Covert and Side Channels

Robert Brotzman-Smith

Side Channels

PennState

Side Channels

- Side-channel attacks extract information by observing implementations on systems
- These attacks do not rely on code vulnerability
  - Ex) Buffer overflows, SQL injection, etc.
- Do not rely on theoretical weaknesses of algorithms

Example Side Channel

- Timing
- CPU Cache
- Power Usage
- Electromagnetic field
- Acoustic
- Thermal
- Speculation

Timing Side Channels

Timing Side Channels

- Timing side channels work by observing how long a task takes to complete
- They obtain information when an algorithm takes different amounts of time to execute depending on the inputs
- This is particularly problematic when the execution time depends on secret data
- Can be exceptionally dangerous since they do not require the adversary and victim to necessarily share resources

Real Timing Attack

- Example is from libgcrypt which is a common cryptographic library
  - Implements modular exponentiation
  - Commonly used in RSA and ElGamal
- Essentially the algorithm will square and take the modulus every time
- When the current bit is set it will also multiply by the base and again apply the modulus
  - Note that the key here is the exponent
- Notice that based on the key's value the then branch of the if statement is executed
- Thus every time a bit is set in the key, that loop iteration will take more time to execute
  - This can leak many bits of the key quickly

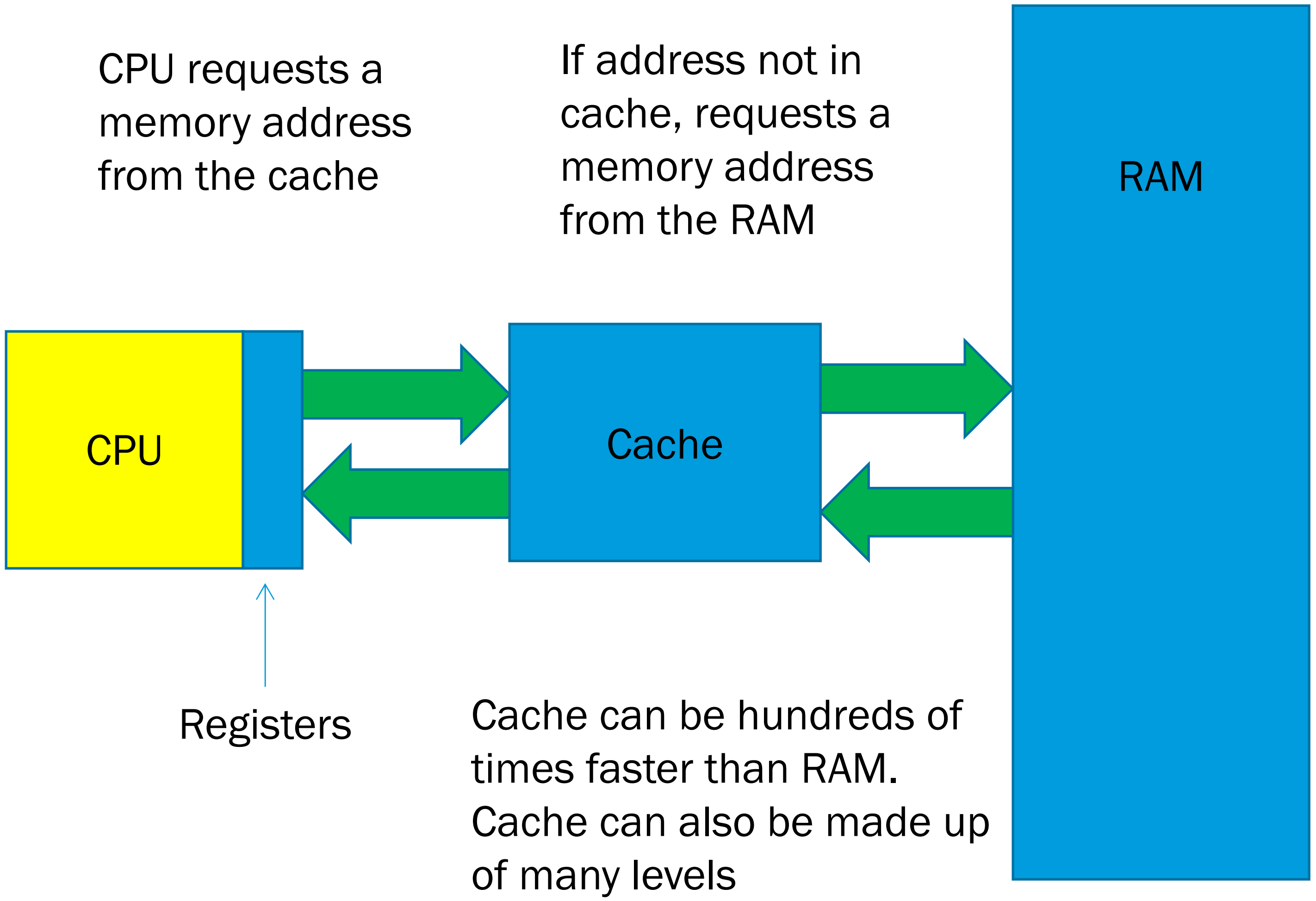
```
void squareNMultiply ()
{
        // details omitted for  brevity
    while (c)
    {
        res = res * res;
        res = res % mod;
        if (((1 << 31 ) & key) != 0)
        {
            temp = res * base;
            temp = temp % mod;
            res = temp;
        }

        key <<= 1;
        c–;
    }
```

Other Timing Attacks

- Not all attacks need a secret dependent branch to cause timing differences
- Some instructions will take different amounts of time to execute based on their operands
  - Ex) division
- More timing variation can occur based on where data is located in the memory hierarchy
  - i.e. registers, cache, ram, disk

Cache Side Channels

CPU Cache Attacks: Preliminaries



CPU requests a memory address from the cache

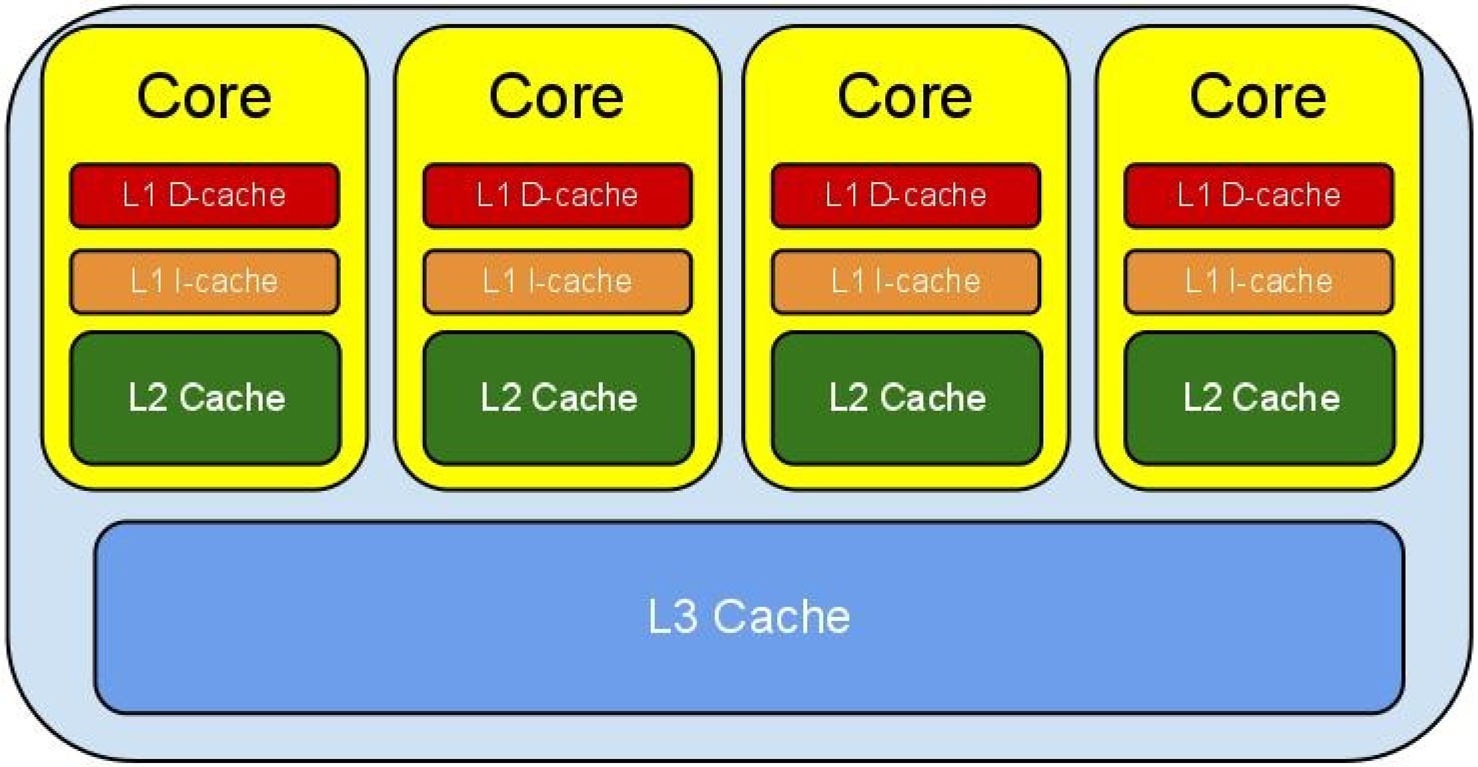
If address not in cache, requests a memory address from the RAM

Registers

Cache can be hundreds of times faster than RAM. Cache can also be made up of many levels

CPU Cache Attacks: Preliminaries

- CPU caches are also broken into slower and faster memory
  - L1 faster than L2 faster than L3
- CPU caches store both data and instructions



PennState

CPU Cache Attacks: Preliminaries

- Modern CPU caches are typically N-way set associative
  - Opposed to being directly mapped or fully associative
- In N-way set associative caches, there are cache size/N sets
  - Ex) 32 kb 8-way cache will have ~4000 cache sets
- Memory can be mapped to one cache set and the processor will apply a replacement policy to each cache set
  - Ex) least recently used, pseudo least recently used, not most recently used
- The replacement policy is typically what allows adversaries to learn information through a side channel
  - This is because most commercial processors use a replacement policy that is related to recent program behavior

CPU Cache Attacks

- Cache side-side channel attacks leverage the state of the cache to infer sensitive data used during program execution
  - State refers to memory addresses present in the CPU cache
  - The key insight is that as programs execute the state of the cache is constantly being updated
- Since CPU caches are very fast, these side channels can leak large amounts of data quickly
  - 100's of kilobytes/second
- Cache side channel attacks are often categorized into three cases
  - Time
  - Access
  - Trace

Side Channel Categories

- Time
  - Adversary is able to observe the total execution time of some target piece of code
  - Can be launched remotely
  - Leaks the least amount of information
- Access
  - Adversary is able to determine whether certain memory addresses are cached
  - Requires shared cache with victim
  - Leakage is limited by how long it takes to probe memory addresses
- Trace
  - Adversary knows the order memory addresses are cached
  - Requires shared cache with victim
  - Fine grained traces are difficult to achieve
  - Usually used to analyze countermeasures

CPU Cache Side Channel Overview

CPU cache improves performance by storing recently used data in fast memory

Information about recent program execution is in the cache state

Cache side channel attacks infer what data is in the cache

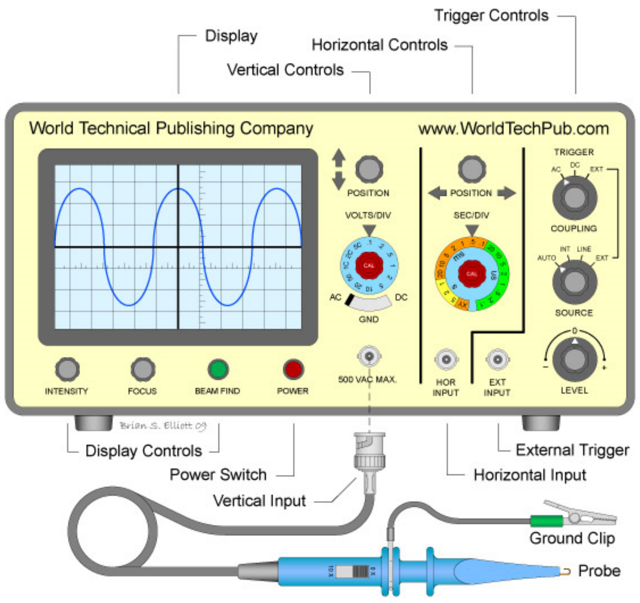
Determining What is Cached

- Many access based attacks to determine the cache state
  - Flush+Reload
  - Flush+Flush
  - Prime+Probe
- The goal of each one is to determine whether or not a set of memory addresses has ben accessed by a victim process
  - Usually the target memory locations will be related to some sensitive data used by the process

Power Side Channels

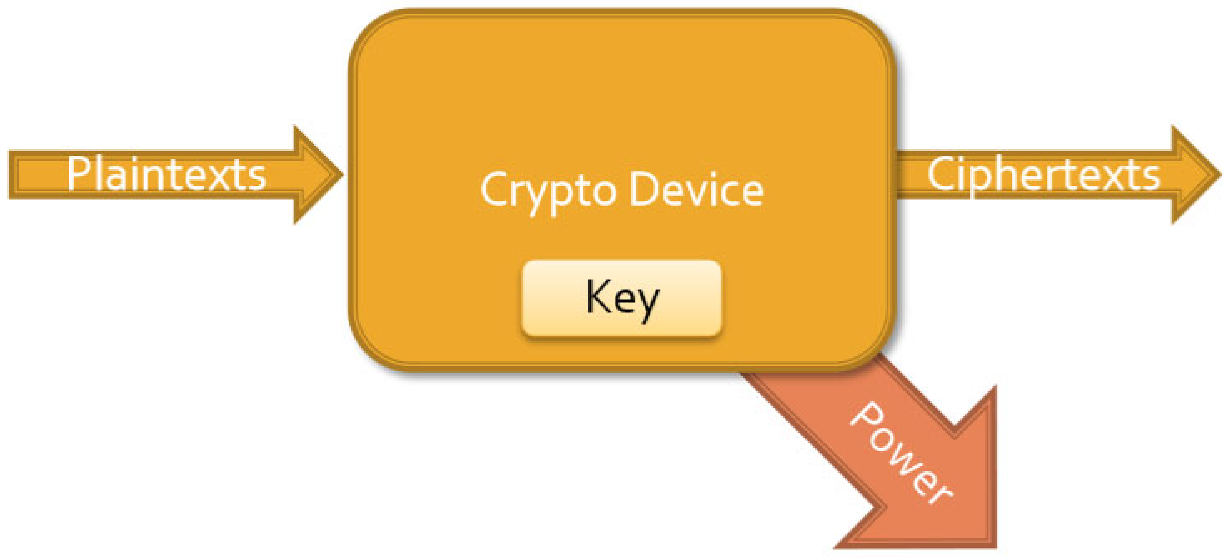
Power Side Channel

- Power side channels leverage fluctuations in a machine's power usage when performing different operations
  - Power usage depends on what instruction is being executed
  - Also can depend on the operands of the instructions
- Typically an adversary will require physical access to the machine to measure power usage
  - Uses oscilloscope to measure voltage



Power Side Channel Attack Model

- Adversary may control the cipher or plaintexts
- Goal is to learn the key used by the device doing the encryption/decryption

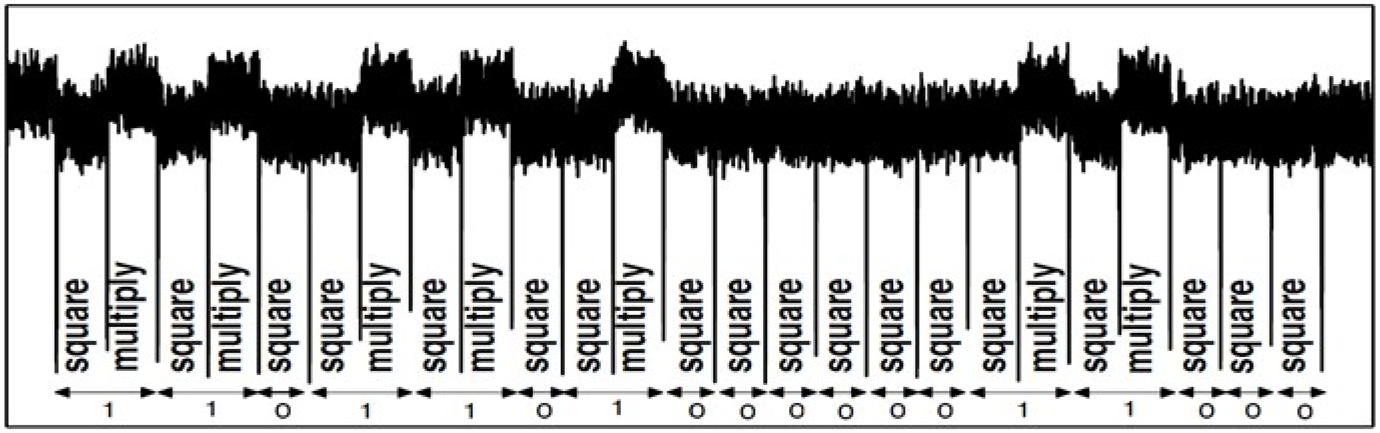


Power Analysis Types

- It is commonly divided into three categories
  - Simple Power Analysis (SPA)
    - Adversary can learn information by visually looking at power trace
  - Differential Power Analysis (DPA)
    - Adversary applies statistical techniques to learn information
      - Ex) Difference of means, Correlation, error correction, etc
  - High-Order Differential Power analysis (HO-DPA)
    - Considers data from multiple source simultaneously
    - Data must be synchronized by time

Simple Power Analysis

- Example shows square-and-multiply algorithm used to compute modular exponentiation



- We can clearly see the extracted key here is: 110110100000011000
- This approach can be used to learn a key of arbitrary length

Speculation Side Channels

Out-of-Order & Speculative Execution

- Out-of-order execution happens anytime another instruction is executed before previous instruction(s) are retired
- Speculative execution happens when the result of a branch is unknown and execution proceeds down a guessed branch
- Both of these optimizations help to maximize the CPU resources

Transient Instructions

- Any instruction which can be executed out of order and leaves measurable side effects
  - Ex) memory loads/stores
- These instructions are key to create any kind of side channel or covert channel

Branch Prediction

- Modern processors use heuristics to improve their guess of a branch value before it is known
- The processor will then execute instructions assuming the guess is correct
- If the guess is correct nothing needs to be done
- If the guess is wrong, the processor needs to roll back any changes it made to the program's state
  - This does not include everything such as the cache state
- The branch predictor is often shared between processes
  - Allowing an adversary to train the branch predictor from their process

Spectre

1

Trains branch predictor to execute a branch not taken by normal execution

2

Speculatively execute instruction(s) which reveal some sensitive data

3

Use side channel to recover sensitive data

- Flush+Reload
- Evict+Reload

PennState

Spectre Variants

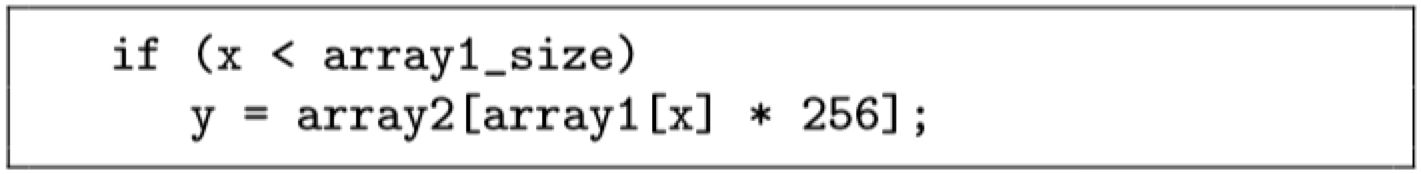
Exploit conditional branches

- Influence branch predictor to incorrectly guess result of condition
- CVE-2017-5753

Exploit indirect branches

- Train predictor that an indirect branch will execute an attacker specified gadget
- CVE-2017-5715

Spectre Exploiting Conditional Branches

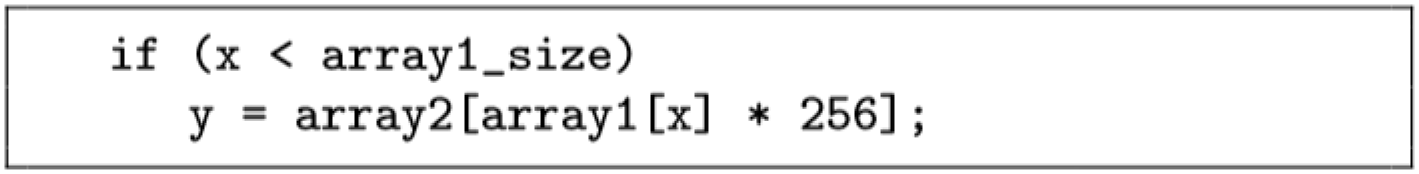


```
if (x < array1_size)
    y = array2[array1[x] * 256];
```

Listing 1: Conditional Branch Example

- Assumptions:
  - x is chosen by the attacker and can read anything in memory (possibly a secret value)
  - Array1_size is not in the cache
  - The branch predictor has been trained to predict true for the branch condition

Spectre Exploiting Conditional Branches



```
if (x < array1_size)
    y = array2[array1[x] * 256];
```

Listing 1: Conditional Branch Example

- The Attack:
  1) Since array1_size is not cached and branch predictor guesses true, the true branch will begin execution
  2) x was chosen by attacker and reads secret byte k
  3) Array2 will then access its (k*256)th element
  4) Attacker then checks to see what element from array 2 was accessed via a side channel attack from another process

Meltdown

1

Attacker identifies address in kernel space to read

2

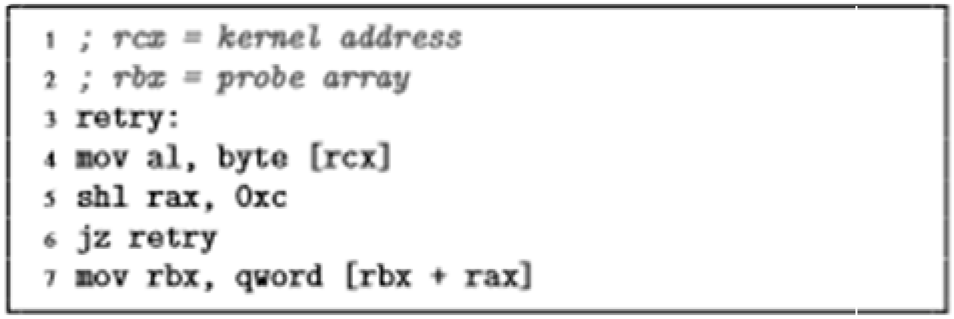
Transient instructions are used to force address into the cache

3

Another process observes the cache line used by the transient instruction(s)

PennState

Meltdown’s Core



```
1 ; rcx = kernel address
2 ; rbx = probe array
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```

- Line 4 reads a byte from the kernel address space
- Line 5 improves throughput of covert channel by multiplying kernel data by 4096
    - Ensures each possible value read will be placed in a separate page
- Line 6 keeps performing the attack until something is read
- Line 7 is the transient instruction which modifies the cache
    - It will probably get executed due to out of order execution
- *Note that this instruction sequence can be placed into a transaction avoiding the exception from being raised (but the effect on the cache will persist)

Side-Channel mitigations

Timing Channel Mitigations

- Randomize control flow
- Insert random noise
- Padding
  - Adds instructions to control flow paths to make them uniform
- Constant time algorithms
  - Requires code rewriting
- Constant time instructions
  - Means all instructions take the same amount of time
  - Requires all instructions to take the same amount of time as the slowest instruction

Cache Side-Channel Mitigation

- Preload/Pin data to the cache
  - Prevents detectable changes to the cache state
- Write side channel resistant code
  - Requires code rewriting and knowledge regarding how to avoid side channels
- Cache partitioning
  - Each processes gets its own part of the cache
- Random cache accesses
  - Makes it more difficult to distinguish legitimate cache accesses

Power Side-Channel Mitigation

- Make timing synchronization more difficult
  - Modulate cpu frequency
  - Add delays to the program execution
  - Randomize the execution of the program
- Make power usage uniform
- Add noise to power consumption
- Change cryptographic keys often
  - Power analysis often requires many traces

Side Channel Detection

Side Channel Detection

- Detecting side channel attacks in progress is important
- Allows further defensive action to be taken
  - Such as isolating the malicious process or killing it
- Particularly important in cloud environments
  - Since many users will share the same hardware

How to Detect Cache Side Channels

- Most state-of-the-art tools use performance monitors
- These monitors keep track of various metrics
  - L1 cache hits/misses, l2 cache hits/misses, cycles, etc
- Tools typically look for abnormal cache hit/miss rates
- Recent work has shown that using transactional memory can also be used to detect cache-based side channels

Identifying Side Channels in Programs

- Identifying side channels in software can be a challenging problem
- Particularly because code that looks seemingly innocuous can leak large amounts of data
- Recall the AES example
- Simply making a memory access dependent on a secret key can leak half of an encryption key
- How can we identify code that potentially reveals information ahead of time?

Overall

- Cache aware symbolic execution can be used to identify cache-based side channels
- It will indicate the exact line of code causing the side channel and the kind of side channel
  - Ex) either key dependent branch or array access
- Allows users to iteratively remove the side channel and then check again and see if there are more side channels to fix
- Guarantees that if no cache-based side channels are reported that none are possible
  - i.e. the analysis is sound

Acknowledgements

- [1] http://www.cs.tau.ac.il/~tromer/istvr1516-files/lecture3-power-analysis.pdf
- [2] https://en.wikipedia.org/wiki/Oscilloscope#/media/File:WTPC_Oscilloscope-1.jpg
- [3] Steganography and Covert Channels, K. Reiland, W. Oblitey, S. Ezekiel, J. Wolfe
- [4] https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwjJtKvnw8fgAhXMnOAKHU6AC0wQFjAAegQICRAC&url=https%3A%2F%2Fwww.cs.clemson.edu%2Fcourse%2Fcpsc420%2Fpresentations%2FSpring2007%2FCovert%2520Channels.ppt&usg=AOvVaw0FhHuxgRjmuXZmHE5GWNUc
- [5] Topics in Cryptography:Lecture 7, Moni Naor
- https://incoherency.co.uk/image-steganography/

PennState