Lecture Notes on Jan/12/2023

Chapter 2:
Basic Computer Security Concepts

CYENG 225: Microcontroller Essentials for Cyber Applications

Instructor: Dr. Shayan (Sean) Taheri

Gannon University (GU)

Chapter 2 Overview

- **Major Items**
  - Basics of computer security needed for understanding secure processor architecture designs.
  - Discussion on the trusted computing base, security threats to a system, and threat models.
  - Brief overview of information leaks and side channels for understanding protections against side-channel attacks.
  - Security concepts of confidentiality, integrity, and availability.
  - Basics of symmetric-key cryptography, public-key cryptography, and secure hashes.
  - Importance of good sources of randomness.
  - Overview of physically unclonable functions (PUFs) and their applications.

Trusted Computing Base

- **Trusted Computing Base (TCB)**
  - Available in secure processor architecture.
  - Formed by the hardware components and the software components that work together and provide some security guarantees, as specified by the architecture.
  - **Important Distinction**
    - **Hardware Secure Architectures** where the hardware is protecting the software but all security mechanisms are solely in hardware.
    - **Hardware-Software Secure Architectures** where hardware works with some software (usually privileged software such as the operating system or the hypervisor) to protect other software. → More flexibility, but typically increases the size of the TCB.
  - Larger TCB → Less desirable as more lines of software code (and likewise lines of hardware description code) are assumed to be correlated with more potential bugs and consequently security vulnerabilities.

Trusted Computing Base (Cont.)

- **TCB and Computer System**
  - A typical computer system can be broken down into a number of distinct hardware components, e.g., processor cores, processor caches, interconnect, Dynamic Random-Access Memory (DRAM), etc.
  - These hardware components interact with each other as well as with the different software components as the system executes.
  - The hardware components which are dependent upon to provide security form the hardware TCB.
  - The software components (if any) which are dependent upon to provide security form the software TCB.
  - These Hardware and Software Components work together and provide certain security guarantees that are assumed to be trusted, and together form the **whole TCB**.

Trusted Computing Base (Cont.)

- Correct System Operation → TCB Correctness.
- **Goal of Secure Processor Architectures** → Ensuring that the trusted hardware components can work together with the trusted software components to provide the desired security properties and protections for the software (e.g., trusted software modules or enclaves, whole user applications, or even containers or VMs) running on the system.
- **System Components** → Trusted and Untrusted Hardware and Software Parts.
- Untrusted → Not trusted (i.e., not necessarily malicious) for the correct system operation.
- **Design of a Secure Processor Architecture** → The TCB construction and the system security properties should be properly considered, regardless of the effects of the untrusted parts.
- Each Untrusted Entity → A potential attacker that tries to break the security of the system → Consider all possible attacks by all the untrusted entities, unless a specific threat is explicitly not protected against.
- Threats → Untrusted Entities (i.e., Internal) and Physical Attacks (i.e., External).
- Trusted Parts → Explicitly determine the correct system operation.
- Any issue with a trusted part → No assurance for the whole system protection.
- Malicious or buggy trusted part is possible due to poor design.

Trusted Computing Base (Cont.)

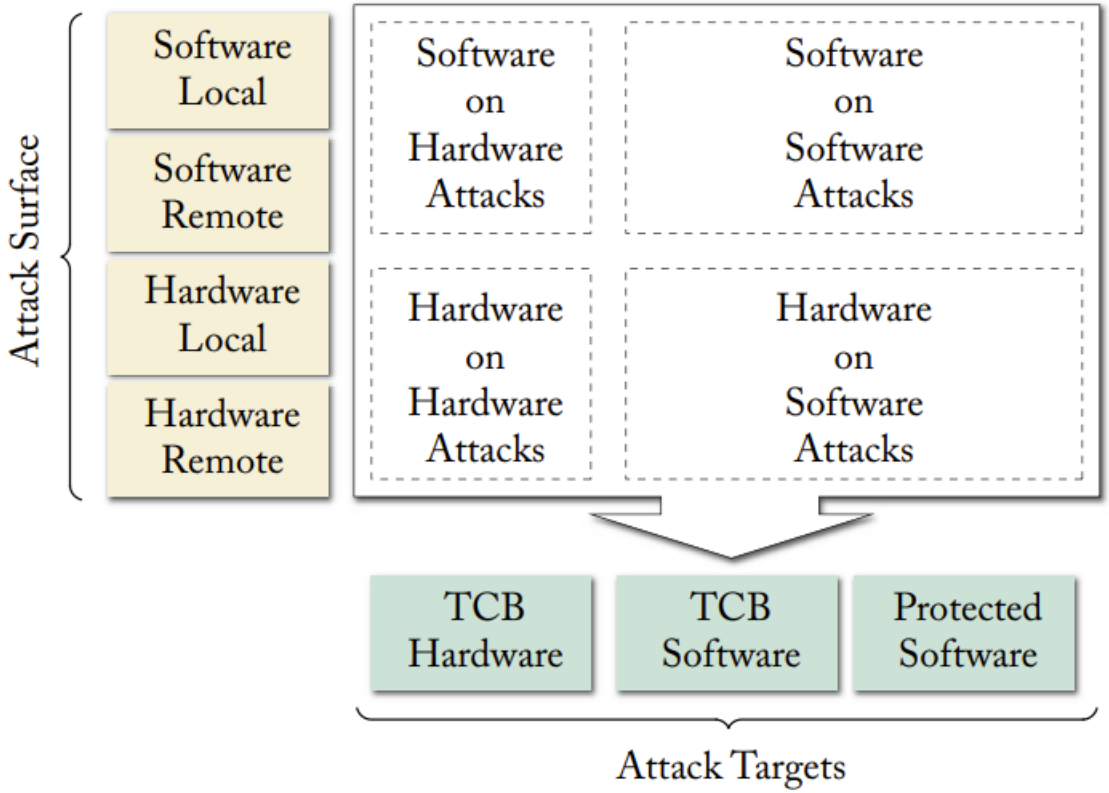
- Trustworthiness is a qualitative designation indicating whether the entity will behave as expected, is free of bugs and vulnerabilities, and is not malicious.
- **Trust Vs. Trustworthy:**
  - **Trust** → The hardware or software entities from the TCB.
  - **Trustworthy** → Design and implementation of the TCB entities.
- Formal Security Verification → Making sure that the design is correct.
- Bugs and Malicious Modifications (e.g., hardware Trojans) → They can occur at any stage of supply chain and product lifecycle, such as design, implementation, testing, and manufacturing.
- **Experts in Secure Computer Architecture** → Ensuring that the protocols, interactions, interfaces, and use of encryption and hashing among the trusted components are suitable from security perspective.
- Unusable products can be untrustworthy → The sensitive information (e.g., encryption keys) should be destroyed.

Trusted Computing Base (Cont.)

- **Kerckhoffs' Principle** → Avoid Security through Obscurity!
- **Security through Obscurity** → Attempting to secure the system by making the operation of the TCB secret and hoping that any potential attackers are not able to reverse engineer the system and break it. → Problem: Existence of many failed security systems. → Example: breaking into Intel's Management Engine that runs secretive code.
- Operation of the TCB of a security system should be publicly known and should have no secrets other, then the cryptographic keys. → Breaking the system is possible only if we have the cryptographic keys.

Security Threats to a System

- **Processor** + **Whole Hardware** + **Software** → **Term**: **Attack Surface** → They can be vulnerable to a number of security threats. → Exploitation by adversaries to execute different types of attacks.
- The attack surface is the combination of all the attack vectors that can be used against a system.
- Individual attack vectors are different ways that an attacker can try to break system's security.



- ✓ **The potential attack surface of a secure processor.**
- ✓ **The terms external and internal refer to whether the potential attack is from within the system or from outside.**
- ✓ **The targets of the attacks can be either the TCB or the software that is being protected (by the TCB).**

Security Threats to a System (Cont.)

- **TCB and Securiy** → It is assumed to be trustworthy.
- Attack vectors could be through hardware or software:
  - **External Attackers** (attacker is not executing code on the target computer nor physically near the computer system they are attacking).
  - **Internal Attackers** (the attacker is running code on the system he or she is trying to attack, or has physical access to the system).
- **Hardware Attacks** could come from untrusted hardware (not in the TCB), or external physical attacks (e.g., physical probing of the memory or data buses).
- **Software Attacks** could come from untrusted software (not in the TCB) or the software that is supposed to be protected (but due to bugs or malicious behavior tries to attack the system on which it is running).
- The targets of the attacks: (1) **TCB Hardware**; (2) **TCB Software**; and (3) **Software to Get Protected**.
- Software-on-software attacks could be untrusted operating system attacking software that is being protected.
- Software-on-hardware attacks could be untrusted software using cache side-channel attacks to learn secret information from a processor cache's operation.
- Hardware-on-software attacks could be an untrusted memory controller trying to extract information from DRAM memory.
- Hardware-on-hardware attacks could be untrusted peripheral trying to disable memory encryption engine.

Basic Security Concepts

- Analyzing and designing a secure processor architecture involves deciding about what properties the system will provide for the protected software, within the limitations of the threat model.
- The basic properties are: **confidentiality, integrity, and availability**.
- Authentication mechanisms are important to consider.
- As part of integrity checking and also of authentication, **understanding freshness and nonces** is a crucial aspect.
- It is important to **distinguish security from reliability**, and keep in mind that security assumes reliability is already in place.

Symmetric-Key Cryptography

- To ensure data confidentiality, symmetric- or private-key cryptography is needed.
- In symmetric-key cryptography, data encryption and decryption processes use the same secret key.
- When protecting data, there is the plaintext **p** which is encrypted into a resulting ciphertext **c** by the encryption function which also uses some private key **k**.
- Thus, the encryption process is: **c = Enc(k, p)**.
- To get back the plaintext data, decryption is needed: **p = Dec(k, c)**.
- Note, both encryption and decryption use the same key in symmetric-key cryptography.
- It is required that the encrypted ciphertext looks almost random to someone who does not posses the key **k**.
- Given ciphertext, an attacker should not be able to learn neither the key nor the plaintext data.
- Symmetric-key algorithms can be broken down into block ciphers and stream ciphers.

Public-Key Cryptography

- In public-key cryptography, also called asymmetric-key cryptography, data encryption and decryption processes use different keys.
- For confidentiality protection, the input is a plaintext **p** which is encrypted into a resulting ciphertext **c** by the encryption function which uses the public key **pk**.
- The encryption process is: **c = Enc(pk, p)**.
- To get back the plaintext data, decryption is needed: **p = Dec(sk, c)**.
- The decryption uses the secret key **sk**.
- Given a **pk** it should be infeasible to find out what is the secret key **sk**, which depends on hardness of certain mathematical problems, such as factoring of large numbers, e.g., RSA.
- The advantage of public-key cryptography is that **pk** can be given to anybody, and they can encrypt the data or code using this **pk**.
- Meanwhile, only the user, program, or hardware module in possession of the **sk** can decrypt the data.
- **For integrity, public-key cryptography can be used in "reverse" direction when used in digital signatures or message authentication codes**.
- The **sk** can be used to create a digital signature, and anybody with access to the **pk** can verify the signature - but cannot make a new valid signature as they do not have **sk** nor can they get the **sk** from knowing **pk**.

Random Number Generation

- Most security and cryptographic algorithms and protocols depend on **good sources of random numbers**, especially for key generation.
- There are **pseudo-random number generators (PRNGs)**, which use an algorithm to expand a seed into a long string of random-looking numbers.
- The numbers are not truly random, as given knowledge of the seed and the algorithm; anybody can re-generate the same sequence of random-looking numbers.
- There are also **true random number generators (TRNGs)**, which generate truly random numbers.
- For example, physical phenomenon like electrical noise or temperature variations can be used as sources of randomness.
- True random numbers are hard to obtain at high rate, thus many times TRNGs generate a small true random number, the seed, and a PRNG is used to expand that seed into a long string of random numbers.
- As long as the seed is truly random, and never accessible to potential attackers, then the resulting PRNG output can be used in secure manner.
- **Secure Computer Architects (SCAs)** often assume existence of sufficient randomness, and hardware and circuit designers are ones focusing on how to create such circuits.
- Designers should however realize that TRNGs can be manipulated, such as by inserting backdoors into processor's random number generator.

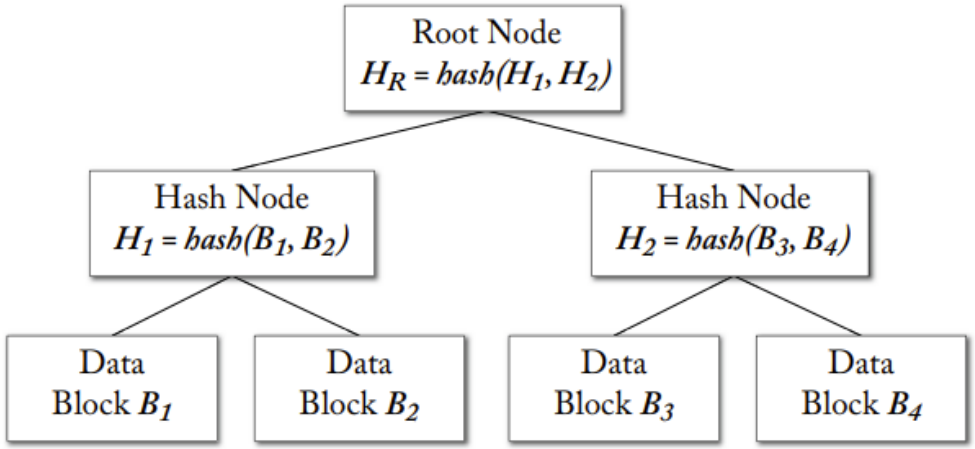
Secure Hashing

- To ensure data integrity, secure hashes are needed. A secure hash algorithm is a cryptographic hash function.
- A secure hash maps input data, **m**, of variable size to a fixed size output, **h** - the output is simply called the hash of the input data.
- **Secure hash is a one-way function**, and it should be infeasible to mathematically invert it and deduce the input data given the hash value.
- There are three properties that strong cryptographic hashes should have.
  - **Pre-Image Resistance**: Given a hash value **h** it should be infeasible to find any message m such that **h = hash(m)**.
  - **Second Pre-Image Resistance**: Given a specific input **m1** it should be infeasible to find different input **m2** such that **hash(m1) = hash(m2)**.
  - **Collision Resistance**: It should be difficult to find two random messages **m3** and **m4**, where **m3 != m4**, such that **hash(m3) = hash(m4)**; due to the birthday paradox, it is possible to find two such random messages that hash to the same value much more readily than one may expected.

Secure Hashing (Cont.)

- Hash functions are used to compute the hash value, sometimes called **digest** or **fingerprint**, of some input data.
- Given **the hash size is fixed**, and much smaller than the data size in most cases, it is easier to store and protect the hash value, rather than the original data.
- Given same input m, the hash will always be the same h, and anybody with access to m can compute h.
- Common application of hashes is in authentication and integrity checking.
- Example: A hash can be computed for a large file, then the file can be sent to an untrusted storage while the hash is kept in a safe location. Later, when the large file is read again, its hash can be recomputed and checked against the stored value to make sure there was no modification to the file (note this does not protect against replay attacks, or if someone is able to change the hash value stored in the secure location).
- When checking integrity or authentication, freshness needs to be ensured; often a nonce is included as part of the hash (i.e., hash data concatenated with the nonce).

Example of a hash tree with four data nodes, showing internal hash nodes, and the root hash. The value of the root hash depends on all the data nodes' values.



Public Key Infrastructure

- **Public key infrastructure (PKI)** is a set of policies and protocols for managing, storing, and distributing certificates used in public-key encryption.
- In PKI, there is a trusted third party which can distribute digital certificates that vouch for correctness of public keys pk of different entities, and allows for verification and decryption of public-key encrypted data without having to directly talk to each sender to get their key.
- The **trusted third party** is the **certificate authority**.
- The authority is responsible for verification of the certificates that it receivers.
- It then distributes the certificates to other users, signed with its own private keys.
- Certificates for the certificate authorities are usually pre-distributed (e.g., browsers come with built-in list of certificates for certificate authorities).
- If there is a problem with a certificate authority, then **the PKI infrastructure will break**.
- Example: A user can by mistake trust a certificate authority they should not, or a malicious certificate authority could equivocate and give different information to different users, enabling man-in-the-middle type attacks.

Physical Unclonable Function

- Each secure processor should be uniquely identified and have a unique set of cryptographic keys.
- This can be achieved by "**burning in**" the unique information at the factory.
- However, such an approach requires extra cost (since each chip has to be written with the unique information), and a potentially malicious manufacturer may keep information about all the secret keys they have burned into their products.
- As an alternative, researchers have recently proposed **Physical Unclonable Functions (PUFs)**, and they could be used to generate unique information per-chip.
- PUFs leverage the unique behavior of a device, due to **manufacturing variations**, as **a hardware-based fingerprint**.
- A PUF instance is extremely difficult to replicate, even by the manufacturer.
- Many uses of PUFs have been presented in literature: **authentication and identification**, hardware-software binding, remote attestation, and secret key storage.
- PUFs can also be leveraged for random number generation as well as for recently proposed virtual proofs of reality.
- PUF implementation is mostly domain of hardware and circuit designers, but architects can leverage them in their security architectures.
- At the architecture level, PUFs are most often abstracted away as modules that **give a unique, unclonable fingerprint of the hardware**.

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 2: Basic Computer Security Concepts", Pages 5-23.

Questions?