Introduction

The introduction provides an overview of essential aspects within the realm of computer security. It highlights the foundational security properties that form the basis of a robust security framework.

The first set of basic security properties includes:
- Confidentiality: This property aims to thwart unauthorized disclosure of information.
- Integrity: Its purpose is to prevent any unauthorized modification of information.
- Availability: Ensures reliable access to information.
- Authentication: Involves proving the claimed identity, applicable to both data and entity authentication.

Complementing these basic properties are several auxiliary security considerations, such as:
- Non-Repudiation: A measure to prevent false denial of performed actions.
- Authorization: Determines the permissible actions for an entity, often expressed as "What Alice can do."
- Auditing: Involves securely recording evidence of performed actions.
- Attack Tolerance: Refers to the system's ability to provide a certain level of service even after experiencing failures or attacks.
- Disaster Recovery: Ensures the system can recover to a secure state.
- Key-Recovery, Key-Escrow: Mechanisms related to the management of cryptographic keys.
- Digital Forensics: Techniques and processes for investigating and analyzing digital systems.

The discussion further delves into various security mechanisms employed to enforce these properties, including:
- Random Numbers: Used, for example, in generating Initialization Vectors.
- Pseudo Random Numbers: Generated algorithmically to mimic randomness.
- Encryption/Decryption: Techniques to secure information.
- Hash Functions: Ensure data integrity.
- Hash Chain (Inverted): Utilized for secure information storage.
- Message Integrity Code (MIC): Protects the integrity of messages.
- Message Authentication Code (MAC and HMAC): Ensures message authenticity.
- Digital Signatures: Used for non-repudiation.
- Key Exchange Protocols: Facilitate secure key sharing.
- Key Distribution Protocols: Ensure secure distribution of cryptographic keys.
- Time Stamping: Adds a temporal dimension to security measures.

The discussion also touches upon the types of attackers, distinguishing between **insiders** and **outsiders**, as well as the two main types of attacks: **passive attacks**, where the attacker can only read information, and **active attacks**, where the attacker can read, modify, generate, or destroy information.

1. Overview

This topic delves into the realm of computer security, which is defined as the safeguarding of automated information systems with the aim of achieving the relevant objectives of preserving the integrity, availability, and confidentiality of information system resources, encompassing hardware, software, firmware, information/data, and telecommunications.

Key challenges in computer security are highlighted, emphasizing its complexity, the necessity to anticipate potential attacks, the counter-intuitive nature of certain procedures, the involvement of algorithms and confidential information, and the critical decision-making regarding the deployment of security mechanisms. It also underscores the dynamic interaction, often portrayed as a battle of wits, between attackers and administrators.

Vulnerabilities within a system are explored, noting that compromised system resources may lead to corruption, leakage, or unavailability. Various types of attacks, categorized as passive or active, insider or outsider, are introduced.

Countermeasures, which are the means employed to address security attacks, are discussed in terms of prevention, detection, and recovery. However, it's acknowledged that these countermeasures may introduce new vulnerabilities, resulting in residual vulnerability. The ultimate goal is to minimize risk within given constraints.

The consequences of security threats are outlined, including unauthorized disclosure, deception, disruption, and usurpation. Network security attacks are classified as either passive (eavesdropping) or active (modifying/faking data), each presenting unique challenges in detection and prevention.

The X.800 Security Architecture, designed for the OSI model, provides a systematic framework for defining security requirements, characterizing approaches to satisfy these requirements, and outlining security attacks, mechanisms, and services.

A strategic approach to computer security is proposed, involving specification and policy development, implementation of mechanisms for prevention, detection, response, and recovery, and the ongoing assessment of correctness and assurance. In essence, the comprehensive understanding of computer security encompasses policy development, strategic implementation, and continuous evaluation of the effectiveness of security measures.

2. User authentication

User authentication is a fundamental building block of computer security, forming the basis for access control and user accountability. It involves verifying the identity claimed by a system entity through two steps: identification (specifying an identifier) and verification (binding the entity, often a person, to the identifier). This process is distinct from message authentication.

There are **four** means of authenticating a user's identity, based on something the individual **knows**, **possesses**, **is** (static biometrics), or **does** (dynamic biometrics). These can be used alone or in combination, but each has its own set of issues.

Password authentication is a widely used method where the user provides a name/login and a password, which is then compared with the stored password for the specified login. While common, it has vulnerabilities such as offline dictionary attacks, specific account attacks, and password guessing. Countermeasures include intrusion detection, account lockout mechanisms, and encrypted network links.

Hashed passwords, a key part of user authentication, have evolved for better security. Improved implementations use stronger hash/salt variants, with systems like MD5 or Blowfish. However, password cracking techniques, including dictionary attacks and rainbow table attacks, remain concerns.
A rainbow table attack is a type of password cracking attack that involves using precomputed tables to reverse cryptographic hash functions. The objective is to find the original plaintext password from its hash value. Rainbow table attacks exploit the fact that many users choose weak passwords, and even if passwords are hashed, common hash functions can be quickly reversed using these precomputed tables.

Users often choose weak passwords, making them susceptible to guessing. Proactive password checking, using rule enforcement and user advice, aims to eliminate guessable passwords while remaining memorable.

Token authentication involves objects the user possesses, such as embossed cards, magnetic stripe cards, memory cards, or smartcards. Memory cards store but do not process data, while smartcards have their processor, memory, and I/O ports, executing protocols to authenticate with readers or computers.

Biometric authentication relies on physical characteristics for user identification. While offering unique templates, it faces challenges of false matches and non-matches, requiring a threshold to balance error rates.

Remote user authentication over networks is more complex, involving challenge-response mechanisms to protect against eavesdropping, replay, and other attacks.

Authentication security issues include client attacks, host attacks, eavesdropping, replay attacks, trojan horses, and denial-of-service attacks. Overall, user authentication methods require careful consideration and continuous improvement to address evolving security challenges.

3. Denial of Service

Denial of Service (DoS) is an action that disrupts authorized network, system, or application usage by depleting resources like CPU, memory, bandwidth, and disk space. Various attack methods aim at network bandwidth, system resources, or application resources, and DoS attacks have been a persistent issue.

Classic DoS attacks include simple flooding pings, causing congestion by overwhelming lower-capacity links from higher-capacity ones. Source address spoofing involves using forged source addresses to generate high-volume traffic, making it challenging to trace the actual source. SYN spoofing targets a server's ability to respond to connection requests, overwhelming system resources.

Flooding attacks are classified based on the network protocol used, such as ICMP Flood, UDP Flood, and TCP SYN Flood.
Distributed Denial of Service (DDoS) attacks involve multiple compromised systems forming a botnet to generate higher traffic volumes.

Reflection attacks exploit normal network behavior, using spoofed source addresses to flood a target with responses from multiple servers.
DNS Amplification attacks leverage DNS requests with a spoofed source address to flood a target with larger responses.

Defenses against DoS attacks include blocking spoofed source addresses, implementing rate controls in upstream distribution networks, modifying TCP connection handling, and blocking IP directed broadcasts. Application attacks can be managed with "puzzles" to distinguish legitimate requests, and general system security practices should be maintained.

Responding to attacks requires a robust incident response plan with contacts for ISPs, standard filters, network monitors, and Intrusion Detection Systems (IDS) to detect abnormal traffic patterns. Identifying the type of attack, capturing and analyzing packets, implementing filters, and tracing packet flow back to the source are crucial steps in mitigating and preventing future attacks. Contingency plans and regular updates to incident response plans are essential for effective defense against DoS attacks.

4. Intrusion Detection

Intruders pose a significant threat, ranging from benign to serious trespass, involving unauthorized logins, privilege abuse, and software-based threats like viruses and trojan horses. Classes of intruders include **masqueraders** (individuals or entities that attempt to gain unauthorized access by pretending to be someone or something they are not), **misfeasors** (authorized users who misuse their privileges, either intentionally or unintentionally), and **clandestine users** (access systems or data without being noticed).

Examples of intrusion encompass remote root compromises, web server defacement, password guessing, copying sensitive data, running packet sniffers, distributing pirated software, accessing networks through unsecured connections, impersonating users, and exploiting unattended workstations.

Hackers, motivated by the thrill of access and status, may be benign but consume resources, necessitating countermeasures like Intrusion Detection Systems (**IDS**), Intrusion Prevention Systems (**IPS**), and Virtual Private Networks (**VPNs**). Criminal enterprises, organized groups of hackers, pose threats to corporations and governments, primarily targeting credit cards on e-commerce servers.

Insider attacks, among the most challenging to detect, involve employees exploiting access and system knowledge, often motivated by revenge or entitlement. Intrusion techniques aim to gain access or increase privileges, utilizing methods like buffer overflows or password acquisition.

Security intrusion is defined as unauthorized access attempts, and Intrusion Detection is a service monitoring system events to provide real-time warnings of unauthorized access attempts. Intrusion Detection Systems (IDSs) are classified as host-based and network-based, involving sensors, analyzers, and user interfaces.

Host-Based IDS monitors single host activity and employs anomaly detection (threshold and profile-based) and signature detection. Audit records, fundamental for intrusion detection, include native audit records and detection-specific audit records.
An "**audit record**" is a log containing detailed information about events and activities within a computer system. These records are crucial for auditing and computer security as they provide a comprehensive trail of what occurs within a system.

Audit records can be of two main types:

1. <u>Native Audit Records</u>: These are provided by the operating system itself. They contain fundamental information about events occurring within the system, such as logins, file accesses, and system changes.

2. <u>Detection-Specific Audit Records</u>: These are specific to intrusion detection systems (IDS) and are tailored to the task of identifying and monitoring potential security breaches. They often log individual elementary actions, including details such as the subject (user), action, object, exception condition, resource usage, and timestamp.

Anomaly detection involves threshold detection and profile-based methods, analyzing past behavior and deviations. Signature detection observes system events and applies rules to identify intruders, employing rule-based anomaly detection and rule-based penetration identification.

Network-Based IDS (**NIDS**) monitors network traffic at selected points, using sensors inline or passively. Intrusion Detection Techniques include signature and anomaly detection, with sensors sending alerts and logs to improve detection parameters and algorithms.

Distributed Adaptive Intrusion Detection involves collaborative efforts and information sharing among IDSs. **Honeypots**, decoy systems filled with fabricated information, divert and hold attackers to collect activity information without exposing production systems.

5. Firewalls and Intrusion Prevention Systems

Firewalls and Intrusion Prevention Systems (IPS) play a crucial role in safeguarding Local Area Networks (LANs) against potential security threats arising from internet connectivity. While internet access is essential, it introduces risks, making it imperative to secure workstations and servers. Firewalls act as a perimeter defense, serving as a centralized point for security enforcement.

In terms of capabilities, firewalls define a single choke point, monitor security events, and offer a convenient platform for Internet functions. However, they have limitations, such as being unable to fully protect against certain attacks that can bypass the firewall, potential gaps in internal threat protection, and vulnerabilities related to improperly secured elements.

Types of Firewalls:

1. **Packet Filtering Firewall:** Applies rules based on packet header information to filter incoming and outgoing packets.
2. **Stateful Inspection Firewall:** Reviews packet header information while maintaining data on TCP connections, offering enhanced security for TCP traffic.
3. **Application-Level Gateway:** Acts as a relay for application-level traffic, requiring proxy code for each application, providing more secure filtering but with higher overheads.
4. **Circuit-Level Gateway:** Sets up two TCP connections, relaying segments without examining contents, suitable for trusted inside users with lower overheads.

The choice of firewall basing involves options like bastion hosts, individual host-based firewalls, and personal firewalls, each catering to specific security needs. **Intrusion Prevention Systems** (IPS) have emerged as a recent addition, integrating In-Line Network or Host-Based Intrusion Detection Systems (IDS) with the ability to block traffic.

Host-Based IPS identifies attacks using signature and anomaly detection techniques, allowing tailoring to specific platforms and incorporating sandboxing for monitoring applet behavior.
Incorporating Sandboxing: Sandboxing involves running potentially malicious code or applications in a restricted environment, isolated from the rest of the system. This controlled environment, known as a sandbox, allows the IPS to observe the behavior of the code without risking harm to the actual system.
Monitoring Applet Behavior: Applets are small applications or software components. The Host-Based IPS, by incorporating sandboxing, can closely monitor the behavior of these small programs. This monitoring helps identify any malicious activities or unexpected behavior exhibited by the applets.

Network-Based IPS, on the other hand, functions as an inline NIDS capable of discarding packets or terminating TCP connections, using signature and anomaly detection for identifying malicious packets.

The overall security approach includes various firewall topologies, such as host-resident firewalls, screening routers, single/double bastion configurations, and distributed firewall setups. Unified Threat Management (UTM) products further contribute to a comprehensive security infrastructure, providing an effective means of protection against diverse security threats.
Unified Threat Management (UTM) products refer to comprehensive security solutions designed to provide multiple security features and functions within a single, integrated platform. UTM solutions are aimed at simplifying the management of various security measures and protecting against a wide range of cyber threats. These products typically combine several security features into a unified and centrally managed system, offering a holistic approach to network security.

6. Buffer Overflow

Buffer Overflow is a widespread attack mechanism that has been present since the 1988 Morris Worm and has evolved through various instances like Code Red, Slammer, Sasser, and others. Despite known prevention techniques, it remains a major concern due to the legacy of widely deployed buggy code and continued careless programming techniques.

Buffer Overflow Basics occur due to programming errors allowing more data to be stored than the capacity available in a fixed-sized buffer. Overwriting adjacent memory locations leads to the corruption of program data, unexpected transfer of control, memory access violations, and execution of code chosen by attackers.

Memory is a flat sequence of bytes, and memory protection involves marking areas as readable, writable, or executable. Interpretations of memory, such as types, are abstractions defining how byte ranges are interpreted. Buffer Overflow Attacks require identifying vulnerabilities, understanding buffer storage in memory, and determining potential corruption.

Stack Buffer Overflow occurs when the buffer is on the stack, potentially overwriting critical control items. The x86 stack, managed by a Stack Pointer, follows a Last In, First Out (LIFO) structure. **Shellcode**, supplied by attackers, is often saved in an overflowed buffer and traditionally transfers control to a shell.

Buffer Overflow Defenses include **compile-time** and **run-time** approaches. Compile-time defenses involve using modern high-level languages, safe coding techniques, language extensions, and safe libraries. Run-time defenses include non-executable address space, address space randomization, guard pages, and more.

In the realm of fortifying software against the perils of buffer overflow attacks, one notable defensive strategy is the incorporation of a "**canary**." When a program is susceptible to buffer overflows the inclusion of a "stack canary" or "canary value" emerges as a crucial defensive measure.

Within the program's execution flow, prior to storing critical elements such as the function's return address on the stack, a strategically chosen value, known as the canary, is introduced. This canary value is not arbitrary; it is typically either a randomly generated or pre-established value chosen during the program's initialization phase.

As the function nears its return, a vigilant check is implemented to validate the integrity of the canary. This entails scrutinizing whether the canary value remains unaltered throughout the execution process.

The canary serves as an indicator of potential buffer overflow attempts. If an attacker endeavors to overflow a buffer and tamper with the return address, they are compelled to manipulate the canary as well. Any inconsistency in the canary value signals a potential threat, highlighting an attempted buffer overflow.

Detecting a modified canary triggers proactive measures to mitigate potential risks. These measures may include aborting the program's execution or preventing the execution of maliciously tampered code.

In essence, the canary functions as a vigilant sentinel, diligently overseeing the integrity of the stack. Its presence significantly reinforces the system's resilience against malevolent attempts to exploit buffer overflows, contributing to a more secure and robust software environment.

Other Overflow Attacks include stack overflow variants, heap overflow, global data overflow, format string overflow, and integer overflow. Replacement Stack Frame and Return to System Call are stack overflow variants, while Heap Overflow targets buffers in the heap. Global Data Overflow attacks buffers in global data.

The landscape of buffer overflow attacks is diverse, with evolving variants likely to be discovered. Defenses aim to prevent these attacks through a combination of compile-time and run-time strategies.

7. Operating System Security

Operating System Security is a critical aspect of system administration, involving careful planning, consolidation, and ongoing maintenance to safeguard against cyber threats. This process covers both general OS security and specific implementations on Linux and Windows systems.

First and foremost, meticulous planning is essential for installation, configuration, and maintenance to ensure the security of the operating system and key applications. This includes basic hardening measures that, if adopted, can prevent a significant percentage of recent cyber intrusions.

In the planning phase, understanding the system's purpose, user categories, authentication methods, information management, system access to external hosts, system administration details, and additional security measures is crucial. System deployment should be a planned process that assesses risks, secures the underlying OS and applications, protects critical content, and employs network protection mechanisms.

OS hardening involves installing and patching the OS, configuring it to remove unnecessary services, applications, and protocols, and setting up users, groups, and permissions. Additional security controls, such as antivirus, host-based firewalls, and IDS, may be installed and configured. Testing the security measures is crucial to ensure that the implemented steps address the system's needs effectively.

Ongoing security maintenance includes monitoring and analyzing logging information, regular backups, recovery from security compromises, regular testing of system security, and using appropriate software to patch/update critical software.

Linux's traditional security model relies on **discretionary access controls** (DAC), where users or processes with "root" privileges have extensive permissions. Patch management is crucial for installed server applications, requiring secure configurations and regular updates with security patches.

Network access controls are vital for securing the network, and tools like TCP wrappers are essential for checking access. Linux uses syslogd or Syslog-NG for logging, and effective log management involves balancing the number of log files used, managing their size, and configuring application logging.

User management principles emphasize careful setting of file/directory permissions, using groups to differentiate between roles, and exercising extreme care in granting/using root privileges. Running processes as unprivileged users/groups is crucial for system security, and logging is an essential resource that Linux achieves using syslogd or Syslog-NG.

Application security is a vast topic, covering various security features implemented similarly across different applications. Issues such as running as unprivileged user/group, running in a chroot jail, modularity, encryption, and logging are reviewed.

Running in a chroot jail confines a process to a subset of the file system, containing the effects of a compromised daemon. Modularity in applications is highly prized for providing a smaller attack surface.

Windows security architecture involves components like the Security Reference Monitor (SRM), Local Security Authority (LSA), Security Account Manager (SAM), and Active Directory (AD). Windows privileges are systemwide permissions assigned to user accounts, categorized as dangerous or benign.

Windows system hardening is the process of shoring up defenses, reducing exposed functionality, and disabling features to achieve attack surface reduction. Stripping privileges from an account soon after an application starts is a common practice in Windows.

Overall, operating system security requires a comprehensive approach to ensure the integrity, confidentiality, and availability of systems in the face of evolving cyber threats.

8. Access Control

Access control is a fundamental aspect of computer security, focusing on preventing unauthorized use of resources. The key principles involve user authentication, access control functions, and the assignment of access rights to users and groups. Various access control policies, including discretionary access control, mandatory access control, and role-based access control, play a crucial role in managing system security.

Access control requirements emphasize reliable input, fine and coarse specifications, least privilege, separation of duty, open and closed policies, policy combinations, conflict resolution, and administrative policies. The essential elements of access control include subjects (entities that access objects), objects (controlled resources), and access rights (ways subjects access objects, like read, write, execute).

Discretionary access control, often implemented through an access matrix, defines the access rights of subjects to objects. The access matrix is typically sparse and can be decomposed by either row or column. Protection domains, representing sets of objects with associated access rights, may be static or dynamic.

In the context of UNIX, file administration involves inodes, hierarchical directory structures, and file access control mechanisms like "set user ID" (SetUID) or "set group ID" (SetGID). UNIX systems also support Access Control Lists (ACLs), enabling the specification of additional users or groups with associated permissions beyond standard permissions.

Role-Based Access Control (RBAC), as defined by the NIST RBAC Model, restricts system access based on users' assigned roles and responsibilities. This model simplifies access management, ensuring that users have appropriate permissions without unnecessary access. RBAC is widely adopted as a security model, contributing to enhanced system security and streamlined access control.

9. Malicious Software

Malicious software, commonly known as malware, poses a significant threat to computer systems by exploiting vulnerabilities. This category includes various types of programs, such as viruses, worms, logic bombs, and backdoors. The sophistication of malware makes it a formidable challenge for computer security.

Key malware terminology includes viruses, worms, logic bombs, Trojan horses, backdoors, mobile code, auto-rooter kits, spammers, flooders, keyloggers, rootkits, zombies, and bots. These terms categorize different malicious programs based on their characteristics and actions.

Viruses are software pieces that infect programs, modifying them to include a copy of the virus, specific to the operating system and hardware. They go through phases of dormancy, propagation, triggering, and execution. Macro viruses, targeting office applications, gained prevalence in the mid-1990s.

E-mail viruses/worms, like Melissa, exploit vulnerabilities in applications and rapidly propagate through email systems. Countermeasures against malware involve prevention, detection, identification, and removal. The evolution of antivirus technology has led to signature scanners, heuristics, identification of actions, and combination packages.

Compression viruses use compression techniques to infect files, making detection challenging. Virus classification includes boot sector, file infector, and macro virus, with concealing strategies like encrypted, stealth, polymorphic, and metamorphic viruses.

The term "Digital Immune System" refers to cyber defenses inspired by biological immune systems, aiming to detect, prevent, respond to, and recover from cyber threats. Behavior-blocking software analyzes software behavior to prevent malicious activities in real-time.

Worms are self-replicating programs that propagate over networks. The Morris Worm, a notable example, targeted UNIX systems in 1988. Recent worm attacks like Code Red, SQL Slammer, Mydoom, and Stuxnet have caused significant disruptions.

Proactive worm containment involves preemptive measures to detect and control worms early. Network-based worm defense focuses on security measures at the network level to prevent worm spread.
Bots, when coordinated, form botnets and may take over computers for hard-to-trace attacks.
Rootkits, installed for admin access, make malicious and stealthy changes to the host operating system. Rootkit System Table Mods refer to unauthorized modifications to system tables, allowing rootkits to maintain a persistent and concealed presence. Countermeasures against rootkits include various strategies to detect and eliminate these stealthy threats.

10. Software Security

Software security is a critical concern as many vulnerabilities stem from poor programming practices. Issues, such as insufficient validation of program input, contribute to security flaws, as seen in the Open Web Application Security Top Ten. While software quality focuses on accidental failures, security faces intentional attacks exploiting vulnerable code.

Defensive programming, also known as secure programming, emphasizes checking assumptions and errors comprehensively. Handling program input is a common source of failure, where data from various sources must be validated for size and type. Assumptions about buffer size may lead to buffer overflow, a vulnerability not always detected by testing.

Interpreting input, whether binary or text, requires validation before use. Failure to validate may result in injection attacks, like SQL injection, influencing program execution. Cross-Site

Scripting (XSS) is another common exploit where an attacker injects malicious scripts into trusted websites, impacting unsuspecting users.

To counter these threats, validating input syntax, considering alternate encodings, and validating numeric input are essential. Writing safe program code involves correct memory use, addressing race conditions in shared memory, interacting with the operating system, managing system calls, and handling temporary files securely. Ultimately, software security should be a design goal from the outset, emphasizing prevention and robustness against intentional attacks.