ANNOUNCEMENT

- Checkpoint II presentations will be on 5/21
  - 8-10 min presentation + 3 min Q&A
  - Presentation MUST cover:
    - 1 slide on your research topic
    - 1 slides on your research goal(s)
    - 1-2 slides on your hypothesis and evaluation design
    - 1-2 slides on your preliminary results [very important]
    - 1 slide on your next steps until the final presentation

CS 578: CYBER-SECURITY

PART III: SIDE-CHANNELS

Sanghyun Hong
sanghyun.hong@oregonstate.edu

SAIL
Secure AI Systems Lab

How can we break the isolation?

- Rowhammer breaks **integrity**
- Side-channels break **confidentiality**

DIFFERENTIAL POWER ANALYSIS

CACHE-BASED TIMING SIDE-CHANNEL: FLUSH+RELOAD

PRELIMINARIES ON MEMORY ARCHITECTURE

- The X86 cache
  - Memory architecture

PRELIMINARIES ON MEMORY ARCHITECTURE

- The X86 cache
  - Memory architecture
  - Memory access latency:
    - DRAM >>> L3 cache >> L2 cache > L1 cache > Registers
    - Cache access is much faster than DRAM access

PRELIMINARIES ON CACHE

- The X86 cache
  - Memory architecture
  - Memory access latency:
    - DRAM >>> L3 cache >> L2 cache > L1 cache > Registers
    - Cache access is much faster than DRAM access
  - The cache exploits access localities
    - Temporal locality: a program will likely access the same memory address multiple times
    - Spatial locality: a program will likely access nearby memory addresses

PRELIMINARIES ON CACHE

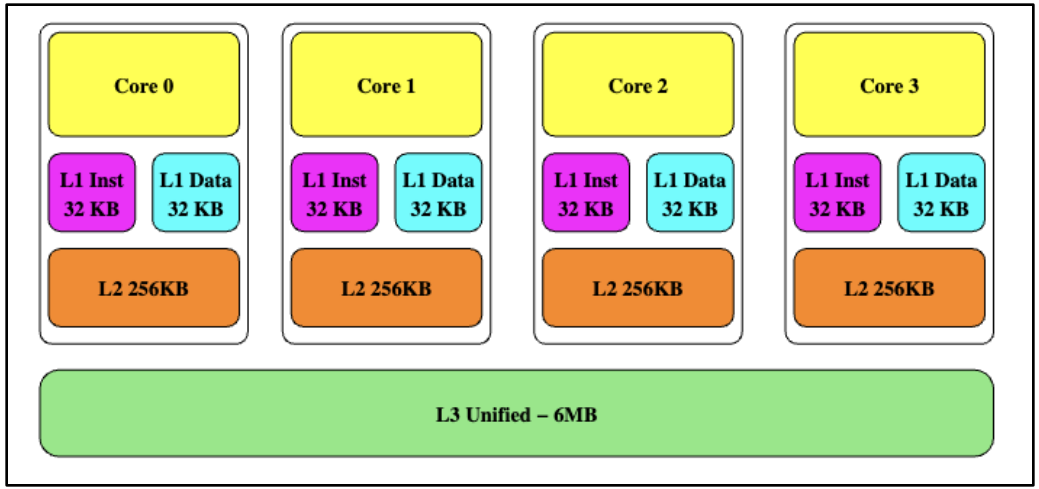
- The X86 cache
  - Memory architecture
  - Memory access latency:
    - DRAM >>> L3 cache >> L2 cache > L1 cache > Registers
    - Cache access is much faster than DRAM access
  - The cache exploits access localities
    - Temporal locality: a program will likely access the same memory address multiple times
    - Spatial locality: a program will likely access nearby memory addresses
    - x86 system:
      - Divides memory into blocks (= lines in cache)
      - Stores lines recently accessed by a program

PRELIMINARIES ON CACHE – SHARED BETWEEN CORES

- The X86 cache
  - Memory architecture
  - Memory access latency:
    - DRAM >>> L3 cache >> L2 cache > L1 cache > Registers
    - Cache access is much faster than DRAM access
  - The cache exploits access localities
    - Temporal locality: a program will likely access the same memory address multiple times
    - Spatial locality: a program will likely access nearby memory addresses
    - x86 system:
      - Divides memory into blocks (= lines in cache)
      - Stores lines recently accessed by a program
  - The last-layer-cache (LLC: L3) is shared across multiple cores
    - Improve the system performance
    - Think of a shared library in memory used by multiple programs

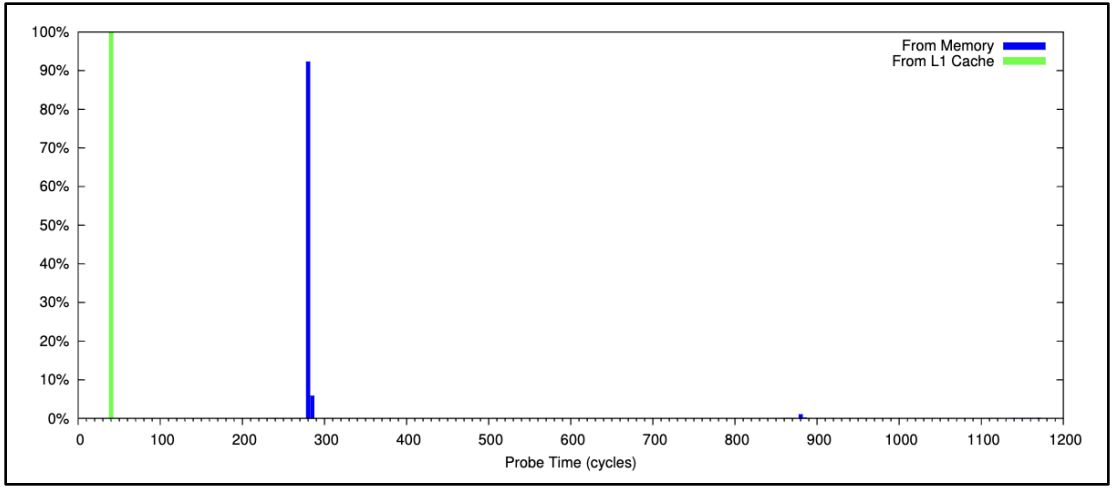
PRELIMINARIES ON CACHE – CONSISTENCY

- The X86 cache
  - Memory and cache are often in inconsistent states
  - In case of this cache conflict, system flushes the cache line
    - clflush: one can flush the cache line
    - Think about: what happens if one flushes a cache line intentionally



FLUSH+RELOAD TECHNIQUE

- Timing side-channel attack
  - Exploit the cache flush to leak information on the victim's memory access
  - Assumptions:
    - The victim and the attacker access the shared code (e.g., shared libraries)
    - The victim's process and the attacker's process can be on the same or in different cores
    - The attacker can flush the cache line intentionally

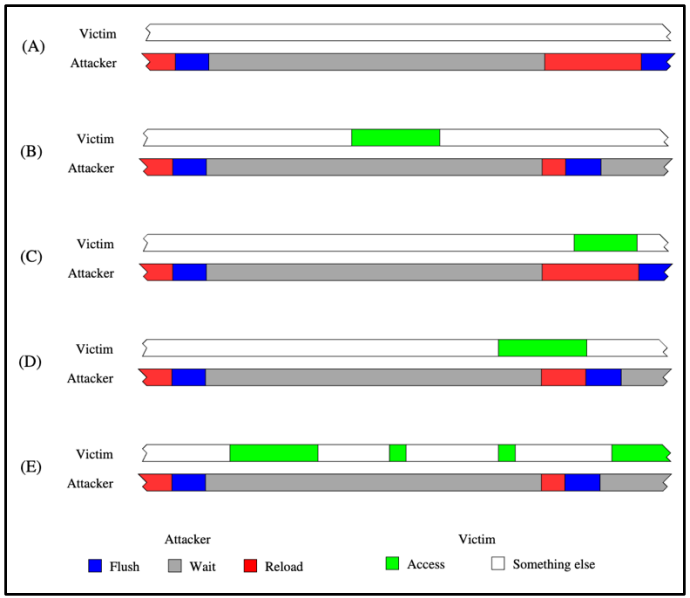


FLUSH+RELOAD TECHNIQUE

- Timing side-channel attack
  - Exploit the cache flush to leak information on the victim's memory access
  - Assumptions:
    - The victim and the attacker access the shared code (e.g., shared libraries)
    - The victim's process and the attacker's process can be on the same or in different cores
    - The attacker can flush the cache line intentionally
  - Flush+Reload procedure
    - Step 1: The attacker first flush the cache line (or lines)
    - Step 2: They will wait for a few cycles (e.g., 2000 CPU cycles)
    - Step 3: They will access the same cache line(s) again
    - Step 4: Measure the time it takes to load the data
      - Slow access: the data has not been accessed by the victim
      - Fast access : the data is accessed by the victim
    - Repeat Step 1-4 forever

FLUSH+RELOAD TECHNIQUE

- Timing side-channel attack
  - Flush+Reload procedure
    - Step 1: The attacker first flush the cache line (or lines)
    - Step 2: They will wait for a few cycles (e.g., 2000 CPU cycles)
    - Step 3: They will access the same cache line(s) again
    - Step 4: Measure the time it takes to load the data
      - Slow access: the data has not been accessed
      - Fast access : the data is accessed by the victim
    - Repeat Step 1-4 forever



FLUSH+RELOAD DEMONSTRATION – RSA

- RSA operation … why?

FLUSH+RELOAD DEMONSTRATION – RSA

- RSA operation
  - Public key: e N
  - Private key: d (that satisfies ed = 1)
  - To ciphertext: $C = M^e \bmod N$
  - To plaintext: $C^d \bmod N$
    - $(M^e)^d \bmod N$
    - $M^{ed} \bmod N$
    - M mod N (N is a really large prime, so mostly it's N)

FLUSH+RELOAD DEMONSTRATION – RSA

- RSA operation
  - Public key: e N
  - Private key: d (that satisfies ed = 1)
  - To ciphertext: $C = M^e \bmod N$
  - To plaintext: $C^d \bmod N$
    - $(M^e)^d \bmod N$
    - $M^{ed} \bmod N$
    - M mod N (N is a really large prime, so mostly it's N)
  - Implementation of the modular exponentiation
    - Square and multiply algorithm (see the right)
    - In here e is equal to d above
    - For clear bits: square – reduce
    - For set bits : square – reduce – multiply – reduce

Square and multiply algorithm

$x \leftarrow 1$
for $i \leftarrow |e|\text{-}1$ **downto** 0 do
  $x \leftarrow x^2 \bmod n$
  if $(e_i = 1)$ **then**
    $x = xb \bmod n$
  endif
done
return x

FLUSH+RELOAD DEMONSTRATION – GNUPG

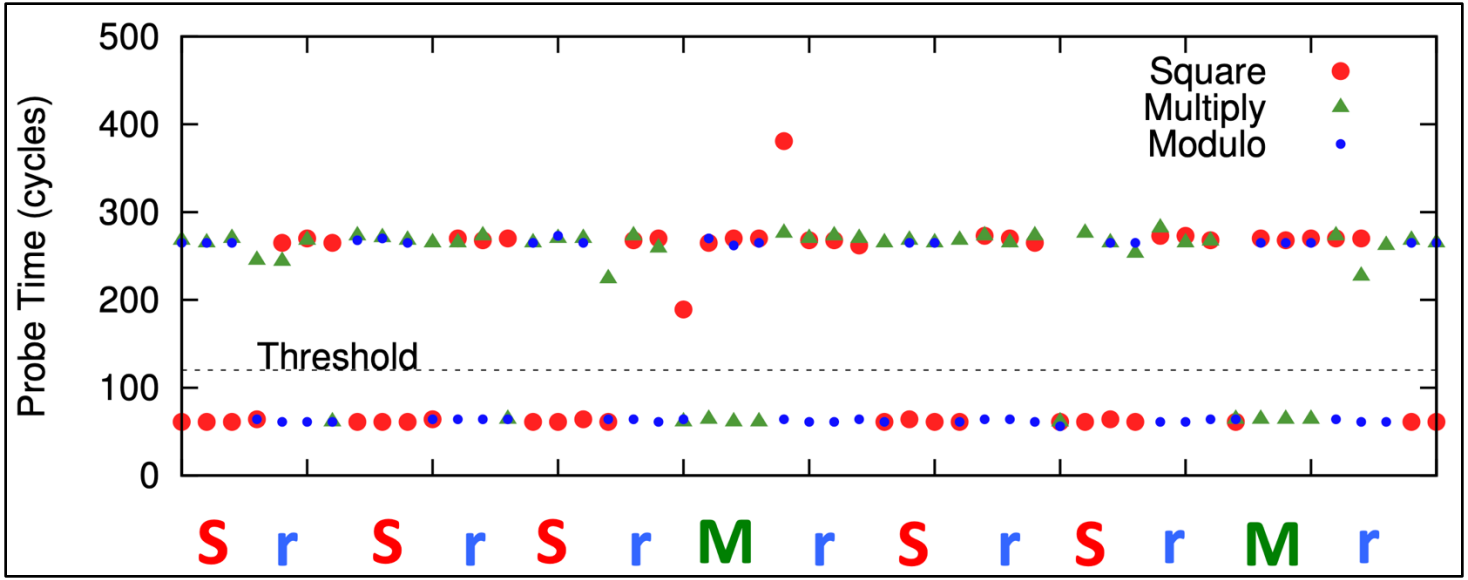
- GnuPG (GPG) operation … why?

FLUSH+RELOAD DEMONSTRATION – GNUPG

- GnuPG (GPG) operation
  - GPG uses the RSA algorithm
    - Encryption and digital signatures
    - 0 bit: square – reduce
    - 1 bit: square – reduce – multiply – reduce

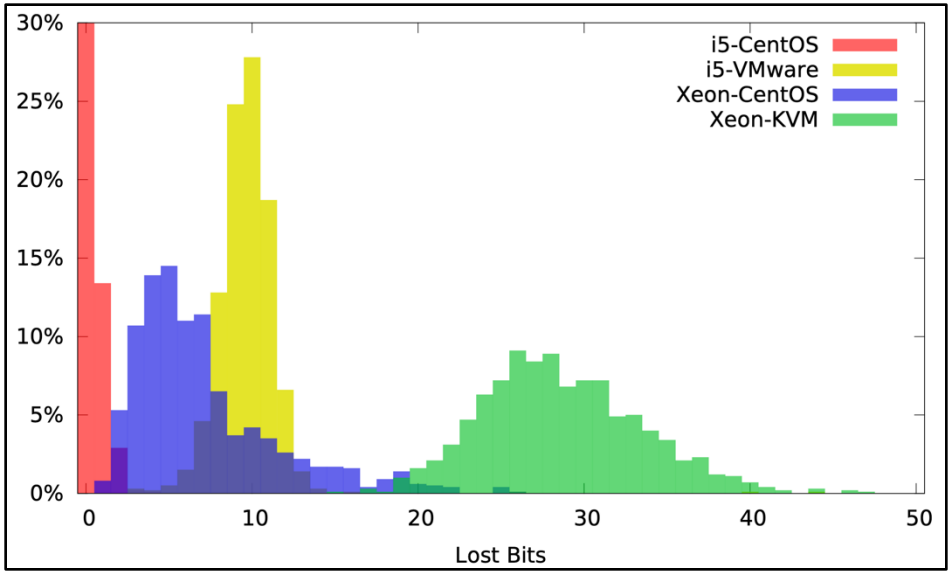
FLUSH+RELOAD DEMONSTRATION – GNUPG

- GnuPG (GPG) operation
  - Run Flush+Reload
    - Extract the sequence of operations of the modular exponentiation
    - Each Flush+Reload attempt is 2048 cycle – reconnaissance



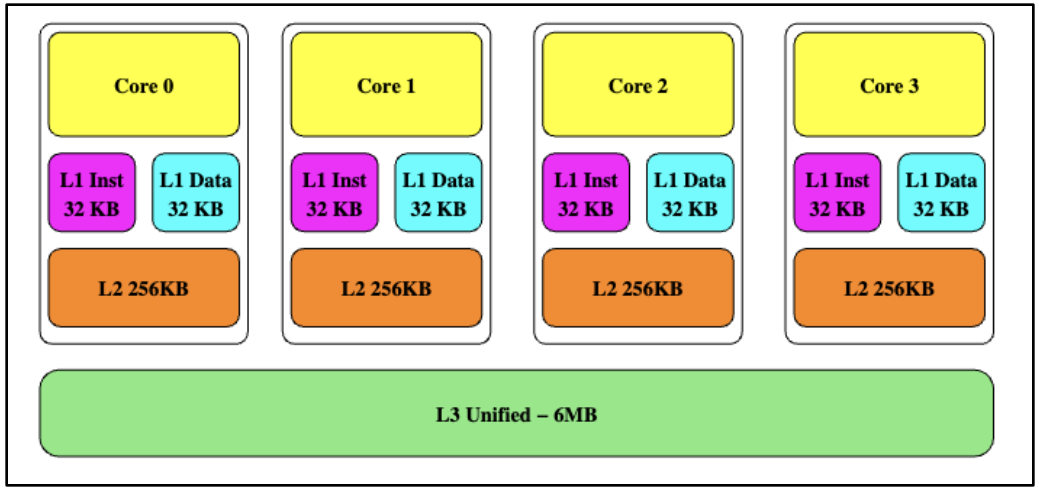
FLUSH+RELOAD DEMONSTRATION – GNUPG

- GnuPG (GPG) operation
  - Run Flush+Reload
    - Extract the sequence of operations of the modular exponentiation
    - Each Flush+Reload attempt is 2048 cycle – reconnaissance
    - The attack can have false negatives



FLUSH+RELOAD IMPLICATIONS

- Cross-VM attacks
  - The attacker and the victim VMs are on the same host, but on different cores
  - The attacker VM can spy the behaviors of the victim
    - If the attacker knows what software libraries used by the victim
    - It's easier to do that for commodity software



FLUSH+RELOAD MITIGATIONS

- Countermeasures
  - Hardware-based non-inclusive cache
  - Disable page sharing in the OS
  - Obfuscation
  - Software-level diversification

Thank You!

Sanghyun Hong

https://secure-ai.systems/courses/Sec-Grad/current

SAIL
Secure AI Systems Lab