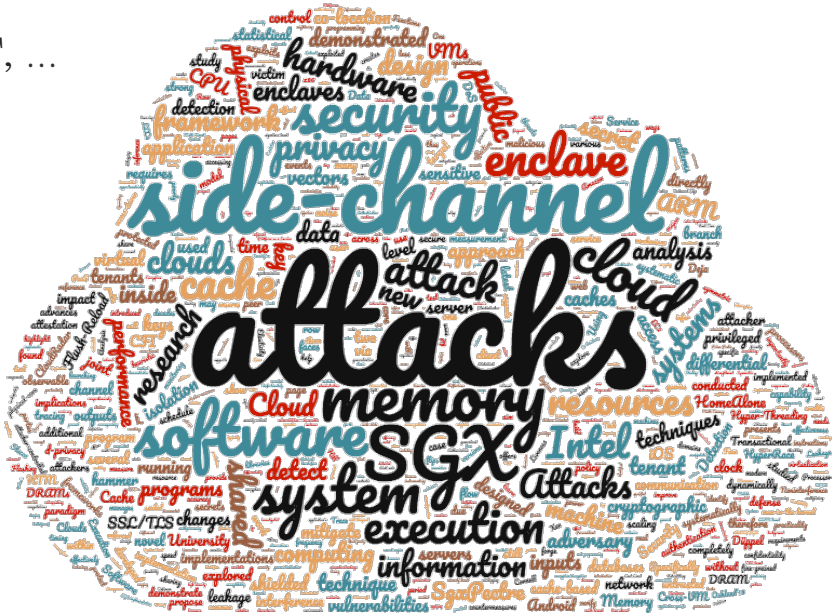
How to Protect Data Even After Operating System Compromise

Yinqian Zhang, Ph.D.
Assistant Professor
Computer Science & Engineering
The Ohio State University

A Note of My Research

- Computer system security in general
  - Side-channel security of computer systems
  - Cloud computing and its security
  - Authentication and authorization
  - Security of blockchain, mobile phones, IoT, …



Topics of This Talk

How to Protect Data Even After Operating System Compromise

Not Really …

Nice! This must be a talk about data encryption !!!

Topics of This Talk

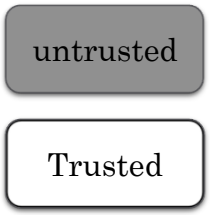
How to Protect Data Even After Operating System Compromise

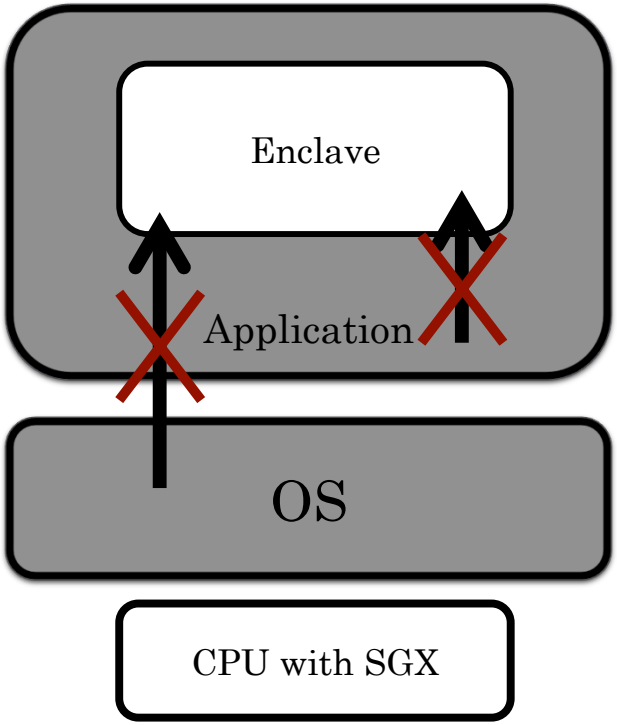
- A technology called Intel SGX
- Research in my lab to improve SGX security

Why Data Encryption is NOT The Solution

- **Data-in-use must be unencrypted**
- **Operating system compromise => data breach**

Intel Software Guard Extension (SGX)





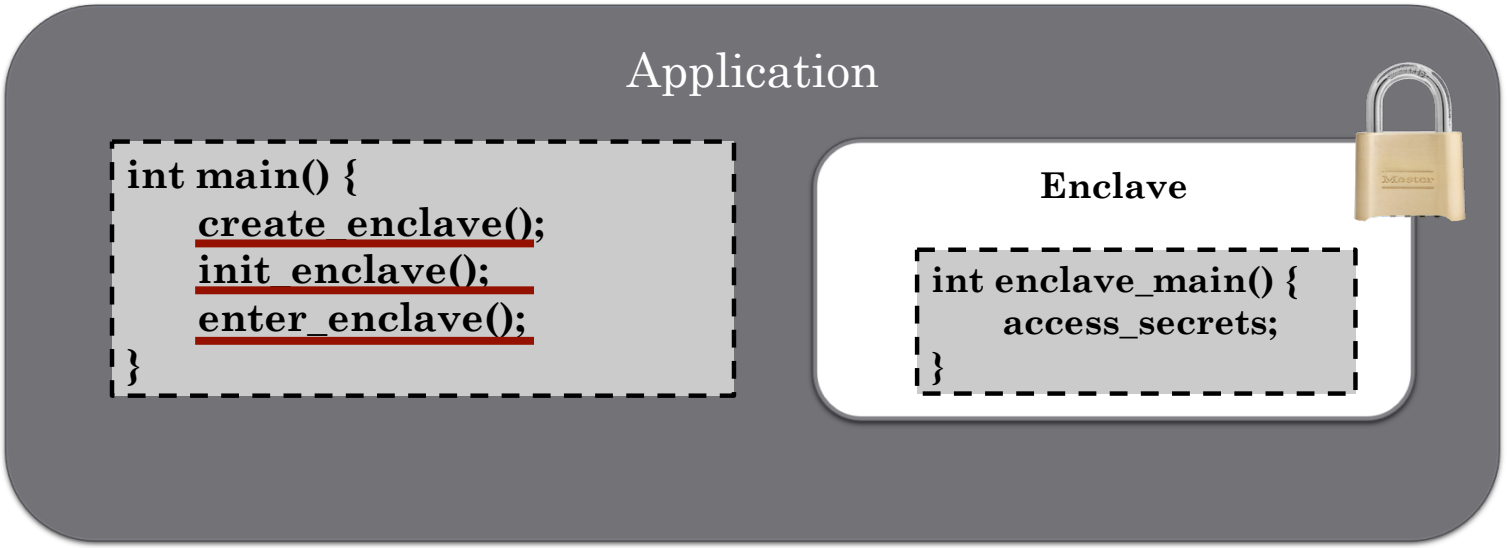
How Does SGX Work?

- Memory isolation
  - Confidentiality and integrity: Enforced at the operating system, BIOS, VMM, SMM, or TEE layers even in the presence of privileged malware.
- Memory encryption
  - The processor boundary becomes the attack surface perimeter—all data, memory, and I/O outside this perimeter is encrypted.
- Remote attestation
  - A remote party can verify an application enclave identity and securely provision keys, credentials, and other sensitive data to the enclave.
- Sealed storage
  - An enclave application may encrypt data to be stored in persistent storage for later use.

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

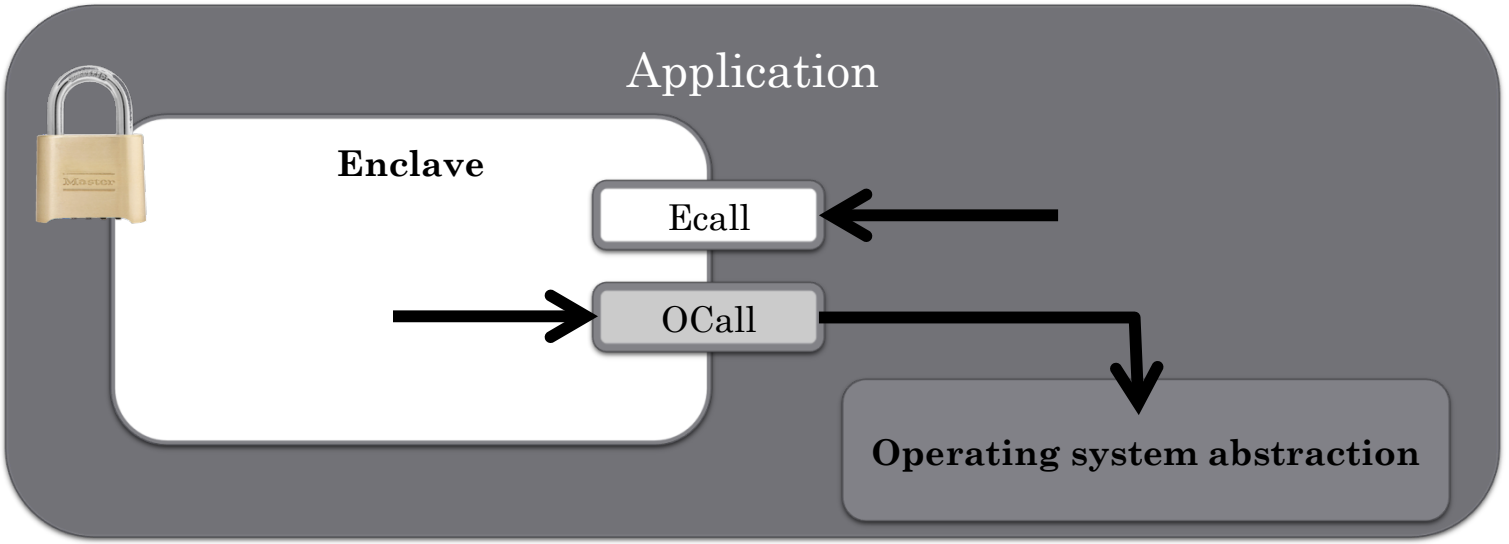
Properties of SGX Enclaves

- A software application must be re-written into two components
  - A trusted component runs inside the enclave
  - An untrusted component runs outside
  - The trusted component is loaded as a dynamic library



Properties of SGX Enclaves (Con't)

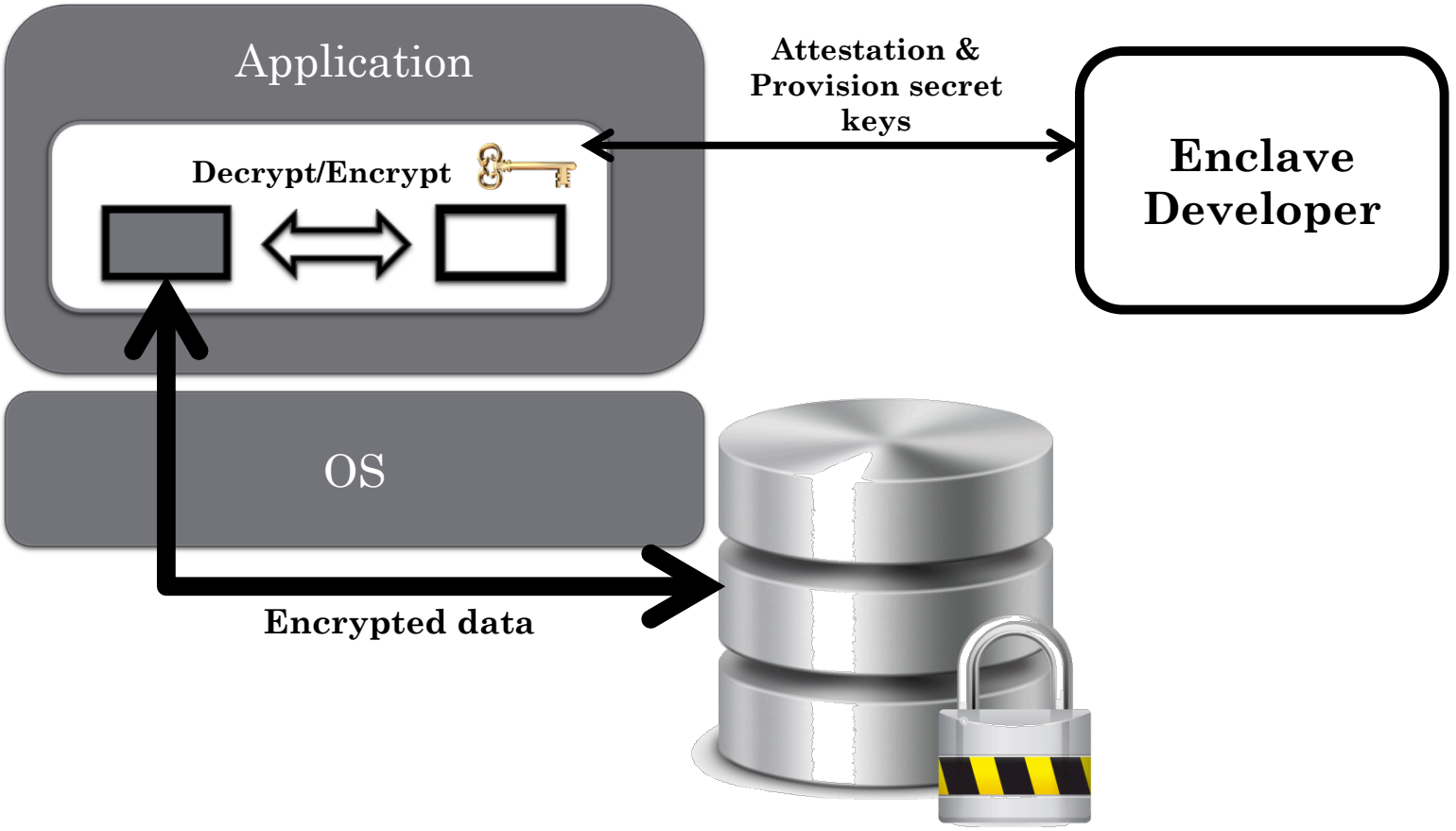
- After enclave init, OS no longer has access to the internal memory of an enclave.
- Application uses enclave functions through ECall
- Enclave performs system calls through OCalls



Properties of SGX Enclaves (Con't)

- After initiation, the enclave may initiate a remote attestation to verify to the enclave owner:
  - The launched enclave has the same measurement (cryptographic hash of all enclave memory pages) as expected by the enclave owner
  - The enclave runs on a legitimate Intel CPU with SGX supports
  - A 64-byte data buffer can be bound to the attestation result
- A secure communication channel can be established between the enclave and its owner
  - Diffie-Hellman key exchange
  - Public key of the enclave can be sent as part of the 64-byte attested data

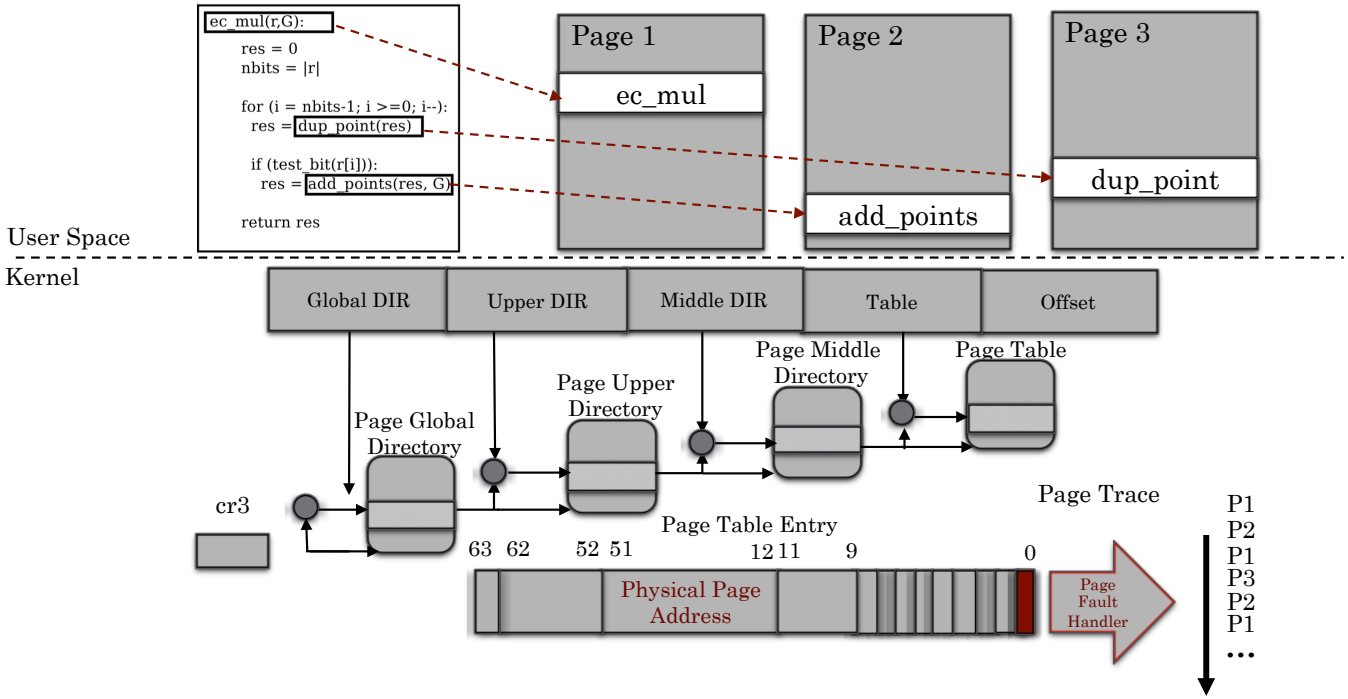
Using SGX to Protect Data Security



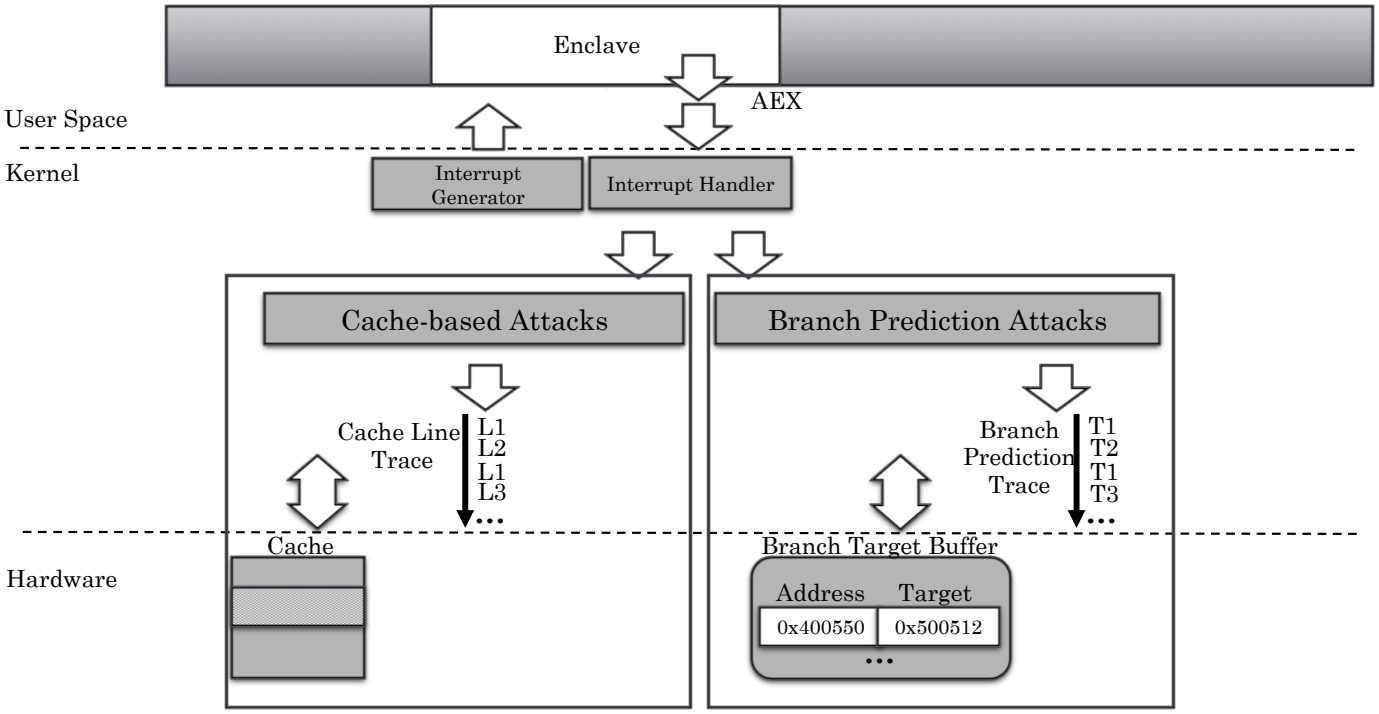
Side-Channel Attacks against Intel SGX

- What are side-channel attacks
  - Indirect confidentiality breaches
  - Inference attacks on sensitive data
- History of side-channel attacks
  - 1990s: power analysis or timing analysis of smart cards
  - 2000s: Attacks from Hyperthreads on Intel CPU using caches, BTB, FU
  - 2010s: Multicore systems, virtualization, cloud, mobile, etc.
- Side-channel attacks against SGX enclaves
  - Exception-based side-channel attacks
  - Interrupt-based side-channel attacks
  - Hyperthreading side-channel attacks
  - Cross-core side-channel attacks

Exception-based Side-Channel Attacks



Interrupt-based Side-Channel Attacks



Roadmap of My Research

- Understanding side-channel hazards of Intel SGX
  - Memory side-channel attack surfaces (CCS'17a)
- Detecting side-channel vulnerabilities in enclave programs
  - Dynamic analysis tools for detecting sensitive control-flow vulnerabilities in SSL/TLS (CCS'17b)
- Compiler-assisted runtime defenses
  - Timed execution for detecting side-channel attacks at runtime (AsiaCCS'17)
  - Contrived data race for detecting Hyper-Thread co-location (S&P'18)
- Branch target injection & SGX side channels
  - SgxPectre attacks: Exploiting speculative execution in SGX enclaves

Roadmap of My Research

- Understanding side-channel hazards of Intel SGX
  - Memory side-channel attack surfaces (CCS'17a)
- Detecting side-channel vulnerabilities in enclave programs
  - Dynamic analysis tools for detecting sensitive control-flow vulnerabilities in SSL/TLS (CCS'17b)
- Compiler-assisted runtime defenses
  - Timed execution for detecting side-channel attacks at runtime (AsiaCCS'17)
  - Contrived data race for detecting Hyper-Thread co-location (S&P'18)
- Branch target injection & SGX side channels
  - SgxPectre attacks: Exploiting speculative execution in SGX enclaves

Roadmap of My Research

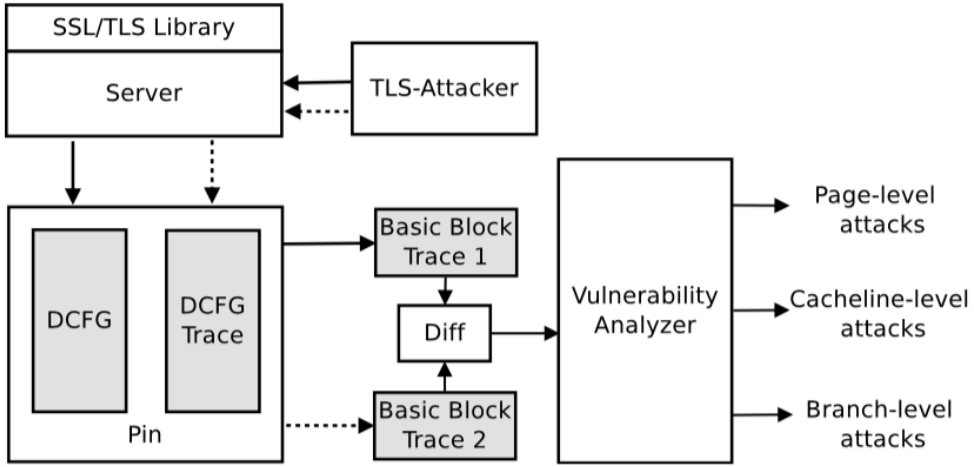
- Understanding side-channel hazards of Intel SGX
  - Memory side-channel attack surfaces (CCS'17a)
- Detecting side-channel vulnerabilities in enclave programs
  - Dynamic analysis tools for detecting sensitive control-flow vulnerabilities in SSL/TLS (CCS'17b)
- Compiler-assisted runtime defenses
  - Timed execution for detecting side-channel attacks at runtime (AsiaCCS'17)
  - Contrived data race for detecting Hyper-Thread co-location (S&P'18)
- Branch target injection & SGX side channels
  - SgxPectre attacks: Exploiting speculative execution in SGX enclaves

The Problem: Oracle Attacks against SSL/TLS

- Ciphersuites considered secure in non-SGX settings are vulnerable in enclaves
- Bleichenbacher attacks: decrypting PMS and derive the session key
  - OpenSSL 1.0.2j (1024-bit RSA key): 19,346 queries in 28 minutes 20 seconds
  - OpenSSL 1.0.2j (4096-bit RSA key): 57,286 queries in 91 minutes 39 seconds
- CBC padding oracle attacks: decryption SSL/TLS encrypted message
  - TLS v1.2 in GnuTLS 3.4.17: 48,388 queries in 51 minutes to decrypt one block (16 bytes)
  - mbedTLS-SGX: 25,717 queries in 29.5 minutes to decrypt one block
- SSL/TLS implementation in SGX requires additional consideration of man-in-the-kernel attacks

Stacco: A Differential Analysis Tool

- Packet generator
  - TLS-Attacker
- Trace recorder
  - DCFG tools by PinPlay
- Diff tools
  - Comparing basic-block traces
- Vulnerability analyzer
  - Page-level analysis
  - Cacheline-level analysis
  - Branch-level analysis



Main Results

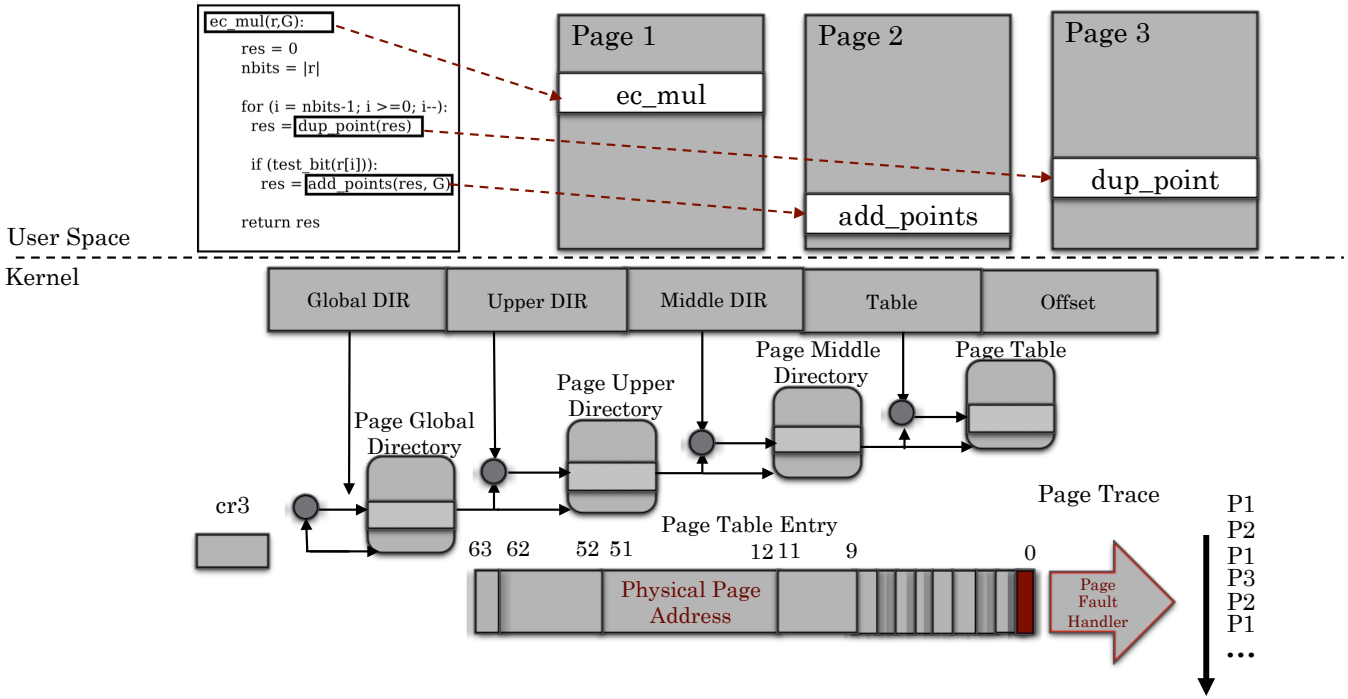
Table 1: Experiment results of the differential analyses. B: vulnerable to branch-level attacks? C: vulnerable to cacheline-level attacks? P: vulnerable to page-level attacks? D: Differentiable; N: Not differentiable; N/A: unable to test.

	Test Name	OpenSSL 1.0.2j			GnuTLS 3.4.17			mbedTLS 2.4.1			WolfSSL 3.10.0			LibreSSL 2.5.0		
		B	C	P	B	C	P	B	C	P	B	C	P	B	C	P
Bleichenbacher attacks	PKCS#1 Conformant	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
	Wrong Version	D	D	D	D	D	D	D	D	D	D	D	N	D	D	D
	No 0x00 Byte	D	D	N	D	D	D	D	D	D	D	D	N	D	D	N
	0x00 in Padding	D	D	D	D	D	D	D	D	D	D	D	N	D	D	D
	0x00 in PKCS Padding	D	D	N	D	D	D	D	D	D	D	D	D	D	D	N
	PMS Size=0	D	D	D	D	D	D	D	D	D	D	D	N	D	D	D
	PMS Size=2	D	D	D	D	D	D	D	D	D	D	D	N	D	D	D
	PMS Size=8	D	D	D	D	D	D	D	D	D	D	D	N	D	D	D
	PMS Size=16	D	D	D	D	D	D	D	D	D	D	D	N	D	D	D
	PMS Size=32	D	D	D	D	D	D	D	D	D	D	D	N	D	D	D
	Exploitable	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Padding Oracle attacks	Padding Length Byte XOR 1	D	D	N	N/A	N/A	D	D	D	D	D	D	D	D	D	D
	Padding Length Byte = 0x00	D	D	N	N/A	N/A	D	D	D	D	D	D	D	D	D	D
	Padding Length Byte = 0xFF	D	D	N	N/A	N/A	D	D	D	D	D	D	D	D	D	D
	Last Padding Byte XOR 1	D	D	N	N/A	N/A	D	D	D	D	D	D	D	D	D	D
	Last Padding Byte = 0x00	D	D	N	N/A	N/A	D	D	D	D	D	D	D	D	D	D
	Last Padding Byte = 0xFF	D	D	N	N/A	N/A	D	D	D	D	D	D	D	D	D	D
	Exploitable	✓	✓	✗	N/A	N/A	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

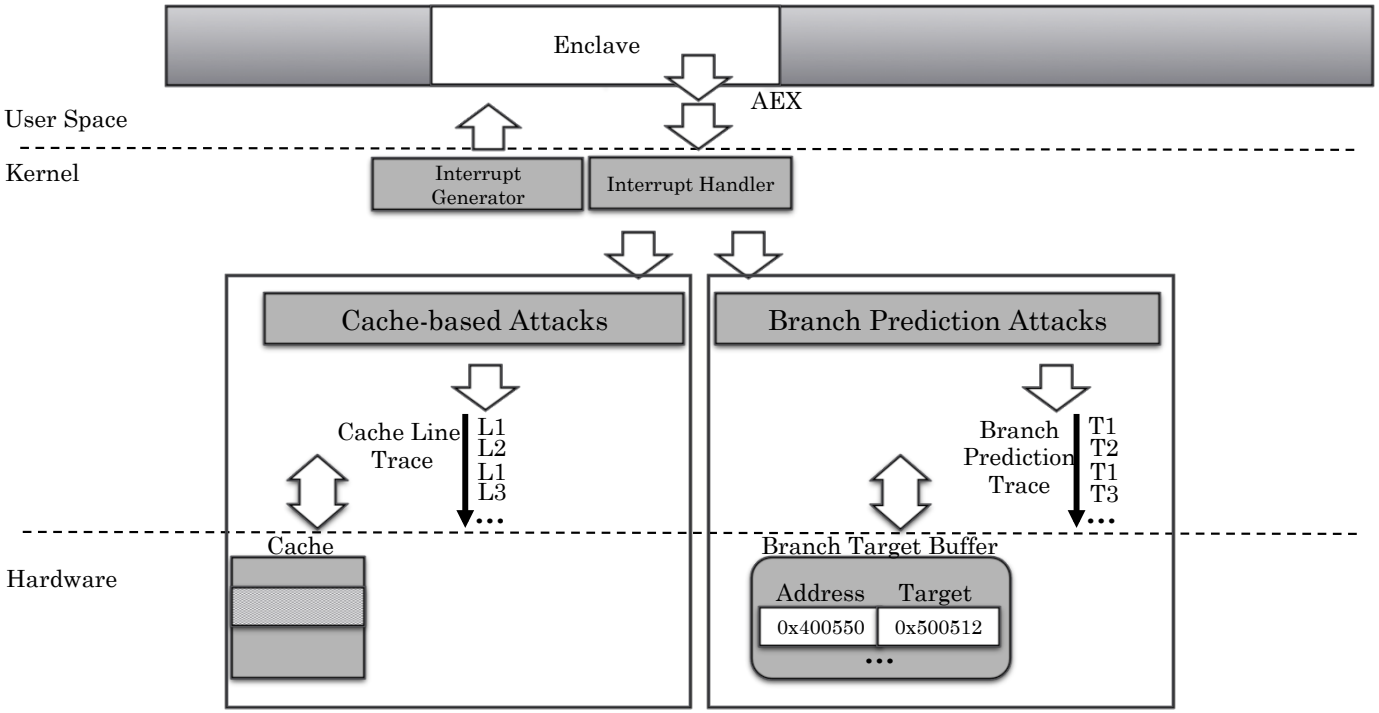
Roadmap of My Research

- Understanding side-channel hazards of Intel SGX
  - Memory side-channel attack surfaces (CCS'17a)
- Detecting side-channel vulnerabilities in enclave programs
  - Dynamic analysis tools for detecting sensitive control-flow vulnerabilities in SSL/TLS (CCS'17b)
- Compiler-assisted runtime defenses
  - Timed execution for detecting side-channel attacks at runtime (AsiaCCS'17)
  - Contrived data race for detecting Hyper-Thread co-location (S&P'18)
- Branch target injection & SGX side channels
  - SgxPectre attacks: Exploiting speculative execution in SGX enclaves

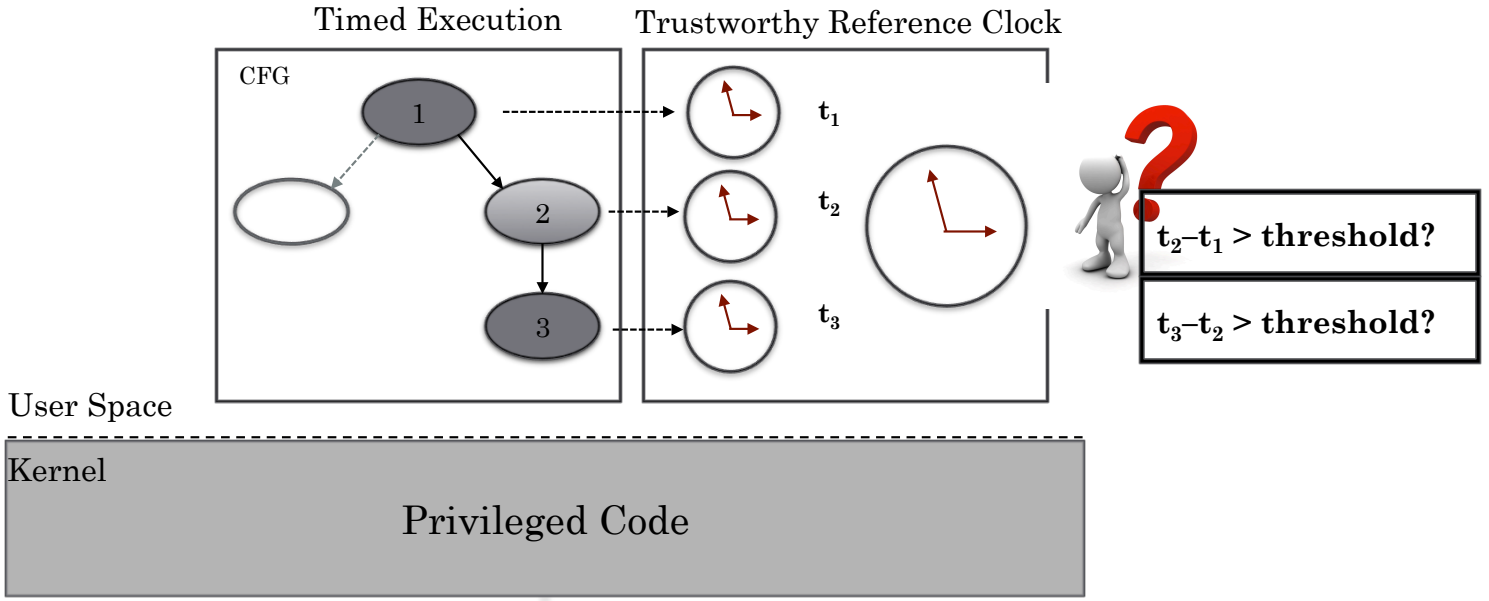
Exception-based Side-Channel Attacks



Interrupt-based Side-Channel Attacks



Déjà Vu: Timed Execution (AsiaCCS'17)



Déjà Vu: Code Snippet for The Reference Clock

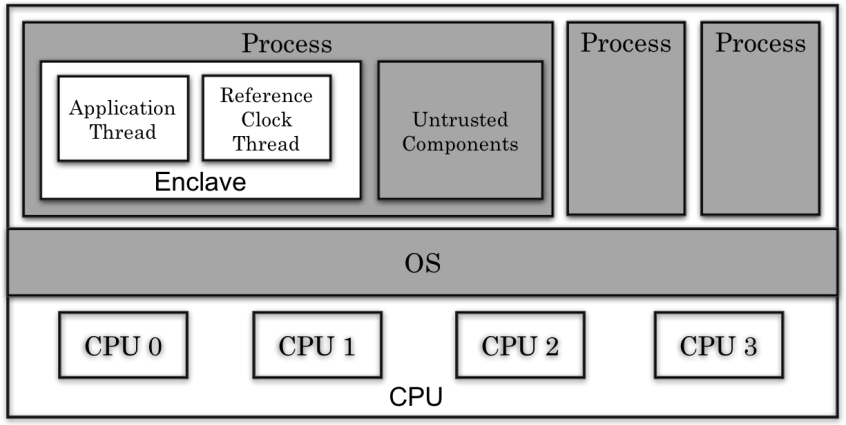
```
1  unsigned int timer; // global variable
2  unsigned int interrupted; // global variable
3  void timer_thread()
4  {
5    unsigned int rand;
6    while (1) {
7      if (_xbegin() == _XBEGIN_STARTED) {
8        __asm volatile ("rdrand %0\n\t"
9          :"=r"(rand));
10       rand = (rand & 0x7) + 1;
11       for(int i = 0; i < rand; i++) {
12         // tasks comprising v cycles
13       }
14       _xend();
15     } else {
16       interrupted += 1;
17       continue;
18     }
19     timer += rand
20   }
21 }
```

TSX transaction

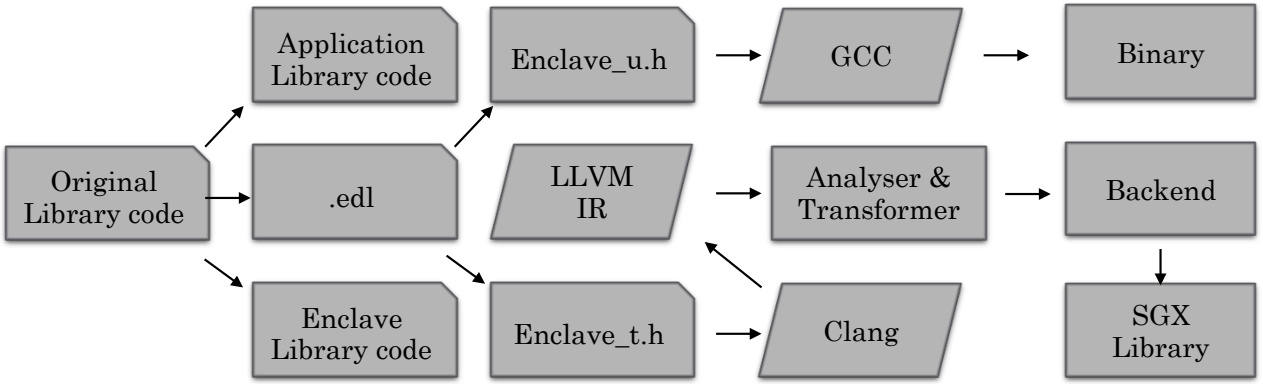
randomness

increase timer outside TSX transaction

Déjà Vu: System Architecture & Programming Model



- Two threads sharing the same enclave
- Compiler–assisted instrumentation



Hyper-Threading Side Channels

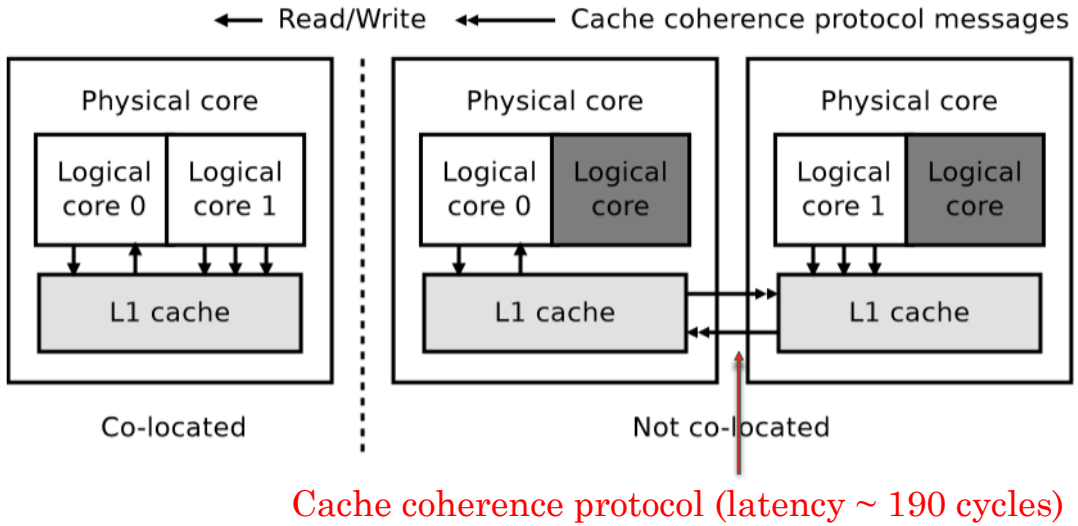
- Hyper-Threading enabled side channels

Side Channels	Shared	Cleansed at AEX	Hyper-Threading only
Caches	Yes	Not flushed	No
BPUs	Yes	Not flushed	No
Store Buffers	No	N/A	Yes
FPUs	Yes	N/A	Yes
TLBs	Yes	Flushed	Yes

- Hyper-Threading facilitated side channels
  - HT-SPM: Hyper-Threading facilitated sneaky page monitoring
- Hyper-Threading thwarted defenses
  - Hyper-Threading invalidates some defense techniques that leverage Intel's Transactional Synchronization Extensions (TSX)

Hyper-Thread Co-Location Tests (S&P'18)

- Two threads operate on the same shared variable ν inside the enclave:
  - Thread T_0
    - **While (1)**
      Writes 0 to ν
      Waits for 10 cycles
      Reads ν
  - Thread T_1
    - **While (1)**
      Writes 1 to ν



- Thread T_0 reads ν=1 with very high probability when co-located
- Thread T_0 reads ν=0 with very high probability when not co-located

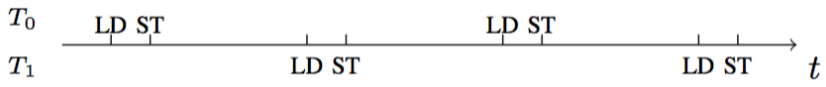
A Refined Design

Thread T_0

```
1  <initialization>:
2     mov $colocation_count, %rdx
3     xor %rcx, %rcx
4     ; co-location test counter
5  <synchronization>:
6     ··· ; acquire lock 0
7   .sync0:
8     mov %rdx, (sync_addr1)
9     cmp %rdx, (sync_addr0)
10    je .sync1
11    jmp .sync0
12  .sync1:
13    mfence
14    mov $0, (sync_addr0)
15 <initialize a round>:
16    mov $begin0, %rsi
17    mov $1, %rbx
18    mfence
19    mov $addr_v, %r8
20 <co-location test>:
21  .L0:
22 <load>:
23    mov (%r8), %rax
24 <store>:
25    mov %rsi, (%r8)
26 <update counter>:
27    mov $0, %r10
28    mov $0, %r11
29    cmp $end0, %rax
30    ; a data race happens?
31    cmovl %rbx, %r10
32    sub %rax, %r9
33    cmp $1, %r9
34    ; continuous number?
35    cmova %r11, %r10
36    add %r10, %rcx
37    shl $b_count, %rbx
38    ; bit length of $count
39    mov %rax, %r9
40    ; record the last number
41 <padding instructions 0>:
42    nop
43    nop
44     ⋮
45    nop
46    mov (%r8), %rax
47    mov (%r8), %rax
48     ⋮
49    mov (%r8), %rax
50    dec %rsi
51    cmp $end0, %rsi
52    jne .L0
53    ; finish 1 co-location test
54 <all rounds finished?>:
55    ··· ; release lock 1
56    dec %rdx
57    cmp $0, %rdx
58    jne .sync0
```

Thread T_1

```
1  <initialization>:
2     mov $colocation_count, %rdx
3     xor %rcx, %rcx
4     ; co-location test counter
5  <synchronization>:
6     ··· ; release lock 0
7   .sync2:
8     mov %rdx, (sync_addr0)
9     cmp %rdx, (sync_addr1)
10    je .sync3
11    jmp .sync2
12  .sync3:
13    mfence
14    mov $0, (sync_addr1)
15 <initialize a round>:
16    mov $begin1, %rsi
17    mov $1, %rbx
18    mfence
19    mov $addr_v, %r8
20 <co-location test>:
21  .L2:
22 <load>:
23    mov (%r8), %rax
24 <update counter>:
25    mov $0, %r10
26    mov $0, %r11
27    cmp $end0, %rax
28    ; a data race happens?
29    cmovg %rbx, %r10
30    sub %rax, %r9
31    cmp $1, %r9
32    ; continuous number?
33    cmova %r11, %r10
34    add %r10, %rcx
35    shl $b_count, %rbx
36    ; bit length of $count
37     mov %rax, %r9
38    ; record the last number
39 <store>:
40    mov %rsi, (%r8)
41 <padding instructions 1>:
42    mov (%r8), %rax
43    lfence
44    mov (%r8), %rax
45    lfence
46    mov (%r8), %rax
47    lfence
48    mov (%r8), %rax
49    lfence
50    mov (%r8), %rax
51    lfence
52    dec %rsi
53    cmp $end1, %rsi
54    jne .L2
55    ; finish 1 co-location test
56 <all rounds finished?>:
57    ··· ; acquire lock 1
58    dec %rdx
59    cmp $0, %rdx
60    jne .sync2
```



More Details behind The Scene

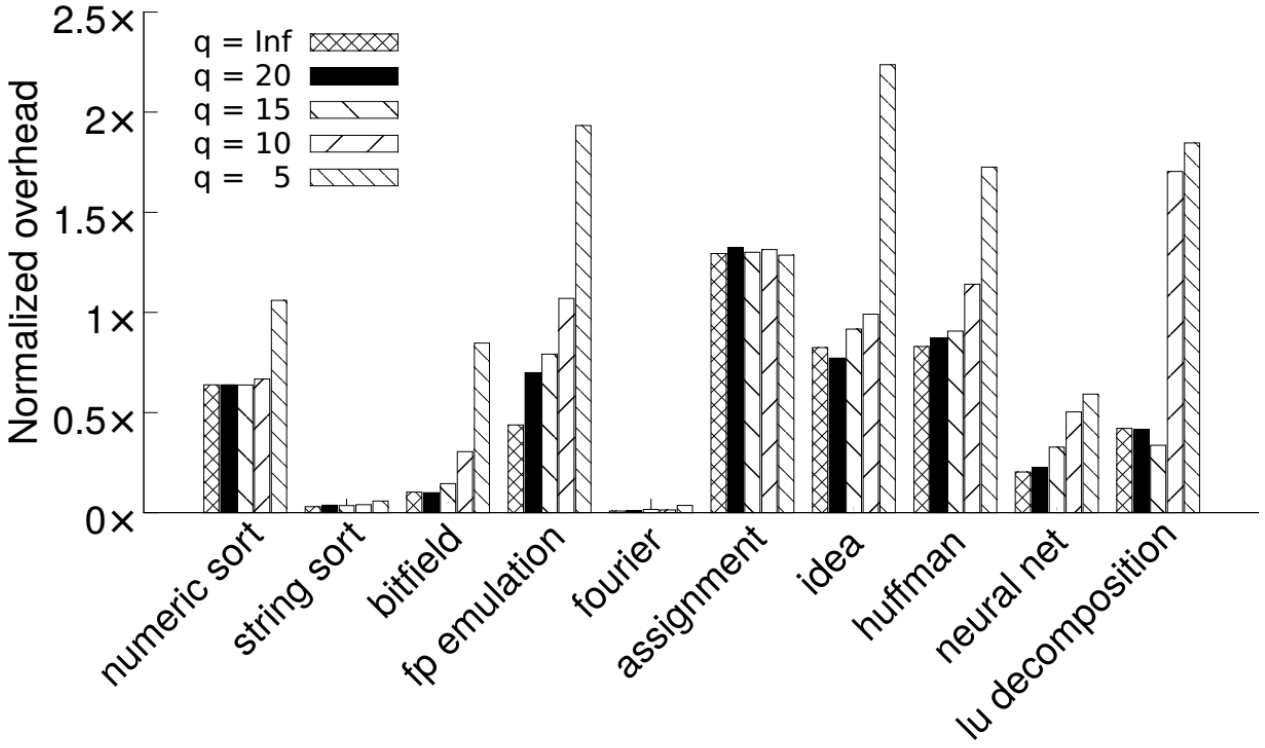
- Co-location test via statistical hypothesis testing:
  - Consider n data race unit tests: X_j, $j = 1, 2, \ldots, n$; probability of passing each test $P(X_j = 1) = p$.
  - Null hypothesis and alternative hypothesis

    H_0: $\hat{p} \geq p$; the two threads are co-located.

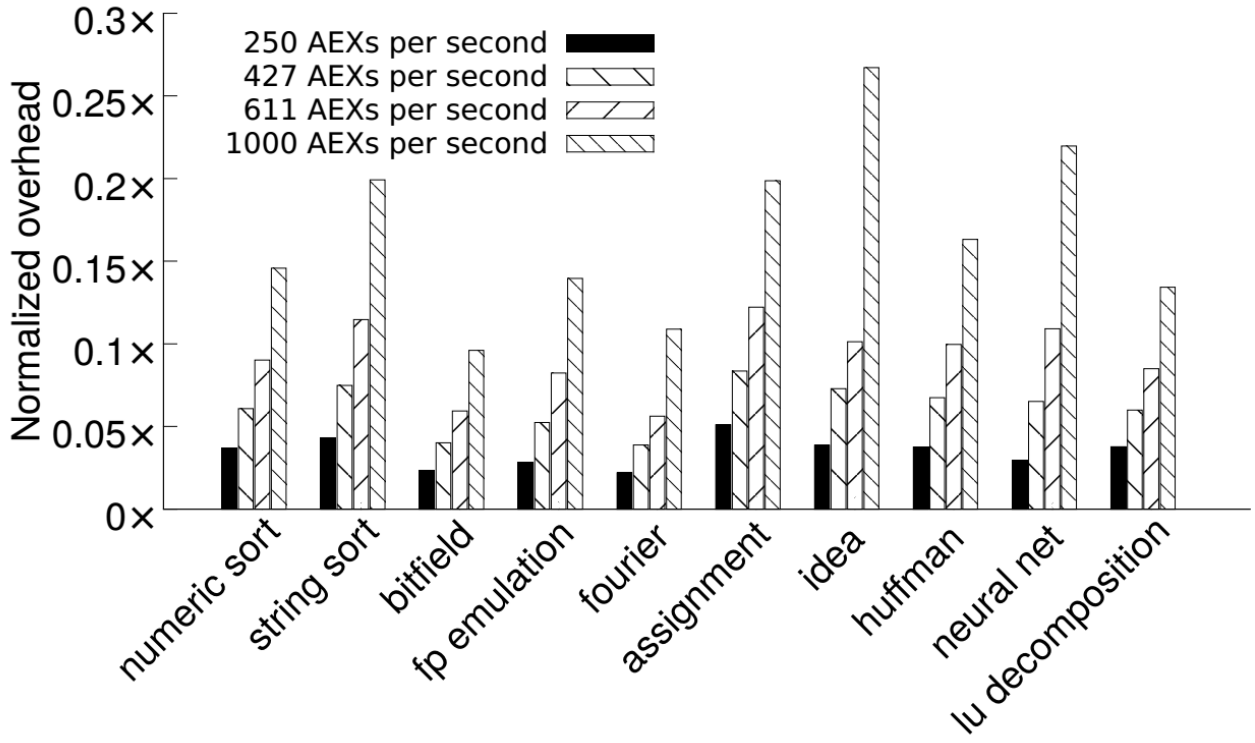
    H_1: $\hat{p} < p$; the two threads are not co-located.

- Threat model
  - Altering the execution speed of the enclave program by (1) causing cache contention, (2) altering CPU frequency, and (3) disabling caching.
- Security analysis
  - Two threads not co-located: the co-location test will fail with very high probability
  - Two threads co-located: the co-location test will fail with very low probability

Performance overhead of AEX detection



Performance overhead of co-location tests



Roadmap of My Research

- Understanding side-channel hazards of Intel SGX
  - Memory side-channel attack surfaces (CCS'17a)
- Detecting side-channel vulnerabilities in enclave programs
  - Dynamic analysis tools for detecting sensitive control-flow vulnerabilities in SSL/TLS (CCS'17b)
- Compiler-assisted runtime defenses
  - Timed execution for detecting side-channel attacks at runtime (AsiaCCS'17)
  - Contrived data race for detecting Hyper-Thread co-location (S&P'18)
- Branch target injection & SGX side channels
  - SgxPectre attacks: Exploiting speculative execution in SGX enclaves

GitHub, Inc. [US] https://github.com/OSUSecLab/SgxPectre

SgxPectre Attacks

Practical Spectre attacks against Intel's SGX enclaves, including Intel signed privileged enclaves, e.g., quoting enclaves.

Overview

SgxPectre Attacks resulted from a research project conducted by security researchers at The Ohio State University. The study systematically explores the insecurity of Intel SGX due to branch target injection attacks and micro-architectural side-channel attacks. The research is one of a series of research projects on SGX side channels in which OSU researchers have been involved.

Software Guard eXtensions (SGX) is a hardware extension available in recent Intel processors. SGX provides software applications shielded execution environments, called *enclaves*, to run private code and operate sensitive data, where both the code and data are isolated from the rest of the software systems. Even privileged software such as the operating systems and hypervisors are not allowed to directly inspect or manipulate the memory inside the enclaves. There are already commercial cloud platforms that utilize SGX to offer customers trustworthy computing environments.

However, it has already been demonstrated that by observing execution traces of an enclave program left in the CPU caches, branch target buffers, DRAM's row buffer contention, page-table entries, and page-fault exception handlers, a side-channel adversary with system privileges may *infer* sensitive data from the enclaves. These traditional side-channel attacks are only feasible if the enclave program already has secret-dependent memory access patterns.

SgxPectre can Extract Data from Intel's SGX Enclaves

March 06, 2018

A recent research paper from the Ohio State University detailed an update of the Spectre attacks on Intel's Software Guard Extensions (SGX). SGX is a feature of modern Intel processors that protects selected code and data from disclosure or modification by containing them in "enclaves". If the attack, named SgxPectre by the researchers, is successful, an attacker would be able to extract data from these enclaves.

While users are still coping with Meltdown and Spectre, a pair of Intel processor vulnerabilities discovered in January of this year, even those serious flaws did not appear to affect SGX enclaves. The enclave security was designed so that even operating systems are not permitted to access what's inside.

Related Posts

- New Exploit Kit Fallout Delivering Gandcrab Ransomware
- Patch Now: New Mirai, Gafgyt Variants Target 16 Flaws Via Multi-Exploits
- Addressing Challenges in Hybrid Cloud Security
- Unseen Threats, Imminent Losses
- PyRoMineIoT Targets, Infects, and Spreads to Vulnerable IoT Devices

yinqian@cse.ohio-state.edu

http://web.cse.ohio-state.edu/~zhang.834/