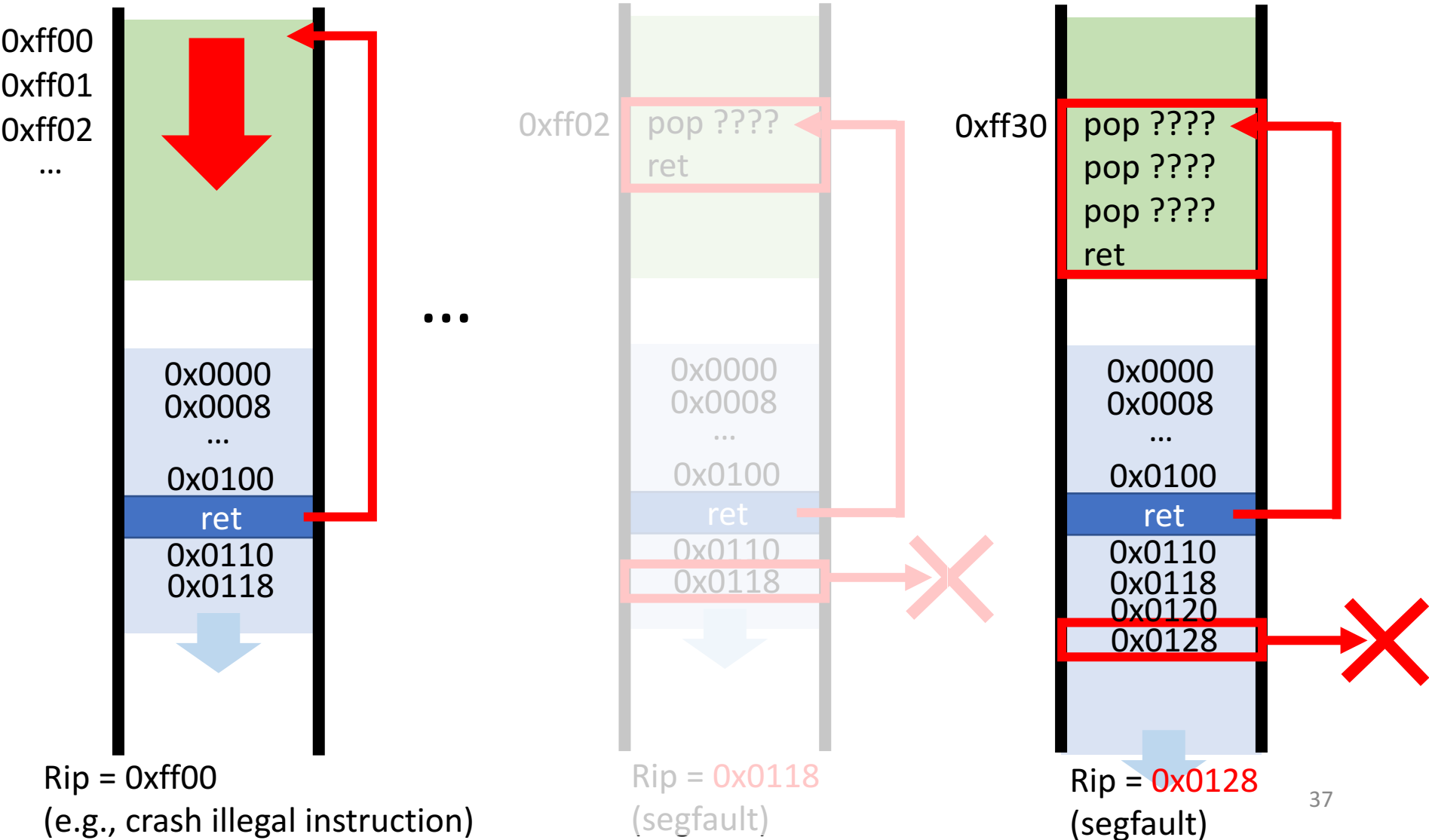


Rip = 0xff00
(e.g., crash illegal instruction)

Step 1. Looking for pop Gadgets



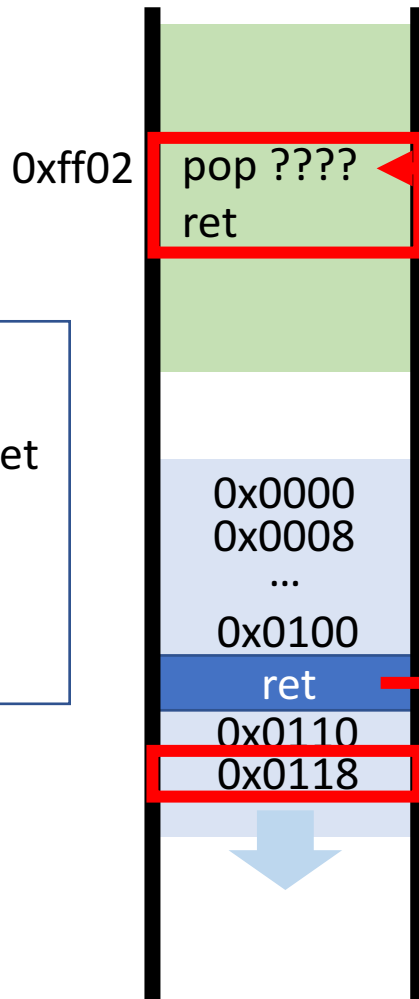
Step 1. Looking for pop Gadgets



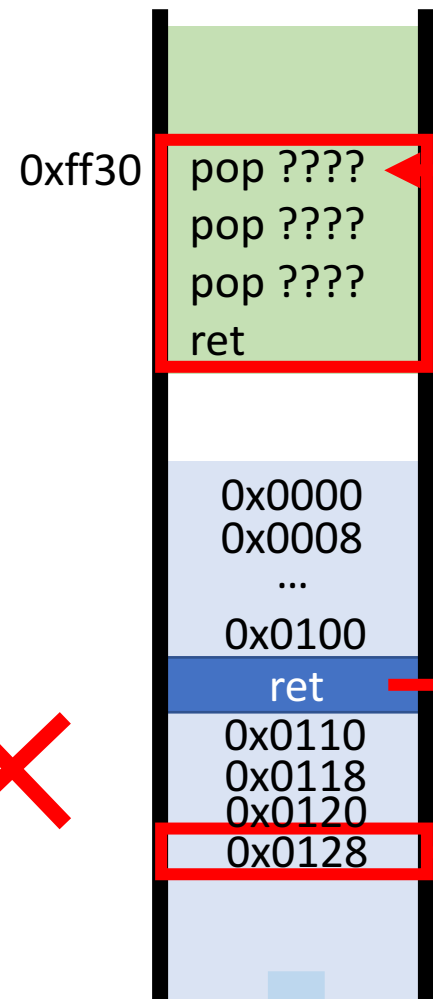
Step 1. Looking for pop Gadgets

Catalog of pop gadgets
(unknown args)

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



Rip = 0x0118
(segfault)



Rip = 0x0128
(segfault)

Step 2. Looking for ENCLU

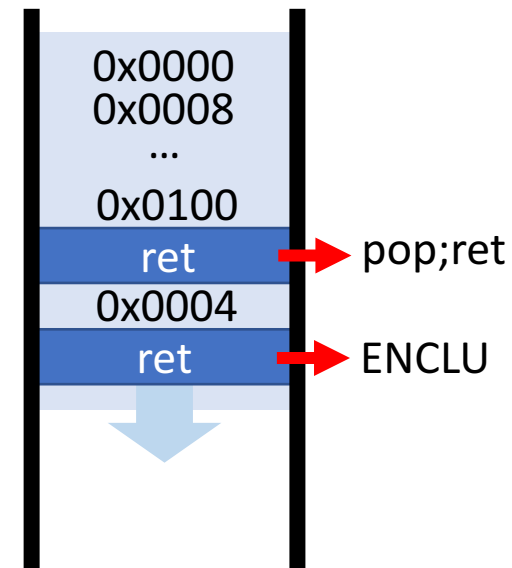
- ENCLU: an inst. dispatches to various leaf functions
 - rax = 0: EREPORT
 - rax = 1: EGETKEY
 - ...
 - rax = 4: EEXIT

Step 2. Looking for ENCLU

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

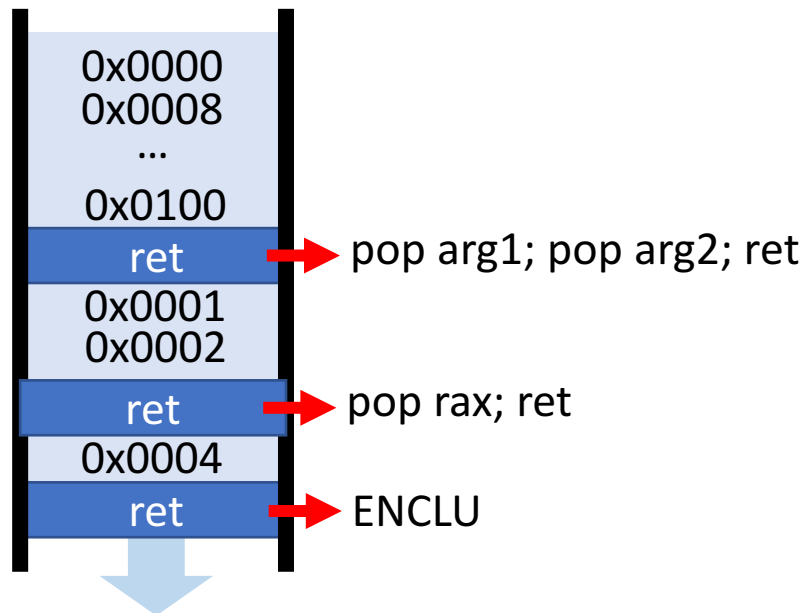
→ Scan code for each “pop????;ret”

→ If gracefully exit, rip = ENCLU



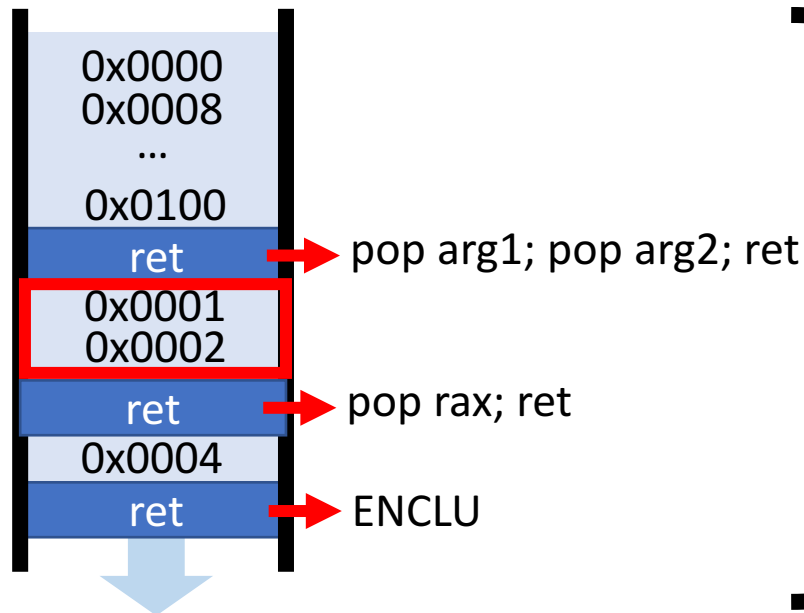
Step 3. Deciphering pop Gadgets

- 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

+ =

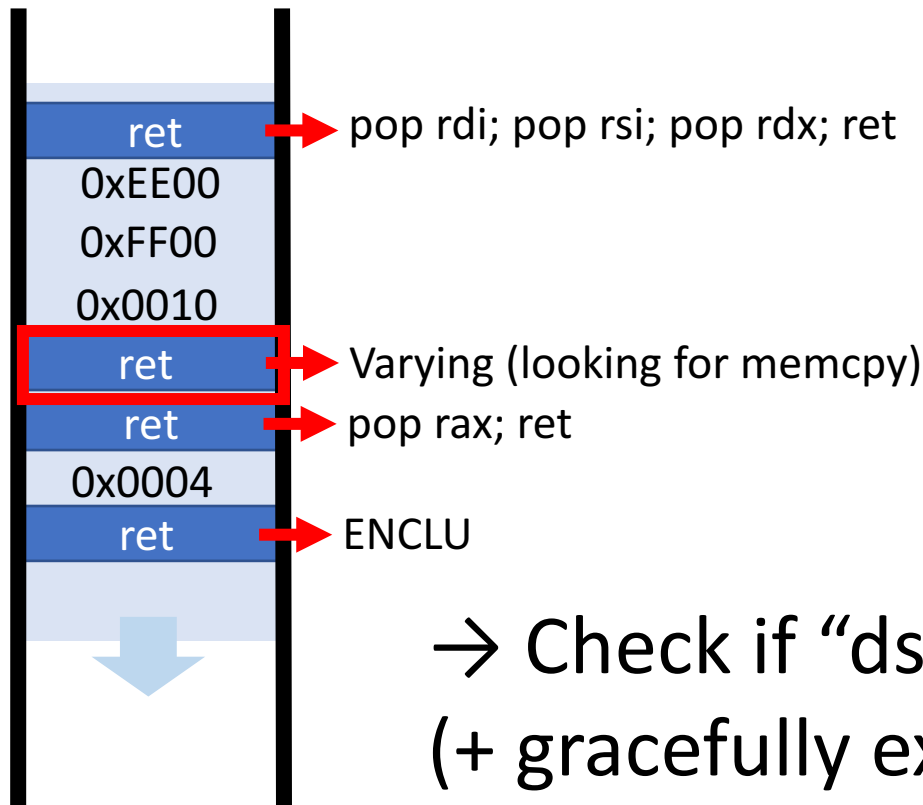
Register file

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

pop rsi
pop rdi
ret

Step 4. Looking for memcpy()

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



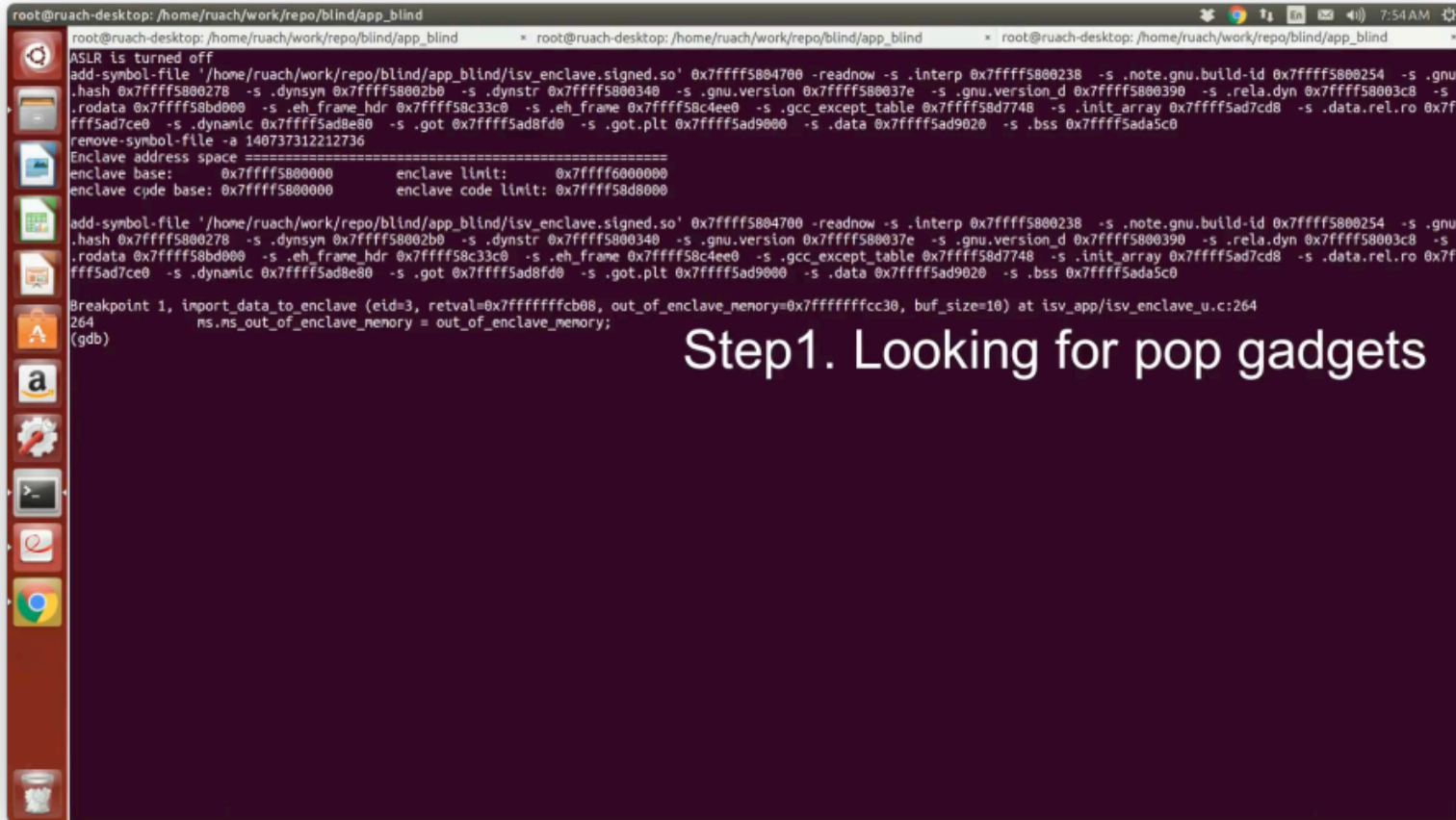
→ Check if “dst” contains 0x10 data (+ gracefully exited)

Gadgets Everywhere (e.g., SDK)

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

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

DEMO: PoC Dark ROP



```
root@ruach-desktop: /home/ruach/work/repo/blind/app_blind
root@ruach-desktop: /home/ruach/work/repo/blind/app_blind
root@ruach-desktop: /home/ruach/work/repo/blind/app_blind
ASLR is turned off
add-symbol-file '/home/ruach/work/repo/blind/app_blind/isv_enclave.signed.so' 0x7ffff5804700 -readnow -s .interp 0x7ffff5800238 -s .note.gnu.build-id 0x7ffff5800254 -s .gnu.hash 0x7ffff5800278 -s .dynsym 0x7ffff58002b0 -s .dynstr 0x7ffff5800340 -s .gnu.version 0x7ffff580037e -s .gnu.version_d 0x7ffff5800390 -s .rela.dyn 0x7ffff58003c8 -s .rodata 0x7ffff58bd000 -s .eh_frame_hdr 0x7ffff58c33c0 -s .eh_frame 0x7ffff58c4ee0 -s .gcc_except_table 0x7ffff58d7748 -s .init_array 0x7ffff5ad7cd8 -s .data.rel.ro 0x7ffff5ad7ce0 -s .dynamic 0x7ffff5ad8e80 -s .got 0x7ffff5ad8fd0 -s .got.plt 0x7ffff5ad9000 -s .data 0x7ffff5ad9020 -s .bss 0x7ffff5ada5c0
remove-symbol-file -a 140737312212736
Enclave address space =====
enclave base: 0x7ffff5800000 enclave limit: 0x7ffff6000000
enclave code base: 0x7ffff5800000 enclave code limit: 0x7ffff58d8000
add-symbol-file '/home/ruach/work/repo/blind/app_blind/isv_enclave.signed.so' 0x7ffff5804700 -readnow -s .interp 0x7ffff5800238 -s .note.gnu.build-id 0x7ffff5800254 -s .gnu.hash 0x7ffff5800278 -s .dynsym 0x7ffff58002b0 -s .dynstr 0x7ffff5800340 -s .gnu.version 0x7ffff580037e -s .gnu.version_d 0x7ffff5800390 -s .rela.dyn 0x7ffff58003c8 -s .rodata 0x7ffff58bd000 -s .eh_frame_hdr 0x7ffff58c33c0 -s .eh_frame 0x7ffff58c4ee0 -s .gcc_except_table 0x7ffff58d7748 -s .init_array 0x7ffff5ad7cd8 -s .data.rel.ro 0x7ffff5ad7ce0 -s .dynamic 0x7ffff5ad8e80 -s .got 0x7ffff5ad8fd0 -s .got.plt 0x7ffff5ad9000 -s .data 0x7ffff5ad9020 -s .bss 0x7ffff5ada5c0
Breakpoint 1, import_data_to_enclave (eid=3, retval=0x7ffff5800000, out_of_enclave_memory=0x7ffff5800000, buf_size=10) at isv_app/isv_enclave_u.c:264
264 ns.ns_out_of_enclave_memory = out_of_enclave_memory;
(gdb)
```

Step1. Looking for pop gadgets

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

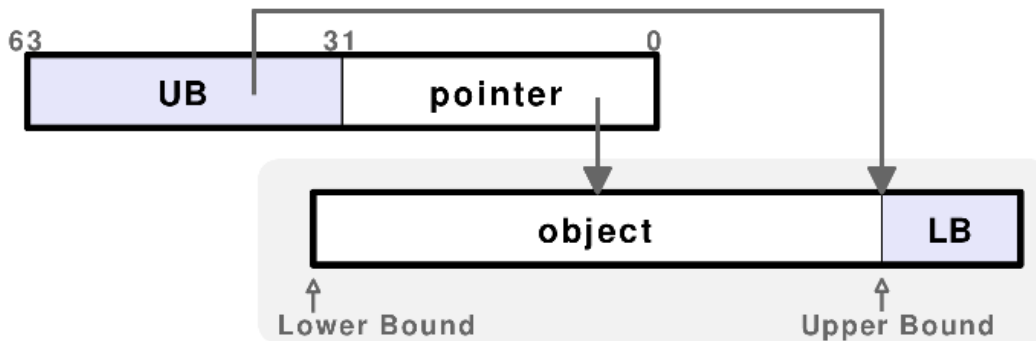
Shielded execution based on Intel SGX provides strong security guarantees for legacy applications running on untrusted platforms. However, memory safety attacks such as Heartbleed can render the confidentiality and integrity properties of shielded execution completely ineffective. To prevent these attacks,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

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

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


Done w/ Memory Safety on SGX?

- SGXBounds is a temporary solution: no temporal safety (i.e., UAF) and SGX likely supports more memory in the future (e.g., large pages)
- Traditional mitigations help (or required)?

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,-0x00(%rbp)
mov   %fs:0x28,%rax
mov   %rax,-0x8(%rbp)
```

prologue

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

epilogue

SGX Mitigation Checklist

- Popular mitigation schemes:

- Stack Canary

- RELRO

- DEP/NX

- ASLR/PIE

Defense: ASLR/SW-DEP inside SGX

- Popular mitigation schemes:

- ✓ Stack Canary

- ✓ RELRO

- ✗ DEP/NX

- ✗ ASLR/PIE

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

Jacback Seo^{*‡}, Byoungyoung Lee^{‡§}, Seongmin Kim^{*}, Ming-Wei Shih[‡],
Insik Shin^{*}, Dongsu Han^{*}, Taesoo Kim[‡]

^{*}KAIST [‡]Purdue University [§]Georgia Institute of Technology

{jacback, 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 code footprint, which is not supported as library in Haven [12]. This paper introduces SGX-Shield for Intel SGX [28, 29], which mitigates memory corruption in unsafe programming models by providing a memory layout in an assembly language.

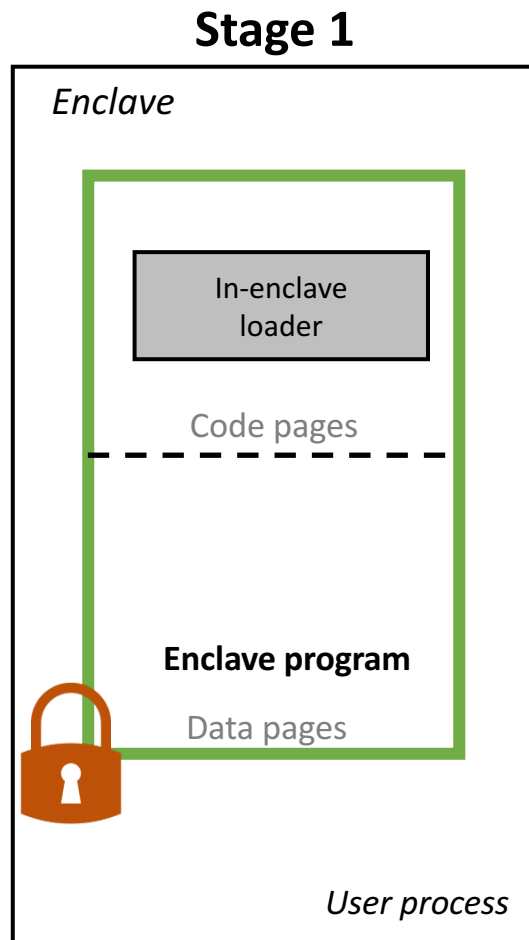
NDSS'17

Challenges for Mitigation Schemes

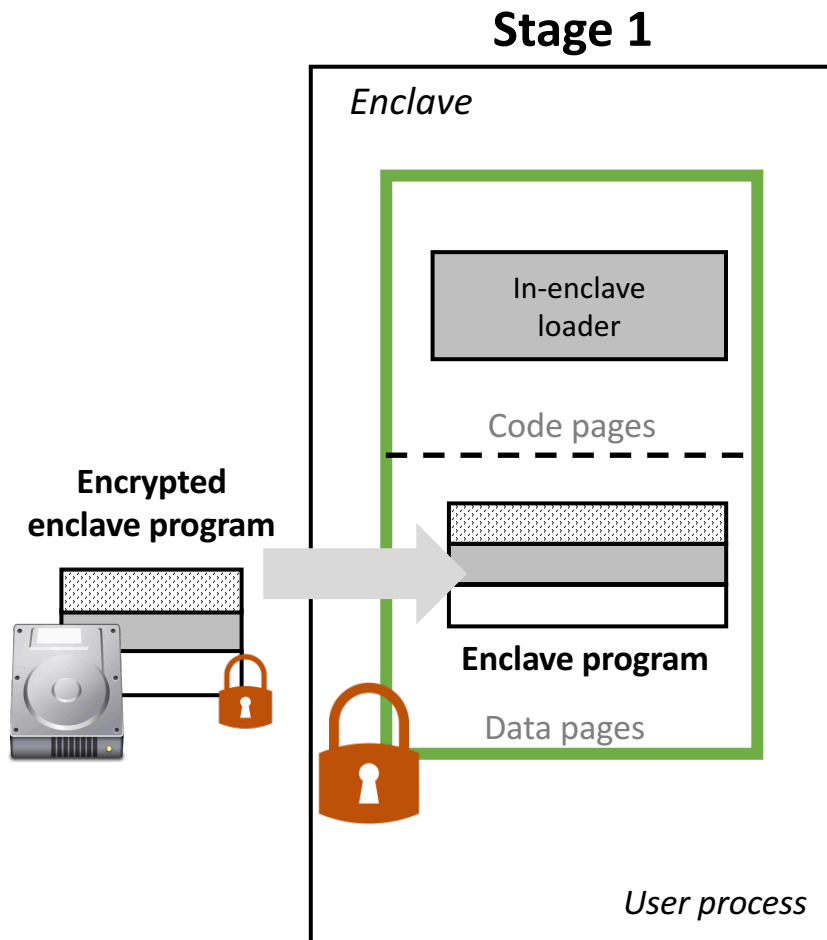
It is non-trivial when an attacker is the kernel:

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

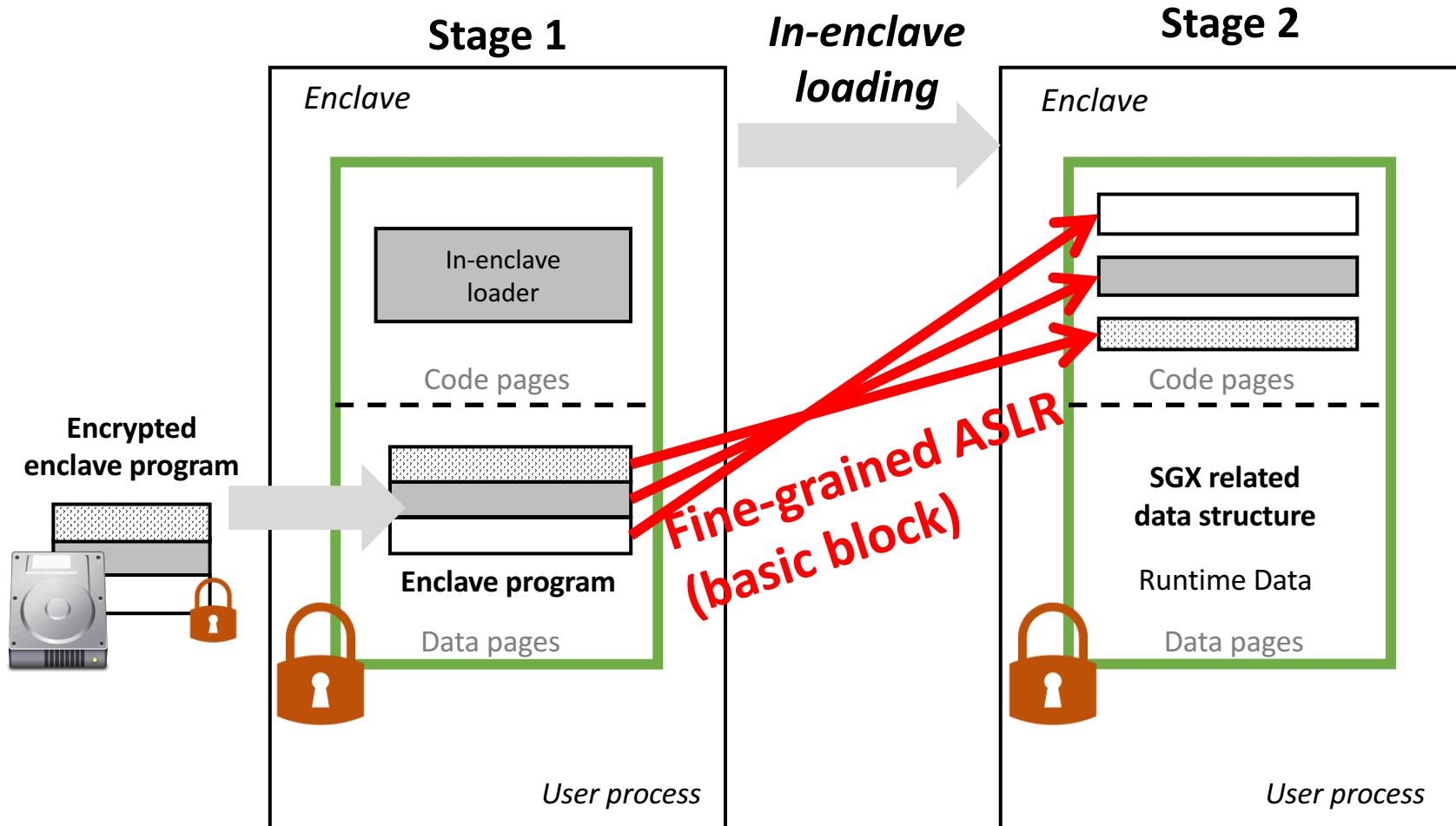
SGX-Shield's Approach: In-enclave Loading



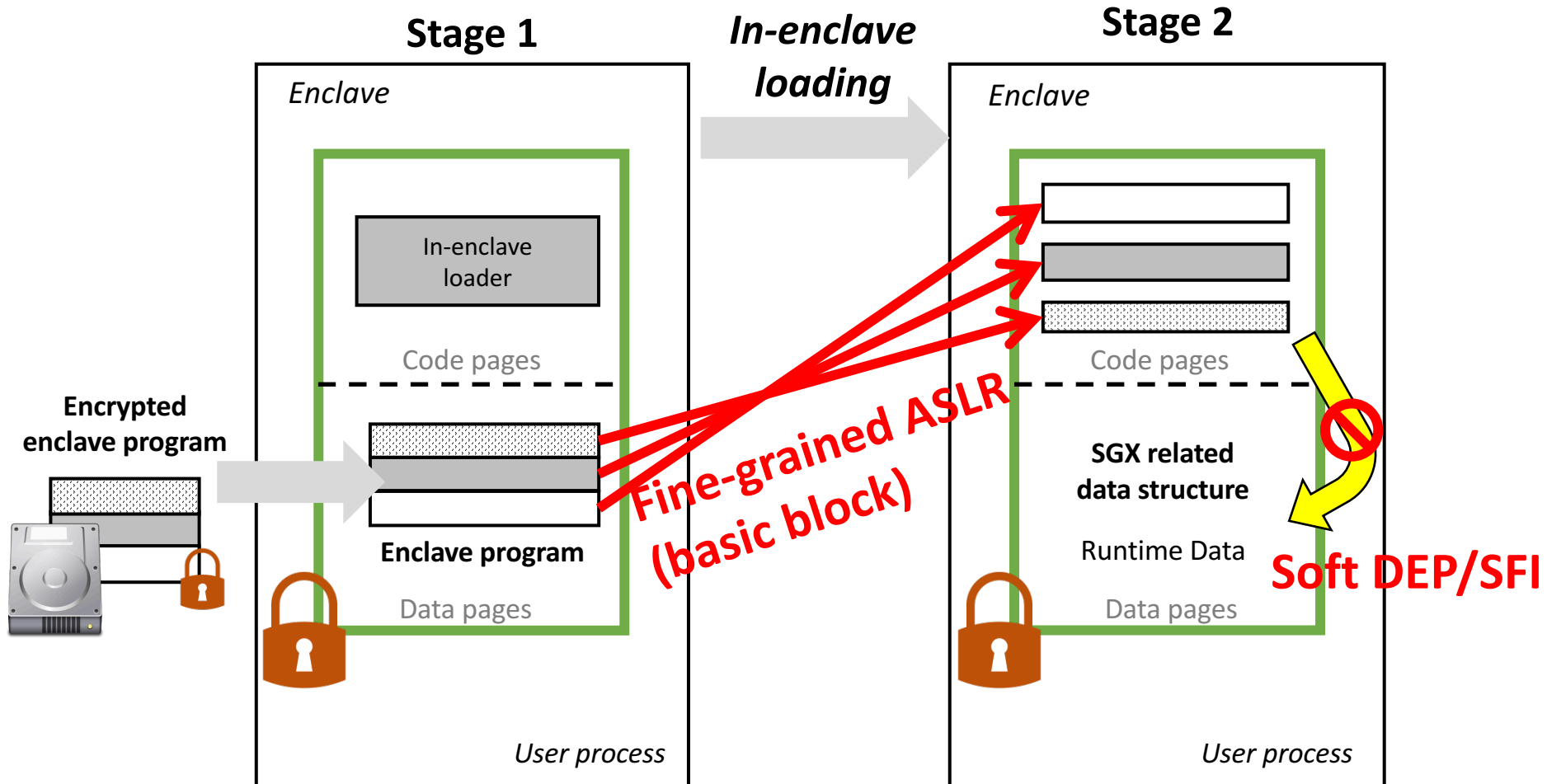
SGX-Shield's Approach: In-enclave Loading



SGX-Shield's Approach: In-enclave Loading



SGX-Shield's Approach: In-enclave Loading



DEMO: SGX-Shield

```
sgx3: ~ (ssh)  
mingwei@sgx3:~/workspace/sgx-attack/SGX-Shield/loader$ make clean  
mingwei@sgx3:~/workspace/sgx-attack/SGX-Shield/loader$ mak
```

Compile the enclave with SGX-Shield

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

★ Uninitialized Padding Problem

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

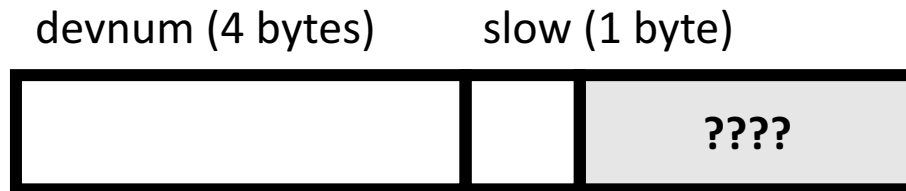
★ Uninitialized Padding Problem

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

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

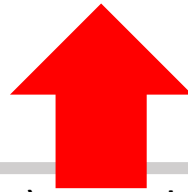
★ Uninitialized Padding Problem

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

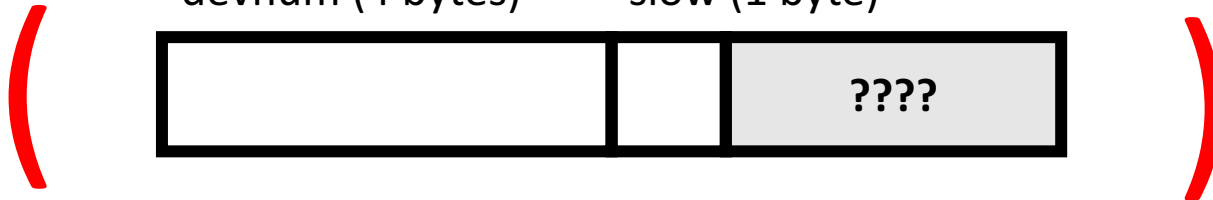


```
struct usbdevfs_connectinfo {  
    .devnum = 1,  
    .slow = 0,  
};
```

★ Uninitialized Padding Problem

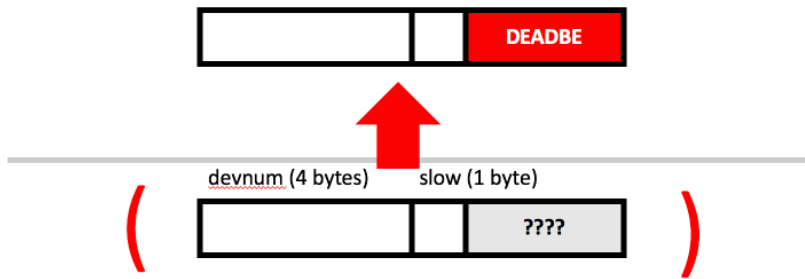


devnum (4 bytes) slow (1 byte)



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

★ Uninitialized Padding Problem



UniSan: Proactive Kernel Memory Initialization to Eliminate Data Leakages

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

ABSTRACT

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

1. INTRODUCTION

As the de facto trusted computing base (TCB) of computer systems, the operating system (OS) kernel has always been a prime target for attackers. By compromising the kernel, attackers can escalate their privilege to steal sensitive data in the system and control the whole computer. There are three main approaches to launch privilege escalation attacks: 1) direct code reuse attacks [17]; 2) k2usr attacks [17]; and 3) code reuse attacks (e.g., Return-Oriented Programming (ROP) and Return-Oriented Programming (ROP) prevention) protection has been de

CCS'16

Ecall/Ocall: EDL Interface for SGX

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

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 {
    void e/ocall_test_struct(test_struct ts);
}
```

Leaking Uninitialized Secure Enclave Memory via Structure Padding (Extended Abstract)

Sangho Lee Taesoo Kim

Georgia Institute of Technology

Abstract

Intel 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 structure variables contain padding bytes, uninitialized

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 functions need to be copied to normal memory first and then copied back to the enclave caller later. The Edger8r tool creates an such edge

cs.CR/25 Oct 2017

arXiv'17

DEMO: SGX Bleed POC

```
251 printf("    ret.val3: 0x%lX\n", ret.val3);
252
253 unsigned char *ptr = (unsigned char*)&ret;
254 printf("    ret: ");
255 for (int i = 0; i < sizeof(ret); ++i) {
256     if (i % 8 == 0) printf("\n    | ");
257     printf("%0X ", ptr[i]);
258     if (i % 8 == 7) printf("|");
259     if (i == 15) printf(" <= copied via padding!");
260 }
261
262 /* Destroy the enclave */
263 sgx_destroy_enclave(global_eid);
264 return
265}
266
```

Sensitive data leaked via padding!

```
sizeof(t1->val2): 1
sizeof(t1->val3): 8
t1->val  = 0x1111222233334444
t1->val2 = 0x99
t1->val3 = 0x5555666677778888
*t1:
| 44 44 33 33 22 22 11 11 |
| 99 AD BE EF DE AD BE EF | <= uninit. va
lues
| 88 88 77 77 66 66 55 55 |

4. return *t1

[main]
ret: 0x1111222233334444 (addr: 0x7fff9ab35a50)
sizeof(ret): 24
sizeof(ret.val1): 8
sizeof(ret.val2): 1
sizeof(ret.val3): 8
ret.val1: 0x1111222233334444
ret.val2: 0x99
ret.val3: 0x5555666677778888
ret:
| 44 44 33 33 22 22 11 11 |
| 99 AD BE EF DE AD BE EF | <= copied via
padding!
| 88 88 77 77 66 66 55 55 |

~/sgx-bleed/poc/SGX_PADDING_POC master* sangh
o@sgx-workstation 12s
```

-UU-: @----F20 App.cpp Bot (251,0) Git:master (C++/l AC

Using Rust SGX SDK?



Rust SGX SDK

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

v0.9.0 Release

Almost there! Rust SGX SDK v0.9.0 is coming up as a beta version of the future v1.0.0, with the as well as many new features! Also we added support for programming SGX untrusted part in it's easy to port Rust crates to the SGX trust execution environment and write the whole SGX refer to [release_notes](#) for further details.

Good news! Our poster 'Rust SGX SDK: Towards Memory Safety in Intel SGX Enclave' [\[pdf\]](#) has Please kindly cite our poster if you like Rust SGX SDK!

<https://github.com/baidu/rust-sgx-sdk>

Using Rust SGX SDK?

- A promising direction to address traditional attack vectors
- But, it still suffers from SGX-Bleed!

New Attack Vectors

- Page table attack
- Branch shadowing attack
- ★ • Rowhammer against SGX

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 numbers of vulnerabilities in popular feature-rich commodity operating systems has inspired a long line of work on exploiting these vulnerabilities to gain unauthorized access to sensitive information. This paper presents a new class of attacks that exploit deterministic side channels to reveal sensitive information without the need for a trusted channel or a trusted component in the operating system. These attacks are deterministic and can be used to reveal sensitive information in a wide range of operating systems, including those that are widely used in enterprise environments.

SP'15

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

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
kaj

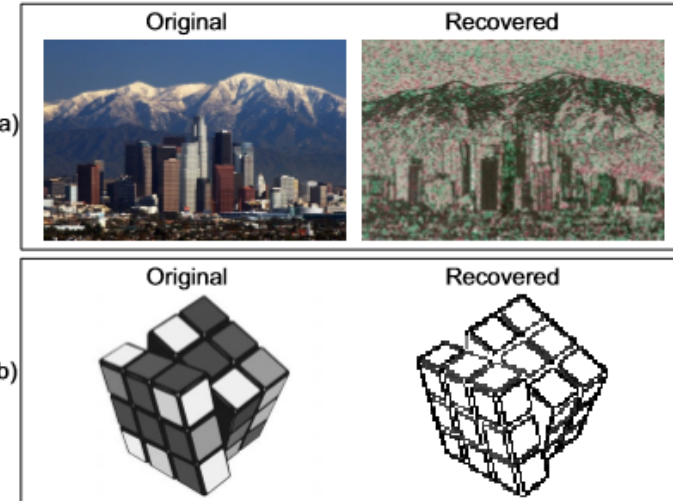
Frank Piessens
imec-DistriNet, KU Leuven
frank.piessens@cs.kuleuven.be

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

Abstract

Protected module architectures, such as Intel SGX, enable strong trusted computing guarantees for hardware-enforced enclaves on top of a potentially malicious operating system. However, such enclaved execution environments are not immune to side-channel attacks. In this paper, we present a new class of attacks that exploit deterministic side channels to reveal sensitive information without the need for a trusted channel or a trusted component in the operating system. These attacks are deterministic and can be used to reveal sensitive information in a wide range of operating systems, including those that are widely used in enterprise environments.

ware to make it relatively easy to run legacy applications with hardware prevention. An essential hardware prevention mechanism is the ability to read or write a memory location, or



Sec'17

DEMO: Page Fault Attack

```
mingwei@sgx3:~$ sudo dmesg -wH
```

```
[Oct24 04:28] [SGX PAGE TABLE ATTACK HELPER] Init done
```

```
[ +10.435019] [SGX PAGE TABLE ATTACK] Set trap the range 0x00007ffff0008000 - 0x00007ffff0015000
```

```
[ +8.567998] [SGX PAGE TABLE ATTACK] Access page access: 0x00007ffff0015000
```

```
[ +0.000030] [SGX PAGE TABLE ATTACK] Access page access: 0x00007ffff0008000
```

```
[ +0.000005] [SGX PAGE TABLE ATTACK] Access page access: 0x00007ffff0009000
```

```
[ +0.000003] [SGX PAGE TABLE ATTACK] Access page access: 0x00007ffff000b000
```

```
[ +0.000110] [SGX PAGE TABLE ATTACK] Access page access: 0x00007ffff000d000
```

```
[ +0.000003] [SGX PAGE TABLE ATTACK] Access page access: 0x00007ffff000f000
```

```
[ +0.000003] [SGX PAGE TABLE ATTACK] Access page access: 0x00007ffff0011000
```

```
[ +0.000003] [SGX PAGE TABLE ATTACK] Access page access: 0x00007ffff0013000
```

Page sequences:

test()

process_secret()

fun0() - 's'

fun2() - 'e'

fun4() - 'c'

fun6() - 'r'

fun8() - 'e'

fun10() - 't'

The secret input is "secret"!!

"sgx3" 04:28 24-Oct-17

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, including kernel exploits, hardware Trojans,

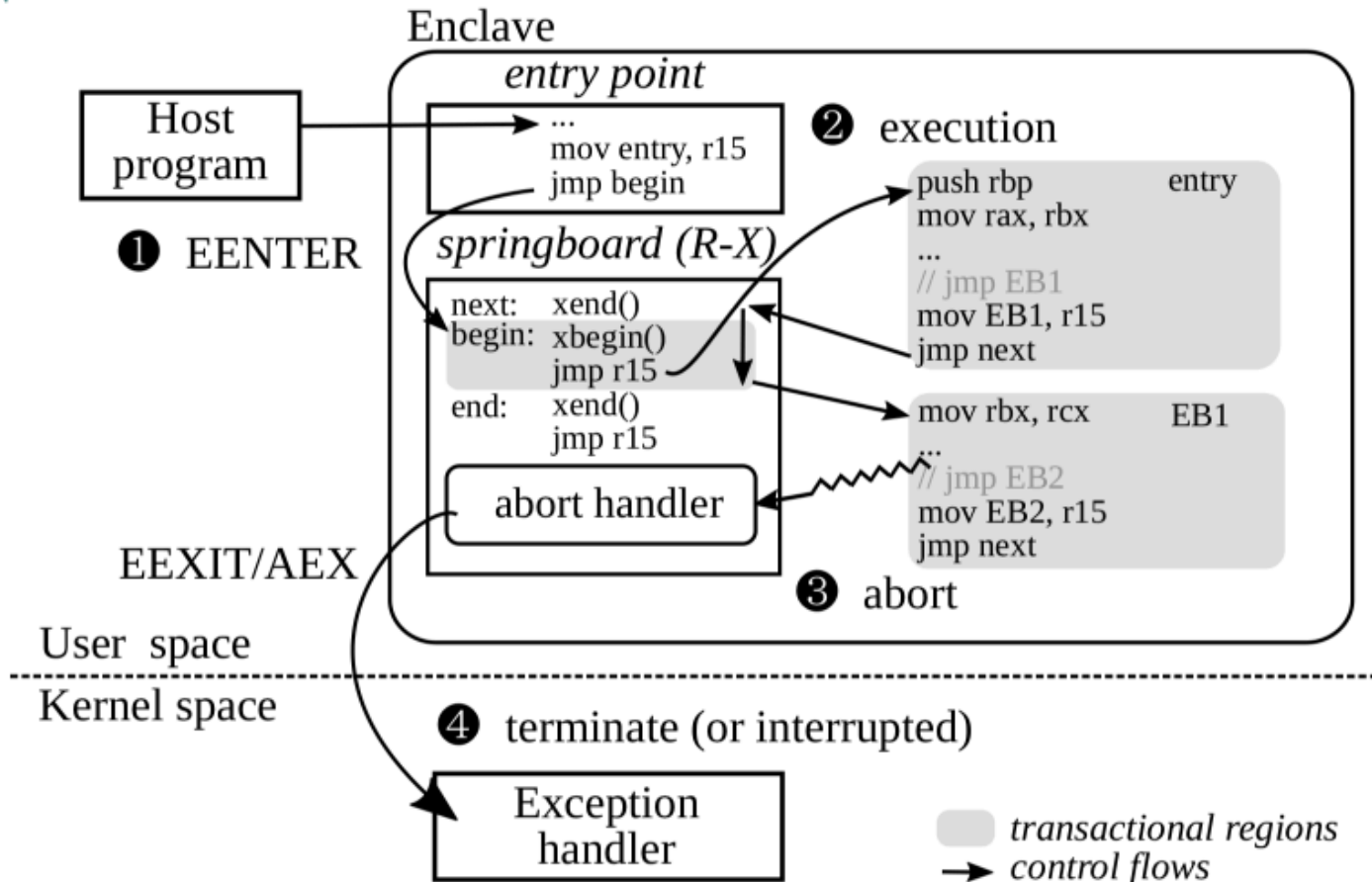
NDSS'17

Key Idea: TSX Isolates Faults!

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

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

Design of T-SGX (Compiler)

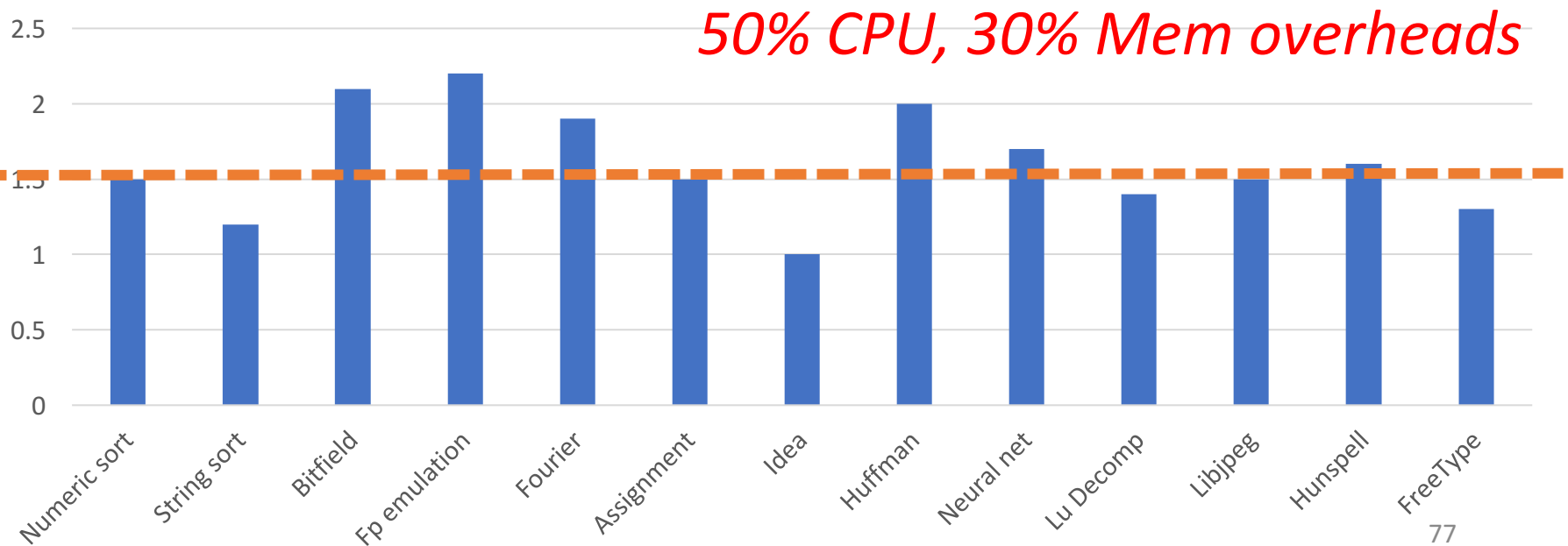


T-SGX: Eradicating Page Faults

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

T-SGX: Eradicating Page Faults

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



DEMO: T-SGX

```
mingwei@sgx3:~$ sudo dmesg -wH
[Oct31 20:45] [SGX PAGE TABLE ATTACK HELPER] Init done
[ +5.214116] [SGX PAGE TABLE ATTACK] Set the trap range 0x00007ffff0007000 - 0x00007ffff0014000, spr
ingboard page 0x00007ffff0032000
[ +5.631861] [SGX PAGE TABLE ATTACK] Access page access: 0x00007ffff0032000

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

**Attacker can only observe
single-page information!**

New Attack Vectors

- Page table attack
- Branch shadowing attack
- ★ • Rowhammer against SGX

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.

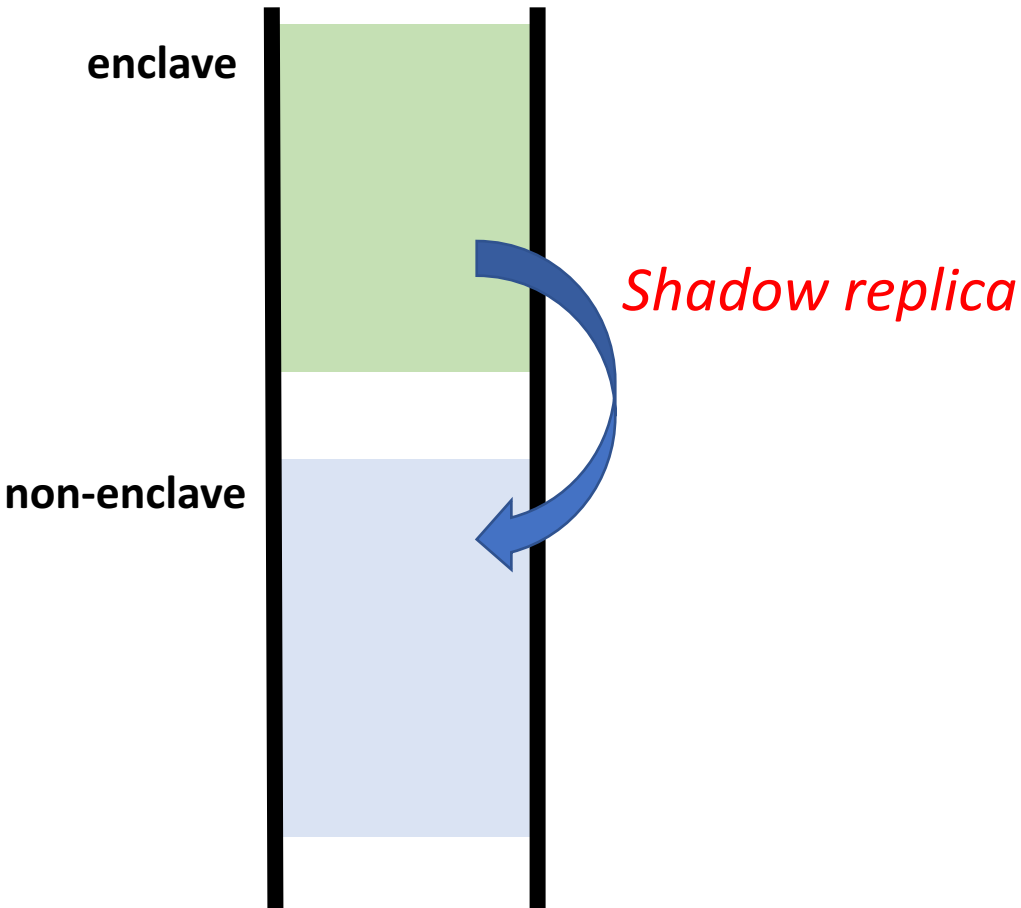
we need either to fully trust the operator, which is problematic [16], or encrypt all data before uploading them to the cloud and perform computations directly on the encrypted data. The latter can be achieved through homomorphic encryption, which is still slow, or through secure preserving encryption, which is still slow. This is especially when we use a private cloud or personal workstation,

SEC'17

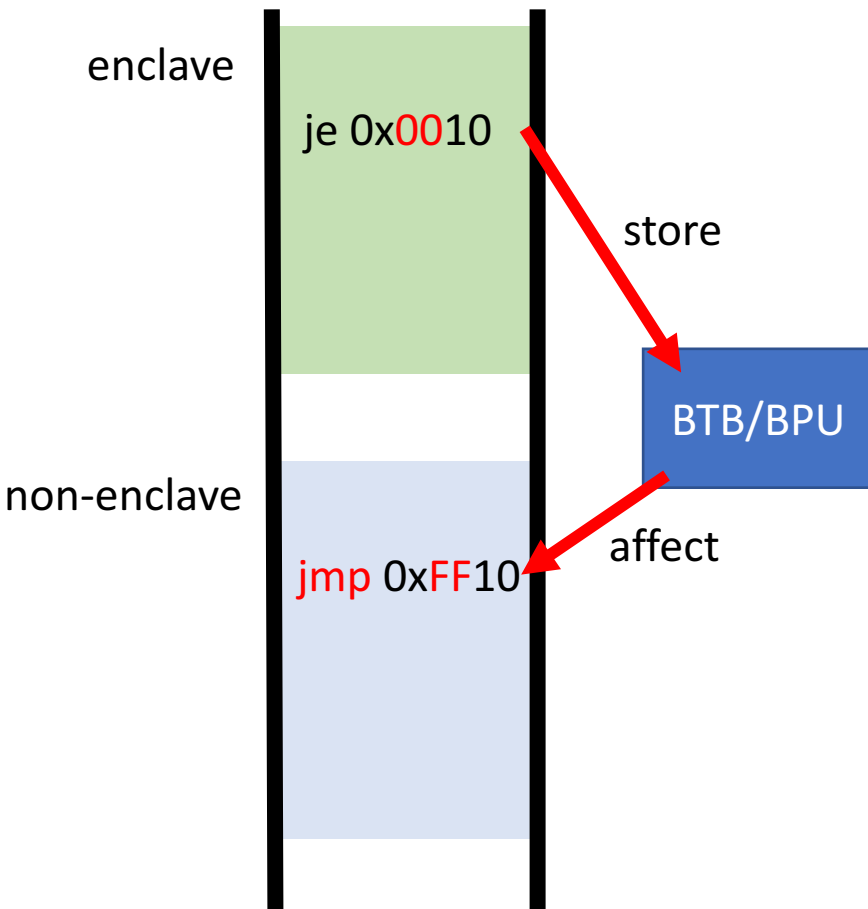
Idea: Exploiting New HW Features




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

Our Approach: Branch Shadowing

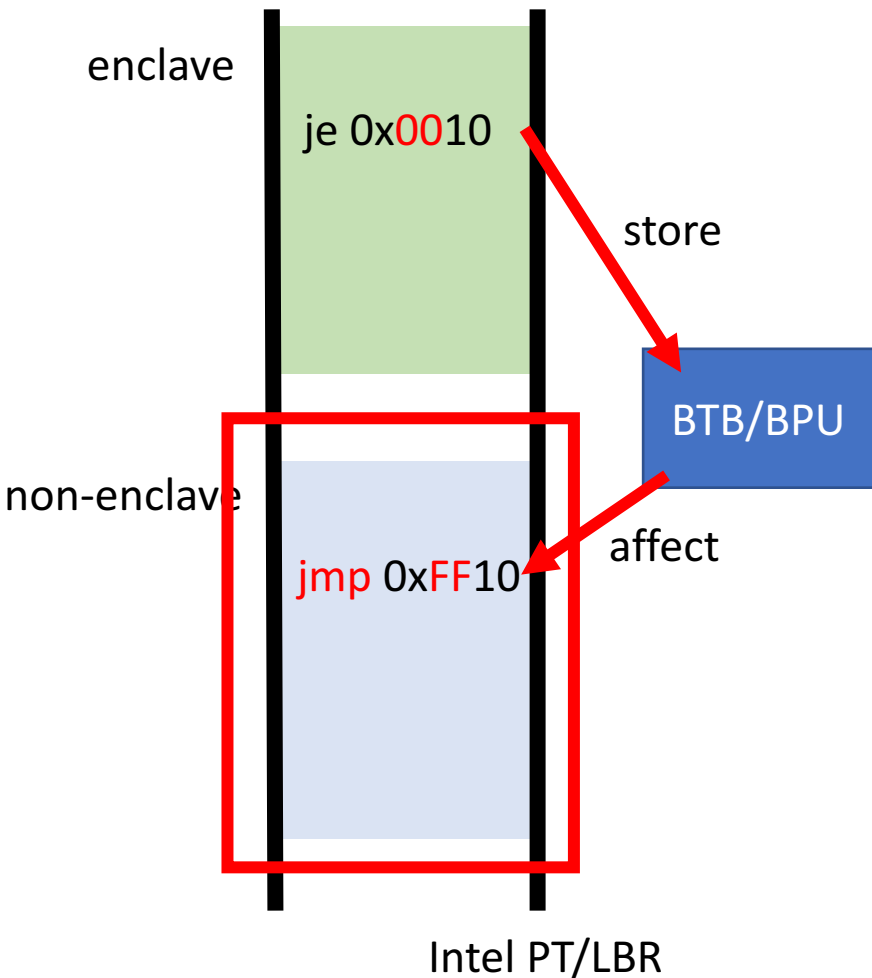


Our Approach: Branch Shadowing



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

Our Approach: Branch Shadowing

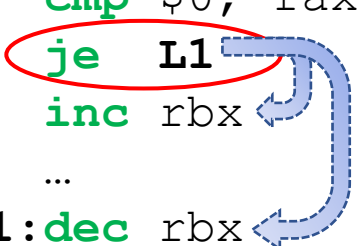


- are mapped onto the same branch prediction buffer
- is a shadow copy of an enclave program forced to take all branches (e.g., je → jmp)
- Monitor with LBR/PT and extract branch prediction results indirectly

Branch Prediction 101

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

```
...  
  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
```

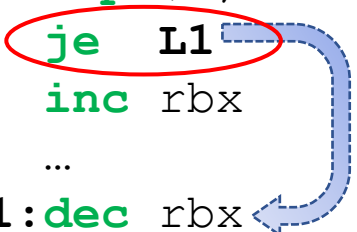
Make this prediction if

- 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

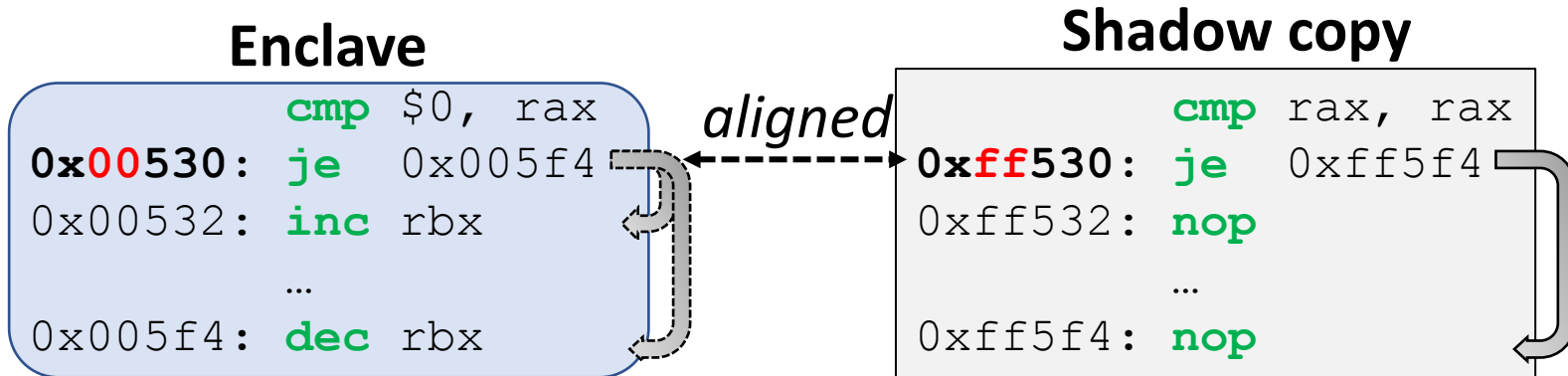
Example: Inferring Cond. Branch

Enclave

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

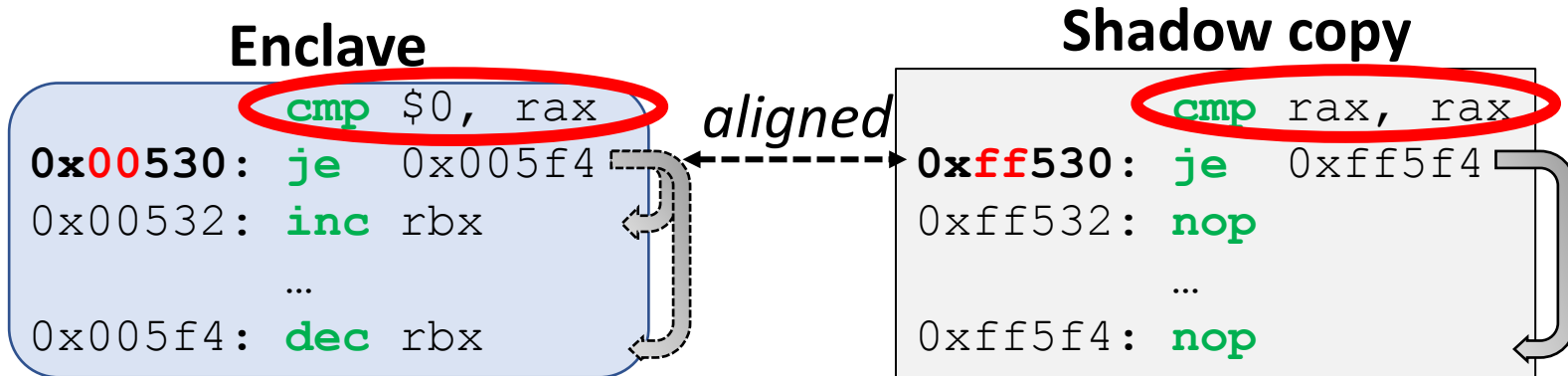
A dashed grey arrow originates from the right side of the instruction at address 0x00530 and points to the right side of the instruction at address 0x005f4, indicating a conditional branch.

Example: Inferring Cond. Branch



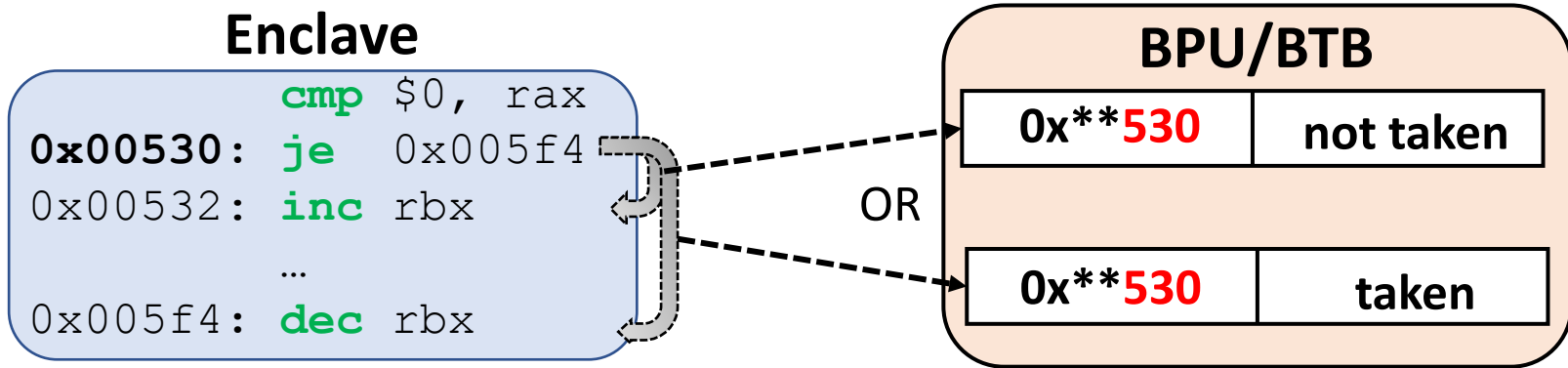
- Prepare a shadow copy w/
 - Colliding conditional branches

Example: Inferring Cond. Branch

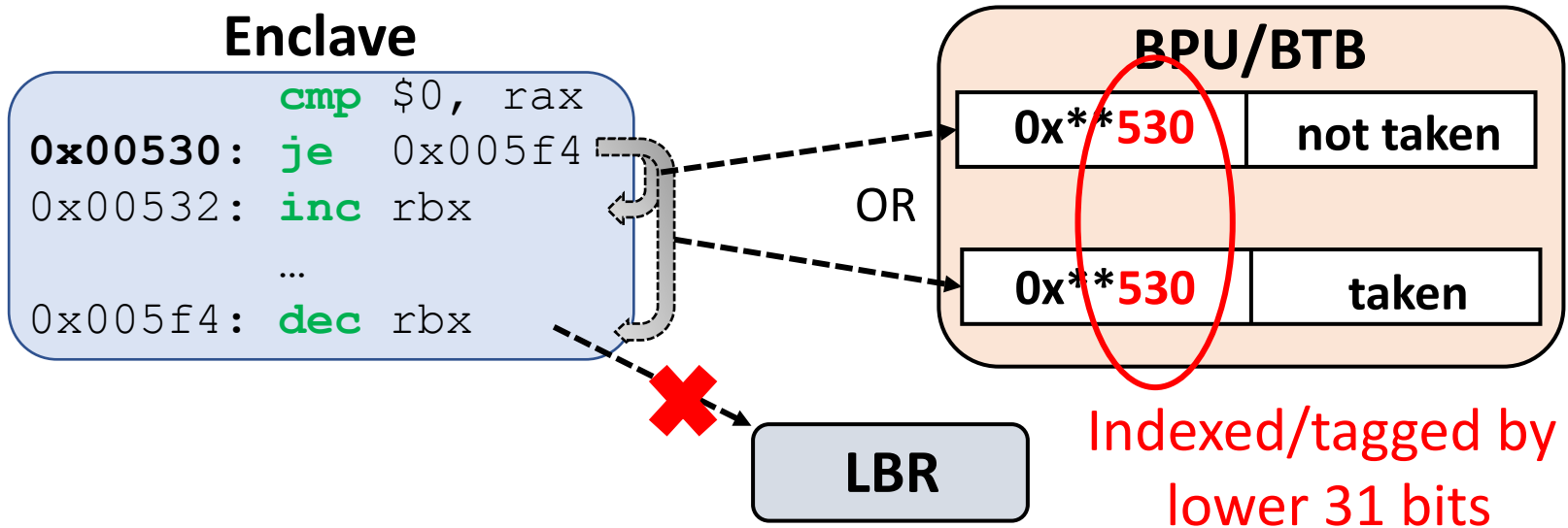


- Prepare a shadow copy w/
 - Colliding conditional branches
 - Always to be taken (to be monitored by LBR)

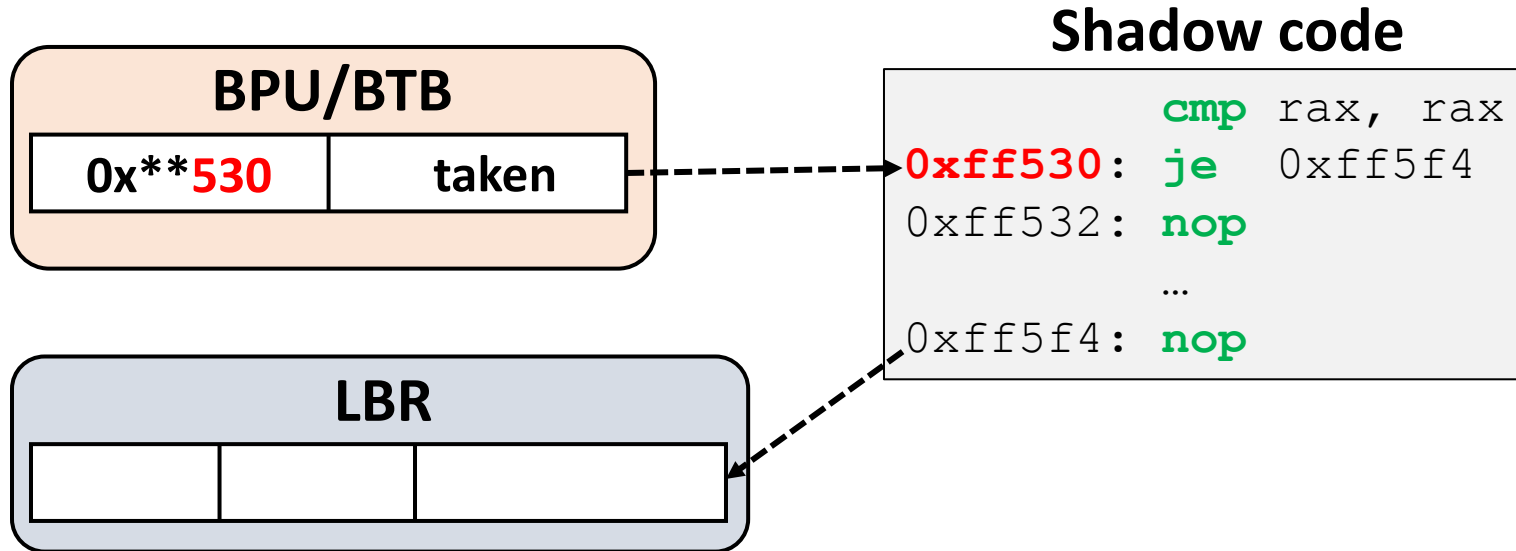
Example: Inferring Cond. Branch



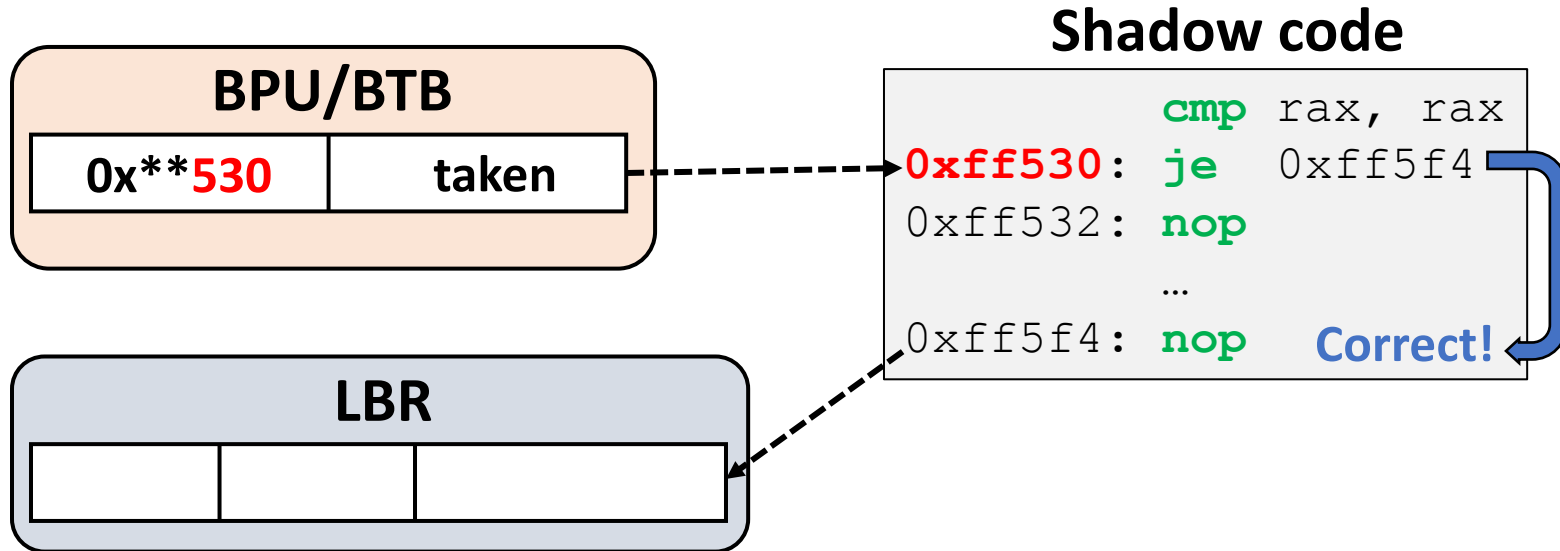
Example: Inferring Cond. Branch



Example: Inferring Taken Branch

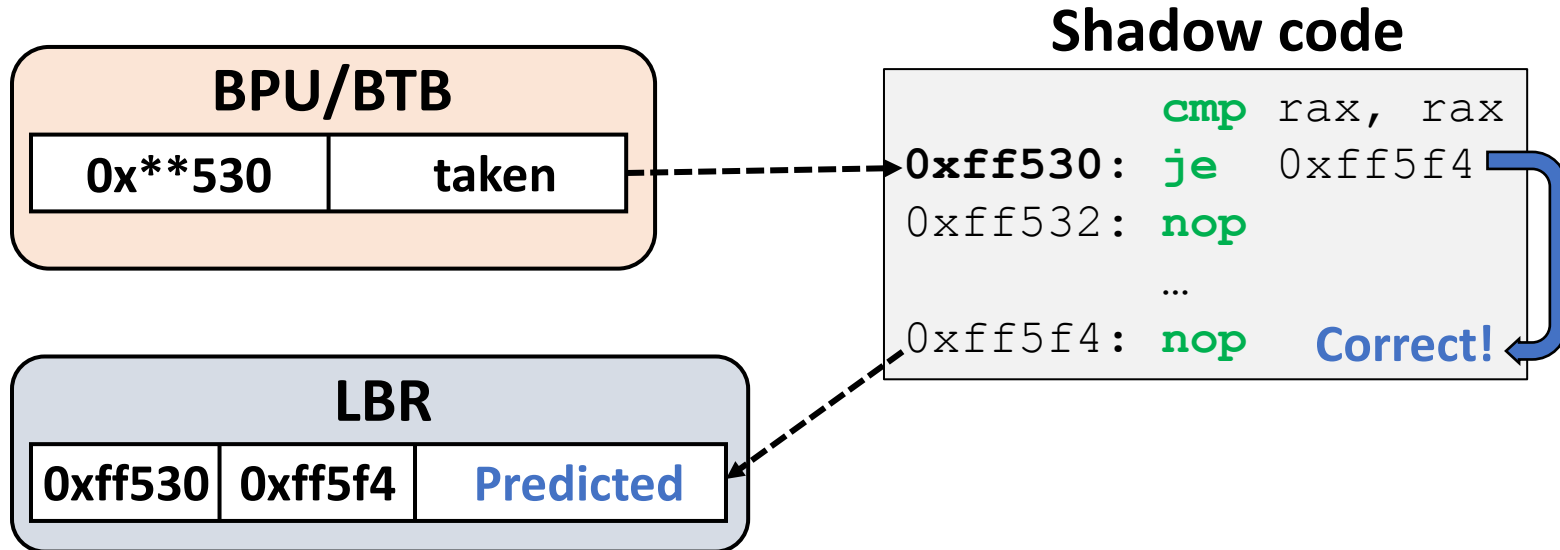


Example: Inferring Taken Branch



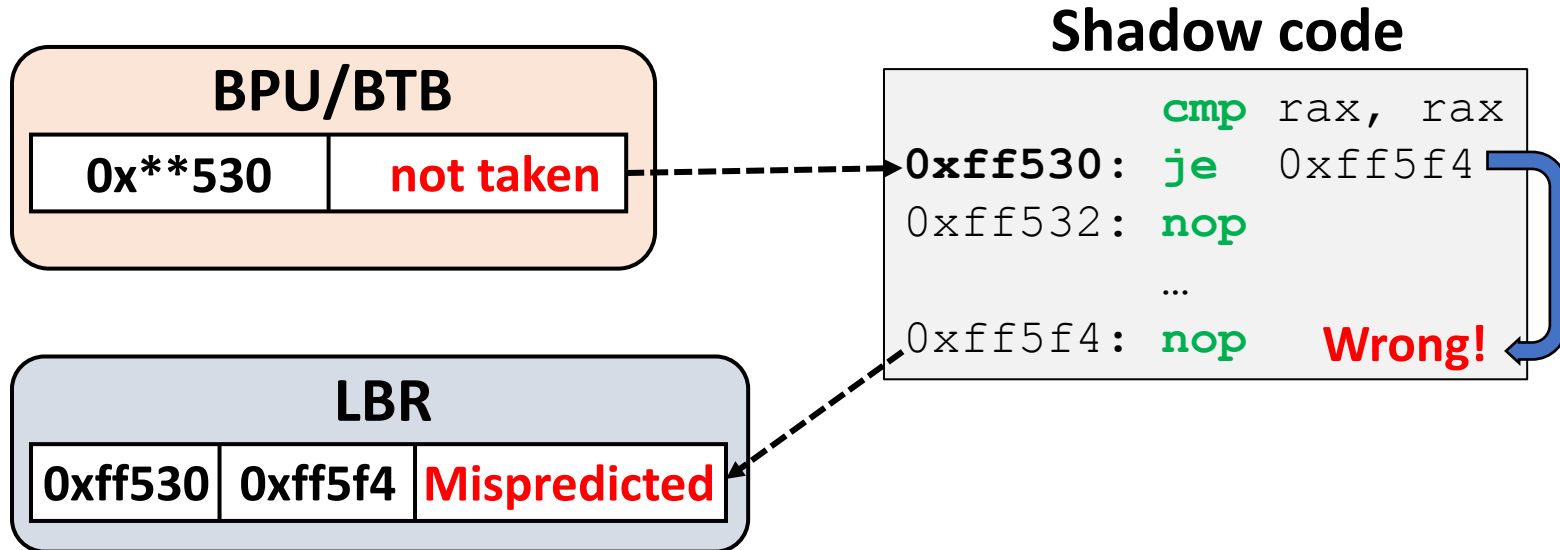
- BPU/BTB *correctly predicts* the execution of the shadow branch using the history

Example: Inferring Taken Branch



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

Example: Inferring **Not-taken** Branch



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

Enabling Single Stepping!

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

Example: Attacking RSA Exp.

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

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

Sliding-window
exponentiation of mbedtls

Example: Attacking RSA Exp.

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

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

Sliding-window
exponentiation of mbedtls

Taken only when ei is zero

Example: Attacking RSA Exp.

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

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

Sliding-window
exponentiation of mbedtls

Taken only when ei is zero

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

DEMO: Branch Shadowing Attack

**Branch Shadowing Attack
- RSA exponentiation -**

What Else?

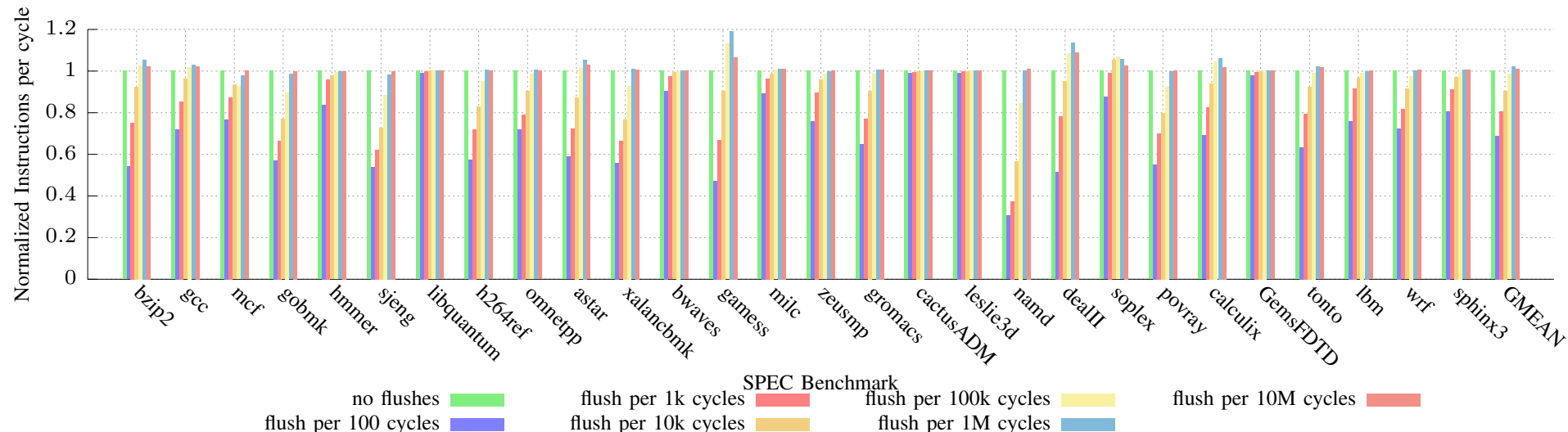
Program/Function	Description	Leakages
libc/strtol	Convert a string into an integer	The sign and length of an input Hexadecimal digits
libc/vfprintf	Print a formatted string	The input format string
LIBSVM/k_function	Evaluate a kernel function	The type of a kernel (e.g., linear, RBF) The number of features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

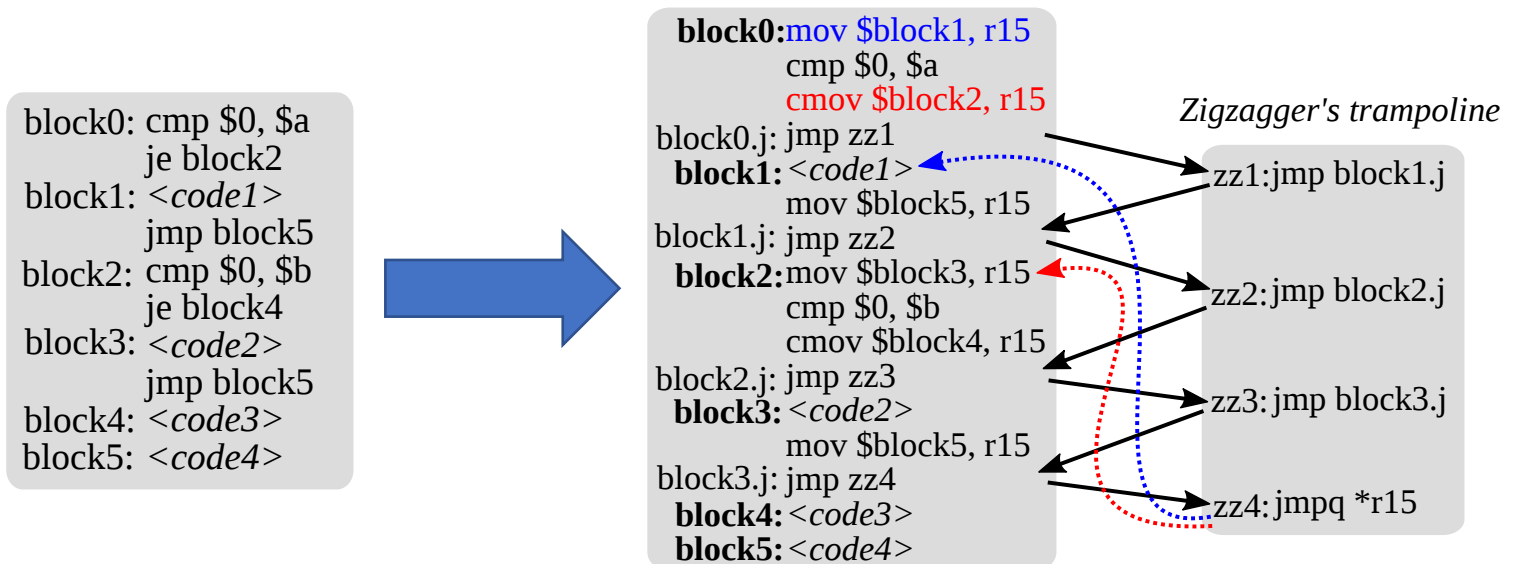
Defense: Flushing Branch States (Hardware)

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



Defense: 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.



Defense: Obfuscating Branch (Software/Compiler)

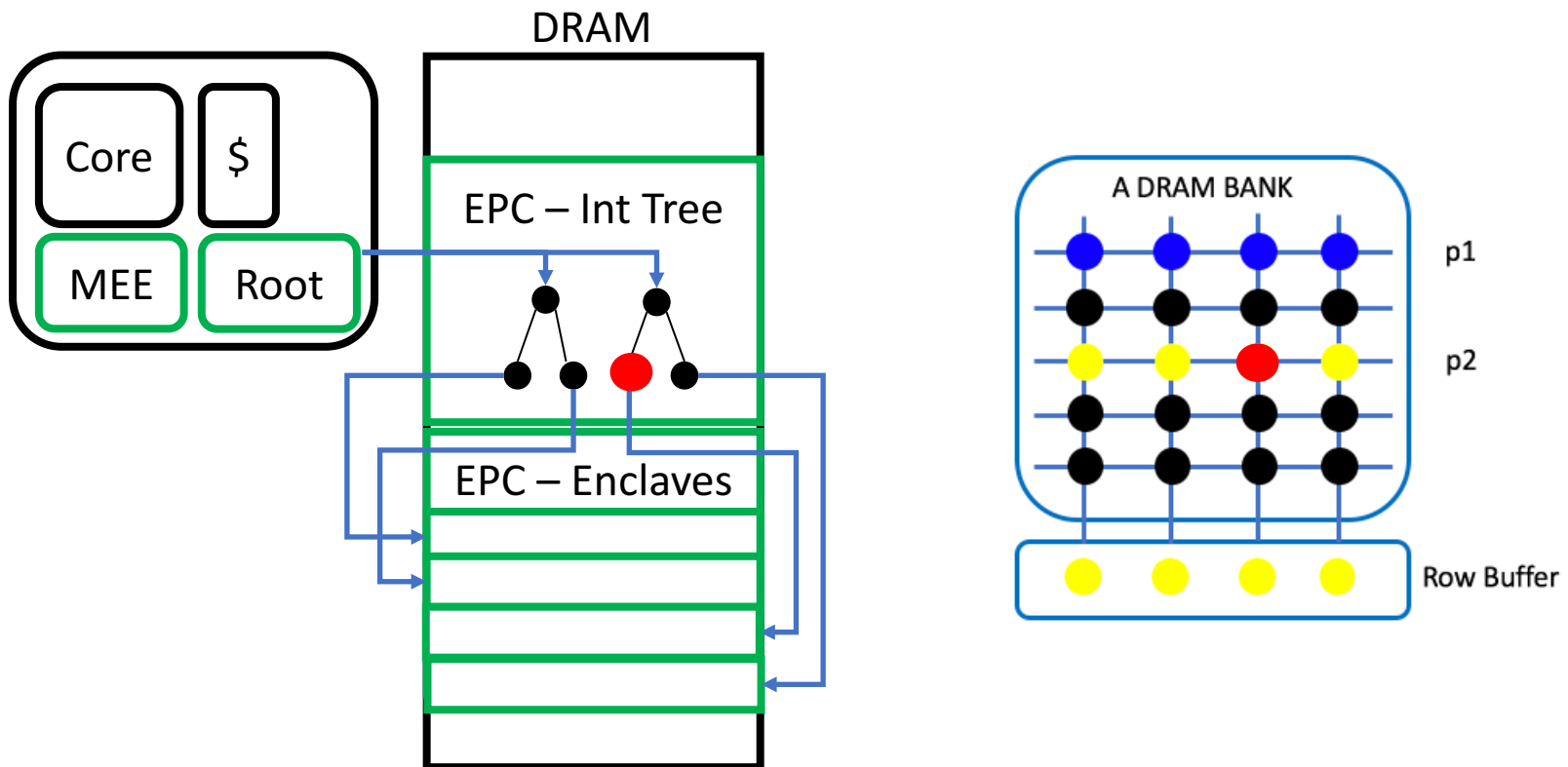
- LLVM-based implementation
- Overhead (nbench): $\leq 1.5\times$
- Just mitigate the attack, don't solve it completely

New Attack Vectors

- Page table attack
- Branch shadowing attack
- ★ • Rowhammer against SGX

SGX-Bomb: Rowhammer Attack

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



SGX-Bomb: Rowhammer Attack

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

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

SGX-BOMB: Locking Down the Processor via Rowhammer Attack

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

Sangho Lee
Georgia Institute of Technology
sangho@gatech.edu

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

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 Environment (TEE) enable secure computation in a user program without relying on the operating system. Intel Software Guard Extensions (SGX) [18] is a commodity hardware-based TEE imple-

System Software for Trusted Execution '17

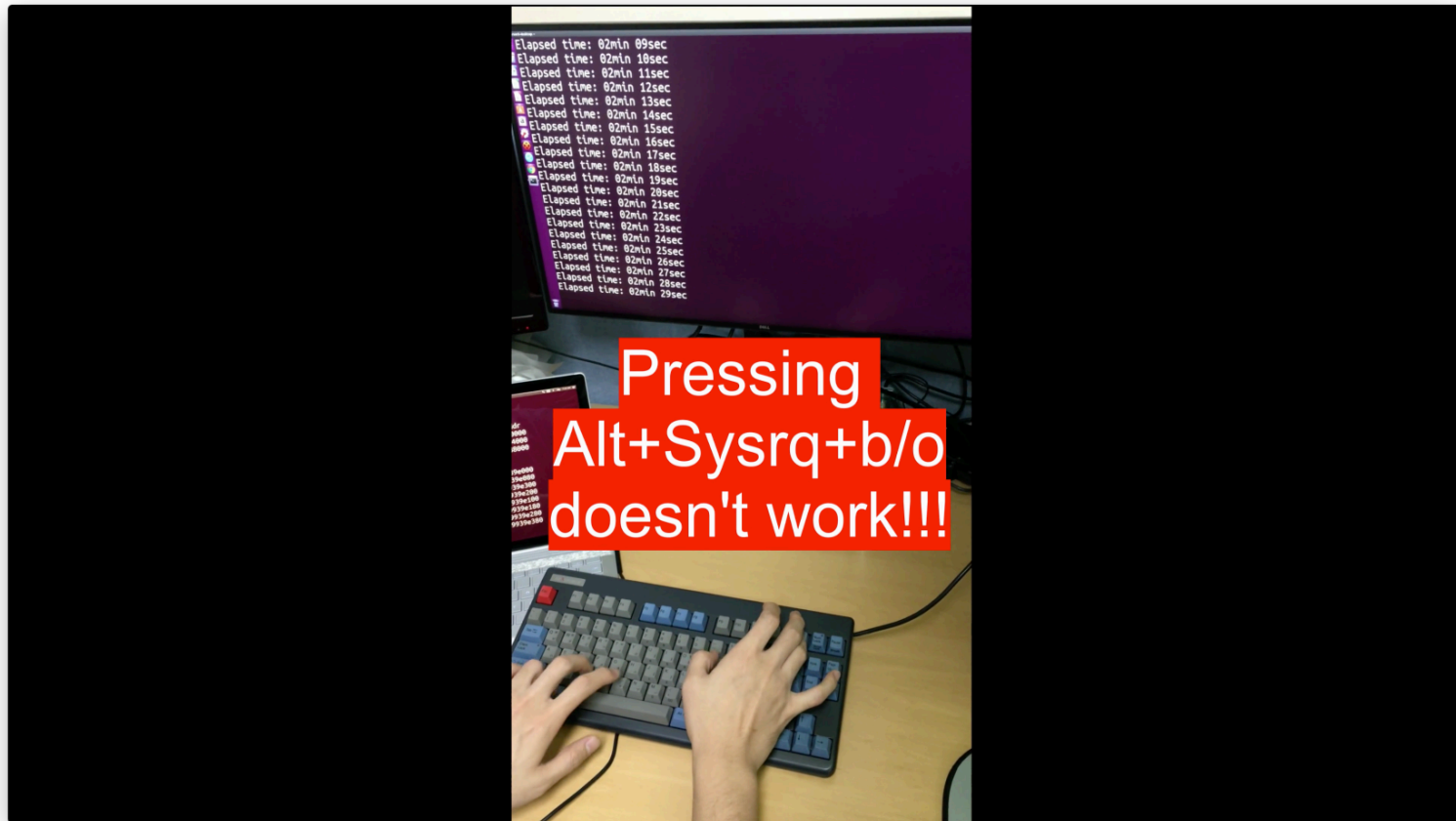
About Integrity Violation

- 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

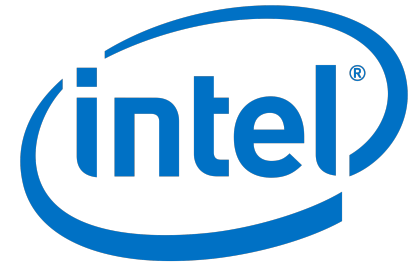
- Easier to trigger than normal rowhammer
i.e., a single, arbitrary bit in EPC region (128MB)
- Harder to detect
 - Not notifiable in terms of resource usages
 - Popular defenses (e.g., in Linux) rely on PMU (e.g., cache misses) that is not possible for enclaves

DEMO: SGX-Bomb



Defenses against SGX-Bomb

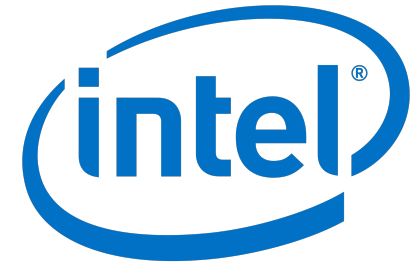
- Use LPDDR3 compliant to Intel's Pseudo-TRR (Target Row Refresh)
 - ECC can't completely block (easy to trigger multiple bits)
- Potential defenses:
 - Using Uncore PMU
 - Row-aware memory allocation for EPC regions



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



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

Backup

Local APIC Timer

- OS can program the timer interrupt (belonging to the local APIC).
- Recent Linux kernel uses the TSC-deadline mode.
 - Schedule the next timer interrupt with the time stamp counter (TSC) value

```
tsc = rdtsc();  
wrmsrl(MSR_IA32_TSC_DEADLINE,  
       tsc + (((u64) delta) * TSC_DIVISOR));
```


Modified Local APIC Timer

```
1 /* linux-4.4.23/arch/x86/kernel/apic/apic.c */
2 ...
3 // manipualte the delta of TSC-deadline mode
4 unsigned int lapic_next_deadline_delta = 0U;
5 EXPORT_SYMBOL_GPL(lapic_next_deadline_delta);
6
7 // specify the virtual core under attack
8 int lapic_target_cpu = -1;
9 EXPORT_SYMBOL_GPL(lapic_target_cpu);
10
11 // a hook to launch branch shadowing attack
12 void (*timer_interrupt_hook)(void*) = NULL;
13 EXPORT_SYMBOL_GPL(timer_interrupt_hook);
14 ...
15 // update the next TSC deadline
16 static int lapic_next_deadline(unsigned long delta,
17                               struct clock_event_device *evt) {
18     u64 tsc;
19
20     tsc = rdtsc();
21     if (smp_processor_id() != lapic_target_cpu) {
22         wrmsrl(MSR_IA32_TSC_DEADLINE,
23              tsc + (((u64) delta) * TSC_DIVISOR)); // original
24     }
25     else {
26         wrmsrl(MSR_IA32_TSC_DEADLINE,
27              tsc + lapic_next_deadline_delta); // custom deadline
28     }
29     return 0;
30 }
31 ...
```

- Exported hooks to perform attacks
- delta=1000 was the minimum value we could use (i7-6700K).
- About **50 ADD instructions** were executed between two timer interrupts.

```
32 // handle a timer interrupt
33 static void local_apic_timer_interrupt(void) {
34     int cpu = smp_processor_id();
35     struct clock_event_device *evt = &per_cpu(lapic_events, cpu);
36
37     if (cpu == lapic_target_cpu && timer_interrupt_hook) {
38         timer_interrupt_hook((void*)&cpu); // call attack code
39     }
40     ...
41 }
```

Last Branch Record

- Record the information of recently **taken** branch instructions (Skylake: up to 32)
 - Branch instruction address (from)
 - Target address (to)
 - Prediction result (success/failure)
 - Elapsed core cycles between LBR entry updates
- Selectively record branch information
 - Branch type: conditional/indirect, function call/return
 - Space: User and/or kernel

```

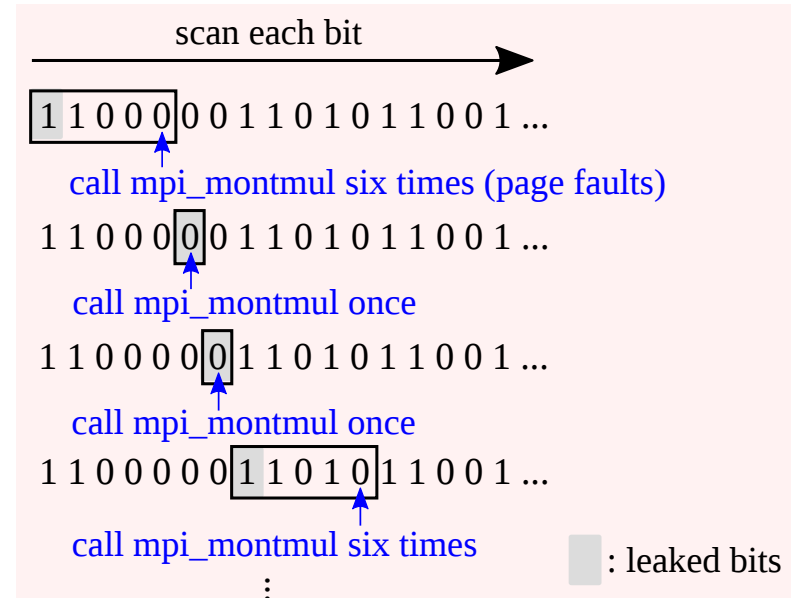
1  /* Sliding-window exponentiation: X = A^E mod N */
2  int mbedtls_mpi_exp_mod(mbedtls_mpi *X, const mbedtls_mpi *A,
3                          const mbedtls_mpi *E, const mbedtls_mpi *N,
4                          mbedtls_mpi *_RR) {
5      ...
6      state = 0;
7      while (1) {
8          ...
9          // i-th bit of exponent
10         ei = (E->p[nblimbs] >> bufsize) & 1;
11
12         // cmpq 0x0,-0xc68(%rbp); jne 3f317; ...
13         * if (ei == 0 && state == 0)
14             continue;
15
16         // cmpq 0x0,-0xc68(%rbp); jne 3f371; ...
17         * if (ei == 0 && state == 1)
18             + mpi_montmul(X, X, N, mm, &T);
19
20         state = 2; nbits++;
21         wbits |= (ei << (wsize-nbits));
22
23         if (nbits == wsize) {
24             for (i = 0; i < wsize; i++)
25                 + mpi_montmul(X, X, N, mm, &T);
26
27                 + mpi_montmul(X, &W[wbits], N, mm, &T);
28                 state--; nbits = wbits = 0;
29         }
30     }
31     ...
32 }

```

Page-fault Attack?

```
/* 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);
        ...
        if (nbits == wsize) {
            for (i = 0; i < wsize; ++i)
                mpi_montmul(X, X, N, mm, &T);
            mpi_montmul(X, &W[wbits], N, mm, &T);
            ...
        }
    }
    ...
}
```



Differentiate these two function calls

**Recognizable bit fraction: $(1+\text{window size})/2$
(~30% if window size is five)**

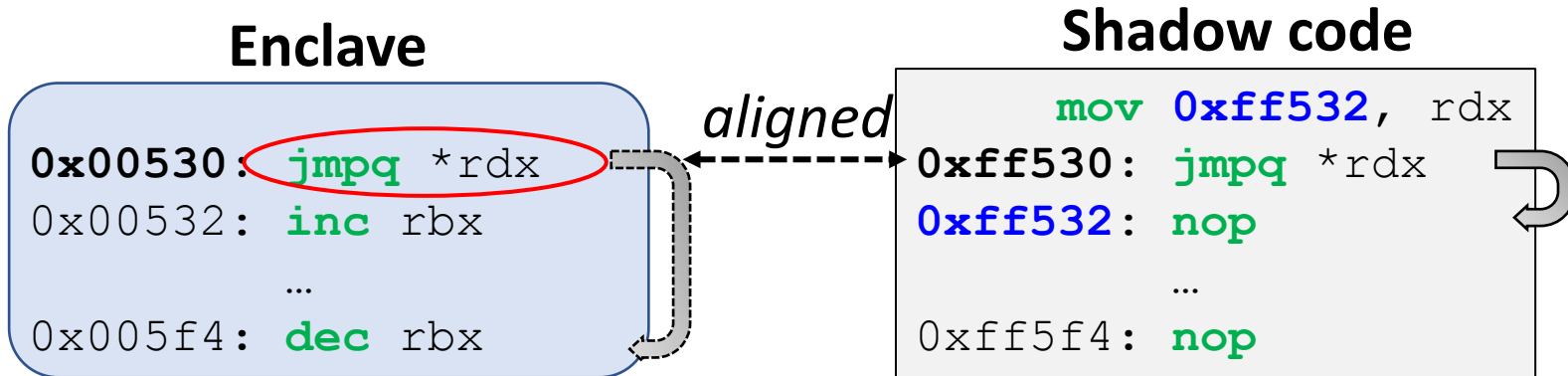
Inferring Indirect Branch

Enclave

```
0x00530: jmpq *rdx  
0x00532: inc rbx  
...  
0x005f4: dec rbx
```

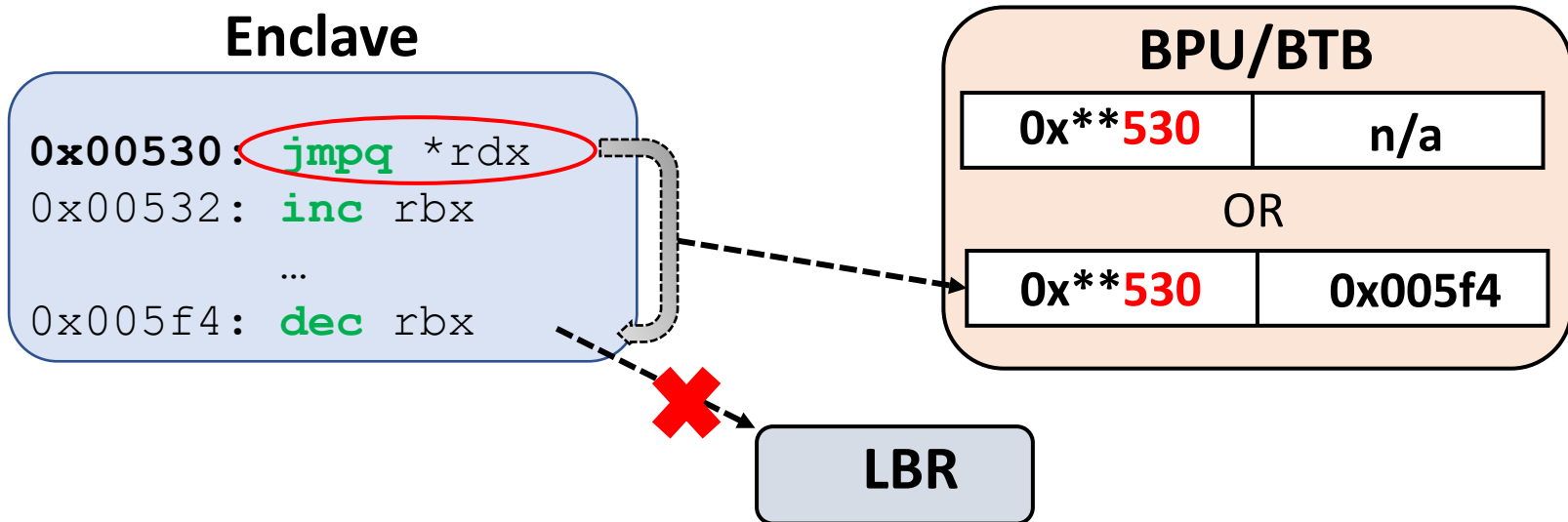
- Infer whether a target indirect branch in an enclave has been executed

Inferring Indirect Branch



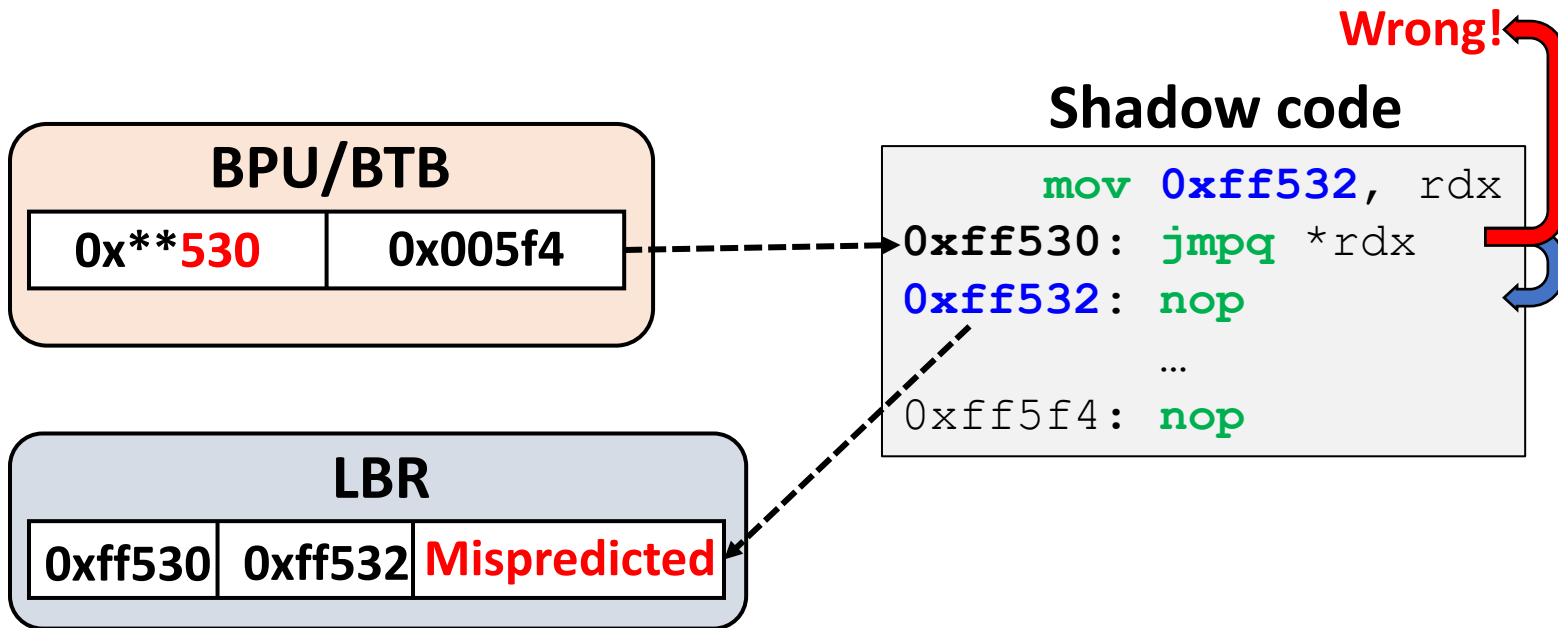
- Infer whether a target indirect branch in an enclave has been executed
- Prepare shadow code for a target branch
 - Colliding indirect branch
 - Jump to the next instruction
- The execution of the shadow branch is affected by the target branch.

Inferring Indirect Branch



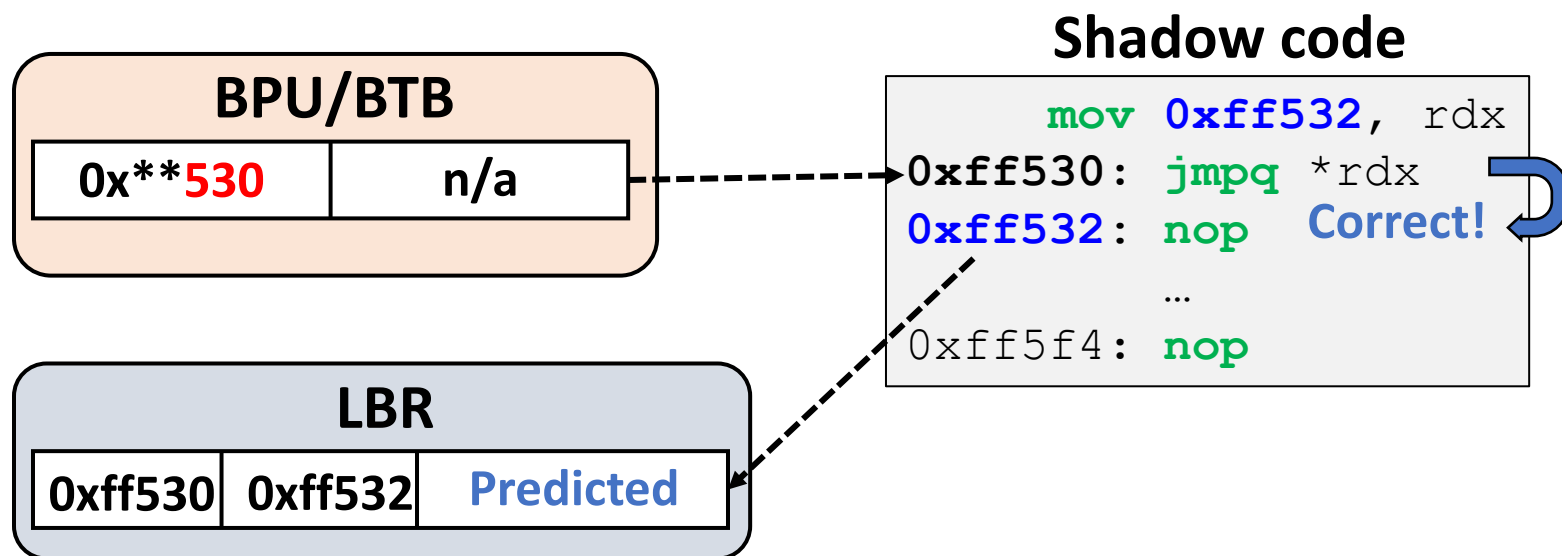
- BPU/BTB is updated according to the execution of the target branch.
- LBR ignores branch execution inside an enclave.

Inferring Indirect Branch (Executed)



- BPU/BTB *mispredicts* the execution of the shadow branch.
- LBR reports the corresponding branch information.
 - **Mispredicted** → The target branch **has been executed**.

Inferring Indirect Branch (Not Executed)



- BPU/BTB *correctly predicts* the execution of the shadow branch.
- LBR reports the corresponding branch information.
 - **Predicted** → The target branch **has not been executed**.