Multi-Core Memory Hierarchies

Lecture 12:
Memory System Security

Tong Dong

http://mprc.pku.edu.cn/~tongdong/MMH
What is Security?

Cornerstone security properties (CIA):

- **Confidentiality: “Read”**
  - Prevent disclosure of information to an unauthorized entity
    - Example attack: Eavesdropping

- **Integrity: “Write”**
  - Prevent unauthorized modification without detection
    - Example Attack: Corruption attacks (code injection)

- **Availability**
  - System and services are available when requested by legitimate users
    - Example Attack: Denial of Service
More Security Aspects

- Access control policy
  - Specifies which principals can access which objects/resources
  - Authentication and Authorization are essential aspects

- Accountability and Attribution
  - Specifies if/how actions can be tied to attacks

- Non-repudiation
  - Ability to hold one responsible to messages/events

- Anonymity
  - Ability to carry out actions without identification

- Privacy
  - Right to determine how one’s personal information is distributed

- Distinction between Security (confidentiality) & Privacy
  - Confidentiality is the obligation to protect secret information
  - Privacy is the right to protect distribution of personal information
A threat model defines
- the threats that are being considered
- the threats that are not being considered
  - e.g., Consider threats/attacks on the confidentiality and integrity of sensitive data, but not Denial of Service threats/attacks.
- the basic assumptions of the computing model

An attack is an instantiation of a threat

Attacks violate security properties of a system
Security Policies vs. Security Mechanisms

- **Security Policy**
  - specifies what and who is allowed access to what resources or information, when.

- **Security mechanisms**
  - implements the security policy

- **Trusted vs. Trustworthy**
  - Trusted – depended on to maintain the security policy
  - Trustworthy – designed to be secure, dependable
  - A trusted component may not be trustworthy, and vice versa

- **Trusted Computing Base (TCB)**
  - the hardware, software or networking components that must be correct and un-corruptible, otherwise the security policy may not hold
Access Control (AAA)

- **Authentication: Who are you?**
  - Authenticate human/machine to machine
  - What you know, What you have, What you are.

- **Authorization: What are you allowed to do?**
  - Restrict actions of authenticated users
  - Who can do what to which object?

(Subject, object, rights)

<table>
<thead>
<tr>
<th></th>
<th>Final Grades</th>
<th>Homework Grades</th>
<th>Exam Prep</th>
<th>Lecture Slides</th>
</tr>
</thead>
<tbody>
<tr>
<td>Professor</td>
<td>R, W</td>
<td>R</td>
<td>R, W</td>
<td>R, W</td>
</tr>
<tr>
<td>TA</td>
<td>R</td>
<td>R, W</td>
<td>R, W</td>
<td>R</td>
</tr>
<tr>
<td>Student</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>R</td>
</tr>
</tbody>
</table>

Access Control Matrix
Classes of Cryptography

- Symmetric Key Ciphers
  - Useful for protecting Confidentiality

- Cryptographic Hash functions
  - Useful for protecting Integrity

- Public Key Ciphers
  - Useful for longer-term identity: authentication, digital signatures and non-repudiation

- Cryptographic access control: put encrypted data in public, but control access to keys

- Cryptographic Protocol: SSL, SSH, IPSec, etc.
Cryptography

- **Symmetric-key crypto:**
  - Shared secret key used for encryption and decryption

- **Public-key (asymmetric-key) crypto:**
  - Encryption with Public key of recipient, decryption with Private key of recipient
  - Signing with Private key of sender, Verification with Public key of sender
Block Ciphers

Examples:

AES = Advanced Encryption Std.

DES = Data Encryption Std.

3DES = Triple DES

Many others
Model of Operations

(a) ECB Mode

Plaintext

\[ P_{i-1}, P_i, P_{i+1}, \ldots \]

\[ \rightarrow \]

\[ E_K, E_K, E_K, \ldots \]

\[ \rightarrow \]

\[ C_{i-1}, C_i, C_{i+1}, \ldots \]

Ciphertext

(b) CBC Mode

Plaintext

\[ P_{i-1}, P_i, P_{i+1}, \ldots \]

\[ \rightarrow \]

\[ \text{XOR} \]

\[ \rightarrow \]

\[ E_K, E_K, E_K, \ldots \]

\[ \rightarrow \]

\[ C_{i-1}, C_i, C_{i+1}, \ldots \]

Ciphertext

(c) Counter Mode

Plaintext

\[ P_{i-1}, P_i, P_{i+1}, \ldots \]

\[ \rightarrow \]

\[ \text{XOR} \]

\[ \rightarrow \]

\[ \text{CTR}_{i-1}, \text{CTR}_i, \text{CTR}_{i+1}, \ldots \]

Ciphertext

Electronic Code Book Mode

Cipher Block Chaining Mode

Counter Mode
Cryptography Hash

- Acts like a fingerprint of a message
Memory Integrity Tree

Merkle hash tree for Integrity Verification of large amounts of data
Security Profiles: Software Attacks

- **Software Attacks**
  - Malware & viruses
  - Social engineering

- **Non-invasive Hardware Attacks**
  - Side channels (DEMA, DPA)
  - Physical access to device – JTAG, bus probing, IO pins, etc.

- **Invasive Hardware Attacks**
  - Well resourced and funded
  - Unlimited time, money & equipment

- **TrustZone® technology-based TEE**

- **SmartCards/HSMs**
Severe Software Vulnerabilities

- Buffer errors: 23%, 22%
- Code injection: 4%
- SQLi: 9%
- Resource management: 6%, 5%
- Access control: 9%
- Input validation: 13%
- XSS: 12%
- Other categories: 40%, 41%
- 31%, 30%
Eternal War in Memory

Make a pointer go out of bounds
Make a pointer become dangling

Use pointer to write (or free)
Use pointer to read

Modify a data pointer
Modify code ...

Modify a code pointer ...

Modify a data variable ...

Output data variable

... to the attacker specified code

... to the address of shellcode / gadget

... to the attacker specified value

Interpret the output data

V.A. Address Space Randomization
V.B. Data Space Randomization

VII.A. Data integrity
VIII.A. Code Pointer Integrity

Code Integrity

VIII.B. Control-flow integrity
VII.B. Data-flow Integrity

Use pointer by indirect call/jump
Use pointer by return instruction
Use corrupted data variable

Execute available gadgets / functions
Execute injected shellcode

Code corruption attack
Control-flow hijack attack
Data-only attack
Information leak

Non-executable Data / Instruction Set Randomization
Software Attacks: Dangling Pointer

```c
// Original program
int main() {
    char *p1, *p2;
    p1 = malloc(32);
    strcpy(p1, "hello");
    p2 = p1;
    strcat(p2, "world");
    free(p1);
    // p1 and p2 become dangling pointers
    strcat(p2, "?"');
}
```

```c
// Transformed program
int main() {
    char *p1, *p2;
    p1 = DDMalloc(32);
    strcpy(*((char**)p1), "hello");
    p2 = p1;
    strcat(*((char**)p2), "world");
    free(*((char**)p1));
    *((char**)p1) = NULL;
    strcat(*((char**)p2), "?"');
}
```

![Figure 1: An Example of Heap Space Dangling Pointer](image)

(a) Point-to Relations for Aliases in the Original Program in Fig. 1

(b) Point-to Relation for Aliases in the Transformed Program in Fig. 1
Software Attack: Buffer Overflow

- Code injection attack
  - StackGuard: stack canaries
  - Data Execution Prevention (DEP):
    - $W \oplus X$
    - Non-Executable (NX) Memory

```c
void f(void) {
    char buffer[20];
    if (gets(buffer) != NULL) 
        ...
}
```

Input 1: “0123456789”

Input 2: “0123456789012345678901234567890123456789…”

Input 3: “01234567890123456789…\xR1\xR2\xR3\xR4\x31\xC0…”
ROP/JOP: Return/Jump-Oriented Programming

- ROP and JOP: Turing-complete
- Address Space Layout Randomization (ASLR)
- Control-Flow Integrity
  - Shadow Call Stack
  - Branch Regulation
- Pointer Integrity

(a) The ROP model

(b) The JOP model
No-control Data Attacks

- Control-Flow Bending
  - Printf-Oriented Programming

- Counterfeit Object Oriented Programming (COOP)
  - Virtual function table in C++

- Data-Oriented Programming

- Data Flow Integrity and Isolation

```c
void or(int* in1, int* in2, int* out) {
  printf("%s%s\n", in1, in2, out);
  printf("%s\n", out, out);
}

void not(int* in, int* out) {
  printf("%*d%\s\n", 255, in, out);
  printf("%s\n", out, out);
}

void test(int in, int const, int* out) {
  printf("%*d%*d%\n", in, 0, 256-const, 0, out);
  printf("%s\n", out, out);
  printf("%*d%\s\n", 255, out, out);
  printf("%s\n", out, out);
}

char* pad = memalign(257, 256);
memset(pad, 1, 256);
pad[256] = 0;
void single_not(int* in, int* out) {
  printf("%*d%\s%hhn%\s%\s\n", 255, in, out,
            addr_of_argument, pad, out, out);
}
```

Figure 7: Gadgets for logic gates using printf.
Security Profiles: Hardware Attacks

Cost/Effort To Attack

TrustZone® technology-based TEE

SmartCards/HSMs

Invasive Hardware Attacks
- Well resourced and funded
- Unlimited time, money & equipment

Non-invasive Hardware Attacks
- Side channels (DEMA, DPA)
- Physical access to device – JTAG, bus probing, IO pins, etc.

Software Attacks
- Malware & viruses
- Social engineering

Cost/Effort to Secure
Trust Computing

- **TPM (Trusted Platform Module)**
  - Trusted Computing Group
  - Evaluation Assurance Level: EAL4+
  - Intel TXT
  - Server/Desktop/Laptop Market

- **TEE (Trusted Execution Environment)**
  - Global Platform
  - Evaluation Assurance Level: EAL2+
  - ARM TrustZone
  - Mobile Phone/Tablet Device Market

- Secure Boot
- Cryptographic acceleration
TEE System Architecture

[Diagram showing the TEE system architecture with layers of privilege and components like REE, TEE, Trusted OS Components, Trusted Kernel, and Trusted Peripherals.]
TEE Realization Examples

ARM

Intel

AMD
Intel TXT: Trusted Execution Technology

- Intel® TXT and VT-x Support
- Intel® TXT and VT-d Support
- BIOS AC Module, Preboot TXT Init Code
- Intel® Software SINIT AC module
- 3rd Party SW MLE, Hosted OS Apps, etc.
- TPM Support
- Intel Chipset

- Xeon®
- Flash
Trusted Platform Module (TPM 1.2)
TurstZone: Two CPUs virtualized in One

- TEE with only one Entry/Exit
- Keys only used in secure world
- Offers Integrity (secure boot) and Confidentiality
- CPU MHz/resources are dynamically shared
- Two domains in same machine
  - Difficult to give precise “overhead” values since secure and non-secure tightly integrated from design standpoint
- Use exceptions to move between modes
In 2015, Intel released the Skylake micro-architecture featuring a secure execution technology - Software Guard Extensions (SGX)

- SGX allows secure execution in user-space (ring 3) in a container called a secure-enclave, which is shielded from the OS, VMM, and SMM.
  - Ideally, no vulnerability or intentionally malicious code in any of these layers should compromise the confidentiality or the integrity of the secure-enclave.
  - No probing of physical buses outside the processor chip should compromise the security, as the memory is encrypted as well.

Figure 1: Interaction between the application and the secure-enclave: control is transferred to the enclave via ecall; requests for OS API calls are processed via ocalls.

Hardware-enhanced security Research

- **Hardware enabled isolation**
  - Static versus dynamic partitioning of resources for trusted versus untrusted software
  - Defend from “attacks from below”, e.g., untrusted OS or HV or both

- **Secure Processors (Cryptographic access control and isolation)**
  - Secure execution environment for access to keys and decrypted information
  - Hardware support for Secure key generation, management and storage
    - Master keys and key derivation
    - Secure storage
    - True Random Number Generators (TRNG)
    - Physically Unclonable Functions (PUF)
  - Mitigate information leakage thru covert channels and side channels
    - Newest control channel attacks for Intel SGX.

- **Dynamic Information Flow Tracking (DIFT)**
  - Track trusted or tainted data or control
  - Explicit versus implicit information flow tracking
  - Minimize false positives (usability) and false negatives (security)
Hardware-enhanced security Research

- **Attestation and Trust Evidence**
  - How can the system assure the user that his desired security properties are being provided?
  - What information can a system collect to provide evidence so that the user can “trust” it?

- **Moving Target Defense**
  - Randomization and other techniques that thwart attack strategies that depend on known vulnerabilities, fixed mappings or locations, or predictable values

- **Software-hardware security verification**
  - Combine software, hardware and network protocol security verification
  - Scale to realistic systems (with accurate abstractions)
  - Compose security verification of subsystems

- **Security Metrics**
  - How can we meaningfully evaluate if one system is more secure than another, for some security property?
Secure Hardware Research

- **Detect and Mitigate Side-Channel Attacks**
  - Leak critical information through correctly implemented hardware subsystems
  - HW or SW attacks on hardware resources
    - Unlike SW vulnerabilities, the HW is functioning correctly -- but leaking secrets!
  - Many types of side-channels: Power, timing, acoustic, caches, memory bus, branch prediction, E&M, fault-induced, etc.

- **Memory integrity (physical attacks)**
  - What if attacker changes the information written at a given memory address?
  - Faster Memory Integrity Trees, e.g., Bonzai Merkle tree
  - Memory integrity trees for multicore systems

- **Supply chain Security**
  - What if design is changed at some stage of chip implementation, fabrication or delivery?
  - Fake chips, old chips with limited lifetimes. Malicious chips

- **Hardware Trojans**

- **Security of CAD tools** that generate and verify hardware chip designs

- **IPcores and SOC security and trustworthiness**
XOM: eXecution-Only Memory

- Software Piracy
- XOM: allows instructions stored in memory to be executed but not otherwise manipulated.
  - Internal Compartment built from a session key: a process in one compartment cannot read data from another compartment
  - Encrypted External Memory
  - Tagging Data

---

Mondrian Memory Protection (MMP) allows arbitrary permissions control at the granularity of individual words.

OS Protection: Segment and Page Granularity

Protection Domain: a lightweight context that determines permissions for executing code.

Permission Table: compressed permissions information in held in main memory

Protection Lookaside Buffer (PLB)

Secure Processors

- Tamper-evident (TE) environments
- Private and authenticated tamper-resistant (PTR) environments
- Trusted Computing Base (TCB)
- Integrity Verification and Encryption

Figure 1: Our secure computing model.

Figure 2: A binary hash tree. Each internal node is a hash of the concatenation of the data in the node’s children.

Program Vulnerabilities: buffer overflow which is very difficult to protect as the **first step** of an attack.

Prevent the **final step**, namely, the unintended use of I/O inputs

- Changing a code pointer for indirect jumps, or inject malicious code at a place that will be executed without requiring malevolent control transfer.

**Dynamic Information Flow Tracking** to track I/O inputs and monitor their use: Tagged Memory

Memory Attacks

- The SoC as the trust boundary
  - Memory Modification, Rollback, Cut & Paste
- Cold-boot attack, Memory Bus Monitoring, DMA Attacks, Induce Memory Errors and Exploit it.
- Sentry: a new security approach – storing users’ sensitive data on the ARM SoC rather than in DRAM.
  - iRAM: Scratchpad Memory
  - Cache locking
  - Encrypts the memory of sensitive applications and OS subsystems when a mobile device transitions to a screen-locked state

<table>
<thead>
<tr>
<th>In Scope Attacks</th>
<th>Out of Scope Attacks</th>
<th>iRAM</th>
<th>Locked L2 Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cold boot</td>
<td>software attacks (malware)</td>
<td>Safe</td>
<td>Safe</td>
</tr>
<tr>
<td>Bus monitoring</td>
<td>physical side-channel attacks</td>
<td>Safe</td>
<td>Safe</td>
</tr>
<tr>
<td>DMA attacks</td>
<td>code-injection</td>
<td>Safe (ARM TrustZone)</td>
<td>Safe</td>
</tr>
<tr>
<td></td>
<td>JTAG attacks</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sophisticated physical attacks</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(e.g., using electron microscope)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Security analysis of different storage alternatives to DRAM.
ORAM: Oblivious RAM

- ORAMs make the sequence of memory locations accessed indistinguishable from a random sequence of accesses, from a cryptographic standpoint.

L. Ren, et al. 2013. Design space exploration and optimization of path oblivious RAM in secure processors. ISCA-40. (96/19)
Cold Boot Attacks

- Memory Encryption
- DDR3/4 Scrambling is also leaky.
- New Stream Cipher Engines
  - Low-latency, low-power, high throughput
- AES-CTR and Chacha8: Counter-based stream ciphers
  - Permitting to perform keystream generation without having the corresponding plaintext or ciphertext

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Maximum Freq.(GHz)</th>
<th>Cycles per 64B</th>
<th>Maximum Pipeline Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>2.4</td>
<td>13</td>
<td>5.4</td>
</tr>
<tr>
<td>AES-256</td>
<td>2.4</td>
<td>17</td>
<td>7.08</td>
</tr>
<tr>
<td>ChaCha8</td>
<td>1.96</td>
<td>18</td>
<td>9.18</td>
</tr>
<tr>
<td>ChaCha12</td>
<td>1.96</td>
<td>26</td>
<td>13.27</td>
</tr>
<tr>
<td>ChaCha20</td>
<td>1.96</td>
<td>42</td>
<td>21.42</td>
</tr>
</tbody>
</table>

Table II: Cipher Engine Performance (45nm). This table provides the speed of the five cipher engines analyzed. All implementations were synthesized to a 45nm silicon-on-insulator technology. The latencies presented here do not include potential queuing delays.
Side-Channel Attacks

- Side-Channel Attacks target implementation characteristics of crypto algorithms
  - Execution time,
  - Power consumption and thermal
  - Resource sharing: Cache, MMU Cache, NoC, Memory Controller and Bus, etc.
  - Sounds
  - Radiation

- Side-Channel Attacks are serious threats to information security
  - Hard to detect. do not require special privileges or equipment, and their behavior may not be overtly harmful.
  - Not easy to eliminate without affecting performance, e.g., disabling a cache or disabling sharing of bus wires
  - Can break various systems, because side-channel attacks target the physical features of implementations of algorithms, rather than a specific algorithm.
  - Destroy the entire confidentiality of data or programs, because they recover cryptographic keys, instead of decrypting specific data
Cache-based Side-Channel Attacks

- Cache based side-channel attacks exploit the difference in the access times of cache hits and misses.
- Partitioning based approaches
- Randomization based approaches

<table>
<thead>
<tr>
<th>Miss Based Attacks</th>
<th>Timing Based Attacks</th>
<th>Access Based Attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type 1:</strong> Observe if victim uses the attacker-evicted cache line(s), causing victim’s longer execution time for an entire security-critical operation. <em>E.g., Evict-and-time attack</em></td>
<td><strong>Type 2:</strong> Observe if victim evicts the attacker’s cache line(s), causing the attacker to later miss on these cache line(s) resulting in longer memory access time. <em>E.g., Prime-and-probe attack</em></td>
<td></td>
</tr>
</tbody>
</table>

| Hit Based Attacks | Type 3: Observe if victim reuses memory lines fetched by his previous memory accesses, causing victim’s shorter execution time for an entire security-critical operation. *E.g., Cache Collision attack* | Type 4: Observe if attacker uses the victim-fetched cache line(s), which causes the attacker’s shorter memory access time. *E.g., Flush-and-reload attack* |

<table>
<thead>
<tr>
<th></th>
<th>Type 1</th>
<th>Type 2</th>
<th>Type 3</th>
<th>Type 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA Cache</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SP Cache</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PL Cache</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Nomo Cache</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Newcache</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>RP Cache</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>RF Cache</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RE Cache</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Noisy Cache</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Access-based cache attacks

- **Prime and Probe attack (Miss-based attack)**
  - If victim evicts attacker’s cache line

- **Flush and Reload attack (Hit-based attack)**
  - If attacker reuse victim-fetched cache line(s)
Partition-Locked and Random Permutation Cache ISCA 2007

- Software Cache Side-Channel Attacks
  - By an unprivileged user program performing simple timing measurements based on cache misses
  - The root cause: Cache Interference

- Partition-Locked cache (PLcache)
  - To eliminate cache interference

- Random Permutation cache (RPcache)
  - To randomize cache interference

Figure 4. Access handling procedure for PLcache

Figure 5. A logical view of the RPcache

Software techniques to mitigate the information leakage problem in caches always hurts performance, 2~4x slowdown.

Hardware Technique (ISCA’07) cause cache under-utilization.

NewCache

- Direct-mapped cache: fast access time and power efficiency
- Dynamic memory to cache mapping and longer cache index: low miss ratio and security
- Security-aware cache replacement algorithm (SecRAND)

---

Preloading and Software Permutation
HPCA 2009

- Software Cache based Side-Channel Attacks
  - Access-driven attacks: hits or misses
  - Timing-driven attacks

- Hardware-based defense lack the flexibility to adapt to newly developed attacks

- Integrated hardware/software mitigation approaches
  - Preloading
  - Informing Load Instruction: user-level exception raised when informing load misses
  - Software permutation scheme assisted by informing loads

\[
\begin{align*}
  r &= \text{random\_number;} \\
  & \quad \text{// from hardware random number generator (i.e. the one} \\
  & \quad \text{// used by the RPcache)} \\
  i\_max &= \text{number\_of\_tables;} \\
  j\_max &= \text{table\_size} / \text{table\_element\_size}; \\
  & \quad \text{for (i=0; i < i\_max; i++) // Fetch each protected table} \\
  & \quad \text{for (j=0; j < j\_max; j +=} \\
  & \quad \quad \text{cache\_line\_size/table\_element\_size}) \\
  & \quad \quad \text{Prefetch( T[(i XOR r) \% i\_max][(j XOR r) \% j\_max]);} \\
  & \quad \text{// Fetch each protected cache line in a random order}
\end{align*}
\]

Fig 3. Code of the informing load exception handler

Fig 5. Address permutation by swapping both the pointers and the data.

HPCA-15. (84/9)
New Cache-based Side-Channel attacks Classification:
- Contention based attacks (Previous):
- Reuse based attacks (New): exploit the reuse of a previously accessed (and cached) security-critical data to correlate the addresses of two memory accesses.

Insight: the fundamental demand fetch policy of a cache is a security vulnerability that causes the success of reuse based attacks.

Random Fill Cache: random cache fill within a configurable neighborhood window, which takes advantage of the random access pattern found in cryptographic algorithms

Random Access Pattern
Catalyst HPCA 2016

- Secret information in one VM can be extracted by another co-resident VM using high-bandwidth, low-noise side channel attacks on the last-level cache (LLC)
- Intel CAT (Cache Allocation Technology)
- CATalyst: a pseudo-locking mechanism which uses CAT to partition the LLC into a hybrid hardware-software managed cache.
  - Using CAT as a coarse-grained mechanism to partition the LLC into secure and non-secure partitions
  - The secure partition is loaded with cache-pinned secure pages.

(a) The **PRIME+PROBE** attack

(b) The **FLUSH+RELOAD** attack
Catalyst HPCA 2016

- Security guarantees
  - No eviction of secure pages by malicious code
  - No overlapping of secure pages between different active VMs (security domains)

- Page coloring based defenses

![Diagram](image)

**Figure 3:** CATalyst architecture, with finer-grained, page-level partitioning of 2 ways of the LLC.

<table>
<thead>
<tr>
<th>COS0</th>
<th>M9</th>
<th>M8</th>
<th>M7</th>
<th>M6</th>
<th>M5</th>
<th>M4</th>
<th>M3</th>
<th>M2</th>
<th>M1</th>
<th>M0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>COS1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>COS2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>COS3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 4:** Example bit mask configuration for CAT as a pseudo locking mechanism.

**Figure 5:** Procedure to pin one secure page into the secure partition.

**Input:** Secure page $P$ to be pinned. $C$: code that loads a secure page, $P$

1) Disable local interrupts
2) Access $C$
3) Access one word in $P$
4) Set current logical processor to COS3
5) cfflush $P$
6) Access $P$
7) Set current logical processor back to COS0
8) Enable local interrupts
Covert Channel:

- A communication channel where a sender encodes information by modulating a condition that is detectable by a receiver.
- A Trojan process to communicate sensitive information to a spy process that shares some physical resource, such as:
  - File system objects
  - Network stacks/channels
  - Input devices
  - Caches and/or micro-architectural structures, such as branch predictors

J. Chen and G. Venkataramani. 2014. Cc-hunter: Uncovering covert timing channels on shared processor hardware. MICRO-47. (48/12)
ReplayConfusion: Detecting Cache-based Covert Channel Attacks Using Record and Replay, MICRO 2016

- Cache-based Convert Channel Attacks
- How to detect covert channel communication, and block future information leakage
- Observation:
  - These attacks are the malicious accesses highly tuned to the mapping of addresses to the caches;
  - Following a distinctive cadence as bits are being passed between Trojan and spy.
- ReplayConfusion
  - Record and deterministic Replay (RnR)
  - The replay execution uses a different mapping of addresses to the caches.
  - If the difference function between the cache miss rate timelines of recording
    - and replay is both sizable and retains a periodic pattern, it indicates that there is an attack.

Fig. 4. Taxonomy of cache-based covert channel communication protocols.

Fig. 6. High-level ReplayConfusion architecture.

Inclusive Cache Hierarchies in Intel Processors

Observation: when the spy wants to evict the probe address from the shared cache, the address is also practically always in the private cache of the core running the victim process.

SHARP (Secure Hierarchy-Aware cache Replacement Policy)
- A shared cache’s replacement algorithm to prevent a process from creating inclusion victims in the caches of cores running other processes.
- A slightly modified `clflush` instruction to thwart these attacks.

**Figure 2: SHARP replacement algorithm.**

1. More lines to consider?
2. Obtain information on the presence of the line in private caches
3. Is the line in any private cache?
4. Is the line present only in the requestor’s private cache?
5. Increment the alarm counter
6. Evict a random line
7. Evict the selected line
8. Is alarm counter > threshold?
9. Generate interrupt
10. End

**Figure 3: SHARP defending against a single-threaded attack.**

**Figure 4: SHARP defending against a multi-threaded attack.**

Memory Controller Side Channel

- Temporal Partitioning, TP (HPCA’14), incurs high performance overhead.
- Bank Triple Alternation, BTA (HPCA’15), improves the performance of TP by restricting DRAM schedules to enforce consecutive requests access different banks.
- Rank Partitioning, RP (HPCA’15), suffers from the limited number of ranks.
- SecMC-NI (NI, Non-Interference), allows multiple requests to be issued within a short period by interleaving accesses to different banks and ranks.
- SecMC-Bound, hides most (not all) timing interferences by delaying memory responses, and provides an information theoretic bound on information leakage by bounding the number of cases that the interference is visible to a program.

![Figure 3: SecMC-NI scheduling example.](image)

![Figure 4: Bank conflict in SecMC-NI scheduling.](image)

![Figure 5: SecMC-NI scheduling with rank interleaving.](image)

MITTS works by having a simple hardware traffic shaper for each core or thread that can shape memory traffic into a prescribed distribution.

MITTS not only controls the rate based on processor core’s feedback, but also controls the distribution of request inter-arrival times using an online auto-tuner.

- Bin-based Bandwidth Shaper
- Bin Credits Replenishment
- Network Traffic Shaping:
  - Leaky Bucket and Token Bucket

![Credit Deduction Given the Request Inter-Arrival Time. Four bins have credits in left example.](image)

Figure 5: Hardware Memory Shaper

Figure 6: MITTS Hardware Implementation.
Camouflage: Memory Traffic Shaping to Mitigate Timing Attacks, HPCA 2017

- Camouflage: a hardware solution to mitigate timing channel attacks not only in the memory system, but also along the path to and from the memory system (e.g. NoC, memory scheduler queues).
- Novel Idea: shaping memory requests’ and responses’ inter-arrival time into a pre-determined distribution for security purposes, even creating additional fake traffic if needed.
- Three different memory traffic shaping mechanisms for different security scenarios by having Camouflage work on requests, responses, and bi-directional (both) traffic.

Figure 1. An Example of Timing Leakage. Attacker measures its own response latency to estimate a co-scheduled VM’s memory traffic.

Memory Bus Side Channel

- **Memory Bus Security**
  - Data and address confidentiality: Oblivious RAM (ORAM) with ~100x bandwidth consumption.
  - Data integrity and freshness: Merkle Trees
  - Timing channel.

- **InvisiMem (ISCA’07):** smart memories (memories with compute capability) with packetized interface (as opposed to the DDR interface) can be taken advantage of to design an ultra-low overhead secure processor.

- **3D/2.5D DRAM and Intel Software Guard Extensions (SGX)**

---

**Figure 1:** Smart memory based secure designs. a) InvisiMem_far b) InvisiMem_near

**Figure 4:** Existing client-host remote attestation and key exchange (left). Smart memory authentication and key exchange under InvisiMem (right).

---

S. Aga and S. Narayanasamy. 2017. InvisiMem: Smart Memory Defenses for Memory Bus Side Channel. ISCA-44. (14)
HotCalls ISCA 2017

- SGX primitive call function: 50x ~ 100x slow
- HotCalls: based on a synchronization spin-lock mechanism and provide a 13-27x speedup over the default interface.

---

Enclave

Request call
1. Acquire lock
2. Set data
3. Set call_ID <- function_ID
4. Mark “Go”, release lock
5. Acquire lock
6. Is “Done” set?
   - No: release lock and go to 5

Untrusted Code

Poll for call
1. Acquire lock
2. Is ”Go” set?
   - No: release lock and go to 1
3. Release lock
4. Execute( call_ID, data )
5. Acquire lock
6. Mark “Done”, release lock

Shared Memory

Spinlock void *data call_ID Go | Done

---

Figure 10: Optimizing throughput with HotCalls and No-Redundant-Zeroing. The measurements are normalized to running without SGX. Memcached, by nature, is a memory-intensive application, and therefore optimization is limited by the performance of the memory encryption engine.

Figure 11: Optimizing latency with HotCalls and No-Redundant-Zeroing. The values above the bars are in milliseconds: For openVPN the values are the average ping round-trip time. For memcached and lighttpd the values are the server’s average response latency.

Leveraging speculation in side channel attack

- **Speculation**
  - Execute instructions early before you know that are definitely needed
  - Buffer results until instruction execution is certain (commit) than change the state
  - Maintain precise exceptions: delay any encountered exception until instruction is for certain
    - Because instructions are speculated they may appear to cause an exception that doesn’t really exist because the instruction will be discarded.

- **The Hole that Meltdown and Spectre Exploit:**
  - Speculation can change the microarchitectural state
  - Can be observed by via the side channel.
While our conditional is resolved, we may speculatively execute the inner block

- This execution will be discarded when the conditional is resolved…
- …But, time subsequent loads on accessible_memory exposes bits of private_memory

- Variant 1: Caller attacks gadgets using accessible_memory side channels
Two key examples

- Browsers: JITs run in a shared address space. Here, `private_memory` could be another tab or website; it can potentially use its own execution environment for an accessible_memory side-channel.
  - Resolution: Site isolation, all tabs in their own process

- Kernels and Hypervisors: User-space (or a guest) is usually directly mapped, providing a convenient accessible_memory side-channel. `private_memory` typically includes all host memory!
NetSpectre

- Exploit the Spectre v1 hole without running any code!
  - Break-in from a remote machine via LAN or within the cloud

- NetSpectre: leak gadget and transmit gadget

- Leak gadget:
  - Send “valid” packets to train predictor in the networking code
  - Send “invalid” packet & leak data in shadow

- Transmit gadget: cannot observe state directly.
  - Use network request to change/measure cache state.

- Can also leak through memory, I/O, and AVX instructions.

- “Noisy” channel: leaks 0.25-1 bit/minute but attack is completely remote
Security Challenges

- **Software flaws:**
  - a. HW’s job: reduce these and minimize damage
  - b. Support function, which works if programmers take advantage
    - i. Must be effective
    - ii. Must be efficient

- **Hardware flaws**
  - a. Cannot allow this--no matter how much performance could be gained!
  - b. Hard to fix and fixes may cost more than the “HW optimization” gained.
    - i. Next generation Intel processors will probably not fix Spectre v1.
  - c. Lots of us missed this problem and for about 10-15 years.
But what about variant 1?

- We don’t have the luxury to know the boundary.
- No existing constructs which allow software to advertise the hardware within a single address space.

Cascade Lake implements hardware mitigations against targeted side-channel methods.

<table>
<thead>
<tr>
<th>Variant</th>
<th>Side-Channel Method</th>
<th>Mitigation on Cascade Lake</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variant 1</td>
<td>Bounds Check Bypass</td>
<td>OS/VMM</td>
</tr>
<tr>
<td>Variant 2</td>
<td>Branch Target Injection</td>
<td>Hardware + OS/VMM</td>
</tr>
<tr>
<td>Variant 3</td>
<td>Rogue Data Cache Load</td>
<td>Hardware</td>
</tr>
<tr>
<td>Variant 3a</td>
<td>Rogue System Register Read</td>
<td>Firmware</td>
</tr>
<tr>
<td>Variant 4</td>
<td>Speculative Store Bypass</td>
<td>Firmware + OS/VMM or runtime</td>
</tr>
<tr>
<td></td>
<td>L1 Terminal Fault</td>
<td>Hardware</td>
</tr>
</tbody>
</table>

Cascade Lake SP expected to provide higher performance over software mitigations available for existing products.
Summary

- Security Basic
  - CIA: Confidentiality, Integrity, Availability
  - Tread Model and Attack
  - Access Control, Cryptographic, Protocol
  - Security Profiles, Policies and Mechanisms

- Hardware Security
  - Hardware-enhanced Software Security
    - TPM, TEE, SGX
    - XOM: eXecute-Only Memory for software privacy
    - Capability-based Security: Protection Table and Protection lookaside buffer for individual words.
    - Information Flow Tracing
    - Control Flow Integrity
  - Secure Hardware
    - Memory Bus Encryption and Integrity Verification
    - ORAM
    - Side-channel attack: cache, memory controller using randomization and partition
    - Convert-channel attack: cache, memory controller using detection.