Lecture Notes

Chapter 3:
Secure Processor Architectures

CYENG 225: Microcontroller Essentials for Cyber Applications

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)

Chapter 3 Overview

- **Major Items**
  - Introduction on the main features of secure processor architectures, as an extension of general processor architectures.
  - Examples of real-world attacks on existing processor architectures. → Motivation for research and design of secure architectures.
  - Discussion on features of general-processor architectures.
  - Introduction on secure processor architectures, their features, especially concerning different execution levels and privileges.
  - Examples of existing academic and commercial architectures.
  - Lists for secure processor architecture assumptions and limitations of what the architectures can achieve.

Real-World Attacks

- These **real-world attacks** further motivate the need for secure processor architectures, and also give a warning of how wrong assumptions about the hardware behavior (e.g., the decay of DRAM after power is turned off), or unintended consequences of performance improving features (e.g., speculative execution), can result in vulnerabilities and attacks.
- As secure processor architectures are built upon normal processors, the same bugs and vulnerabilities can affect them as well, and should be guarded against.
- These vulnerabilities are in addition to any intentional hardware trojans or modifications to the system.
- **The processor vulnerabilities can be in the ISA, microarchitecture, circuits, or devices, and they can be used to break the system or bypass protections offered by the system**.
- The attacks range from ones that simply crash the system, to attacks allowing one to steal data from different processes, kernel, or form other VMs.
- Some of the attacks can be deployed remotely without physical access to the system, only requiring the attacker to run some code on the target machine, while other attacks require physical presence, e.g., to probe the circuits or remove DRAM chips.
- Of the security-related hardware bugs or vulnerabilities, the recent and most well-known ones are the **Coldboot attack** and the **Rowhammer attack** (both affecting DRAM) along with Spectre and Meltdown.
- These vulnerabilities and resulting attacks exemplify the range of threats that processors face, from ones requiring physical access to cool down the memories to later steal data, to ones that can be executed remotely on cloud computing servers.
- They also show that any component in the computer system can be vulnerable, e.g., a problem with the DRAM may be just as damaging as problem with the main processor chip itself.

Real-World Attacks – Coldboot

- A **Coldboot attack** can be used to steal information from DRAM when the system is powered off.
- **Coldboot exploits physical phenomenon that data stored in DRAM does not disappear as soon as the power is turned off**.
- Rather, **with the DRAM refresh disabled**, or **the power all-together turned off**, **the charges on capacitors in the DRAM cells (which are used to store the data) slowly decay**.
- Thus, **the basic assumption that DRAM is a volatile memory that loses contents instantly when powered off is not true**.
- In the Coldboot attack, researchers have shown that data, such as encryption keys, can be extracted from DRAM chips after computer is powered off.
- To extend the amount of time available before charges in the capacitors decay, DRAM chips can be easily cooled down with a can of compressed air spray (kind of used to clean computer keyboards or other electronics from dust).
- Cooling DRAM further slows down the decay, allowing one to remove a DRAM module from the computer, transfer it to another computer and dump the data.
- Alternatively, a computer can be quickly shut down while DRAM is cooled, and rebooted into a malicious OS that reads off the DRAM data before it could have decayed.
- A variety of solutions can be used to protect against Coldboot attack, but all **require extra software or hardware to effectively erase data**, rather than wait and assume the data will be lost due to the DRAM cell decay.
- In software, secret keys need to be explicitly erased.
- In hardware, battery-backed DRAM could use stored energy from the batteries to explicitly zero out the memory contents when external power is lost or refresh is disabled.

Real-World Attacks – Rowhammer

- A **Rowhammer attack** can be used to alter bits in memory locations not accessible to the attacker process or application.
- Rowhammer is a different vulnerability of the DRAM, but one which is also related to how data is stored as charges on capacitors in DRAM.
- Most computer systems rely on isolation to separate programs or VMs from one another.
- The isolation is enforced through page tables or other mechanisms for checking which physical memory a process can access.
- However, as shown in Rowhammer, **accessing DRAM cells in a specific pattern can cause data to be altered in other DRAM cells** - there is no explicit violation of the isolation mechanisms, but rather the physical devices' properties cause data to change in memory locations what were not actually accessed by the attacker.
- To realize the attack, first, attacker process' data and the victim's data need to be in adjacent DRAM rows. → Next, the attacker can repeatedly access its own data in the DRAM rows adjacent to the victim's data.
- After a large number of iterations, some of the bits in the victim's data will change their value.
- The attack is built on the principle that the **charges in certain DRAM cells will flip if there is repeated electrical activity nearby**, i.e., memory access, in the adjacent cells.
- Which cells flip their value depends on **the manufacturing variations** of the DRAMs and is different from one device to another.
- Thus, the attack requires the data to be in very specific locations, and not all DRAM rows in a DRAM module may be susceptible to this attack.
- Protections against this type of attack can include hardware modifications to **make DRAM cells less prone to flipping bits under repeated stress of accesses to adjacent cells**.
- In software, the memory of victim and attacker processes can be allocated such that it is not in adjacent DRAM rows.

Real-World Attacks – Meltdown

- **Meltdown vulnerability** can be used to break isolation between user applications and the operating system.
- Meltdown exploits side effects of the out-of-order and speculative execution features on today's processors to enable user applications to read arbitrary memory locations of the operating system kernel that have been mapped into the address space of the user process.
- Today's operating systems **map the kernel into the address space of every process** to allow for, e.g., fast interrupt handling that does not require changing address spaces.
- The isolation between the user application and the kernel memory locations is based on a privilege level bit indicating the current execution privilege level (typically user or kernel).
- On a memory access, the privilege level is checked.
- **The out-of-order and speculative execution features** of Intel processors, but possibly others as well, allow for speculative execution instructions to improve performance; and they should nullify any changes in the processor state if the speculation was incorrect.
- However, speculative execution of data loads also influences the processor cache.
- Although the processor may properly clean up its state after any speculative execution, if the speculatively loaded data remains in the processor cache, it can leak information, as was demonstrated with Meltdown (and related Spectre attack).
- In a simplified example of Meltdown, there may be an instruction that causes a trap in a user program, followed by an access to a memory location in the kernel, labeled **`data`** in below example, and further access, labeled **`probe_array`** in below example, that uses the data from that kernel memory location as an address to access another memory location. A sample code form is:

```
raise_exception();
access(probe_array[data * 4096]);
```

Real-World Attacks – Meltdown (Cont.)

- If the out-of-order and speculative execution logic speculatively executes the memory access (e.g., before computing that the instructions should not happen due to the trap), it will read the data from kernel memory location and use it as an address for the probe array load.
- The accessed kernel data is never visible to the user application.
- However, **the memory content** that was accessed based on the address derived from the data in the kernel memory is left present in the cache.
- Subsequently, by doing **a cache side-channel attack**, the attacker application can probe which memory locations are in the cache, and from there can directly derive the value of the kernel's data.
- Thus, out-of-order and speculative execution, combined with a cache side-channel attack, **allow user applications to bypass protection checks** and read any data that is mapped into the user application's address space.
- A **hardware solution** to Meltdown is to do privilege checks on the speculatively executed data early in the speculation process (processors such as from AMD were not found to be vulnerable as they do the checks before the memory access is speculatively executed).
  - Such hardware changes require micro code updates or replacement of the processor which can be costly.
- A **software solution** is to not map so much kernel data into the address space of user applications, however, this will have performance impact, e.g., on interrupts.
- Meltdown does not use branch prediction for achieving speculative execution; it relies on instructions that will cause a trap.
- Meltdown leverages delayed privilege checks to allow applications to access kernel memory locations.

Real-World Attacks – Spectre

- **Spectre vulnerability** can be used to break isolation between different applications.
- Spectre exploits speculative execution of instructions following **branch instructions**.
- Spectre allows forcing a victim application to leak its secrets to a different, attacker, application.
- **Spectre and Meltdown Attacks** leverage cache side-channels analysis to actually find out what the secret data is (of the kernel in case of **Meltdown**, or of the victim application in case of **Spectre**).
- According to the researchers who found Spectre vulnerability, speculative execution capabilities found in processors from Intel, AMD, and ARM all currently have Spectre vulnerability.
- Consequently, Spectre affects most processors in use today, but in practice may be difficult to exploit as it requires a mix of techniques to achieve a practical attack.
- In a simplified example, an attacker needs to **train a branch predictor to mis-predict on a certain branch instruction address**.
- As a branch predictor is shared by all processes running on the same CPU, an attacker who knows the code of the victim can create an application that has branches at same addresses, but the branch **outcome** is **different**.
- For the victim, as an example, it can have an access to an array, result of which is used to access a second array; with an if statement to guard the address range for the first array access—to prevent accesses beyond its bounds. A sample code form is:

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

- Since **the attacker has trained the branch predictor to mis-predict** on this `if` branch check, it can cause the processor hardware to assume the branch is not taken (if statement is **true** actually) and execute the out-of-bounds access with a very large **x** to the first array, which will be used by the second array access.

Real-World Attacks – Spectre (Cont.)

- The processor will eventually compute that there was a mis-prediction and discard any data, **`y`**.
- However, **the processor cache will have already been occupied with the data from array accesses**.
- As the array element accessed depends directly on value of data in **`array1[x]`**, in a final step, the attacker can perform a cache side-channel attack and **measure timing of the accesses** to figure out which memory location was brought into the cache during the speculative execution and thus **find the value of the data** at **`array1[x]`**.
- In addition to abusing the branch predictor, other variants of the attack are possible such as by **using indirect branches**.
- A **hardware solution** for this type of attack would be to disable the speculative execution and branch prediction.
- However, **the performance impact** would be significant.
- **Another hardware solution** would be not to share the branch predictor (e.g., have multiple separate predictors in hardware), but such solution may not scale.
- A **software solution** may be that applications could also be isolated by having one application only running alone on one processor.
- Since processors do not share branch predictors, this would prevent an attacker from influencing the branch predictor state - but again performance (negative) impact would be significant.
- **Another software solution** is to for loads inside branches to act as memory fences (or insert explicit memory serialization instructions). → This can prevent memory loads (which modify cache state used in the side channel part of the attack) from executing until prior instructions have finished.

Other Bugs or Vulnerabilities

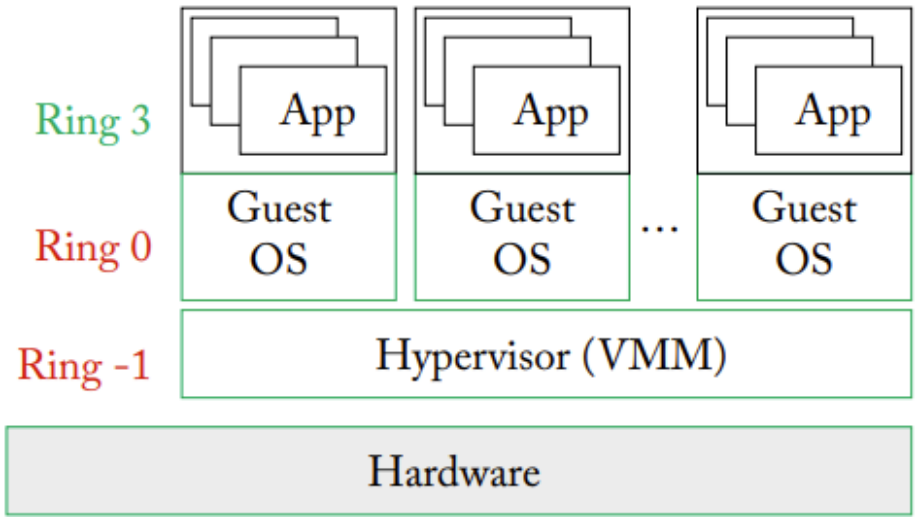
- Normal processors suffer from **variety of bugs**, which are published regularly by the processor manufacturers in their errata documents, or which are listed on numerous web pages.
- An analysis of over 300 bugs from these errata documents has found almost 10% were security-critical.
- **Attacks leveraging processor bugs**, or simply **abusing some processor functionality**, can be used against features such as the **System Management Mode (SMM)**, to escalate privileges of the attacker, or **Message Signaled Interrupts (MSI)** mechanisms to break **Virtual Machine (VM)** isolation.
- The vulnerabilities are by no means limited only to processors or memories. → For example, researchers have demonstrated that vulnerabilities in GPUs can be used to break isolation and steal data from different programs sharing the same GPU.
- **Attacks do not have to also focus just on the compute related components**.
- Thermal sensors have been abuse to leak information in multicore processors.
- Features such as **dynamic voltage and frequency scaling (DVFS)** have also been abused, and researches have shown that they can be manipulated to change timing of operations and introduce faults, allowing them to leak secrets such as encryption keys from protected environments.

Other Bugs or Vulnerabilities (Cont.)

- Many of such attacks stem from **competing design goals (and parameters)**.
- The main goals of processor architects are to **improve performance, reduce area/space, or reduce energy/power consumption**.
- Each type of optimizations, however, can bring about potential vulnerabilities.
- Performance enhancing features are a prime example of sources of potential attacks → **Example:** Processor caches allow for creating abstraction of large and fast memory, but they also allow cache side-channel attacks due to timing differences in memory accesses that they create.
- Reduction of area can also lead to potential attacks, e.g., using more densely packed transistors and electronics can reduce cost of a chip, but can lead to attacks such as Rowhammer in DRAMs, where close proximity of memory cells allows them to interact electrically in ways that break higher-level assumptions about how the memory operates.
- Power features such as the dynamic voltage and frequency scaling can cut processor power, but also can be abused to change timing of operations and cause **faults** that result in attacks.

General-Purpose/Secure Processor Architectures

- Secure processors are built on top of general-purpose processor architectures, and expand them with new security features.
- At any point in time, code running on the system is executing at one of **the privilege levels, or rings**.
- The privilege level determines what the code or instructions can and cannot do.
- Traditionally, modern computer systems use **ring-based protections** to separate privileged and unprivileged software.
- During code execution, hardware keeps track of the current privilege level (ring) in which the code is running.
- **The different rings considered in commodity processors are: user (ring 3), various semi-privileged code (ring 2 and 1), and the operating system kernel (ring 0).**
- To provide further functionality, over time, new features have been added, resulting in addition of new privilege levels, such as the hypervisor (ring -1).



A diagram of the typical software levels in a modern processor. The green outline shows the components most often considered trusted in a modern processor.

VMM: Virtual Machine Manager.

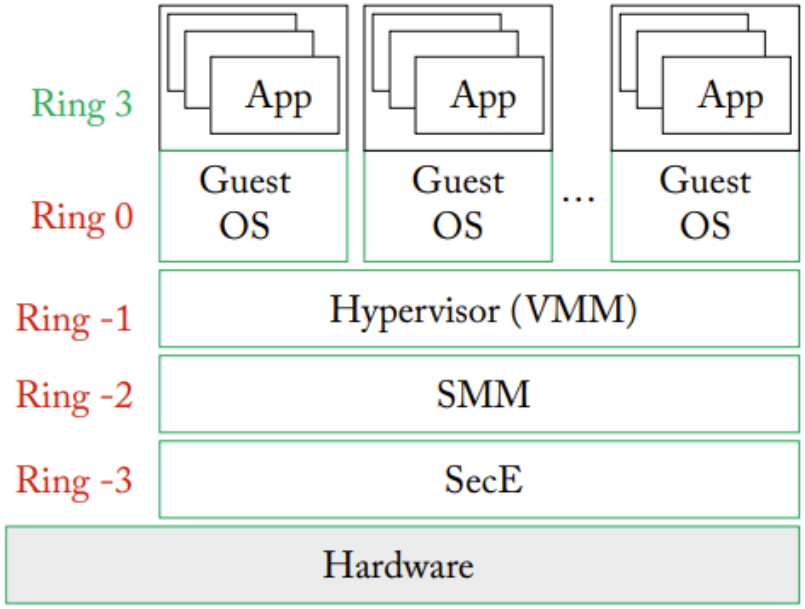
General-Purpose/Secure Processor Architectures (Cont.)

- When executing instructions on a processor, certain instructions are restricted to be only available in **privileged mode** (e.g., ring 0) while others can be executed in any privilege level.
- Memory access checks are performed to ensure only privileged code can access some memory locations.
- A privilege change from **a lower to higher privilege** can happen through **special instructions** (such as a system call initiated by the application or a VM exit initiated by the guest VM) or by **a hardware event** (such as a fault, an interrupt, or a signal on a physical pin on the processor chip).
- Entrance to more privileged modes has to be guarded so that less privileged code cannot elevate its privileges when not authorized.
- It is the duty of the software and hardware controlling the more-privileged execution to validate the inputs before it acts on them.
- **Any fault that occurs while code is in a particular privilege mode can affect all other code in that or any less privileged (i.e., same or higher ring number) code.**
- Such faults, however, should not have impact on any more privileged code (i.e., lower ring number). → For example, if a guest OS crashes (ring 0), then the hypervisor (ring -1) should still keep operating correctly.
- Compromised or malicious operating system can attack all the applications in the system.
- Likewise, compromised or malicious hypervisor (i.e., **malicious host OS**) can attack all the operating systems in the system (i.e., all of the guest OSes).
- Secure processor architectures address some of these issues by adding new privilege levels for trusted management software, prevent some lower levels (more privileged today) from having access to higher levels, or add horizontal privilege separation within levels.
- This aims to help **reduce the software TCB**, which otherwise today contains all the software from the operating system down.

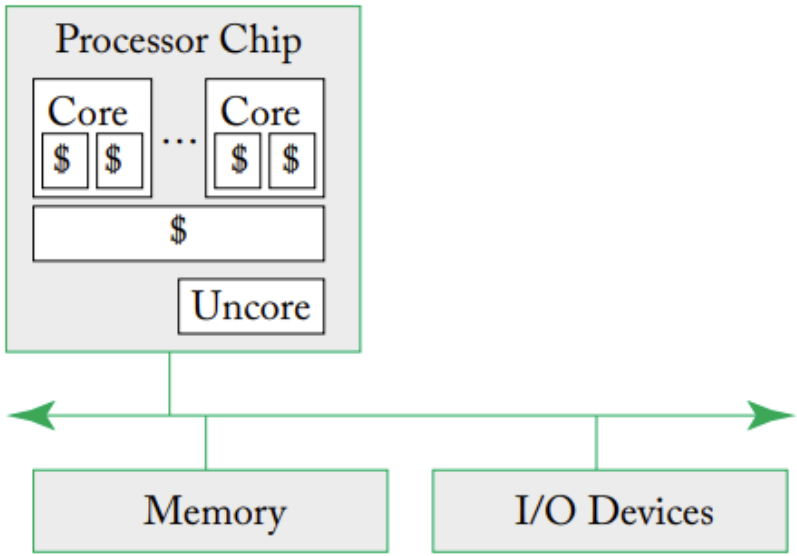
General-Purpose/Secure Processor Architectures (Cont.)

- Starting in the early 2000s, commodity processor vendors have started to **add hardware virtualization assistance** to their products, with Intel VTx and AMD-V extensions.
- In today's commodity processors, **hypervisor mode (ring -1)** is typically implemented as an extra privileged mode for the main processor cores.
- Hardware changes only/mainly affect the processor.
- In addition to the mode itself, extra additions such as hardware nested page tables have been introduced (a concept of software-based shadow page tables has been used before hardware supported nested page tables were introduced).

A diagram showing new privilege levels that have been added over time. New levels are leveraged to protect more trusted code, as it manages the system.



A diagram of the typical hardware components in a modern processor. The green outline shows the components most often considered trusted in a modern processors.



General-Purpose/Secure Processor Architectures (Cont.)

- With addition of these components, it is natural that hardware inside the processor, such as **table-lookaside buffers (TLBs)**, is expanded to work with the new memory management additions.
- A **typical computer system** today contains the main processor, and other components such as memories or input and output (I/O) devices.
- **Snooping on the system bus** is possible and can be used to extract information communicated on the bus.
- Compromised or malicious devices can attack other components of the system.
- **Secure processor architectures**, add new features to the processor, or the other components, so that some of the other components can be untrusted (and thus not part of the TCB).
- Secure Processor Architectures add new hardware and software features to provide **Trusted Execution Environments (TEEs)** wherein software executes protected from some of the software and hardware threats (according to an architecture's threat model).
- Secure processor architectures enhance general-purpose processor with new protection features.
- They provide new or alternate privilege levels and utilize software or hardware changes to facilitate protection of software (software modules, applications, or even VMs).
- The new privilege levels include **system management mode/SMM (ring -2)** or **platform security engine (ring -3)**.
- System Management Mode (Ring -2) is a privilege level originally introduced in Intel processors, which is more privileged than the hypervisor mode.
- The SMM code is typically part of the firmware, and the SMM mode can be entered only through a special System Management Interrupt (SMI) by asserting a pin on the processor chip's package or I/O access to a specific port.
- The goal of SMM to provide some management functionalities, even if the operating system or hypervisor is compromised.
- The SMM code is typically very small and provided by the computer manufacturer without means for users to analyze, check, or update the code.
- These restrictions create a **security through obscurity situation**, and have led to exploits, e.g., SMM rootkits.
- Platform Security Engine (Ring -3) is an even more privileged level.

General-Purpose/Secure Processor Architectures (Cont.)

- **Platform management engine** is typically a separate, small processor **fully independent** of the main processor of the computer.
- The management engine can access resources of the computer even if the operating system or hypervisor is compromised or has crashed.
- If management engine is **a separate chip**, it can have separate power connection from the main processor, memory or other components allowing it to stay on while the whole computer is offline → For example, it can be used to power on computers remotely (management engine has interface to network as well so it can receive remote commands).
- In addition, it often has **reserved memory regions** that it can use for **code or data**, which cannot be accessed by other components, leading to the designation of ring -3 that is not controllable by any other code in the system.
- In case of Intel's ME, it was usually embedded into the motherboard's north bridge, although with newer designs it may be in another part of the system, or even within same package as the main processor.
- With introduction of AMD's SEV and memory encryption technologies, AMD's chips include a Platform Security Processor, which has many similarities to the Management Engine.
- **The goal of the highest privileged level** is to be able to control system execution and emulate some hardware features using a very small, embedded processor.
- However, similar to SMM, the even more privileged security engine usually, in commercial products, contains proprietary code that is usually a trade secret, with infrequent updates.
- New privileged execution modes can also be introduced to separate privileges horizontally.
- These new privileges can be made orthogonal to existing protection levels.
- Furthermore, architectures can be designed to break the linear relationship (where the lower level is always more privileged than a higher level).
- Reducing the number of trusted levels makes the TCB smaller, which is always one of the design goals for secure processor architectures.
- Some architectures, such as Bastion, have been designed to assume untrusted operating system; hypervisor and all the levels below work together to protect the **Trusted Software Modules (TSMs)**.

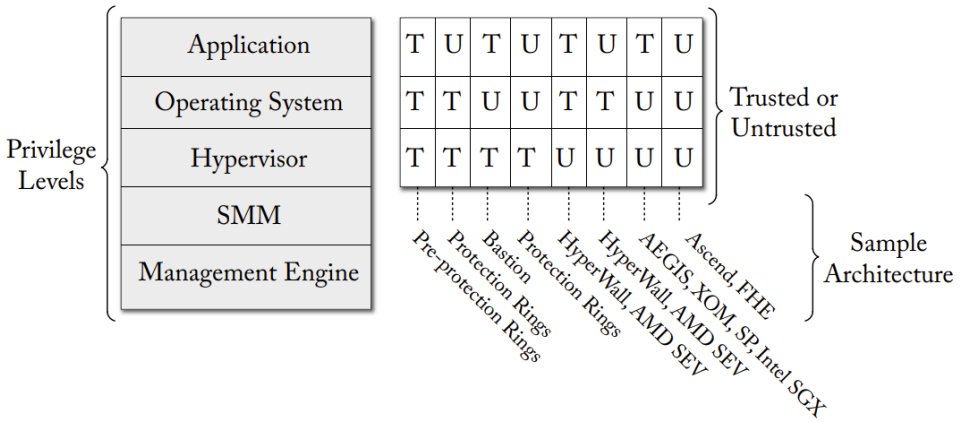
General-Purpose/Secure Processor Architectures (Cont.)

- There are also alternative designs which do not necessarily use **the linearly ordered set of privilege levels** at all.
- Capability-based architectures associate "**capabilities**" with different resources (such as memory locations or hardware modules).
- To access or make use of a resource, the requester needs to have a token or a capability that allows it the access.
- In such scenarios, the threat models will look different as the trusted and untrusted components (and potential attackers) are not discussed in terms privilege levels, but sets of capabilities that each entity possesses.
- **By introducing the different new levels**, adding horizontal level separation, or designing the system to reduce the software TCB and consider some levels untrusted, different architecture for different threat models can be achieved.
- Especially, various combinations of trusted and untrusted levels in a system result in designs that protect from different software threats and attacks.
- Typically, each processor architecture is composed of **multiple execution privilege levels**.
- At each privilege level, e.g., user level, there can be multiple software entities, e.g., there are many applications running on a typical workstation at one time, or there are numerous VMs (operating systems) running on a server.
- The processor architecture threat model has to specify which of the privilege levels are trusted and which are untrusted.
- The threat model has to specify if entities at the same level are **mutually trusting**.
- A threat model has to say which of the levels and entities are trusted and which are not.
- Modern computer system is composed of one or more separate physical chips.
- Physical probing of wires, changing chips, and even modifying the physical chips form the different hardware attacks that secure processor architectures aim to mitigate where possible.
- When designing a secure processor architecture, memory is often singled out as the component that should not be trusted as it is a passive component which can be easily remove and swapped with a different one.
- **Physical probing inside the memory** is typically not considered, but a memory chip can be always moved into another computer and its contents read out, e.g., the Coldboot attack.
- **I/O devices** are also singled out as untrusted as they are typically from different, potentially untrusted manufacturing sources.

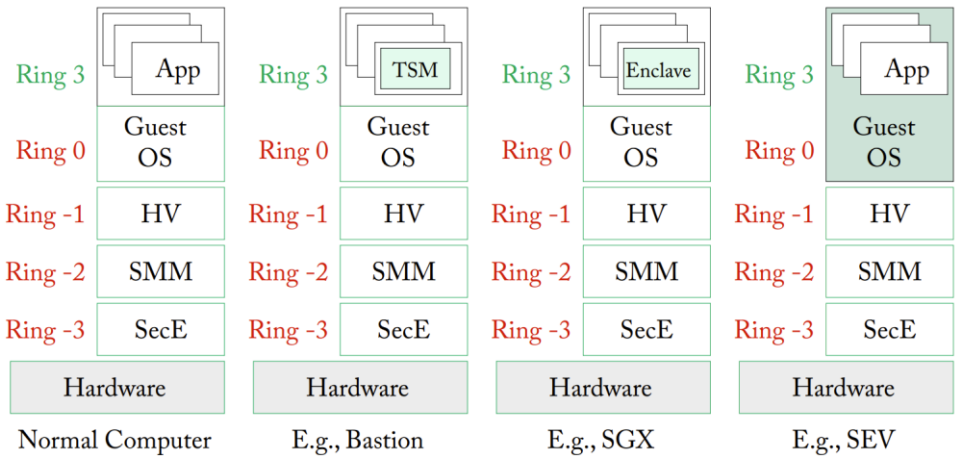
General-Purpose/Secure Processor Architectures (Cont.)

- Finally, **the interconnect is simple** to physically probe as well and should not be trusted.
- In **a package-on-package configuration**, a CPU chip is physically located below a memory chip. → This can greatly reduce wire lengths and accelerate the data transfer between memory and CPU.
- Because of the tight integration, it is now more difficult to probe the interconnect between CPU and memory, especially as attackers may have to physically disassemble the package-on-package configuration, which increases the difficulty of the attack.
- The **3D integrated systems** may consider memory to be trusted again due to the higher difficulty of the attacks. → On the other hand, tight integration of many components may lead to new types of attacks or side channels.
- Key parts of the hardware TCB can be implemented as dedicated circuits or actually as firmware or other code running on dedicated processor, especially code running on dedicated processor seems to be **preferred approach** in industry.

Given a set of privilege levels, each of these can be trusted or untrusted, depending on the secure processor design. The figure shows different combinations of trusted and untrusted levels and corresponding sample secure architecture that implements the assumptions about which parts are trusted and untrusted. The sample architectures listed are Bastion, HyperWall, AMD SEV, AEGIS, XOM, SP, Intel SGX and Ascend. While not a processor architecture, Fully Homomorphic Encryption (FHE) is also included in this figure as it is one potential solution for processing data when all of the software is untrusted.



Example of how different architectures consider some of the privilege levels untrusted, and these are not in the TCB. Green outline shows which levels are trusted for these sample architectures. The three examples are Bastion [35], Intel's SGX, and AMD's SEV architectures.



Examples of Secure Processor Architectures

- **Academic Architectures**
  - Researchers begun to be interested in protecting software applications (code) from hardware attacks and modification of the contents of the off-chip memories.
  - Protections against operating systems were later introduced.
  - Once hypervisors were introduced, they were co-opted to work with the hardware to provide the protections.
  - As the hypervisor code begun to bloat, the hypervisor begun to be considered an untrusted entity and new protections were added to protect code and data against untrusted hypervisors as well.
  - Some architectures consider all software to be untrusted and explore how to perform computation on encrypted data.
  - Most of the designs focus on **single-processor systems**.
  - Multiprocessor security has focused mostly on the communication aspect (securing communication between multiple processors and memories), while individual processors in a secure multiprocessor system are often secured using ideas from single-processor designs.

Examples of Secure Processor Architectures (Cont.)

- **Commercial Architectures**
  - The security features were most common in main frame computers.
  - Designers of later microcomputers did not incorporate many explicit security features, but in the late 2000s industry begun to again present designs such as Sony, Toshiba, IBM's Cell Broadband Engine (with its security processor vault), Dallas Semiconductor's Secure Microprocessor Chip (with its encrypted memory and self-destruct mode), ARM's TrustZone, Intel's SGX, and AMD's SEV.
  - **The commercial solutions leverage many of the academic ideas, but also their own pragmatic ideas needed to actually deploy the products**.
  - One pragmatic feature, and potential weakness, of the architectures comes from use of dedicated security processors (inside the main processor).
  - These are used to realize some of the **"hardware" features**, such as managing the protections, or updating the page tables.
  - Having all features in pure hardware is likely impractical due to time-to-market or cost constraints, so some **"hardware" features have to be pushed to software**.

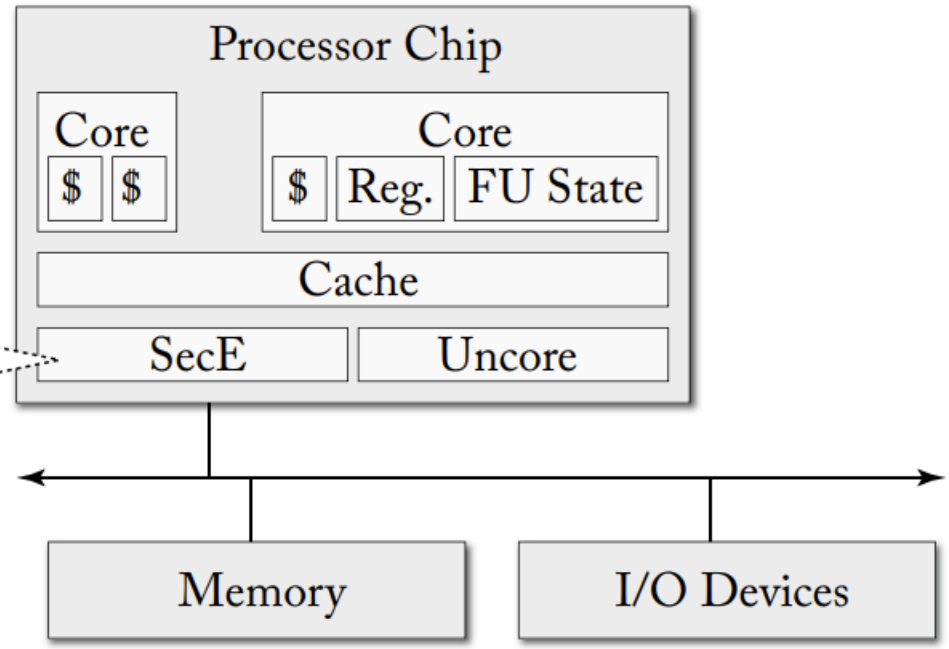
Examples of Secure Processor Architectures (Cont.)

Custom logic or hardware state machine:

- Most academic proposals

Code running on dedicated processor:

- Intel ME = ARC processor or Intel Quark processor
- AMD PSP = ARM processor



The hardware security engine, SecE in the figure, can be realized as dedicated circuits, or it can be implemented as a processor running some management code.

Secure Processor Architectures – Assumptions

- Secure processor architecture designs typically involve **a number of assumptions**.
- **Trusted Processor Chip Assumption**
  - The key to most secure processor architecture designs is the trusted processor chip assumption.
  - It is assumed that the processor chip is **the trust boundary** for the hardware TCB.
  - Everything in the processor chip is trusted, and everything outside is not trusted.
- **Small TCB Assumption**
  - To prevent or minimize problems due to the TCB, the TCB should be small.
  - It is further assumed that a smaller hardware and software TCB imply **better security**.
  - The small TCB assumption is derived from two general ideas.
  - First, less software code means it can be more likely audited, verified, and it will contain fewer bugs.
  - Second, less hardware code likewise means it can be audited, verified, and will contain fewer bugs.
- **Open TCB Assumption**
  - Using **Kerckhoffs's Principle from cryptography**, the TCB should not contain any secrets, and it is assumed that the TCB can be inspected and analyzed.
  - Especially, operation of the TCB should be **publicly known** and should have no hidden functionality.
  - Only secrets should be the cryptographic keys, to prevent security through obscurity.

Secure Processor Architectures – Limitations

- Secure processor architectures are not a full solution to computer security problems, especially, secure processor architectures do not, usually, deal with the physical realization of the processors themselves.
- The general area of **hardware security** (as opposed to the architecture security) covers topics that are orthogonal to secure processor architecture design, but should be considered by architects when they think about their architectures.
- **Physical Realization Threats**
  - Secure processor architectures assume that the manufactured chip, and especially the hardware, is correct.
  - Research on hardware trojans, however, shows that **malicious hardware can be inserted** after the design time, e.g., at the foundry where the processors are manufactured.
  - On the other hand, **hardware trojan defense** research shows how such trojans can be detected.
- **Supply Chain Threats**
  - In addition to **modification of the chip**, which may in practice be very difficult, there are issues of the supply chain.
  - Modern servers, embedded system, etc., contain **intellectual property (IP)** from many designers and they are manufactured in variety of locations before being finally **assembled** into the finished product.
  - At any of these stages in the supply chain, **a malicious component can be inserted** into the system. → For example, CPU can be correct, but the memory chip is malicious.
  - **Fingerprinting hardware modules** and **identifying** correct ones, such as through use of PUFs, is one possible defense.

Secure Processor Architectures – Limitations (Cont.)

- **IP Protection and Reverse Engineering**
  - Security should not be through obscurity, thus the design and hardware of the secure processor architectures should be known.
  - Still, in a number of scenarios the designers may want to keep the hardware implementation a secret, e.g., due to fears of others stealing their intellectual property.
  - **Camouflaged logic, split-manufacturing, and other approaches** can help prevent attackers from **reverse engineering** the design.
  - Camouflaged logic aims to prevent one from deducing the circuit design, and behavior, by looking at the physical layout of the transistors.
  - Meanwhile, **split-manufacturing** involves producing a processor chip in two or more foundries, where each one processes only few levels of the design.
  - Split-manufacturing research shows how to divide the design into different parts (usually the design is split into back end of line, BEOL, and front end of line, FEOL) and each part is processed by a different foundry such that at a foundry cannot deduce final design based just on the parts it is processing.
  - A related topic to **Intellectual Property (IP) Protection** is threats of over-production.
  - Ideas of hardware odometers have been presented where new hardware features are used to ensure only legitimate devices can be authenticated.
  - Re-use or recycling of devices is dangerous as old, worn-out parts can be sold as new.
  - Odometer features can give indication about the age of the integrated circuit chip and whether it is new, or has been used for extended period of time.

Secure Processor Architectures – Limitations (Cont.)

- **Side- and Covert-Channel Threats**
  - **Side- and covert-channel attacks** can be used to leak the information based on the physical emanations (e.g., power, thermal, electro-magnetic).
  - Such attacks can be damaging and used to leak sensitive information - typically the goal is to get the encryption key.
  - A distinction needs to be made between **information leaks due to the design of the logic**, most often timing channels, and **information leaks due to physical implementation**.
  - At the architecture level, the logic-related channels should be eliminated, e.g., due to processor caches. → The physical implementation related channels may still remain, e.g., EM channels.
- **What Secure Processor Architectures Are Not**
  - Secure processor architectures are not **hardware security modules (HSMs)** such as IBM CryptoCards.
  - HSMs are dedicated, hardware modules that have extra physical security compared to typical processors (and secure processor architectures).
  - HSMs may have tamper-resistant and tamper-evident coatings (e.g., the module may try to erase keys and shut down if it detects physical tampering, typically thanks to a wire mesh or other sensors that, when tampered with, signal an intrusion).
  - They further may be battery-backed, to ensure power-cycling of the whole system does not affect them (and that they have power to execute any proactive defenses in case of an attack where the rest of the system is shut down).
  - **Secure processor architectures mimic some HSM features** (e.g., memory encryption) but usually do not deploy the more extreme measures (e.g., physical coatings).

Secure Processor Architectures – Limitations (Cont.)

- **What Secure Processor Architectures Are Not (Cont.)**
  - A distinctive feature of secure processor architectures from HSMs is that they rely on platform features and modify only the architecture and digital logic.
  - Secure processor architectures typically are not concerned with physical design, but aim to provide as much security as possible only through the architecture and digital logic level.
  - Secure processor architectures are also not security accelerators, such as dedicated devices for speeding up encryption or decryption.
  - They almost always have **dedicated hardware for acceleration of encryption, hashing, or public-key cryptography**.
  - These features are used by the secure processor architectures' hardware to speed up the new protections it offers - a separate accelerator may still be present on a system if needed, e.g., for high-speed encryption or decryption of network traffic.
- **Alternatives To Hardware-Based Protections: Homomorphic Encryption**
  - A **theoretical alternative** to the myriad of **hardware-based security modifications**, or need to introduce new hardware architectures, may be **Fully Homomorphic Encryption (FHE)**.
  - **In fully homomorphic encryption, operations are performed on the ciphertext and result in the creation of new ciphertext, which can later be decrypted to see the results of the computation**.
  - **Importantly, the new ciphertext does not leak any information about the results, which can only be accessed by an entity with the proper decryption key**.
  - Currently, fully homomorphic encryption suffers from **two practical limitations**.

Secure Processor Architectures – Limitations (Cont.)

- **Alternatives To Hardware-Based Protections: Homomorphic Encryption (Cont.)**
  - **First Limitation** → The operations are very slow, making them impractical.
  - **Second Limitation** → Code is not protected as the protections only extend to the data.
  - Despite the limitations, with fully homomorphic encryption, the hardware manufacturer is **not** trusted. → Actually, there is not a TCB anymore in some sense.
  - **All the inputs, intermediate data, and outputs are encrypted**, so there is no plaintext information anywhere on the system that could leak out.
  - If the hardware manufacturer is trusted, many operations can be done on a secure processor at high speed. → **Hardware-Based Protection Approach**
  - Any bugs or vulnerabilities could allow sensitive data to leak out, giving **motivation** of using cryptographic approaches such as FHE that do not depend on trusting the hardware nor the processor manufacturer. → **Fully Homomorphic Encryption Approach**

Assignment

- **Reading Assignment:**
  - Zferer, J., 2018. **Principles of secure processor architecture design**, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 9048, pp.1-175.
    - ✓"Chapter 3: Secure Processor Architectures", Pages 25-42.

Questions?