Lesson 1: The Security Problem and Violation Categories

The Definition of a Secure System In an ideal world, we define a computer system as being **secure** if every single one of its resources—including the hardware, the software, and the data—is used and accessed exactly as the designers intended it to be, no matter what external or internal circumstances occur. However, you must understand that this absolute state of security is technically **unachievable**. Because of the sheer complexity of modern operating systems and the constant evolution of technology, it is impossible to account for every possible variable. Therefore, our goal in security is not perfection, but rather to make the cost of a successful attack so high that it deters most intruders from even trying.,

Understanding Threats, Attacks, and "Crackers" When we talk about security, we use specific terminology. An **intruder**, often referred to in technical circles as a **cracker**, is the person trying to break into the system. It is important to distinguish between a **threat**, which is merely the potential for a security violation to happen, and an **attack**, which is the actual, active attempt to break those defenses. These attacks aren't always born of malice; they can be: **accidental**, such as a student accidentally writing a bad piece of code that crashes a server. Or a **malicious** attack. From a defensive standpoint, it is much easier to protect against accidental mistakes than to stop a malicious attacker who is purposefully searching for weaknesses to exploit.

The Five Major Categories of Security Violations When a security breach occurs, it generally falls into one of these five functional categories:

1. **Breach of Confidentiality:** Unauthorized reading of data.
2. **Breach of Integrity:** Unauthorized modification of data.
3. **Breach of Availability:** Unauthorized destruction of data.
4. **Theft of Service:** Unauthorized use of resources.
5. **Denial of Service (DoS):** Prevention of legitimate use. This is a specific type of attack aimed at overwhelming a system so that legitimate users cannot access it. By flooding a computer with more requests than it can handle, the attacker prevents it from doing any useful work.

The Problem of "Too Much" Security Finally, we must consider the human element of security. While we want to protect our systems, there is a constant tension between **security** and **usability**. If a system is "too secure"—for example, requiring a 50-character password that must be changed every hour—it becomes so difficult to use that people will find ways to bypass the security entirely, or they simply won't be able to get their work done. Finding the right balance is a central challenge in security implementation.

Lesson 2: Attack Methods and the Four Levels of Defense

In this lesson, we shift our focus from the broad categories of security violations to the specific techniques attackers use to execute them and the multifaceted strategy we employ to stop them. You must understand that security is a "chain," and an attacker only needs to find one weak link to bypass all your other efforts.

Standard Methods of Security Attacks

Attackers generally do not try to break down a front door if they can find a way to trick the system into letting them in through a side entrance. These four methods are the most common ways they attempt to bypass authentication and gain unauthorized access:

- **Masquerading(breach authentication):** This is a direct attempt to breach authentication where the attacker **pretends to be a legitimate, authorized user**. By successfully assuming another person's identity, the attacker can **"escalate" their privileges**, gaining access to files or commands that their own account would never be allowed to touch.
- **Replay Attack:** In this scenario, an attacker captures a valid data transmission—like a login request or a financial transfer—and then sends it again later. They might **send it exactly as it was captured or modify the message** slightly before re-sending it to trick the system into performing the action a second time.
- **Man-in-the-Middle (MITM) Attack:** This is a sophisticated interception where an intruder inserts themselves directly into the **data flow** between a sender and a receiver. The attacker **masquerades** as the sender when talking to the receiver, and as the receiver when talking to the sender, effectively controlling and potentially altering the entire conversation without either party knowing a third person is involved.
- **Session Hijacking:** Rather than trying to guess a password or crack encryption, the attacker waits for a legitimate user to successfully log in and establish a session. The attacker then "hijacks" or **intercepts that active, already-established session**, allowing them to bypass the authentication process entirely since the system thinks they are the already-verified user.

The Four Essential Levels of Security Defense

Because it is impossible to achieve absolute security, our strategy is to make the cost—in terms of time, effort, and money—so high for the perpetrator that it deters them from continuing their attempt. Security is **as weak as the weakest link** in the chain. Therefore, to be effective we must implement security at these four distinct levels:

1. **Physical Level:** This is the most basic layer and involves protecting the actual hardware. You must secure the **data centers** where servers live, the servers themselves, and any connected terminals or workstations to ensure an attacker cannot simply walk up and plug in a malicious device or steal a hard drive.
2. **Human Level:** Even the best software cannot protect a system if the users are compromised. This level focuses on training people to avoid **"social engineering"** tactics like **phishing** (fake emails), and even low-tech methods like **dumpster diving**, where attackers literally look through a company's trash to find written-down passwords or sensitive documents.
3. **Operating System Level:** This involves using the built-in **protection mechanisms** of the OS itself. We must ensure that the code is thoroughly **debugged** to remove vulnerabilities and that strict access controls are in place to manage what different programs and users are allowed to do.
4. **Network Level:** Since most modern systems are connected, we must guard the communication lines. This means protecting against **intercepted communications** where someone "listens" to your data, as well as preventing service **interruptions** and the massive **Denial of Service (DoS) attacks** we discussed in Lesson 1.

Lesson 3: Program Threats

In this lesson, we move from the broad methods of attack into the world of **Program Threats**, which are specific pieces of software designed to exploit vulnerabilities in your system's code. These threats come in many variations and go by many different names, but they all share the fundamental goal of tricking a computer into performing unauthorized actions.

Trojan Horses and Trap Doors

One of the most classic program threats is the **Trojan Horse**, which is a code segment that misuses its environment by appearing to be a useful or harmless program. It specifically exploits the mechanisms that allow programs written by one user to be executed by another user, hiding its malicious intent behind a "useful" facade. Modern examples of Trojan Horses include spyware, pop-up browser windows, and "covert channels" used for secret data transmission; remarkably, the material notes that up to **80% of all global spam** is actually delivered by systems that have been silently infected with spyware.

Closely related to this is the **Trap Door**, which is a secret "back entrance" to a system, such as a specific user identifier or a hidden password that allows someone to bypass all normal security procedures. These are particularly dangerous because they can be built directly into a compiler—the software that translates code into running programs—making them nearly impossible to detect since the malicious "door" doesn't appear in the program's original source code.

Logic Bombs and Technical Exploits

A **Logic Bomb** is a program **designed to sit quietly and do nothing until a very specific set of circumstances** is met, at which point it "detonates" to initiate a security incident. While a logic bomb waits for a trigger, a **Stack or Buffer Overflow** attack exploits a physical mistake in how a program was written. This occurs when a program fails to check the "bounds" or limits of the data being input into its memory. By sending more data than the program expects, an attacker can write past the intended arguments on the "stack" and overwrite the "return address". When the program finishes its current task and tries to return, it instead jumps to a hacked address that points to malicious code the attacker has secretly loaded onto the stack, resulting in unauthorized access or an immediate escalation of the attacker's privileges.

Viruses: The Self-Replicating Menace

Unlike standalone programs, a **Virus** is a fragment of code embedded within a legitimate program that is designed to be self-replicating and infect other computers. Viruses are highly specialized and are typically designed to target a specific CPU architecture, operating system, or application; they are usually delivered to a new system via email or as a "macro" hidden inside a document. The infection begins when a "virus dropper" program inserts the virus code into your system's files.

There are literally thousands of categories of viruses, including:

- **File or Parasitic Viruses:** These attach themselves to executable files.
- **Boot or Memory Viruses:** These infect the system's startup code or live in the RAM.
- **Source Code Viruses:** These look for and infect the original code written by developers.
- **Polymorphic Viruses:** These change their own code every time they replicate to avoid having a recognizable "virus signature" that security software could spot.
- **Stealth, Tunneling, and Armored Viruses:** These use advanced techniques to hide from the operating system, bypass security layers, or make it extremely difficult for researchers to analyze their code.
- **Encrypted Viruses:** This is a category of virus that uses encryption to hide its internal code, making it much harder for antivirus software to detect its "signature".
- **Multipartite Viruses:** These are particularly resilient viruses that can infect multiple different parts of a system (such as both files and the boot sector) at the same time.

The Modern Evolution of Threats

You should understand that the nature of these attacks has changed over time, moving away from simple "science experiments" conducted by individuals and becoming professional tools for **organized crime**. A **zombie computer** is a system that has been compromised by an attacker and can be controlled remotely without the owner's knowledge. These individual "zombies" are linked together to form a **botnet**, a massive network used to deliver spam or launch **Distributed Denial of Service (DDoS)** attacks.

By using these zombie systems, attackers can launch attacks from multiple locations simultaneously, which significantly decreases their **traceability** and makes it harder for defenders to identify the true source. They also use **keystroke loggers** to silently record every button you press on your keyboard to steal private passwords and credit card numbers.

The "Windows Problem"

The notes also briefly address why Windows is the primary target for most attacks:

- **Monoculture:** It is the most common OS, and a "monoculture" is considered harmful because a single exploit can hit almost everyone if they use **same version of Windows**. making it a highly efficient target for organized crime.
- **Privileges:** Historically, almost every Windows user operated as an **administrator**, making it easier for malware to take control.

Comparisons to Memorize:

- **Logic bombs** are defined by its **trigger mechanism** rather than its method of infection or deception. While other threats are often active immediately, a logic bomb is a program designed to sit dormant and only initiate a security incident when **certain circumstances** or specific conditions are met. Essentially, it is a "timed" or "event-based" threat that waits for the right moment to act.
- **Viruses** are self-replicating code fragments that actively try to spread and infect other programs.
- **Trojan Horses** rely on being mistaken for useful programs to trick users into executing them.
- **Trap Doors** are secret, permanent entry points built into software to bypass normal security.
- **Buffer or Stack Overflows** are technical exploits that take advantage of specific memory-handling bugs in a program's code.

Lesson 4: System and Network Threats

In this lesson, we move beyond individual programs and look at threats that target entire systems and the networks that connect them. While program threats like viruses need a "host" file to live in, system threats are often standalone entities that can move across a network on their own.

Worms: The Self-Propagating Standalone

A **Worm** is a standalone program that uses a **spawn mechanism** to duplicate itself and spread from one system to another. Unlike a virus, it does not need to attach itself to an existing program to function because it is already a program. That is the big difference between a **Worm** and a **Polymorphic Virus.**

A famous historical example is the **Internet Worm**, which caused massive disruption by exploiting three specific weaknesses in UNIX systems:

- **Remote Access Features:** It used standard networking features to jump between machines.
- **Software Bugs:** It exploited well-known bugs in common programs called ***finger*** and ***sendmail***.
- **Trust Relationships:** It took advantage of a mechanism used by **rsh** (remote shell) that allowed "friendly" systems (computers that have established a trust relationship with one another) to access each other without requiring a password.

Port Scanning: Scouting the Target

Before an attacker can launch a successful strike, they often use **Port Scanning** to gather intelligence. **This is an automated process where a program attempts to connect to a range of "ports" on a specific IP address** (or range of addresses) to see which ones respond.

By scanning ports, an attacker can:

- Identify which services are running on a system (like a web server or email server).
- Detect the specific **Operating System** and version the system is running.
- **Tools of the Trade:** Professionals use tools like **nmap** and **nessus** for automated tools used for **network port scanning**, which is like a digital "scouting mission" to see which services are running on a target system.
    1. **nmap:** This tool scans all the ports in a given range of IP addresses to see which ones send back a response. By analyzing these responses, it can identify which services are active and even detect the specific **operating system** and version the computer is using.
    2. **nessus:** This is a more advanced vulnerability scanner. It maintains a massive database of known protocols, bugs, and specific exploits. Once it finds an open port, it checks that service against its database to see if there are any known "weaknesses" or bugs that an attacker could use to break in.

- **Stealth Tactics:** These scans are frequently launched from **zombie systems** to hide the attacker's true identity and decrease **traceability** to make it much harder for the target to trace the activity back to them.

Denial of Service (DoS) and DDoS

The goal of a **Denial of Service (DoS)** attack is simple: overload the target computer so completely that it cannot do any useful work for legitimate users. When these attacks come from multiple sites at once, they are called **Distributed Denial of Service (DDoS)** attacks.

Technical points you should know for the exam:

- **The SYN Handshake(SYN Flood):** In a standard network connection, a computer starts a "handshake" by sending a SYN (Synchronize) request to the server. Normally, the server responds, and the connection is completed with a final confirmation from the user. The attacker sends a massive number of SYN requests to the target system. For every SYN received, the operating system (OS) must set aside some memory and resources to wait for the handshake to finish, making it in a **"Half-Open"** state. Because the attacker never completes the handshake, these are called **"started-connections"** and they exhaust the system's resources until it is so overloaded that the system cannot accept any new requests—meaning legitimate users are blocked from doing any useful work making it a **Denial of Service (DoS)** attack.
- **Popularity vs. Attack:** One of the hardest challenges for a network administrator is telling the difference between a website that is under attack and one that is just **really popular** and receiving a sudden burst of legitimate traffic.
- **Motivations:** As we discussed in Lesson 1, these can be **accidental** (like a student writing bad code that "forks" too many processes) or **purposeful**, often used for things like extortion or punishment.

Lesson 5: Cryptography as a Security Tool

In this lesson, we explore **cryptography**, which is the broadest and most powerful security tool available to us. While an operating system can easily protect and verify communication *internally* because it manages process IDs and communication ports, it cannot inherently trust the source or destination of messages sent over a network.

Without cryptography, a local network is vulnerable to unauthorized hosts, and on a wide-area network (WAN) or the internet, an IP address alone is never enough to establish true authenticity because it can be easily spoofed(an attacker faking a **host name** or an **IP address** to trick a system into granting access).

Instead, we must use **cryptography** to verify the actual source and destination of a message

The Core Purpose of Cryptography

Cryptography uses secret "keys" to force constraints on who can send or receive messages. By using these mathematical secrets, we can achieve three critical security goals:

- **Confirmation of Source:** Proving exactly who sent the message.
- **Receipt by Specific Destination:** Ensuring that only the intended recipient can read the content.
- **Trust Relationships:** Establishing a secure bond between a sender and a receiver even over an insecure medium.

The Mathematical Components of Encryption

The mathematical components are essentially a formal way of describing a "lock and key" system using three specific groups of data and two main "actions" or functions. To understand how encryption works for your exam, you must know these three sets and two primary functions:

The Three Groups (Sets):

- **Set M (Messages):** This is your **plaintext**. It is the original, readable information you want to protect, like a text message or a password.
- **Set C (Ciphertexts):** This is the **encrypted** version of your message. It is the scrambled, unreadable data that you actually send across the network.
- **Set K (Keys):** These are the **secrets**. A key is the specific "code" or mathematical value required to lock or unlock the information.

The Two Actions (Functions):

- **Encryption (E):** This is the process of using a key (K) to turn a readable message (M) into a scrambled ciphertext (C). Think of this as the "locking" function.
- **Decryption (D):** This is the reverse process. It uses a key (K) to turn the scrambled ciphertext (C) back into the original, readable message (M). This is the "unlocking" function.

The Essential Rule: For this to be secure, it must be **computationally infeasible**—meaning it is practically impossible for even the most powerful computers—to figure out the message or the secret key just by looking at the scrambled **ciphertext**. Thus, a computer holding D(k) can decrypt ciphertexts to the plaintexts used to produce them, but a computer not holding D(k) cannot decrypt ciphertexts.

Lesson 6: Authentication and Digital Certificates

While **encryption** is about keeping data private, **authentication** is focused on constraining the set of potential senders of a message. It acts as a complementary—and sometimes redundant—layer to encryption. Crucially, while encryption **hides the content** and restricts on who can read a message, authentication is about ensuring the **message was not modified in transit** and proving who sent it**.**

The Mathematical Framework For your exam, you should know that the authentication algorithm is built on three sets and two primary functions:

- **The Sets:** Just like encryption, we have a set **K** of keys and a set **M** of messages. However, instead of ciphertexts, we have a third set **A** of **authenticators**.

The Process

1. **Creating the Proof (Function S)**: The sender takes their secret **key** and the **message** and runs them through the **S function**. This produces a unique piece of data called an **authenticator**.
2. **Checking the Proof (Function V)**: The receiver takes the **message** and the **authenticator** and runs them through the **V function**. This function returns **true** only if the correct secret key was used to create that specific authenticator.

Why It's Secure

- Only someone who has the secret **signing key** can create an authenticator that the check will accept as "true".
- Even if an attacker intercepts the authenticators being sent over the network, it is **mathematically infeasible** and impossible for an attacker to derive the secret signing key just by studying the authenticators they intercept.

Symmetric vs. Asymmetric Cryptography

Both encryption and authentication can be used in symmetric and asymmetric systems. There are two fundamental ways we apply these mathematical principles, and they differ significantly in how they operate:

- **Symmetric Cryptography:** This is based on **transformations**. It uses relatively simple bit-level shifts or substitutions to scramble data. it is relatively fast because the same key (or a simple transformation of it) is used for both encryption and decryption. Because it is efficient, it is used for **bulk data encryption**.
- **Asymmetric Cryptography:** This is based on complex **mathematical functions**. It is much more "compute-intensive" than symmetric methods. Because of this high technical demand, asymmetric encryption is typically **not used for bulk data encryption**; instead, it is often used for smaller, more critical tasks like exchanging symmetric keys or verifying identities.

Asymmetric Cryptography and **Digital Certificates** work together to solve a major problem: how do you know you are actually talking to the right person on an insecure network?

1. Asymmetric Cryptography: The Two-Key System

Asymmetric cryptography uses a pair of mathematically related keys:

- **Private Key:** Kept secret by its owner and never shared. Used to **create an authenticator** to be sent with your message.
- **Public Key:** Shared openly and available to anyone. Used to **verify the authenticator** received.

The keys are related in such a way that an authenticator created with the private key can be verified using the corresponding public key.

However since they are entirely different from each other, knowing the public key **does not** reveal the private key.

2. Connecting the Math: Signing and Verification

In **Lesson 6**, we looked at the functions **S (Signing)** and **V (Verification)**. Asymmetric cryptography is what makes those functions possible:

- Given a message **M**, the sender uses their private key with the signing function **S** to produce an **authenticator**(like a signature). Anyone possessing the corresponding public key can later verify that the authenticator was created using the matching private key.
- Anyone else can then use your **public key** that you shared with them in function **V** to verify that the sent authenticator is the correct one. The function verifies that the authenticator is valid for the received message. If either the message or authenticator is modified, verification fails. If the function returns **true**, the receiver knows that the message has not been modified and that it was approved by the holder of the private key corresponding to the supplied public key.

The verification function would only return **true** if the authenticator sent was created using **your private key**. This makes it impossible for an attacker to take your message, generate an authenticator of its own and send it with your public key because the verification function would prove that the new authenticator is invalid since it does not “mathematically work” with your public key.

This verification function **only proves** the following: "The message sent was signed by whoever owns the private key corresponding to this public key."

Does that mean there is no way for an attacker to pretend it’s you? Well, **not quite.**

If an attacker manages to intercept your message before it reaches the receiver, the attacker can then replace your public key with their own and sign a modified message using their private key. Without a trusted way to verify **ownership** of public keys, the receiver may accept the attacker's key as **yours**.

3. The Trust Problem: Who Owns the Key?

This is why we use a **Digital Certificate:** A digitally signed document whose purpose is to bind an identity to a public key. By itself, a public key has no inherent identity attached to it; anyone can generate a key pair and claim ownership of the public key. The certificate solves this problem by providing a trusted statement that a specific public key belongs to a specific entity.

- **The Certificate Authority (CA):** This is a trusted third party (like a digital notary). You provide them with proof of your identity, and they **digitally sign** your public key to certify it belongs to you, creating a **Digital Certificate** for you.
- **The Browser Connection:** To make this work, the public keys of these CAs are **pre-installed in your web browser**. When you visit a secure site, your browser checks the site's certificate against these pre-installed keys to ensure the site is legitimate.
- **Chains of Trust:** CAs can also vouch for other authorities by signing their keys, creating a global **chain of trust**.

Component	Purpose
Authenticator + Verification Function	Protects the message from being modified and proves it was approved by the holder of the corresponding private key.
Digital Certificate	Protects the ownership of a public key by binding it to a verified identity and preventing public-key impersonation.

Lesson 7: User Authentication

Correctly identifying a user is the foundation of security because all protection systems depend on knowing exactly who is accessing the system.

Passwords: The Standard Approach

Passwords are the most common way to prove identity. For them to work effectively, they must be kept secret and managed with strict rules:

- **Management:** Users should change passwords frequently, avoid repeating old ones, and use "non-guessable" combinations.
- **Logging:** The system should log every invalid login attempt to spot hackers, though it should never log the actual password the person typed.

Securing the Password Store

Storing passwords as plain text is dangerous. Instead, systems use several layers of protection:

- **Encryption:** Passwords should be encrypted using an algorithm that is easy to calculate but **very difficult to reverse**. Only the encrypted version is stored, never the original plain text password.
- **Salting:** Is the process of adding extra, unique data to a password before it is run through an encryption algorithm. By adding a **salt**, the service that stores your password ensures that every entry in their database is unique. Without this, a hacker could see that 100 people have the same "scrambled" value and know they all used the same common password. Salting forces the hacker to try and crack every single account one by one, which is often too slow to be worth it.
- **Access Control:** On systems like Unix, these encrypted values are kept in a **hidden** file (such as `/etc/shadow`) that only the **superuser** can read.

Stronger Authentication Methods

When standard passwords aren't secure enough, we use more advanced tools:

- **One-Time Passwords:** A hardware device or special software uses a "seed" to generate a random password that changes constantly and can only be used once.
- **Biometrics:** These identify you by physical attributes, like a fingerprint or a hand scan.
- **Multi-Factor Authentication (MFA):** This is the most secure method. It requires two or more factors to grant access—for example, a password combined with a biometric scan or a physical USB "dongle".

Lesson 8: Implementing Security Defenses

This lesson focuses on the practical strategies used to protect systems, starting with the most common theory: **Defense in Depth**, which uses multiple overlapping layers of security to ensure that if one layer fails, others are still in place.

Policy and Assessment

Before technical tools are deployed, a system needs a plan:

- **Security Policy:** A formal document describing exactly what resources are being secured.
- **Vulnerability Assessment:** The technical process of comparing the actual state of a system or network against the stated security policy to find weaknesses.

(very important) Intrusion Detection Systems (IDS) IDS are designed to detect both attempted and successful breaches. There are two main technical approaches:

- **Signature-based Detection:** Identifies "known bad" patterns of data or behavior. While effective against known and old threats, it cannot see new ones.
- **Anomaly(behaviour) Detection:** Identifies activity that deviates from "normal" established behavior. Its primary advantage is the ability to detect **zero-day attacks** (new threats that haven't been seen before).
- **The Problem:** Both methods can suffer from **false-positives** (flagging safe activity as a threat) and **false-negatives** (missing a real attack).

Firewalling and Network Protection A network firewall is placed between trusted and untrusted hosts to limit access between these two security domains.

- **Bypassing Firewalls:** Firewalls can be bypassed via **tunneling** (hiding a disallowed protocol inside an allowed one, like running Telnet inside HTTP) or **spoofing** (faking a host name or IP address).
- **Types of Firewalls:**
    - **Personal Firewall:** A software layer running directly on a specific host.
    - **Application Proxy Firewall:** Understands specific application protocols (like SMTP) and can control them directly.
    - **System-call Firewall:** Monitors every important system call a program makes and applies rules to allow or deny them.

Computer-Security Classifications (U.S. Dept. of Defense)

The government uses four main divisions to rank the security level of systems:

- **Division D (Minimal Security):** The lowest rank.
- **Division C (Discretionary Protection):** Provides security through auditing. **C1** identifies cooperating users; **C2** allows specific user-level access control.
- **Division B (Sensitivity Labels):** Includes all features of Division C, but every object also has a unique sensitivity label. (Subdivided into B1, B2, and B3).
- **Division A (Formal Design):** The highest rank; uses formal mathematical design and verification techniques to ensure the system is secure.

Virus Protection and The Threat Landscape

Virus protection must defend against an incredibly wide variety of malicious code fragments embedded in legitimate programs. The material identifies several specific categories that protection software must handle:

- **Signature Avoidance: Polymorphic viruses** change their own code to avoid having a recognizable signature, and **encrypted viruses** hide their content entirely.
- **Advanced Tactics: Stealth viruses** hide from the OS, **tunneling viruses** bypass detection by going underneath the OS, and **armored viruses** are specifically designed to be difficult for security researchers to analyze.
- **Infection Vectors:** Viruses can target files (parasitic), boot sectors (memory), macros (email), or even raw source code.

Auditing, Accounting and Logging

This layer of defense **records** all or specific system and network activities so they can be reviewed later.

- **Password Safety:** A critical rule for logging is that the system should log every **invalid access attempt**, but it must **never log the actual password** the user typed.
- **Official Ranking:** Auditing is so important that the U.S. government uses it as a benchmark; **Division C (Discretionary Protection)** is defined specifically by its ability to provide security through auditing.