Lecture Notes

Chapter 4: Trusted Execution Environments

CYENG 225: Microcontroller Essentials for Cyber Applications

Instructor: Dr. Shayan (Sean) Taheri
Gannon University (GU)

Chapter 4 Overview

- **Major Items**
  - It introduces **Trusted Execution Environments (TEEs)**.
  - It presents high-level description of the protections offered to **TEEs** by the **Trusted Computer Base (TCB)**, and how the protections can be realized.
  - It presents a list of existing academic and commercial secure processor architectures, and the types of TEEs they offer as examples of possible design choices.
  - It presents TEE-related assumptions.
  - It lists limitations of today’s TCBs and the TEEs they create.

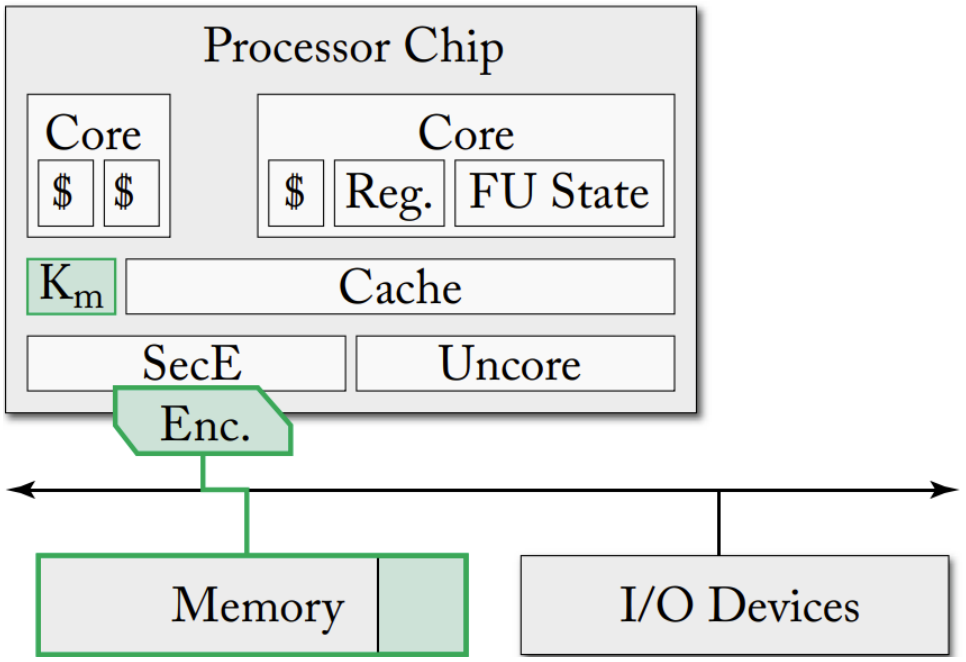
- **TCB is the set of hardware and software that is responsible for creating the TEE environment**.
- Software executing within a TEE is protected from a range of software and hardware attacks.
- The range of attacks that the software is protected from depends on the threat model of the particular secure processor architecture.
- **The relationships between TCB and TEE are**:
  - TEE is created by a set of all the components in the TCB, both hardware and software.
  - TCB is trusted to correctly implement the protections.
  - Vulnerability or successful attack on TCB nullifies the TEE protections.
- Different secure processor architectures focus on protecting **Trusted Software Modules (TSMs)**, also called **Enclaves**, while others on protecting **VMs** or **containers**.
- All of the code inside the TEE is given the same set of protections.
- There are no explicit protections that TEE gives to the different parts of the code in the TEE (except for the differentiation of the usual privilege levels, if the TEE contains a whole VM).
- Users need to carefully consider what code runs within the TEE, especially if they use any **external libraries or unverified code**.

Protecting Software within Trusted Execution Environments (Cont.)

- Protections Offered by the TCB to the TEEs
  - **Confidentiality** and **Integrity** are the two main security properties that the TCB of a secure processor architecture aims to provide for a TEE.
  - Confidentiality and integrity protection is from potential attacks by other software components or hardware components which are not in the TCB.
- **Enforcing Confidentiality through Encryption**
  - Given the trusted processor chip assumption, and that everything outside of the processor chip is untrusted, symmetric key cryptography should be used to protect data going off chip to prevent hardware attacks.
  - The security engine, or other part of the TCB, should encrypt data going out, and decrypt data going in to the chip.
- **Enforcing Confidentiality through Isolation**
  - Protected software can be separated through isolation (controlling address translation and mapping) to prevent software attacks.
  - Naturally, page tables are a well-known mechanism trolling memory allocation.

Protecting Software within Trusted Execution Environments (Cont.)

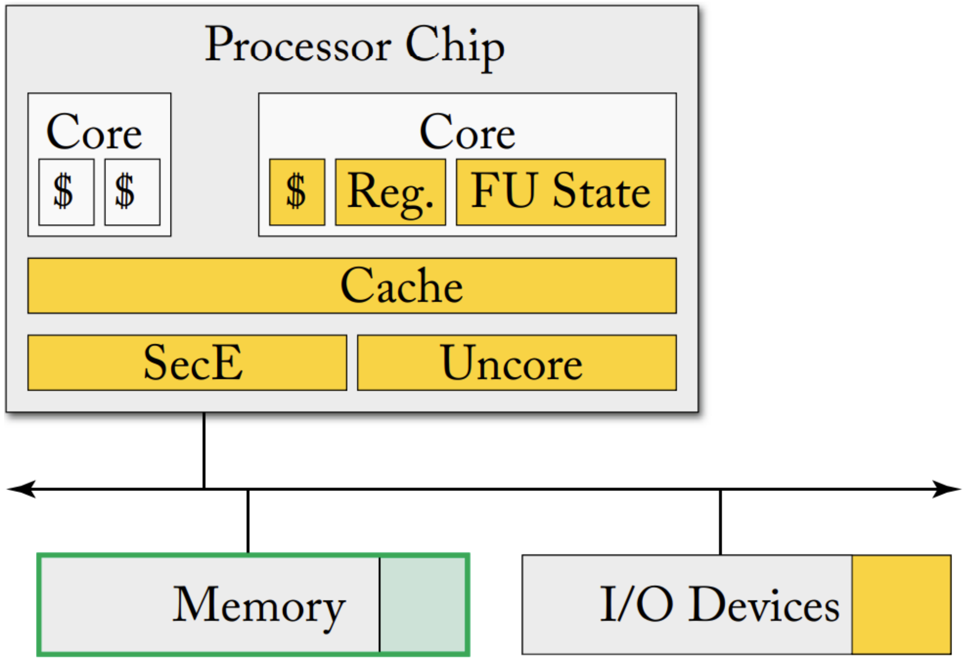
- **Enforcing Confidentiality through State Flushing**
  - State in the processor and elsewhere in the system needs to be flushed to ensure confidentiality from other entities that will later run on the system.
  - While not often considered, any buffer or register in the processor can store data related to execution of a TEE.
- **Enforcing Integrity through Cryptographic Hashing**
  - In addition to integrity protections, cryptographic hashing should be used to protect data going off chip to prevent hardware attacks and modification to the data.
  - All data going off chip (either due to explicit memory operations, or as part of saving and restoring processor state when TEE execution switches between different TEEs) needs to have its integrity protected.



A diagram of a typical processor, with added encryption engine for protecting data going off chip.

Examples of Architectures and TEEs

- Various academic and commercial architectures have explored adding hardware support for securing the executing software at different **granularities**.
- The different secure processor architectures mainly focus on protecting TSMs or Enclaves, or whole VMs.
- In general, one key characteristic of the secure architectures is that there is some shared secret (key) between the TCB and the code running in the TEE, so that the hardware or software of the TCB knows how to decrypt and verify the code and data in the TEE.
- Many of the architectures have concentrated on protecting discrete **Trusted Software Modules (TSMs)**.
- More recently, these are also called Enclaves thanks to the popularity of recent SGX extensions introduced by Intel.

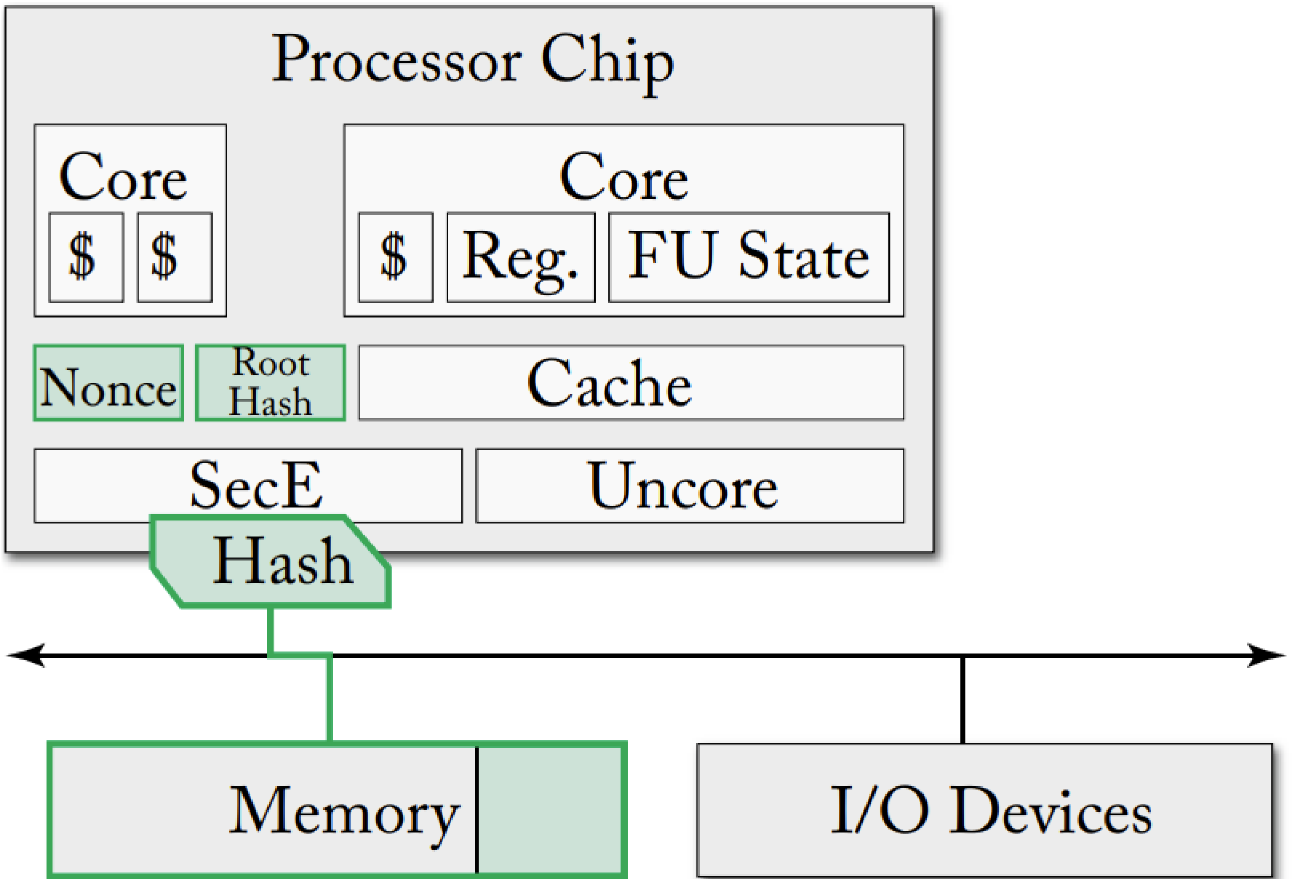


A diagram of a typical processor, showing in yellow the different components of the system that usually contain some state which need to be flushed between executions of different software

Examples of Architectures and TEEs (Cont.)

- **Academic Architectures for Protecting TSMs or Enclaves**
  - eXecute-Only Memory (XOM) has user code stored in memory compartments such that one compartment cannot access another.
  - XOM assumes that external memory is not trusted and, as a result, the data leaving the processor chip is encrypted.
- **Commercial Architectures for Protecting TSMs or Enclaves**
  - Cell Broadband Engine with its processor vault provides hardware isolated execution environments where code executing on a synergistic processing element (SPE) inside the Cell processor can be protected from the code executing in the rest of the system.
  - SPEs are dedicated hardware processing units with associated and dedicated memory.
  - The SPE and the memory can be locked out from the rest of the system and the code executing on the SPE can be isolated from even the main processing unit.
- **Academic and Commercial Architectures for Protecting Whole OSes or VMs**
  - NoHype introduced the idea of eliminating the hypervisor, while still allowing a number of virtual machines to share the processor.
  - Hardware was logically partitioned in NoHype such that each virtual machine obtained its resources on startup, and through the hardware separation mechanisms, one VM could not access resources of other machines.

Examples of Architectures and TEEs (Cont.)



A diagram of a typical processor, showing off-chip memory contents being hashed using a hash engine; and need to store the reference hash value and nonce on chip.

TCB and TEE Assumptions

- TEE designs typically involve a number of assumptions according to the following:
- **No Side Effects Assumption**
  - Secure processor architectures assume no side effects are visible to the untrusted components whenever protected software is executing.
  - Especially, the system is in some state before protected TEE software runs.
  - Next, the protected software runs, often modifying the system and processor state.
- **Bug-Free Protected Software Assumption**
  - The software (code and data) executing within TEE protections is assumed to be bug-free.
  - The goal of any secure processor architectures is to create minimal TCB that realizes a **TEE** within which the protected software resides and executes.
- **Trustworthy TCB Execution Assumption**
  - Any vulnerabilities in the TCB can lead to attacks that nullify the security protections offered by the system to the TEE.
  - Especially, problems in hardware state machines controlling the system could be exploited to nullify TEE protections.

Limitations of TCBs and TEEs

- Hardware-protected execution environments are a great way to protect the critical code and computation. → However, they come with **some limitations and potential pitfalls** for the designers and users.
- **Each limitation** can be seen as **a research challenge** → To find solutions on how to fix the limitations.
- **Vulnerabilities in the TCB**
  - Current designs allow for **TCB-resident attacks**, where the attacker uses the hardware protections to hide from the rest of the system.
  - Because the SMM (**Ring -2**) is more privileged than operating system or hypervisor (**Ring -1**), it may be impossible for system administrator to get rid of SMM rootkit once it is installed, especially, if there is no way to update or recover the SMM code.
- **Opaque TCB Execution**
  - Today, there are often **no means of auditing and accessing the code running** as part of the TCB, especially **code running the "hardware"** that is actually implemented as an embedded processor, notably for the security engine. → **Reminder**: We have Software TCB and Hardware TCB.
  - Proprietary code is usually a trade secret, with **infrequent updates**.
  - **Code signing**, if deployed, further prevents users from themselves updating the code.
  - **This secrecy** introduces type of security through obscurity, and there are well known attacks using the management engine as an attack vector to take over the whole system, e.g., ring -3 rootkits.
  - Code running and managing the TCB should be **fingerprinted**, and possibly **authenticated**.
  - A **hash over the TCB** code can be computed at load time, by the hardware. → Such hash could be available in a read-only register or memory location once the TEE code is loaded, but before the TEE code is executed.
  - Attestation of the TEE could then include attestation of the state of the TCB.

Limitations of TCBs and TEEs (Cont.)

- **TEE-based Attacks**
  - By design, trusted execution environments create **a hardware-protected space** wherein code can execute safely from outside inspection.
  - This creates a number of challenges which, if not addressed, can allow for malicious code to leverage the TEEs as an attack vector, while the hardware features meant to protect TEEs help the attacker from being discovered, or stopped.
- **TEE Code Bloat**
  - **Code Bloat**: Trusted Hardware Complexity or Trusted Software Complexity.
  - It is a potential danger.
  - As the TEEs are used to perform more and more functionality, there is more and more chance of a bug or vulnerability.
  - Over time, researchers and programmers are finding clever ways to put more and more code into the TEEs. → To protect more software elements.

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 4: Secure Processor Architectures”, Pages 43-51.

GANNON UNIVERSITY

Questions?