Basic Computer Security Concepts

Chapters 1,2

[1] J. Szefer, “Principles of secure processor architecture design,” *Synth. Lect. Comput. Archit.*, vol. 13, no. 3, pp. 1–173, 2018.

Security Is a Major Concern

- Software complexity and bugs
  - **Larger software** bases increases the vulnerability (because bug density is constant).
  - OS and hypervisors, although isolates partially the problem, suffer for the same problem (even hypervisors are quite complex today due to performance enhancements)
  - Isolate trusted code from untrusted one (with certain hardware features)
- Side-channel Attacks
  - **Cloud computing** favors co-residency/multi-tenancy
  - Somehow, infer other "apps" internal data is more dangerous
- Physical Attacks
  - Device electrical **probing** can leak valuable information
  - Cloud infrastructure is vulnerable to these kind of attacks
- Humans

Need For Secure Processor Architecture

- Software Size
  - OS is ~Millions of code lines (but **hypervisor too**!)
    - https://www.openhub.net/: Linux ~20M, but **Xen ~1M**, **qemu ~1.5M**, **libbirt ~1M**, etc….)
  - ~20 Bugs per 1000 lines of code. The probability of compromise in-place protection mechanisms (OS or Hypervisor) is high

- Side-Attack Channels
  - **Speculative Execution** might introduce bugs (Meltdown) or just open the door (Spectre)
  - For each performance "trick" at least **a time-based side-channel** attacks (BP, prefetchers, caches, etc…). Can't being disabled

- Physical Access
  - Unable to determine is someone is tampering with the system (behind stage), for example rogue employees or government compelled companies can probe our server
  - In a VMS is trivial but also possible in **Bare Metal hosted system** (v.gr. cold attacks)
  - **IoT** are also susceptible of being tampered (without being aware of it) due to remote location or **Garden Walls**

Trusted Computing Base (TCB) (I)

- HW + SW that work **together** to provide some security guarantees (as expected by the hardware)
  - "Software" means OS or hypervisor (and as small as possible)
  - Hardware components are exposed (via ISA) to achieve the desired guarantees (but software can ignore them)
  - Secure processor architectures goal is to ensure that trusted hardware components are available to the software
- Some parts (HW+SW) in the same system might be **untrusted**
  - Can be exploited by malicious attacks but can't "**penetrate**" TCB portion
- Any modification or **bugs in the TCB** parts might break the security guarantees
  - Trustworthiness is a qualitative designation indicating whether the entity will behave as expected, is free of bugs and vulnerabilities, and is not malicious
- Exhaustive techniques, such as **formal verification**, will be used to guarantee TCB

Trusted Computing Base (TCB) (II)

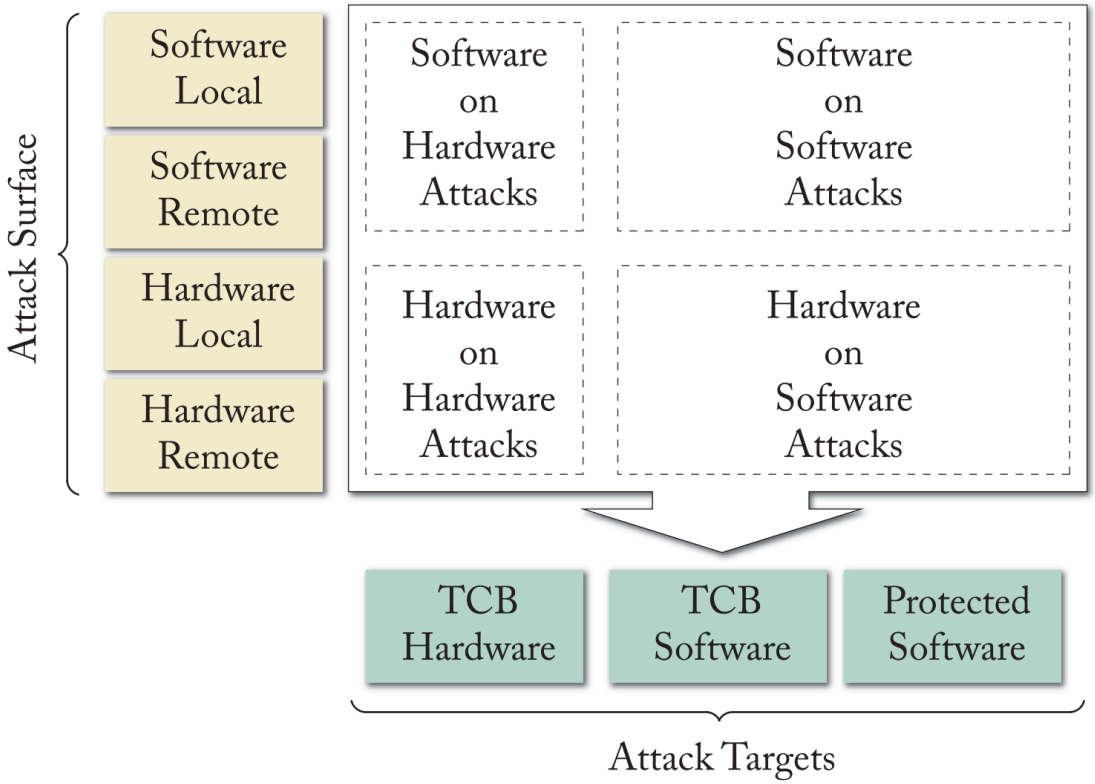
- Architecture designers should ensure that the information exchange between of the components of the system cannot be attacked

- But beyond design hardware trojan can be added during the manufacturing time
  - Supply chain security
  - Foundry malicious behavior detection mechanisms

- Avoid security via obscurity (Kerckhoffs' principle)
  - Design details should be public
  - The only secret are the cryptographic keys to guarantee secure information exchange between components

Security Threats to a Systems: The attack surface

- Combinations of all attack vectors that can be used against a system
  - Non TCB internal hardware or software
  - External hardware (v.gr. cold attacks on memory, physical probing, …)
  - External software (v.gr. Use network services via buffer overflow…)



Security Threats to a Systems: Passive and Active Attacks

- In **passive** attacks only observe the behavior of the system to deduce some information about it
  - **Externally** e.g., EM emission, power consumption, etc…
  - **Internally** e.g., Access to performance monitoring unit (PMU) to deduce TCB protected information (via side-channel attacks)

- In **active** attacks, the attacker will try to modify system behavior (v.gr., Injecting changes in some memory location, usually instructions). Also inject fault in the hardware
  - **Spoofing**: e.g., inject memory commands to read or write protected memory
  - **Splicing**: e.g., mix correct and malicious memory commands
  - **Replay**: e.g., capture and resent memory commands
  - **Disturbance**: e.g., DDoS or repeated access memory location to propagate changes between DRAM Rows (*Rowhammer*)

Security Threats to a Systems

- **Man-in-the-middle Attacks**
  - Intercept, copy and retransmit sensible information (without mangling the information: (passive), or changing (active)
- **Cover Channels**
  - Is a communication channel that was not designed to transfer information
  - Timing, power, thermal, eletro-magnetic emanations, acoustic emanations,...
  - If known at at design time can be prevented (v.gr. cache isolation between process avoids timing). Others require physical access to the system under attack
- **Side Channels**
  - Like cover but the sender does not intend to communicate information to the receiver (but elsewhere)
  - The leaked information is a side effect of the implementation and how HW and SW is used
  - Sender and receiver under control of the attacker to build a "cover" attack
- **Attack bandwidth**
  - Cover and side channel mechanisms, in most cases, are stochastic: extraction rate (bandwidth) depends upon its severity

The Threat Model

- Concise specification of the threats that a given secure processor architecture protects against (can't be anything)
  - TCB: set of trusted hardware and software components
  - Security properties that the TCB aims to guarantee
  - Capabilities of the potential attackers
  - Potential vulnerabilities to address

- Threat to Hardware after the Design Phase
  - Bugs or Vulnerabilities in the TCB
  - Hardware trojans and supply chain attacks
  - Physical Probing and Invasive Attacks

Basic Security Concepts

- **Confidentiality** is the prevention of the disclosure of secret or sensitive information to unauthorized users or entities.
  - If attacker find a breach, sensitive information can leak
  - Can be using side/cover or brute force (e.g., password crackers)
- **Integrity** is the prevention of unauthorized modification of protected information without detection
  - Attacks are focused to bypass integrity checks to gain access to the system (v.gr. bit *s* in certain executable)
- **Availability** is the provision of services and systems to legitimate users when requested or needed
  - Attacks focused on render the system unusable to everybody else (v.gr. DDoS attack)

- Authentication relates to determining who a user or system is

- Freshness: update the authentication requirements across time. Aggregate a nonce/salt in the cypher (v.gr. a counter)

Symmetric-Key Cryptography

- Encryption and decryption uses the same key
  - The encrypted ciphertext looks random to anyone without the key
- Block Cyphers
  - Algorithms that produce the ciphertext from 1 blocks of information (AES uses 16bytes).
  - Information with multiple blocks should be handled blocks: the same block produce the same results (ECB) or change the results according the position of the block in the text
  - Block Cyphers only provide confidentiality: need additional mechanism to provide integrity: hashing (on top of encryption or combined)
- Stream Cyphers
  - Encrypt the text bit by bit with xor pseudo-randomly generated keystreams
  - Faster in hardware but require to process the whole text to access to a part
- Algorithms
  - Block: Old algorithms are unsecure (3DES, DES, RC4). Use AES with a strong key (128-256bits)
  - Stream: ChaCha

Public-Key (asymmetric) Cryptography

- Key do Encrypt is **public** (pk) != key to Decrypt is **secret** (sk)
  - We only need to interchange pk
  - *sk* cannot be inferred from *pk* (depends on hardness of certain mathematic problems, such as large number factorization)
  - In reverse direction (encrypt with *sk* and decrypt with *pk*), can used for integrity (digital signatures or message authentication codes)
- Key encapsulation mechanism
  - Public-key encryption is computationally expensive (if data to transmit is large): use public-key cryptography to interchange a symmetric encryption key
- Post-Quantum Cryptography (**Grover's algorithm**)
  - Algorithms used in current PK, such as RSA, might be computationally vulnerable against brute-force attacks using quantum-computers (theoretically).

Cryptographic Tools

- Randoms Generators
  - PRNG: sequence is deterministic (given a seed)
  - TRNG: using noise (hardware)
- Secure Hashing
  - One way function (mathematically noninvertible) to compute a digest or fingerprint from data
  - Commonly used for integrity checking and authentication
    - In message authentication codes (MAC). Only the entity with the correct cryptographic key can generate/check the hash value
    - Digital signature using the private key to generate the signatures (and public key to authenticate)
    - Hash trees. In large pools of data speed up the hash computation after altering a part (leaves of a binary tree). Nodes are hashes.
  - Most common is SHA-2 or SHA-3 (old SHA1, MD5, MD4 are not secure against current computational power)

Public Key Infrastructure (PKI)

- PKI is a set of policies and protocols for managing, storing, and distributing certificates used in public-key encryption
  - There is a trusted third party (Certificate Authority) which can distribute digital certificates that vouch for correctness of public keys
  - Certificates for the certificate authorities are usually pre-distributed (e.g., browsers come with built-in list of certificates for certificate authorities
- Digital certificates
  - Contains some identifying information about a system or a user and their public key.
  - This information is encrypted with a private key of the certificate authority
  - With the public keys of the certificate authorities is used to check authenticity and content of the certificate
- PKI is used in secure processor in combination with physical unclonable functions (PUF)
  - Depends upon random effects during the manufacturing process
  - Allows Direct Anonymous Attestation (DAA): allows remote authentication of trusted computer (to form a network of TCB) without compromising privacy of the platform

Outline

- Secure Processors Architecture
- Trusted Execution Environments
- Hardware Root of Trust
- Multi-core Protections
- Side-channel Threats and Protections
- Principles of Secure Processor Architecture Design

Warmup: Readings

Meltdown: Reading Kernel Memory from User Space

Moritz Lipp[1], Michael Schwarz[1], Daniel Gruss[1], Thomas Prescher[2],
Werner Haas[2], Anders Fogh[3], Jann Horn[4], Stefan Mangard[1],
Paul Kocher[5], Daniel Genkin[6,9], Yuval Yarom[7], Mike Hamburg[8]
[1]*Graz University of Technology,* [2]*Cyberus Technology GmbH,*
[3]*G-Data Advanced Analytics,* [4]*Google Project Zero,*
[5]*Independent (www.paulkocher.com),* [6]*University of Michigan,*
[7]*University of Adelaide & Data61,* [8]*Rambus, Cryptography Research Division*

Abstract

The security of computer systems fundamentally relies on memory isolation, e.g., kernel address ranges are marked as non-accessible and are protected from user access. In this paper, we present Meltdown. Meltdown exploits side effects of out-of-order execution on modern processors to read arbitrary kernel-memory locations

sor bit of the processor that defines whether a memory page of the kernel can be accessed or not. The basic idea is that this bit can only be set when entering kernel code and it is cleared when switching to user processes. This hardware feature allows operating systems to map the kernel into the address space of every process and to have very efficient transitions from the user process to the kernel, e.g., for interrupt handling. Consequently

Spectre Attacks: Exploiting Speculative Execution

Paul Kocher[1], Jann Horn[2], Anders Fogh[3], Daniel Genkin[4],
Daniel Gruss[5], Werner Haas[6], Mike Hamburg[7], Moritz Lipp[5],
Stefan Mangard[5], Thomas Prescher[6], Michael Schwarz[5], Yuval Yarom[8]
[1] Independent (www.paulkocher.com), [2] Google Project Zero,
[3] G DATA Advanced Analytics, [4] University of Pennsylvania and University of Maryland,
[5] Graz University of Technology, [6] Cyberus Technology,
[7] Rambus, Cryptography Research Division, [8] University of Adelaide and Data61

***Abstract*—Modern processors use branch prediction and speculative execution to maximize performance. For example, if the destination of a branch depends on a memory value that is in the process of being read, CPUs will try to guess the destination and attempt to execute ahead. When the memory value finally arrives, the CPU either discards or commits the speculative computation. Speculative logic is unfaithful in how it executes, can access the victim's memory and registers, and can perform operations with measurable side effects.**

Spectre attacks involve inducing a victim to speculatively perform operations that would not occur during correct program execution and which leak the victim's confidential information via a side channel to the adversary. This paper describes practical attacks that combine methodology from side channel attacks, fault attacks, and return-oriented programming that can read arbitrary memory from the victim's process. More broadly, the paper shows that speculative execution implementations violate the security assumptions underpinning numerous software security mechanisms, including operating system process separation, containerization, just-in-time (JIT) compilation, and countermeasures to cache timing and side-channel attacks. These attacks represent a serious threat to actual systems since vulnerable

leverage hardware vulnerabilities to leak sensitive information. Attacks of the latter type include microarchitectural attacks exploiting cache timing [8, 30, 48, 52, 55, 69, 74], branch prediction history [1, 2], branch target buffers [14, 44] or open DRAM rows [56]. Software-based techniques have also been used to mount fault attacks that alter physical memory [39] or internal CPU values [65].

Several microarchitectural design techniques have facilitated the increase in processor speed over the past decades. One such advancement is speculative execution, which is widely used to increase performance and involves having the CPU guess likely future execution directions and prematurely execute instructions on these paths. More specifically, consider an example where the program's control flow depends on an uncached value located in external physical memory. As this memory is much slower than the CPU, it often takes several hundred clock cycles before the value becomes known. Rather than wasting these cycles by idling, the CPU attempts to guess

https://meltdownattack.com/