Welcome to SOISP

Summary Of Information Security & Privacy

Publicly available at *soisp.perazzolo.ch*

Contents

1 Basic Concepts

1.1 Classes of Malwares

Malware = Malicious software

A Malware can satisfy more than 1 of the following.

Definition
Types of Malware

- **Virus**
  Virus are self-replicating malicious code. It automatically inserts its own code into other clean file.
- **Worm**
  Also self-replicate but across systems
- **Trojan**
  A piece of software that disguises itself as legitimate. Trojan are not self-replicating, meaning the user has to take action.
- **Rootkit**
  Hides itself, extremely hard to detect often control the os (i.e hide from taskmanager)
- **Ransomware**
  Encrypts files, ask ransom for decryption.
- **Nation state malware**
  A Malware designed by nation-state

Commodity Threat

Definition
Commodity Threat

- **Low Barrier to Entry**
- **Untargeted**
- **Generic Ressources**

1.2 Stages of Cyber Attack

Vulnerability / Exploit / Proof of Concept (PoC)

Definition
Vulnerabilities vs Exploits

Vulnerability A weakness in a system that can be exploited by a threat actor to perform unauthorized actions within a computer system.
Exploit A piece of code or technique that takes advantage of a vulnerability to cause unintended behavior in software, hardware, or electronic devices.
Proof of Concept (PoC) = A demonstration that shows how a vulnerability can be exploited, often used to illustrate the potential impact of the vulnerability.

Case study

Vulnerability, Exploit and PoC Example

- Vulnerability: A website's login form doesn't "sanitize" its input. It takes whatever a user types and sends it directly to the database.
- Proof of Concept (PoC): A researcher types ' OR 1=1 – into the username field. Instead of logging in, the website displays an error message that reveals the version of the database. This proves the database is listening to raw commands, but no data was stolen yet.
- Exploit: A hacker writes a script that uses that same flaw to automatically download the entire list of 50,000 user passwords. The script is the "exploit."

Zero-Day vs N-Day Vulnerability

Definition

Zero-Day vs N-Day Vulnerability

A Zero-Day Vulnerability = A software vulnerability that is unknown to those who should be interested in its mitigation (including the vendor of the target software). Since the vulnerability is not known, there are no patches or fixes available, making it a prime target for attackers. *A N-Day Vulnerability* = A software vulnerability that has been known for some time and for which patches or fixes are available. The term "N-Day" refers to the number of days since the vulnerability was disclosed.

2 Cryptography Basics

Definition
Goals of Cryptography

Confidentiality
Ensuring that information is accessible only to those authorized to have access.
Integrity
Maintaining and assuring the accuracy and completeness of data over its entire lifecycle.
Authentication
Verifying the identity of the author or source of information.
Non-repudiation
Ensuring that a party in a communication cannot deny the authenticity of their signature on a document or the sending of a message that they originated.

Naive approach

Using ceaser cipher, unfortunately as soon as you know the method, it only takes 25 attempts to brute force it. This scheme does not respect kirchoff's principle.

Kerckhoffs's Principle

Definition
Kerckhoffs's Principle

A cryptographic system should be secure even if everything about the system, except the key, is public knowledge.

One-time Pad

The one-time pad is perfectly secure but impractical, it requires a key as long as the message, truly random, used only once and securely shared.

2.1 Symmetric Cryptography

Stream Ciphers

A stream cipher use a PRNG to expand a short key into a long keystream, which is then combined with the plaintext bit by bit (often using XOR) to produce the ciphertext.
A $n - k$ bit stretch is stretching k bits into n bits, with $n > k$.

Limitations

Definition
Malleability

A property of some encryption schemes where an attacker can modify the ciphertext in a predictable way, resulting in a corresponding predictable change in the plaintext upon decryption.

To avoid malleability, we can use MAC (Message Authentication Code) to ensure integrity.

Definition
Key Reuse Problem

In symmetric cryptography, reusing the same key for multiple encryption sessions can lead to vulnerabilities, as it increases the risk of key compromise and makes it easier for attackers to analyze patterns in the ciphertext.

To avoid key reuse problem, we can use IV (Initialization Vector) which is a random value

Block Cipher.

Encrypts a fixed size block of data at a time (e.g., 128 bits).
a padding scheme is needed to fill the last block if the message is not a multiple of the block size.
A mode of operation is needed to encrypt messages longer than the block size.

Modes of Operation

Definition
ECB (Electronic Codebook) Mode

A mode of operation for block ciphers where each block of plaintext is encrypted independently using the same key. This can lead to patterns in the plaintext being visible in the ciphertext, making it less secure for certain applications. (We can see the pinguin)

Definition
CBC (Cipher Block Chaining) Mode

A mode of operation for block ciphers where each block of plaintext is XORed with the previous ciphertext block before being encrypted. This introduces dependency between blocks, enhancing security by ensuring that identical plaintext blocks produce different ciphertext blocks. The first block uses an Initialization Vector (IV) to ensure uniqueness. The IV is not secret, it can be prepended to the ciphertext. IV allows to avoid key reuse problem, we can reuse the same key safely if we use a different IV each time.

If a bit is flipped in the ciphertext the whole block is corrupted, but only one bit in the next block is corrupted.

To investigate : Padding oracle attacks

2.2 Data Integrity.

Integrity protects against accidental or intentional data modification.

Cryptographic Hash Functions

Definition
Properties of Hash Functions

Deterministic
The same input always produces the same output.
Fast to compute
Efficiently computes the hash value for any given input.
Pre-image resistance
Given a hash value, it should be computationally infeasible to find the original input.
Second pre-image resistance
Given an input and its hash, it should be computationally infeasible to find a different input that produces the same hash.
Collision resistance It should be computationally infeasible to find two different inputs that produce the same hash value.

3 Authentification and Access Control

3.1 Authentification

> *Definition*
> **Authentification**
>
> The process of verifying the identity of someone or something.

Identification is the act of identifying a subject. Authentication is the act of verifying that identification.

Flavors of authentification:
- Something you know (password, PIN)
- Something you have (smartcard, token)
- Something you are (biometrics: fingerprint, retina, face)

HOTP & TOTP

Hash based One Time Password (HOTP)
uses a counter that increments with each new password generation. The server and client must maintain synchronization of this counter to validate the OTP.

Time-based One Time Password (TOTP)
uses the current time as a moving factor to generate the OTP. Both the client and server must have synchronized clocks to validate the OTP, which typically changes every 30 or 60 seconds.

HOTP is less sensitive to time synchronization issues but requires maintaining a counter, while TOTP provides better security through time-based expiration but relies on accurate timekeeping between the client and server.

U2F (Universal 2nd Factor)

U2F differs from traditional two-factor authentication methods in several ways:
- **Phishing Resistance**: U2F is designed to be resistant to phishing attacks. The U2F device verifies the origin of the authentication request, ensuring that it is communicating with the legitimate service and not a fraudulent one.
- **Simplicity and Usability**: U2F provides a simple user experience. Users only need to insert their U2F device and press a button to authenticate, without the need to enter codes or passwords.
- **Public Key Cryptography**: U2F uses public key cryptography for authentication, which provides a higher level of security compared to traditional methods that rely on shared secrets or one-time codes.
- **No Shared Secrets**: Unlike traditional two-factor authentication methods that rely on shared secrets (like TOTP), U2F uses asymmetric cryptography, eliminating the risk of secret compromise.

Disadvantages:
- **Device Dependency**: Users must have their U2F device with them to authenticate, which can be inconvenient if the device is lost or forgotten.

Error rates for Biometric Authentification

See chapter on Nework security for base rate fallacy discussion

3.2 Kerberos

Kerberos is a network authentication protocol that uses secret-key cryptography to provide secure authentication for users and services in a distributed environment. It operates on the basis of "tickets" that allow users to prove their identity without transmitting passwords over the network.

Components

The main components of Kerberos are:

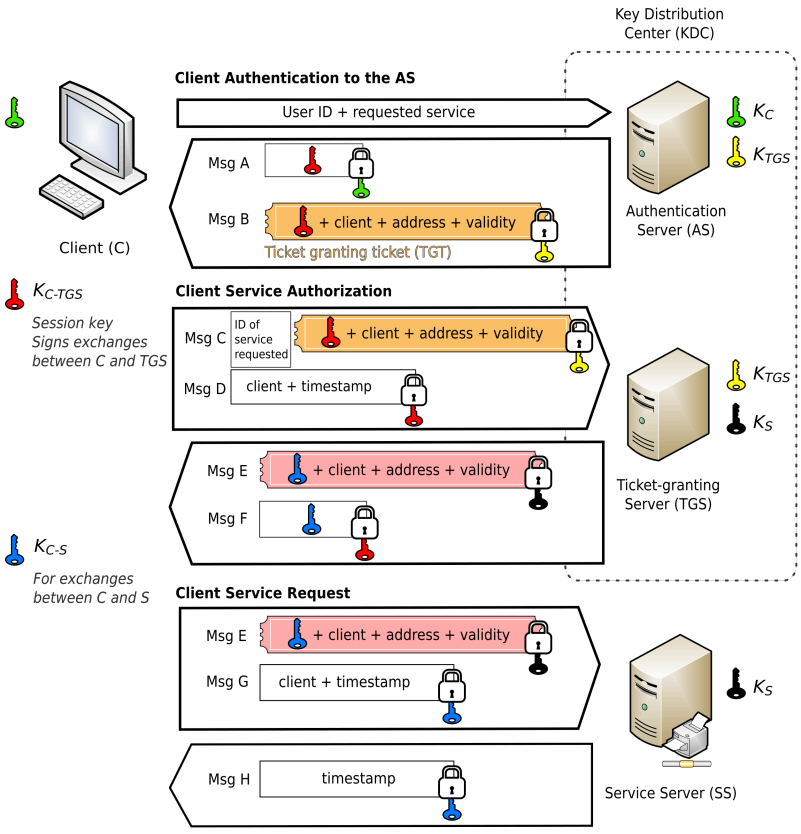
- **Key Distribution Center (KDC)**: The central authority that issues tickets. It consists of two main parts: the Authentication Server (AS) and the Ticket Granting Server (TGS).
  - **Authentication Server (AS)**: Responsible for verifying the identity of users and issuing Ticket Granting Tickets (TGTs).
  - **Ticket Granting Server (TGS)**: Issues service tickets based on the TGT presented by the user.
- **Client**: The user or service requesting access to resources.
- **Service Server (SS)**: The server providing the requested service.

Steps

The main steps in the Kerberos authentication process are:

1. **Initial Authentication**: The client sends a request to the AS for a TGT, providing its identity, via proving knowledge of a shared secret (usually a password).
2. **Ticket Granting Ticket (TGT) Issuance**: The AS verifies the client's identity and issues a TGT, encrypted with TGS's secret key, as well as a session key for communication between the client and TGS, this is encrypted with the client's secret key.
3. **Service Ticket Request**: The client uses the TGT to request a service ticket from the TGS for a specific service.
4. **Service Ticket Issuance**: The TGS verifies the TGT and issues a service ticket for the requested service along with a session key for communication between the client and the service server.

Flow Diagram



Kerberos Authentication Process

3.3 Delegated Authentification: OAuth2.0

3.4 Access Control

Definition
Access Control Models

- **Discretionary Access Control (DAC)**: Access rights are assigned based on the identity of users and/or groups. Owners of resources have the discretion to grant or revoke access to others.
- **Mandatory Access Control (MAC)**: Access rights are regulated by a central authority based on multiple levels of security. Users cannot change access permissions set by the system.
- **Role-Based Access Control (RBAC)**: Access rights are assigned based on roles within an organization. Users are granted permissions based on their role rather than their individual identity.

DAC: ACLs vs Capabilities

In Discretionary Access Control (DAC), Access Control Lists (ACLs) are used to specify which users or groups have access to specific resources. Each resource has an associated ACL that lists the permissions for different users. In contrast, capabilities are unforgeable tokens or keys that grant the holder specific access rights to a resource. Instead of checking an ACL, the system verifies whether the user possesses the appropriate capability to access the resource. The main difference is that ACLs are associated with resources and checked against user identities, while capabilities are associated with users and grant access to resources directly.

RBAC (Role-Based Access Control)

In Role-Based Access Control (RBAC), access permissions are assigned to roles rather than individual users. Users are then assigned to these roles, which determine their access rights within the system. This model simplifies management of permissions by grouping them into roles that reflect job functions or responsibilities within an organization.

MAC (Mandatory Access Control)

In Mandatory Access Control (MAC), access permissions are determined by a central authority based on predefined security policies. Users cannot modify access rights, and access decisions are made based on security labels assigned to both users and resources. This model is often used in environments where data sensitivity is high, such as military or government systems.

Mac and Confidentiality: Bell-LaPadula Model

Rules:

- **Simple Security Property (no read up)**: A subject at a given security level cannot read data at a higher security level.
- **Property (no write down)**: A subject at a given security level cannot write data to a lower security level.
- **Strong Star Property**: A subject can only read and write at the same security level.

The Bell-LaPadula model focuses on maintaining the confidentiality of information by enforcing strict access controls based on security levels. It prevents unauthorized disclosure of sensitive information by ensuring that users can only access data appropriate to their clearance level.

Mac and Integrity: Biba Model

Rules:

- **Simple Integrity Property (no read down)**: A subject at a given integrity level cannot read data at a lower integrity level.
- **Property (no write up)**: A subject at a given integrity level cannot write data to a higher integrity level.

The Biba model focuses on maintaining the integrity of information by preventing unauthorized modification. It ensures that users can only read data from sources of equal or higher integrity and can only write data to sources of equal or lower integrity.

4 Data Security

4.1 Database access Control

SQL Access Control

Most SQL database implement some sort of discretionary access control. In general, they support the GRANT and REVOKE commands to give and take away privileges on database objects such as tables, views, and stored procedures.

Views are a common way to restrict access to specific columns or rows in a table. For example, a view can be created to show only non-sensitive columns of a table or to filter rows based on certain criteria.

Example of view :

```sql
CREATE VIEW public_employee AS
SELECT name, department
FROM employees
WHERE department != 'HR';
```

Example of GRANT :

```sql
GRANT SELECT ON public_employee TO user_readonly;
```

Note that SQLlite does not support access control features like GRANT and REVOKE, instead it relies on file system permissions to control access to the database file itself. This does not allow for fine-grained access control within the database.

SQL also supports role-based access control (RBAC) where roles can be created with specific privileges and then assigned to users. we GRANT to a role, and then assign that role to users.

```sql
CREATE ROLE prof;
GRANT SELECT (name, grade, academic_year) ON com402.students TO prof;
GRANT UPDATE (grade) ON com402.students TO prof;
GRANT prof TO alice;
```

Network access to Databases

Databases can be accessed over a network using various protocols such as TCP/IP, HTTP, or proprietary protocols specific to the database system. To secure network access to databases, several measures can be implemented: Accept connections only from trusted IP addresses or ranges using firewall rules. Either a network firewall or a host-based firewall can be used to restrict access to the database server. We can also use the syntax below to restrict access to specific IP addresses or ranges.

```sql
GRANT SELECT ON com402.students TO 'user'@'192.168.1.%
```

SQL Injection

SQL injection is a code injection technique that exploits vulnerabilities in an application's software by inserting malicious SQL statements into input fields or parameters. This can allow attackers to gain unauthorized access to the database, modify or delete data, or even execute arbitrary commands on the server.

To prevent SQL injection use prepared statements with parameterized queries:

```python
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
user_id = input("Enter user ID: ")
# Using a parameterized query to prevent SQL injection
```

```
cursor.execute("SELECT * FROM users WHERE id = ?", (user_id,))
results = cursor.fetchall()
```

This is called Parameterized Queries (also known as Prepared Statements or Bind Variables). Since the SQL code is defined separately from the user input, the database can distinguish between the two and prevent any malicious input from being executed as part of the SQL statement.

4.2 Salting Passwords

Prevents the attackers from precomputing hashes for common passwords. Also forces the attackers to compute the hash for each password guess individually, which is computationally expensive and time-consuming.

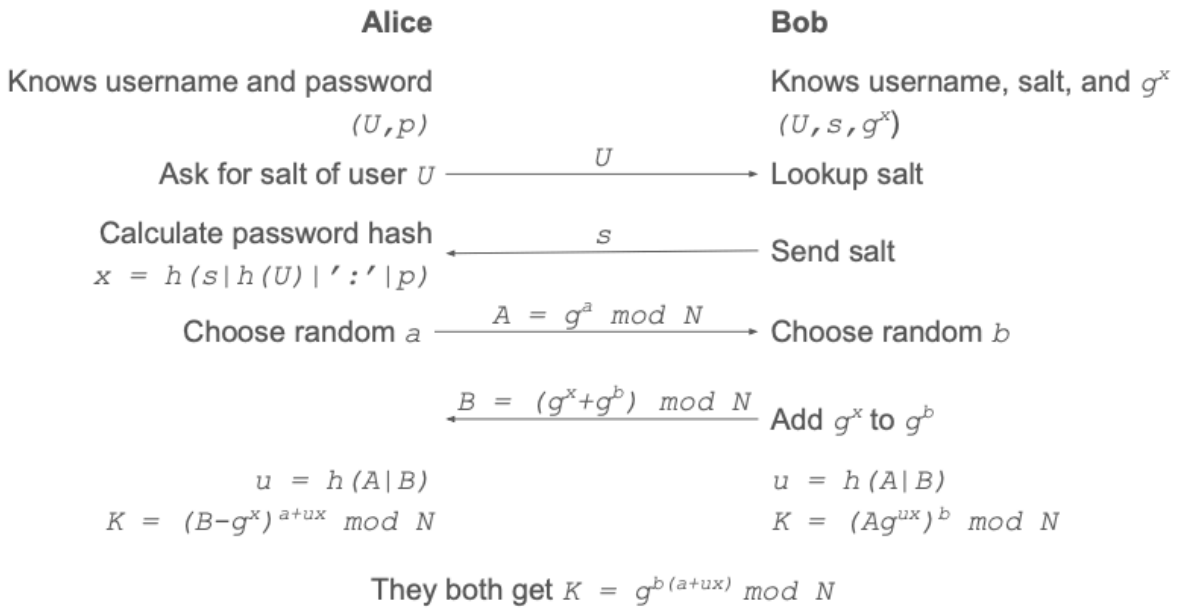
4.3 Rainbow Tables

A rainbow table is a precomputed table used to reverse cryptographic hash functions, primarily for cracking password hashes. Rainbow tables are used to perform a time-memory trade-off attack, where the attacker precomputes the hashes of a large number of possible passwords and stores them in a table. When the attacker obtains a password hash, they can look it up in the rainbow table to find the corresponding plaintext password.

4.4 Secure Remote Protocols (SRP)

A zero knowledge proof authentication protocol that allows a client to prove its identity to a server without revealing its password.

Overview



Secure Remote Protocol (SRP) Overview

5 Programming Language Security

5.1 Type Safety through type systems

A type system is a set of rules that assigns a property called "type" to the various constructs of a computer program, such as variables, expressions, functions or modules. The main purpose of a type system is to reduce bugs in computer programs by defining interfaces between different parts of a program and checking that the parts fit together correctly.
A sound type system guarantees that a program cannot perform an operation on a value that is not appropriate for that value's type.Type safety is a property of a programming language that ensures that operations are performed on data types in a manner consistent with their definitions.

Timing of Type Checking

Definition
Static typing

Type checking occurs during compilation before the program ever runs, requiring variables to have a fixed type.

Advantages
- No run-time overhead
- Find errors early
- Find more errors
- Better for reliability and security

Definition
Dynamic typing

Type checking is performed at runtime, meaning variables are associated with values rather than fixed types and can change during execution.

Advantages
- More flexible (Less boilerplate, overheading in type definitions)
- Better for scripting, rapid prototyping

Inference of Types

Declared Types

Types are explicitly declared by the programmer.

Hindely Milner Type Inference

Declare types for function arguments & return types. Let the compiler infer the rest.

Inferred Types

Types are inferred by the compiler based on the code.

Most modern "industrial" languages are Declared by default but offer Local Inference as a convenience. Truly Inferred (Hindley-Milner) systems are mostly found in the functional programming world

Strictness of Type Checking

Definition
Strict typing

No Implicit Conversions Between Types
The language refuses to mix types. If you try to add a Number to a String, the program crashes or fails to compile. It forces you to convert the data manually.

Definition
Relaxed/Loose typing

Implicit Conversions Between Types
The language automatically converts between types when necessary. For example, adding a Number to a String would convert the Number to a String and concatenate them.

Type Safety and security

Is the only automated mechanism widely used. Language are still turing complete with type systems. Type system finds real errors : No false positives.

Case study
C Type System

1. **Static Typing**: C is statically typed, meaning types are checked at compile time. This helps catch type-related errors before the program runs.

```c
int main() {
    int x = 10;
    x = "Hello"; // Compiler Error: incompatible types when assigning to type 'int' from type 'char *'
    return 0;
}
```

2. **Declared Types**: The programmer must explicitly state the type of every variable, function argument, and return value

```c
float add(float a, float b) { // Types must be explicitly declared
    return a + b;
}
```

3. **Relaxed/Weakly Typed**

allows for many implicit conversions and, more importantly, allows the programmer to bypass the type system entirely using pointers and type casting.

```c
int main() {
    int i = 65;
    char c = i; // Implicit conversion from int to char (Relaxed)

    // Manual override: treating an int as a float pointer
    float *f = (float *)&i;
    return 0;
}
```

4. **Unsound**, we can treat memory containing int as as pointer ect...

Case study
java

1. **Static Typing** As for C
2. **Declared with Local Inference**

```java
// Declared Type (Traditional)
String name = "Gemini";

// Local Inference (Modern Java)
var age = 25; // The compiler infers 'int' based on the value 25
```

3. **Strict Typing** Stricter than C, refuses to perform implic conversion.

```java
double pi = 3.14;
int roundedPi = pi; // Compile-time error: possible lossy conversion

int fixedPi = (int) pi; // Manual conversion required (Strict)
```

Case study
Python

Python is **Dynamic**, types are **inferred** the moment an assignement happens.
It is **Strictly Typed** python will refuse to perform operations like adding a number to a string. (Unlike C)

Case study

Rust

Combines Speed of C with soundness that exceeds even Java.

1. **Static** If doesn't pass Borrow Check at compile time it fails.
2. **Inferred (Hindley-Milner Style)** You rarley have to write types but compiler knows exactly what they are.
3. **Strict** No implicit conversions is allowed

```rust
fn main() {
    let x = 10;          // Inferred as i32 (integer)
    let y: f64 = 5.0;    // Explicitly Declared

    // let sum = x + y;  // ERROR: cannot add `i32` to `f64` (Strict)
    let sum = x as f64 + y; // Manual conversion required
}
```

It is extremly sound, the ownership system ensure type safety (also memory safety)

5.2 Memory Safety

Code can only access data within live regions of memory whose pointer is properly obtained. Divided into two categories: **Spatial Safety** (in-bounds access) and **Temporal Safety** (access to valid objects).

Common memory safety violations includes :
- Buffer Overflows (Spatial Safety Violation)
- Use-After-Free (Temporal Safety Violation)
- Double Free (Temporal Safety Violation)
- Dangling Pointers (Temporal Safety Violation)

Safety Mechanisms

Bounds Checking (check withing arrays)
Lifetime tracking (No double free, no use after free)
Automatic Memory Management (Garbage Collection, Reference Counting, Smart Pointers/Borrow checking)

Note : Adding Bounds checking to C causes too much overhead, due to its nested pointers in part.

Rust achieves memory safety without Automatic Memory Management through its ownership model and borrow checker. At compile time Rust will check and reclaim memory that is no longer in use, preventing memory leaks and dangling pointers. With Automatic Memory Management, there is a runtime cost for tracking memory usage and reclaiming memory, which can lead to performance overhead and latency issues.

Automated Storage Reclamation

Designed to help with temporal safety.

Reference Counting

Reference Counting is immediate, as soon as the count is zero, memory is reclaimed. But it cannot handle cyclic references.

Garbage Collection

Is run when the system is idle or memory is low. (So periodically) Can handle cyclic referenfces Use tracing to find reachable objects. and reclaim unreachable objects. Tracing with mark and sweep consists of two phases:
- Mark phase: Traverse the object graph starting from root references, marking all reachable objects.
- Sweep phase: Iterate through the memory, reclaiming unmarked objects.

May be less expensive than reference counting in some scenarios, but can introduce latency due to pause times during collection.

C++ Smart pointers

`unique_ptr`: Single ownership, cannot be copied, only moved. Automatically deletes the object when it goes out of scope. `shared_ptr`: Shared ownership, multiple pointers can point to the same object. Uses reference counting to delete the object when the last shared_ptr goes out of scope. `weak_ptr`: Non-owning reference to an object managed by shared_ptr. It does not affect the reference count and is used to prevent cyclic references.

Rust Borrow Checker

A value can have only one owner. When the owner goes out of scope, the memory is freed, every thing is a unique_pointer.
In C++ you have to opt-in to use smart pointers, in Rust its the default.
Then it would be annoying to write code if you always had to transfer ownership.
This is why Rust has borrowing.
Borrowing allows multiple references to a value without transferring ownership.
But there are rules: You can have either one mutable reference (&mut T) or any number of immutable references (&T) at a time. This sometimes creates difficulities like in double linked lists.

Unsafe Code In rust Rust allows unsafe code blocks where the borrow checker rules can be bypassed. Put unsafe code in module and wrap it in safe abstractions. Almost all non-trivial Rust code uses some unsafe code.

5.3 Thread Safety

To avoid concurrent bug such as Race Conditions, Deadlocks, Starvation.

Mechanisms

- Global :
  - Locks (Mutex, Read-Write Locks)
  - Atomic Operations
- **Specific to Language** :
  - Java: synchronized keyword, java.util.concurrent package
  - Go: goroutines and channels
  - Rust : ownership and borrowing rules, std::sync module
    In rust owner lifetime must outlive the thread lifetime. No use after free across threads. No data races thanks to the borrow checker.

5.4 Sandboxing & Compartmentalization

Isolating untrusted code to limit its access to system resources and sensitive data. Follow the principle of least privilege.

Sandboxing : Run untrusted code in a restricted environment with limited permissions.

Compartmentalization : Divide the system into isolated compartments with strict access controls between them.

6 Web and Application Security

6.1 SQL Injection

Symbols	Purpose
;	To separate queries
--	To comment out the rest of the query, often critiical to not cause syntax errors
'	To terminate strings
UNION	To combine results from multiple queries
1=1	A tautology that always evaluates to true
DROP TABLE	To delete entire tables
AND/OR/ NOT	To combine or negate conditions

Prevention Techniques:
- Use Prepared Statements (Parameterized Queries)

6.2 LDAP Injection

Very similar to SQL Injection but targets LDAP queries used for directory services.

Symbols	Purpose
*	To match any value
&	To combine multiple conditions
|	To provide alternative conditions
!	To negate a condition
()	To group conditions

Prevention Techniques:

- Input Validation and Sanitization
- Use of Parameterized LDAP Queries

> *Side Quest*
> **SQL-LDAP Translation**
>
> *SQL* : `(status='active' AND type='user') OR department='sales'`
>
> *LDAP* : `(|(&(status=active)(type=user))(department=sales) )`

6.3 Cross-Site Scripting (XSS)

Javascript code injected into web pages.
- **Reflected XSS**: Malicious script is reflected off a web server, such as in an error message or search result.
- **Stored XSS**: Malicious script is permanently stored on the target server, such as in a database.
- **DOM-based XSS**: The vulnerability exists in the client-side code rather than server

Prevention Techniques:
- Input Validation and Sanitization
- Encode data i.e with backslash

6.4 Cookie Security

To prevent user from tampering their session data (i.e changing *role=user* to *role=admin*) we can use HMAC to sign with a secret key only known by the server the data and append the tag to the cookie.

Note that in this course only stateless session management is covered (i.e all session data stored in the cookie), usually encoded in base64 to be ASCII safe.

6.5 Stack Overflow

A stack overflow occurs when a program writes more data to a buffer located on the stack than what is allocated for that buffer. This can overwrite adjacent memory, including control data such as return addresses, leading to undefined behavior, crashes, or security vulnerabilities.

General Notion to know

- **RIP**
  is the re-extended instruction pointer register that holds the address of the next instruction to be executed.
- **Call frame order**
  See below
- **Low endian**
  The least significant byte is stored at the lowest memory address.
  This is not the human readable format, so when reading memory dumps we have to reverse the byte order.

Defense Techniques

W^X / DEP

Write XOR Execute / Data Execution Prevention

Mark memory regions as non-executable, a memory page can be either writable or executable, but not both at the same time.
This preventes shellcode injection attacks where the attacker injects code for example into the stack and then diverts execution to that code.

Stack canaries

Special values placed on the stack that, if altered, indicate a buffer overflow has occurred.

Address Space Layout Randomization (ASLR)

Randomize memory addresses to make it harder for attackers to predict where code and data are located. Stack canaries in C are put Stored Base pointer and the local variables.

Stack frame layout with stack canary on x64/86

Lower Memory Addresses

Stack Component	Role & Notes	Typical Access (x64) *Relative to RBP*

Local Variables	Space for variables declared inside the function (arrays, integers, structs).	`[rbp - 0x20]`
		`[rbp - 0x14]`
		(Negative Offsets)

Stack Canary	**SECURITY GUARD**	`[rbp - 8]` (Usually 8 bytes below RBP)
Stored Base Pointer (Saved RBP)	**FRAME START**. The anchor point. It holds the address of the **previous** function's stack frame so it can be restored later.	`[rbp]` (The address RBP currently holds)
Return Address	The memory address the CPU must jump back to once the current function finishes executing.	`[rbp + 8]` (Positive Offset)

Arguments (on stack)	Parameters passed by the caller. *Note: In x64, the first few arguments usually go into registers, leftovers go here.*	`[rbp + 16]`
		`[rbp + 24]`
		(Larger Positive Offsets)

6.6 Return-Oriented Programming (ROP)

Return-Oriented Programming (ROP) is an advanced exploitation technique that allows an attacker to execute arbitrary code by chaining together small sequences of existing code, called "gadgets," that end with a return instruction. This technique is often used to bypass security mechanisms like Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR).

Code snippets can also be found in shared libraries (like libc) loaded into the process memory.

Modern tools like pwntools can help automate the process of finding gadgets and constructing ROP chains, even when ASLR is enabled.

7 Automated Testing

7.1 Classification of Testing Approaches

Manual, Semi-Automated, and Fully Automated Testing

Manual testing relies on human effort to identify issues,
semi-automated testing uses tools to assist humans, and
fully automated testing employs tools to perform tests without human intervention.

Static vs Dynamic Testing Approaches

Static testing analyzes code without executing it (e.g., static code analysis, formal verification).
Dynamic testing involves executing the code (e.g., unit testing, integration testing, fuzz testing).
Hybrid : testing combine both static and dynamic approaches (e.g., concolic testing).

Soundness vs Completeness

Sound : Testing methods that guarantee no false negatives (i.e., if a vulnerability exists, it will be found).
Complete : Testing methods that guarantee no false positives (i.e., if a vulnerability is reported, it definitely exists).

Only Formal Verification is sound and complete.
Unit testing is sound but not complete.
Fuzz testing is neither sound nor complete.

7.2 Manual Testing

Code Reviews Inspection Walkthroughs Auditing
Unit/Regression/Integration/User Acceptance Testing

7.3 Semi-Automated Testing

1. Static Code Analysis

7.4 Automated Testing

- Formal verification (Using mathematical methods to prove the correctness of algorithms)

Formal Verification (*static*)

Mathematical techniques to prove the correctness of algorithms with respect to a formal specification or property.
Can be both sound and complete, meaning it can guarantee the absence of certain types of errors (soundness) and can prove the presence of certain properties (completeness).

Model Checking (*static*)

explores states mathematically without running the binary in a live environment

Bounded Model Checking

Restricts in depth / number of states explored.

Symbolic Execution (*static*)

Creates a mathematical model of the system, exploring all possible execution paths by treating inputs as symbolic values rather than concrete data.
Still static as it does not run the program with actual inputs.

Concolic Testing (*Dynamic*)

hybrid of concrete and symbolic execution. Runs the program with concrete inputs (Dynamic) to trace a path, then uses symbolic execution (Static) to solve constraints and generate new inputs to explore uncovered paths.
Therefore, it must be classified as hybrid or at least involve dynamic execution.

Fuzzer (*dynamic*)

Role: To automatically generate and send a large number of inputs to a target program to discover vulnerabilities, crashes, or unexpected behaviors.
Synergy: Can be combined with Sanitizers to detect violations during execution.

Common Types:

- **Mutation-based Fuzzers**: Modify existing valid inputs to create new test cases (e.g., AFL, libFuzzer).
- **Generation-based Fuzzers**: Create inputs from scratch based on a specification or protocol (e.g., Peach Fuzzer).

Fuzzing Approaches:

Blackbox Fuzzing: No knowledge of the internal workings of the target program, just throw random inputs and wait for crash
Whitebox Fuzzing: Uses knowledge of the internal structure of the target program to guide input generation. Use heavy math to analyze every branch
Greybox Fuzzing: Combines aspects of both blackbox and white box fuzzing, often using lightweight instrumentation to gather feedback from the target program.

Case study
AFL (American Fuzzy Lop)

In the Lab we used AFL, a popular mutation-based greybox fuzzer.
It uses genetic algorithms to evolve inputs based on code coverage feedback. It instruments the target program to monitor which parts of the code are executed, allowing it to focus on areas that are more likely to contain vulnerabilities.

1. It modifies existing valid "seed" files rather than building inputs from scratch.
2. It uses code coverage feedback to "see" which paths are explored without full logic analysis.

7.5 Sanitatizers (*Dynamic*)

Act as an oracle to make "invisible" bugs observable (usually by forcing a crash).

Synergy: Highly effective when combined with Fuzzing. The Fuzzer generates the input to reach a code path, and the Sanitizer ensures any violation on that path is detected.

Common Types:

- **AddressSanitizer (ASan)**:
  Detects memory errors like out-of-bounds access (buffer overflows) and use-after-free.
- **ThreadSanitizer (TSan)**:
  Detects data races in multi-threaded applications.
- **UndefinedBehaviorSanitizer (UBSan)**:
  Detects undefined behaviors like integer overflows or null pointer dereferences.
- **MemorySanitizer (MSan)**:
  Detects reads of uninitialized memory.

Standard sanitizers (like ASan) add checks during compilation.
The tools listed below achieve the same goal on already compiled binaries using different techniques:

- Valgrind's Memcheck is similar to ASan but works on binaries without recompilation, using dynamic binary instrumentation.
- QASan is another tool that can analyze binaries for memory errors, similar to Valgrind and ASan, but it uses a different approach to instrumentation and analysis.

Since not all bug crashes, Sanitizers enforce security policies at runtime to ensure that violations lead to observable failures.

7.6 Comparison of Techniques

Technique	Scale (LoC)	False Negatives	False Positives	State Explosion
Formal Verification	100	None	No	High
Bounded Model Checking	1,000	Low	No	High
Symbolic Execution	10,000	None	No	High
Concolic Execution	50,000	Medium	No	Medium
Fuzzing	1,000,000	Medium	No	Low
Compiler Warning	100,000,000	High	Yes	No

Bounded Model Checking scales Formal Verification by unrolling loops up to a certain bound (limiting precision)

Concolic execution scales Symbolic Execution by following a concrete path (limiting precision)

BMC, Concolic Execution, and Fuzzing may miss bugs in areas they don't analyze/cover

8 Mobile Security

Android Compilation

Android applications are typically written in Java or Kotlin and compiled into an intermediate bytecode format known as Dalvik bytecode. This bytecode is then executed by the Android Runtime (ART) or the older Dalvik Virtual Machine (DVM) on Android devices. The compilation process involves several steps:

1. **Source Code Compilation**: The Java or Kotlin source code is compiled into Java bytecode using the Java Compiler (javac) or Kotlin Compiler (kotlinc).
2. **Bytecode Conversion**: The Java bytecode is then converted into Dalvik bytecode using the `dx` tool (for older versions of Android) or the `d8` tool (for newer versions).
3. **Packaging**: The Dalvik bytecode, along with resources and manifest files, is packaged into an APK (Android Package) file, which is the format used for distributing and installing Android applications.

Android Application Components

- **/lib** : Contains compiled native libraries (shared libraries) used by the application.
- **/res** : Contains application resources such as images, layouts, and strings.
- **/assets** : Contains raw asset files that the application can read at runtime.
- **AndroidManifest.xml** : The manifest file that describes essential information about the application, including its components, permissions, and hardware requirements.
- **classes.dex** : The Dalvik bytecode file that contains the compiled code for the application.
- **resources.arsc**: A binary file that contains precompiled resources used by the application.
- **META-INF/** : Contains metadata about the APK, including the signature of the application.

About native libraries

Native libraries are compiled code written in languages like C or C++ that can be executed directly by the device's CPU. They are typically used for performance-critical tasks or to access low-level system features that are not available through the Android SDK. Native libraries are packaged in the `/lib` directory of the APK and are specific to the architecture of the device (e.g., ARM, x86). For security analysis, native libraries are hard to analyze compared to Dalvik bytecode.

Android Security Model

Android employs a multi-layered security model to protect user data and ensure the integrity of applications.

1. **Application Sandbox**: Each Android application runs in its own sandboxed environment, isolated from other applications. This prevents unauthorized access to an application's data and resources.
2. **Permissions**: Android uses a permission-based model to control access to sensitive resources and data. Applications must declare the permissions they require in the manifest file, and users must grant these permissions at install time (or at runtime for certain permissions).
3. **Application Signing**: All Android applications must be digitally signed with a certificate before they can be installed. This ensures the authenticity and integrity of the application.
4. **Security Updates**: Android devices receive regular security updates to address vulnerabilities and improve overall security.
5. **Google Play Protect**: A built-in security feature that scans applications for malware and other threats before and after installation from the Google Play Store.

8.1 Attack Vectors on Android

Repackaging an APK

Repackaging an APK involves modifying an existing APK file, typically to inject malicious code or alter the application's behavior. The general steps for repackaging an APK are as follows:

1. **Extract the APK**: Use a tool like `apktool` to decompile the APK and extract its contents.
2. **Modify the Code**: Make the desired changes to the application's code, resources, or manifest file. This may involve injecting malicious code or altering existing functionality.
3. **Rebuild the APK**: Use `apktool` to recompile the modified files back into an APK.

4. **Sign the APK**: Since the original signature is invalidated by the modifications, the repackaged APK must be signed with a new certificate using tools like `jarsigner` or `apksigner`.
5. **Install the APK**: The repackaged APK can now be installed on an Android device for testing or distribution.

Prevention

To prevent APK repackaging attacks, developers can use techniques such as code obfuscation, integrity checks, and secure communication protocols. Users should also be cautious when installing applications from untrusted sources.

- Obfuscation: Making the code harder to understand to deter reverse engineering, concretly for java/kotlin code :
  - Renaming classes, methods, and variables to meaningless names.
  - Removing debug information and comments.
  - Encrypting strings and resources used in the application.
- Integrity Checks: Implementing checks within the application to verify that its code and resources have not been tampered with. This can include:
  - Verifying the application's signature at runtime.
  - Using checksums or hashes to validate the integrity of critical files.
- Secure Communication: Ensuring that all communication between the application and external servers is encrypted using protocols like HTTPS
- User Education: Encouraging users to install applications only from trusted sources, such as the Google Play Store, and to be cautious of applications requesting excessive permissions.
- Play Store Protections: Google Play Store employs various security measures, including malware scanning and app review processes, to detect and prevent the distribution of repackaged or malicious applications.

9 Network Security

9.1 Certificate pinning

Certificate pinning is a security mechanism used to ensure that a client only accepts a specific set of trusted certificates when establishing a secure connection to a server. This technique helps to prevent man-in-the-middle (MITM) attacks by limiting the certificates that a client will accept, even if they are otherwise valid and trusted by a certificate authority (CA). Certificate pinning can be implemented in several ways, including:

- **Public Key Pinning**: The client stores a hash of the server's public key and only accepts certificates that match this hash.
- **Certificate Pinning**: The client stores the entire server certificate and only accepts that specific certificate.

Certificate pinning is commonly used in mobile applications and web browsers to enhance security when connecting to sensitive services, such as banking or e-commerce websites. However, it can also introduce challenges, such as the need to update pinned certificates when they expire or are replaced.

Case study

Attack without Certificate Pinning

Consider a scenario where a user is trying to connect to their bank's website over HTTPS. Without certificate pinning, an attacker could perform a man-in-the-middle (MITM) attack by intercepting the user's connection and presenting a fraudulent certificate that appears to be valid.
The user, trusting the certificate authority (CA) that issued the fraudulent certificate, would proceed with the connection, allowing the attacker to eavesdrop on sensitive information such as login credentials and financial data.
However, if the bank's website implemented certificate pinning, the user's browser or application would only accept the specific certificate or public key pinned by the bank.
In this case, the fraudulent certificate presented by the attacker would not match the pinned certificate, and the connection would be rejected, thereby preventing the MITM attack and protecting the user's sensitive information.

9.2 Proxy Servers

A proxy server acts as an intermediary between a client and a server, forwarding requests from the client to the server and returning the server's responses back to the client. Proxy servers can be used for various purposes, including improving performance, filtering content, and enhancing security. There are several types of proxy servers, including:

- **Forward Proxy**: A forward proxy is used by clients to access resources on the internet. It can cache content, filter requests, and hide the client's IP address.
- **Reverse Proxy**: A reverse proxy is used by servers to handle incoming requests from clients. It can distribute requests across multiple servers, provide load balancing, and enhance security by hiding the server's IP address.

9.3 Firewalls

A firewall is a network security device or software that monitors and controls incoming and outgoing network traffic based on predetermined security rules. Firewalls are used to protect networks and systems from unauthorized access, cyberattacks, and other security threats.

9.4 Demilitarized Zones (DMZ)

Definition
Demilitarized Zone (DMZ)

A Demilitarized Zone (DMZ) is a physical or logical subnetwork that contains and exposes an organization's external-facing services to an untrusted network, typically the internet. The purpose of a DMZ is to add an additional layer of security to an organization's local area network (LAN); an external attacker only has access to equipment in the DMZ, rather than the entire network.

Case study
An Online Sotre with DMZ

Imagine a company that operates an online store. To enhance security, they set up a DMZ to separate their public-facing web servers from their internal network.
The DMZ contains the web servers that handle customer requests, process orders, and display product information. These servers are accessible from the internet but are isolated from the internal network where sensitive data, such as customer information and payment details, are stored.
A firewall is placed between the DMZ and the internal network, allowing only specific types of traffic to pass through. For example, the web servers in the DMZ can communicate with the internal database server to retrieve product information and process orders, but direct access to the internal network from the internet is blocked.
This setup helps to protect the internal network from potential attacks originating from the internet, as any compromise of the web servers in the DMZ does not directly expose the internal network.

9.5 Base rate fallacy

For the exam always transform rate into numbers big enough

1. Extend the rate :
   - TPR = sensitivity = 1 - FNR
   - TNR = specificity = 1 - FPR
2. Apply to a population of size N
3. Compute the actual numbers of TP, FP, TN, FN

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">Disease</td><td rowspan="2">Predictive Value</td><td rowspan="2"></td></tr>
<tr><td>⊕</td><td>⊖</td></tr>
<tr><td rowspan="2">Test</td><td>⊕</td><td>A
True Positive (TP)</td><td>B
False Positive (FP)</td><td>Positive Predictive Value (PPV)
$\frac{TP}{TP+FP} = \frac{A}{A+B}$</td><td>Total Positive Results (A + B)</td></tr>
<tr><td>⊖</td><td>C
False Negative (FN)</td><td>D
True Negative (TN)</td><td>Negative Predictive Value (NPV)
$\frac{TN}{FN+TN} = \frac{D}{C+D}$</td><td>Total Negative Results (C + D)</td></tr>
<tr><td colspan="2" rowspan="2">Sensitivity & Specificity</td><td>Sensitivity
$\frac{TP}{TP+FN} = \frac{A}{A+C}$</td><td>Specificity
$\frac{TN}{FP+TN} = \frac{D}{B+D}$</td><td colspan="2" rowspan="2"></td></tr>
<tr><td>All diseased patients (A + C)</td><td>All non-diseased patients (B + D)</td></tr>
</table>

9.6 Intrusion Detection Systems (IDS)

Signature based

Compares network traffic against signatures from a pattern database, like a WAF does for web traffic

- false negative for unknown attacks
- low false positive rate but still possible

Anomaly based

- Detect unknown attacks
- Higher false positive rate
- Requires training phase

10 Trusted Execution Environments Side Channels

We cannot trust the software due to the complexity of modern software. We also cannot trust the hardware, it comes from untrustworthy manufacturers and can have backdoors.

Definition
Trusted Hardware

A piece of hardware can be trusted if, it always behaves in the expected manner for the intended purpose.

10.1 Key properties of Trusted Hardware:

Attestation

Definition
Attestation

Proves that the system is in a specifc state to an authorised party.
Ability to prove to a remote party that the hardware is in a specific state.

Isolation

Definition
Isolation

Constraints who and what can be accessed.
Code and data are isolated from the rest of the system.

Remark: Unauthorized parties cannot access the data or code. Trusted hardware offers one well-identified interface to access the data. The TCB (Trusted Computing Base) is the set of all hardware, software, and controls that enforce the isolation. Only the TCB can access the data. well defined means

Sealing

Definition
Sealing

Protects private infromation by binding it to platform configurations.
Data can be stored such that it can only be accessed when the system meets specific trusted conditions.

These forms the basis of Trusted Execution Environments (TEE).

10.2 Trusted Hardware Examples

Any hardware that has been certified to perform according to a certain set of requirements under strong adversarial conditions.

Side Quest
Boot cold attack

Boot cold attack is a type of physical attack on a computer system where an attacker with physical access to the machine can extract sensitive information, such as encryption keys or passwords, from the system's memory (RAM) by exploiting the fact that data in RAM can persist for a short period of time after the system is powered off. This is typically done by quickly rebooting the system into a special mode or using external hardware to read the contents of the memory chips directly.

TPM Chip

A small dedicated microcontroller designed to secure hardware through integrated cryptographic keys.

- It can do resaonably fast cryptographic operations.
- TPM has some append only registers called Platform Configuration Registers (PCRs) that can store measurements of software and hardware components.
  - PCRnew = H(PCRold || measurement)

Case study

Measured Boot & TPM Sealing

To detect tampering, the TPM acts as a passive notary. Each stage of the boot process (Firmware → Bootloader → Kernel) measures the next stage by sending its hash to the TPM's PCRs before executing it.

The TPM itself does not "check" these values or halt the boot; that is the job of Secure Boot, which uses digital signatures to verify and stop the process if code is unsigned. The TPM's role is strictly to record the "measurements."

The check happens later during Sealing. When the OS requests an encryption key, the TPM compares the live PCR values against the "healthy" state saved in the secret's policy. If the BIOS or Bootloader was altered, the measurements won't match, and the TPM will refuse to unseal the secret, protecting the data.

Case study

Remote Attestation & Trust

To prove a system's health to a remote server (the Verifier), the TPM acts as a hardware witness. Instead of just "checking in," the Verifier sends a random nonce (challenge) to the computer to ensure the response is fresh and not a replay of a previous successful boot.

The TPM creates a Quote: a data bundle containing the current PCR measurements and the nonce, all digitally signed by the TPM's internal Attestation Key (AK). This signature proves the data came from a genuine, physical TPM and hasn't been modified by the OS.

The Verifier (not the TPM) performs the final check. It inspects the Quote and compares the PCR hashes against its own database of "known-good" configurations. If the measurements match the corporate policy, the Verifier grants access to the network; if a rootkit or unapproved kernel is detected, the server denies the connection.

Hardware Security Module (HSM)

External physical device that manages digital keys for strong authentication and provides cryptoprocessing.

"While the TPM focuses on the integrity of a single device, the HSM focuses on the lifecycle of high-value keys for thousands of applications" - Gemini

- HSM are tamper-resistant and often tamper-evident. The system will flush the keys if tampering is detected.
  - *Remark*: How to flush rapidly, since everything is encrypted, just delete the master key.

Common Uses for HSM

- Payment Processing: When you swipe a credit card, an HSM in the bank's data center verifies your PIN and encrypts the transaction.
- Public Key Infrastructure (PKI): The "Root Certificate" that makes the entire internet trust HTTPS websites is almost always stored inside an HSM.
- Code Signing: Large software companies (like Microsoft or Apple) keep the keys used to sign their OS updates in HSMs to ensure no single disgruntled employee can steal the key and sign a virus.

Case study

HSM Case Study: iPhone Cloud Backups

To ensure End-to-End Encryption for backups, Apple uses a cluster of specialized HSMs. The goal is to allow a user to recover their data via a passcode while preventing Apple or anyone else from brute-forcing it.

1. When you enable encrypted backup, your iPhone encrypts your data with a strong random key. That key is then "wrapped" (encrypted) using **the public key of an Apple HSM**. Apple's servers ($\neq$ HSM) store this encrypted blob but cannot open it because they don't have the HSM's internal private key.
2. The Passcode (pin) Protection (Anti-Brute Force): To recover data on a new phone, you send your passcode to the HSM. The HSM is programmed with an immutable policy: it will only attempt to verify the passcode a limited number of times (e.g., 10). If the passcode is correct, the HSM unwraps the key and sends it back. If you fail 10 times, the HSM destroys the key forever.
3. Preventing "Admin Abuse" (Firmware Trust): To prevent Apple from simply "updating" the HSM firmware to remove the 10-try limit, these HSMs require a specialized, multi-party signing process. In some designs, the hardware is physically modified or locked after the firmware is loaded so that even Apple cannot change the security logic without physically destroying the chip.

Remark: a) On daily use the local Secure Enclaves has the key, it only needs to download it from the HSM once.
b) Face ID data is stored only in the local Secure Enclave. It never leaves the device. Because of this, it cannot be used for remote HSM recovery—the cloud HSM has no idea what you look like.

Intel SGX

Hardware-based TEE implemented in modern Intel CPUs. DRM on the local machine and secure computation on untrusted machines.

"While TPMs and HSMs protect "Data at Rest" (keys on a disk), SGX protects "Data in Use." This allows you to perform calculations on sensitive data (like medical records or financial algorithms) on a cloud server you don't fully trust (like AWS or Azure)." - Gemini

- **Remote Attestation** : Prove to a remote party that a piece of code is running inside a genuine SGX enclave on real Intel hardware.
- **Sealing** : Store data on disk such that only the same enclave on the same CPU can access it later.
- **Isolation** : Enclaves are isolated from the OS and hypervisor. Even with root access, the OS cannot read or modify enclave memory.

Case study
Signal App

How to do private contact discovery. (i.e find which of your contacts are using signal)
1. We want to protect from signal server form knowing your contacts.

This task is called Private set intersection. Note that Signal is in the threat model here.

Solution Using SGX:
- The client sends an encrypted list of contacts to the SGX enclave.
- The enclave decrypts the list and computes the intersection with the signal user database.

To make sure the enclave is not malicious, the client can verify the attestation of the enclave before sending the contacts.

2. Using sealing to store keys.

We still need to trust the signal App.

Case study
Signal Case Study: Private Contact Discovery

How do you find which of your friends use Signal without giving Signal your entire contact list? This is the Private Set Intersection (PSI) problem.

1. The "Blind" Comparison (PSI): Signal runs a service inside an Intel SGX Enclave. Your phone sends an encrypted list of your contacts directly to the enclave. Because it is an SGX enclave, the Signal Server OS cannot see inside. The enclave decrypts the list, compares it against the global user database, and returns only the "matches" to you.
2. Remote Attestation (The Proof): Before your phone uploads any contacts, it performs Remote Attestation. It asks the enclave for a "Quote" signed by the Intel hardware. This proves to your phone that the enclave is:
   a Running on genuine Intel hardware.
   b Running the exact, unmodified open-source code that Signal published (which has been audited to ensure it doesn't "leak" or log contacts).
3. Sealing (Persistence): The enclave uses Sealing to store its own internal keys on the server's disk. This allows the enclave to reboot or update without losing its "identity." Only the same enclave code on that specific server can unseal those keys.
4. The Trust Boundary: While we don't have to trust the Signal Server Admins or the Cloud Provider, we still trust the Signal Application Code. If there were a "backdoor" written into the code inside the enclave, SGX would faithfully execute it. We trust Signal because their enclave code is open-source and verifiable via the attestation hash.

10.3 Side Channel Attacks

Researchers discovered that while the OS cannot read the enclave memory directly, it could sometimes "guess" what was happening inside by watching how the CPU's cache behaved, we'll study two of the most famouse side channel attacks SPECTRE & MELTDOWN

Exploiting unintentional leakages from hardware or software to extract sensitive information.
These leakages can be in the form of timing information, power consumption, electromagnetic leaks, or even sound.

Example Timing attack on RSA, since the square-and-multiply algorithm takes different time depending on the bits of the private key.

Countermeasures

Three categories : Hiding, Masking, and Physical.

- Constant-Time Algorithms: Ensure that operations take the same amount of time regardless of input values.
- Noise Introduction: Add random delays or operations to obscure timing patterns- Faraday Cage: Shielding to prevent electromagnetic leaks.
- Masking

10.4 Mealtdown

Catastrophic attack making it possible to read all memory of a process.

10.5 Spectre

Exploits speculative execution to read memory from other processes. (Branch predictor)

11 Privacy

11.1 The Privacy mindsets

Privacy as confidentiality

- Focuses on protecting data from unauthorized access.
- Emphasizes encryption, access controls, and secure communication.
- "The right to be left alone", minimizing data collection.

Privacy as control

- "The claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others."
- Emphasizes user consent and data ownership.
- Tools: privacy settings, data portability, user consent mechanisms.
- GDPR (General Data Protection Regulation) in the EU.

Privacy as practice

- "The freedom from unreasonable constraints on one's actions and choices in a social context."
- Focuses on social norms and cultural expectations.
- Emphasizes context-aware privacy practices.

11.2 Data anonymization techniques

> *Definition*
> **Quasi-identifier**
>
> Pieces of information that are not of themselves unique identifiers, but are sufficiently well correlated with an entity that they can be combined with other quasi-identifiers to create a unique identifier.

Remarks: *when combined, become personally identifying information*

K-anonymity

- A dataset is k-anonymous if each record is indistinguishable from at least k-1 other records with respect to certain identifying attributes (quasi-identifiers).
- Techniques: generalization (reducing the precision of data) and suppression (removing certain data points).
- Limitations: vulnerable to **homogeneity attack** and background knowledge attacks.

Remarks: *Homogeneity attack occurs when all records in an equivalence class have the same sensitive attribute value, allowing an attacker to infer that value.*

l-diversity

- An extension of k-anonymity that requires each equivalence class to have at least l "well-represented" sensitive attribute values.
- Aims to prevent attribute disclosure by ensuring diversity in sensitive attributes within each group.
- Limitations: may not protect against **skewness and similarity attacks**.

  Remarks: *Skewness attack occurs when the overall distribution of sensitive attributes is significantly different from the distribution within an equivalence class, allowing an attacker to make inferences based on this discrepancy.*

t-closeness

- An extension of l-diversity that requires the distribution of a sensitive attribute in any equivalence class to be close to the distribution of the attribute in the overall dataset.

- Uses a distance metric (e.g., Earth Mover's Distance) to measure closeness
- Aims to provide stronger privacy guarantees by preserving the overall distribution of sensitive attributes.

11.3 Function creep

Definition
Function Creep

The gradual widening of the use of a technology or system beyond the purpose for which it was originally intended, often leading to privacy concerns and ethical issues.

Remarks: *Examples include the use of facial recognition technology for surveillance beyond its initial purpose of security, or the repurposing of health data for marketing without user consent.*

11.4 Differential privacy

Definition
Differential Privacy

The output of an algorithm should not change significantly when a single individual's data is added or removed from the dataset. Mathematically, a randomized algorithm A is ε-differentially private if for all datasets D1 and D2 differing on at most one element, and for all subsets S of the output space of A:

$$\Pr[A(\mathrm{D1}) \in S] \leq e^{\varepsilon} * \Pr[A(\mathrm{D2}) \in S]$$

Remarks:
- ε is the privacy loss parameter; smaller values indicate stronger privacy guarantees.
- Trade-off between privacy and utility: as ε decreases, the amount of noise added increases, which can reduce the accuracy of the results.
- Outliers can significantly impact the effectiveness of differential privacy mechanisms.
- Common mechanisms: Adding noise from Laplace or Gaussian distributions to query results.
- Budget management: Each query consumes part of the privacy budget, limiting the number of queries that can be made while maintaining privacy guarantees. (cumulative privacy loss)

Differential privacy is stronger than anonymization techniques, it assumes that the attacker has access to all other records except the target one, and still cannot infer much about the target record.

12 Machine Learning Security

This section focuses on supervised learning models (labeled data).

Goal

Security in Machine Learning revolves around the following key goals:

- Confidentiality : Ensuring that the data as well as the model parameters are kept secret from unauthorized access.
- Integrity : Ensuring that the output of the model is accurate and has not been tampered with.
- Availability : Ensuring that the model is accessible and usable when needed.

12.1 Attack Models

Opaque-Box Attacks : In an opaque-box attack, the attacker has no knowledge of the model architecture, parameters, or training data. The attacker can only observe the input-output behavior of the model. This type of attack is more challenging for the attacker, as they have limited information to work with.
Gray-Box Attacks : In a gray-box attack, the attacker has partial knowledge of the model, such as its architecture or some of its parameters. This additional information can help the attacker craft more effective attacks.
White-Box Attacks : In a white-box attack, the attacker has complete knowledge of the model, including its architecture, parameters, and training data. This allows the attacker to exploit vulnerabilities in the model more easily.

12.2 Model Stealing

> *Theorem*
> **Stealing a d-dimensions linear model requires d+1 queries**

Note:

- Knowledge about the model can reduce the number of queries needed to steal it.
- Non-linear models require more queries to steal.
- Can be mitigated by limiting the number of queries or adding noise to the output.

12.3 Data Stealing

Extracting sensitive information from the training data used to train a machine learning model.

Membership Inference Attacks

Membership inference attacks aim to determine whether a specific data point was part of the training dataset used to train a machine learning model.
These attacks exploit the differences in the model's behavior when presented with data points that were part of the training set versus those that were not.
Shadow models are often used to perform membership inference attacks.
Overfitting increases the risk of membership inference attacks, as the model may memorize specific data points from the training set.

Side Quest
Shadow Models

Researchers trains a shadow models on similary distribution to gain insights into the meaning of the confidence scores output by the target model.
By analyzing the behavior of the shadow models on known training and non-training data, the attacker can learn to distinguish between members and non-members of the training set based on the target model's output.

Attribute Inference Attacks

Attribute inference attacks aim to infer sensitive attributes of the training data used to train a machine learning model.
The attacker uses the model's output to make educated guesses about the values of specific attributes in the training data.
To mitigate this type of attack, techniques such as differential privacy can be employed during the training process.

Membershiop inference attacks ask the question "Was this data point in the training set?" whereas attribute inference attacks ask "What is the value of this attribute for this data point?"

12.4 Output Manipulation Attacks

Output manipulation attacks aim to manipulate the output of a machine learning model to achieve a desired outcome.
The attacker finds a pertubation that when added to a legitimate input causes the model to misclassify it.
To mitigate this type of attack, techniques such as adversarial training.

Definition
Adversarial Examples

Inputs to machine learning models that have been intentionally designed to cause the model to make a mistake. These examples are often created by adding small perturbations to legitimate inputs, which can be imperceptible to humans but can cause the model to misclassify the input.

Case study
Tesla Autopilot

An example of output manipulation attacks in practice is the Tesla Autopilot system. Researchers have demonstrated that by placing small stickers on road signs, they can create adversarial examples that cause the Autopilot system to misinterpret the signs. For instance, a stop sign with strategically placed stickers can be misclassified as a speed limit sign, leading the vehicle to ignore the stop sign and potentially causing accidents.

This highlights the vulnerability of machine learning models used in critical applications like autonomous driving and underscores the importance of developing robust defenses against adversarial attacks.

12.5 Bias, Fairness and Fallacies

- Biais can be reinforced.
- Ignoring the general prevalence of a class can lead to wrong assuptions about the performance of a model.
- Correlation does not imply causation.

12.6 federated Learning

Definition
Federated Learning

A machine learning approach where multiple decentralized devices or servers collaboratively train a model while keeping the training data localized.
This allows for improved privacy and security, as sensitive data does not need to be shared or centralized.

Case study
Google Gboard

An example of federated learning in practice is Google's Gboard, a virtual keyboard app that uses federated learning to improve its predictive text and autocorrect features. Instead of sending users' typing data to a central server for training, Gboard trains the model locally on users' devices. The model updates are then aggregated and sent back to Google without exposing the raw data, preserving user privacy.

This approach allows Gboard to continuously improve its performance while minimizing the risk of data breaches and maintaining user trust.