Lecture 13 - Intel SGX

Anton Burtsev
November, 2021
Motivation for SGX

• Security and isolation in commodity systems
  • Privilege levels (rings) protect the kernel from user programs
Motivation for SGX

• Security and isolation in commodity systems
  • Privilege levels (rings) protect the kernel from user programs
  • Page tables protect programs from each other
Motivation for SGX

• Security and isolation in commodity systems
  • Privilege levels (rings) protect the kernel from user programs
  • Page tables protect programs from each other
Operating systems haven’t changed for decades

- 40 years old
  - Time-sharing
  - Expensive hardware
  - Overly general

Ken Thompson (sitting) and Dennis Ritchie working together at a PDP-11 (1972)
- 17,000,000 LoC
- 40 subsystems
- 3,200 device drivers
Modern kernels are vulnerable

Linux Kernel Vulnerabilities by Year
Motivation for SGX

- Security and isolation in commodity systems
  - Privilege levels (rings) protect the kernel from user programs
  - Page tables protect programs from each other

- Until one program (malware) attacks the kernel and then attacks any program in the system
TCB of a modern system

- Attack surface is giant
  - OS kernel
    - 17,000,000 lines of code
    - 40 major subsystems
    - 3,200 device drivers
  - Virtual Machine Monitor
    - Hypervisor
    - QEMU emulator
    - Device drivers
    - Parts of host kernel (KVM)/Domain0 (Xen)
Enclaves

- Applications can protect their secrets
  - TCB is small
    - Intel CPU
    - App code itself
  - Protected from malicious
    - BIOS
    - SMM
    - Hypervisor
    - Kernel
- Familiar application environment
SGX enclaves

• Trusted execution environment embedded in the process
SGX enclaves

- Trusted execution environment embedded in the process
  - It’s own code and data
  - Controlled entry points
  - Multi-threading

- Confidentiality

- Integrity
Performance
Performance

- Enters and exits are expensive
  - EEXIT 3,330 cycles
  - EENTER 3,800 cycles
  - Intel SDK adds another 800 cycles
  - Normal syscall is 250 cycles

- Memory is encrypted

- Limited physical memory

Performance

• Enters and exits are expensive
  - EEXIT 3,330 cycles
  - EENTER 3,800 cycles
  - Intel SDK adds another 800 cycles
  - Normal syscall is 250 cycles

• Memory is encrypted
• Limited physical memory

<table>
<thead>
<tr>
<th>Operation</th>
<th>Sequential access</th>
<th>Random access</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>5.6×</td>
<td>5.6×</td>
</tr>
<tr>
<td>WRITE</td>
<td>6.8×</td>
<td>8.9×</td>
</tr>
<tr>
<td>READ and WRITE</td>
<td>7.4×</td>
<td>9.5×</td>
</tr>
</tbody>
</table>

Table 1: Relative cost of LLC misses when accessing EPC vs. accesses to untrusted memory.

• Enters and exits are expensive
  - EEXIT 3,330 cycles
  - EENTER 3,800 cycles
  - Intel SDK adds another 800 cycles
  - Normal syscall is 250 cycles

• Memory is encrypted

• Limited physical memory
  - 128MB (in practice only 90MB available for your application)
  - 40,000 cycles per EPC fault (25K driver, 7K exit/entry, 8K indirect)

Perspective (40,000 cycles per EPC page-fault)

- 4KB page
  - 256 byte DB record
  - 16 records per page
  - Assume we just copy them
    - 50 cycles per record, $50 \times 16 = 800$ cycles per page
    - Inside SGX this number is $40,000 + 800$ cycles per page or $51x$ slower
  - Maybe we lookup record in a hash-table
    - 300 cycles per lookup, $300 \times 16 = 4,800$ cycles per page
    - Inside SGX it’s $44,800$ cycles or $9x$ slower

Performance: KV store (parameter server)

- 10x-33x slowdown
  - 2MB
  - 9000 cycles (inside the enclave) vs 1000 (outside) per-request

Security
Powerful adversary model

• OS + VMM
  • Controlled execution environment
  • Control over page faults
  • Suspending execution
    • Single stepping
  • Flushing caches
SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control

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

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

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

Abstract

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

This paper presents SGX-Step, an open-source Linux kernel framework that allows an untrusted host process to configure APIC timer interrupts and track page table entries directly from user space. We contribute and evaluate an improved approach to single-step enclave execution at instruction-level granularity, and we show how SGX-Step enables the practical construction of an attack that fully leverages the maximal temporal resolution of SGX.

With the increased 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 accessing enclaved code or data directly, while still leaving it in charge of shared platform resources such as system memory or CPU time. With the arrival of Intel’s Software Guard eXtensions (SGX) [6, 7], such strong hardware-enforced trusted computing guarantees are now available on mainstream consumer devices.

Recent research demonstrated, however, that the increased capabilities of a privileged PMA attacker allow her to construct high-resolution, low-noise channels to spy on enclaved execution. Specifically, the past months have seen a steady stream of kernel-level SGX attacks exploiting information leakage from page tables [13, 15], CPU caches [4, 10], or...
Side channel attacks

- Every architectural component of the CPU
  - Branch target buffers
    - S. Lee et al., "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," in USENIX Security, 2017
  - Pattern-history table
    - D. O'Keeffe et al., "Spectre attack against SGX enclave," 2018
  - Caches
    - Brasser et al., "Software grand exposure: SGX cache attacks are practical," in WOOT, 2017
    - J. Gotzfried et al., "Cache attacks on Intel SGX," in EuroSec, 2017
    - M. Hahnel et al., "High-resolution side channels for untrusted operating systems," in USENIX ATC, 2017
    - M. Schwarz et al., "Malware guard extension: Using SGX to conceal cache attacks," in DIMVA, 2017
  - DRAM row buffer
  - Page-tables
  - Page-fault exception handlers
    - Y. Xu et al., "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," 2015
    - S. Shinde and other, "Preventing page faults from telling your secrets," in CCS, 2016
  - Speculative execution
    - J. V. Bulck et al., “Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution,” in USENIX, 2018
Page fault tracing attacks

• Controlled channel attacks

Y. Xu et al., “Controlled-channel attacks: Deterministic side channels for untrusted operating systems,” 2015
Page fault tracing attacks

- Page fault address depends on sensitive data

Y. Xu et al., “Controlled-channel attacks: Deterministic side channels for untrusted operating systems,” 2015
Example: recovering text via spell checker

- Insertions are deterministic
  - Word order is known
  - Observe sequence of page faults

- Lookup exhibits identical sequences

Y. Xu et al., “Controlled-channel attacks: Deterministic side channels for untrusted operating systems,” 2015
Example: recovering text via spell checker

• Wizard of Oz
  • All words
  • 96% accuracy

Y. Xu et al., “Controlled-channel attacks: Deterministic side channels for untrusted operating systems,” 2015
Example: recovering JPEG images

```
GLOBAL(void) jpeg_idct_islow (_decompress_ptr cinfo,
    jpeg_component_info * comptr, JCOEFPTR coef_block,
    JSAMPARRAY output_buf, JDIMENSION output_col)
{

    /* Pass 1: process columns from input... */
    intptr = coef_block;
    quantptr = (ISLOW_MULT_TYPE *) comptr->dct_table;
    wsztr = workspace;

    for (ctr = DCTSIZE; ctr > 0; ctr--) {
        /* Due to quantization, we will usually find that
         * many of the input coefficients are zero,
         * especially the AC terms. We can exploit this
         * by short-circuiting the IDCT calculation for any
         * column in which all the AC terms are zero. In
         * that case each output is equal to the DC
         * coefficient (with scale factor as needed). With
         * typical images and quantization tables, half or
         * more of the column DCT calculations can be
         * simplified this way.
         */
        if (intptr[DCTSIZE+1]==0 && intptr[DCTSIZE+2]==0 &&
            intptr[DCTSIZE+3]==0 && intptr[DCTSIZE+4]==0 &&
            intptr[DCTSIZE+5]==0 && intptr[DCTSIZE+6]==0 &&
            intptr[DCTSIZE+7]==0) {
            /* AC terms all zero */
            ... SIMPLE COMputation ...
            intptr++; quantptr++; wszptr++;
            continue;
        }
        ... COMPLEX COMPUTATION ...
        intptr++; quantptr++; wszptr++;
    }
}
```

- **JPEG**
  - Process 8x8 blocks
  - Function fits on one page
    - Cannot reason about input-dependent page-faults
  - Can reason about number of page-faults
    - Optimizations in the code take shortcuts

Y. Xu et al., “Controlled-channel attacks: Deterministic side channels for untrusted operating systems,” 2015
Cache attacks: Prime + probe

Brasser et al., "Software grand exposure: SGX cache attacks are practical," in WOOT, 2017
Controlled execution environment

- Isolated core
- Execute attack in L1
  - Separate instruction and data caches
  - No self-pollution
- SMT
  - Uninterrupted execution
- Performance Monitoring Counters (PMC)
  - Cache-misses

Brasser et al., "Software grand exposure: SGX cache attacks are practical," in WOOT, 2017
Branch shadowing attack

- SGX does not clear branch history

S. Lee et al., “Inferring fine-grained control flow inside SGX enclaves with branch shadowing,” in USENIX Security, 2017
• SGX does not clear branch history

• Can we extract this information?

• Last Branch Record (LBR)
  • \{from, to, predicted, timestamp\}

S. Lee et al., “Inferring fine-grained control flow inside SGX enclaves with branch shadowing,” in USENIX Security, 2017
**Branch shadowing attack**

- 66% of 1024 RSA private key from a single run
- Full key from 10 runs

---

```c
/* Sliding-window exponentiation: X = A^E mod N */
int mbedtls_mpi_exp_mod(mbedtls_mpi *X, const mbedtls_mpi *A,
                         const mbedtls_mpi *E, const mbedtls_mpi *N,
                         mbedtls_mpi *_RR) {
    ...
    state = 0;
    while (1) {
        ...
        // i-th bit of exponent
        ei = (E->p[nblims] >> bufsize) & 1;
        // cmpq 0x0,0xc68(%rbp); jne 3f317; ...
        *if (ei == 0 && state == 0)
            continue;
        // cmpq 0x0,0xc68(%rbp); jne 3f371; ...
        *if (ei == 0 && state == 1)
            mpi蒙mul(X, X, N, mm, &T);
        state = 2; nbits++;
        wbits |= (ei << (wsize-nbits));
        *if (nbits == wsize) {
            for (i = 0; i < wsize; i++)
                mpi蒙mul(X, X, N, mm, &T);
            if (wbits & [wbits], N, mm, &T);
            state--; nbits = wbits = 0;
        }
    }
    ...
}
```

S. Lee et al., “Inferring fine-grained control flow inside SGX enclaves with branch shadowing,” in USENIX Security, 2017
Possible Defenses
Data-oblivious primitives

- Assignments and comparisons

Non-oblivious

```c
int max(int x, int y) {
    if (x > y) return x;
    else return y;
}
```

Oblivious

```c
int max(int x, int y) {
    bool getX = ogreater(x, y);
    return omove(getX, x, y);
}
```

Data-oblivious primitives

- Assignments and comparisons

### Non-oblivious

```c
int max(int x, int y) {
    if (x > y) return x;
    else return y;
}
```

### Oblivious

```c
int max(int x, int y) {
    bool getX = ogreater(x, y);
    return omove(getX, x, y);
}
```

<table>
<thead>
<tr>
<th>ogreater()</th>
<th>omove()</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov rcx, x</td>
<td>mov rcx, cond</td>
</tr>
<tr>
<td>mov rdx, y</td>
<td>mov rdx, x</td>
</tr>
<tr>
<td>cmp rcx, rdx</td>
<td>mov rax, y</td>
</tr>
<tr>
<td>setg al</td>
<td>test rcx, rcx</td>
</tr>
<tr>
<td>retn</td>
<td>cmovz rax, rdx</td>
</tr>
<tr>
<td></td>
<td>retn</td>
</tr>
</tbody>
</table>

Data-oblivious primitives

- Array access
  - Scan entire array
  - AVX instructions

How does this apply to databases

- It’s possible to build an oblivious database
  - Oblivious primitives for accessing records
  - Oblivious sort for joins
    - Parallel Bitonic sort
    - $N \times (\log(N))^2$

- For a 1M records $\log_2(1,000,000) = 20$
- 10M records $\log_2(10,000,000) = 23$
What’s the future?
What will be fixed in hardware?

**Will be fixed**
- Caches
  - Partitioned caches
- Branch predictors and likely other microarchitectural components of the CPU
- Speculative Taint Tracking (STT)

**Will not be fixed**
- Paging attacks
  - SGX inherently leaves page table under control of the OS
- Memory
  - Enclave’s memory is observable by the OS and hardware attacks
  - ORAM is 10x overhead