366 Exam review

Security concepts : principles Terms to know: Principle , Privilege

Review Questions (1.1)

- What makes up the CIA triad?
  - What does each property mean?
- What are the golden principles?
  - What does each property mean?
- What is the difference between trusted and trustworthy?
- What is the difference between confidentiality and privacy?

Confidentiality (only authorized party can access)
Integrity (Data remains unchanged since created)
Availability (info/resources are available when needed)
Authentication (Proof that a user controls/owns a principle)
Authorization (Proof that a user has the privilege to take the action
Auditability (Ability to link past actions to the principal)

1. Trusted means a software that has been granted authorization or privileges while trustworthy means its secured

Asset: What we want to protect || Policy: Defines what system does/doesn't || Adversary: Entities who violate policies

Review Questions (1.2–1.3)

- What are definitions for the following terms?
  - Asset, security policy, adversaries, threats, attacks, vulnerabilities, mitigations, security controls
- How is risk calculated?
  - What should you use as he probability that an attack will be activated?
- What tool can help estimate impact?
- What is the role of qualitative risk analysis?

mitigation: Counter measures to reduce a threat

risk = likelihood · impact || likelihood = p(attack) · p(vulnerability)

Damage potential
Reproducibility
Exploitability
Affected users
Discoverability

Review Questions (1.4)

- What attributes define an adversary?
- Security should be considered in what phases of the software development lifecycle?
- What types of security assessment are there?

Identity, Objectives, Methods

1. Security should be a part of the entire software life cycle
2. Types of analysis

Review Questions (1.5)

- What is a trust domain?
- Why are diagrams helpful in threat modeling?
  - What is a data flow diagram?
  - What is a user workflow diagram?
- What is an attack tree?
  - What are its component parts?
  - How is it built?
- What is the role of STRIDE?

(check powerpoint pg. 8 for attack tree structure)

diagrams help describe assets, mitigations, : defenses (check pg. 6 : 7 for lect. 2)

Helps enumerate attacker method (Attack tree)

Spoofing identity
Tampering with data
Repudiation
Info disclosure
Denial of service
Elevation of privilege

Review Questions (1.6–1.8)

- What can cause gaps between a threat model and the real world?
- Why do threat models need to evolve?
- What are the following principles?
  - Simplicity and necessity, safe defaults, open design, complete mediation, least privilege, defense in depth
- Is computer security hard? Why or why not?
  - *A high-level understanding is sufficient, you don't need to memorize all 20 points listed in the book*

Invalid Assumpt. focus on wrong thr.

Principles

Simplicity / nec - keep it stupid simple
Safe default - Defualts rarely get changed
Open design - Don't rely on secrecy of your code

Cryptographic building blocks

Review Questions (2.0–2.1)

- Why should you avoid designing your own cryptographic protocols or algorithms?
- What is Kerckhoffs's principle?
- What is a cryptographic key? What is a key space?
  - Why does a key space need to be large?
- What is the difference between a passive and active attacker?
- What is the difference between information-theoretic security and computational security?

Terms:

the security system should remain secure even if everything about how the system works is public

secret value which must remain in secret for encryption to be secure

key space-# of possible keys

ITS provides absolute security that can't be broken even with unlimited computing power (one-time pads)

Comp. security- given fixed computational power : time, an attacker cannot recover the plaintext from ciphertext

Review Questions (2.2)

- What is the difference between symmetric-key and public-key encryption/decryption?
  - What keys are involved with each?
  - Who has access to what keys?
- How does a one-time pad work?
  - What type of security does it have if used properly?
  - How can it be used incorrectly?
- How do other encryption algorithms use one-time pads?

Review Question (2.3)

- What is the difference between symmetric-key and public-key encryption/decryption?
  - What keys are involved with each?
  - Who has access to what keys?
- What key is used for encryption in public-key encryption? Decryption?
- What is hybrid encryption? Why is it needed?
  - How does it work?
- What is the most common public-key cipher?
  - What type of security does it provide?

Cryptographic blocks

Cryptographic Blocks:

Review Questions (2.4)

• What is a digital signature?

—What properties are provided by digital signatures?

The three properties of Digital signatures are:

1. Data Origin authentication (data comes from the owner of the private key)
2. Data Integrity (Data has not changed since it was sent)
3. Non-repudiation (Sender cannot claim to have not sent the data)

• What is the practical significance of non-repudiation?

—Does it ever have a use?

• Should you use the same key for encryption and signing?

It prevents someone from denying they performed an action or sent a message || Creates accountability and establishes trust in digital transactions || EXAMPLES: Legal documents: Digital signatures on contracts that can't be denied later, Email communication: Proving the authenticity of sender for sensitive communications

NO, you should not use the same key for both encryption and signing. This is a security best practice for several important reasons:

- **Different security goals**: Encryption protects confidentiality while signing ensures authenticity and integrity
- **Vulnerability to attacks**: Using the same key exposes you to cryptographic attacks that could compromise both functions

Review Questions (2.5)

• What properties define a cryptographic hash function?

—Three implementation properties

H can be applied to data of any size || H produces a fixed length output || H is fast to compute

—Three security properties

Preimage Resistance (One-way): Given a hash value, it's computationally infeasible to find any input that hashes to that value

Second Pre-image Resistance (Weak collision resistance): Given an input, it's computationally infeasible to find another different input with the same hash

Collision Resistance (Strong collision resistance): It's computationally infeasible to find any two different inputs that hash to the same output

• What is a chroot jail?
—Why has it been largely replaced by? Why?
A technique that changes the apparent root directory for a process and its children || it provides basic isolation but has significant limitations || they're replaced by containers, virtual machines

• What is the difference between discretionary and mandatory access control?
Discretionary Access Control (DAC):

- Owner of resource decides who gets access (e.g., standard Linux permissions)
- Users can change permissions on resources they own
- Access decisions based on identity of users/groups

Mandatory Access Control (MAC):

- System-enforced policies that users cannot override
- Based on security labels/classifications
- Administrator-defined rules that cannot be changed by regular users
- Access governed by system security policy, not owner preference

• What is SELinux?
—What properties does it provide?
—Is it OK to turn it off?
A MAC implementation for Linux developed by the NSA || Role-based access control, Multi-level security, Fine-grained control beyond standard permissions || Generally not recommended for production environments, Turning it off significantly reduces system security

—What type of security does it provide?
Integrity verification (detects if data has been altered)
Authentication (when used in certain protocols)
Does NOT provide confidentiality (doesn't encrypt the data)

• Are hash functions computationally secure or information theoretically secure?
Hash functions are **computationally secure**, not information-theoretically secure
This means their security relies on the computational difficulty of breaking them
Given unlimited computing resources and time, they could theoretically be broken

• What is the relationship between cryptographic hash functions and digital signatures?
Hash functions are a critical component of digital signature schemes
Instead of signing entire messages (inefficient), we:

1. Create a hash of the message (small, fixed size)
2. Sign the hash with the private key

This provides efficiency, integrity, authentication

Review Questions (2.6)
• What is a message authentication code?
—How does it work?
A MAC is a small piece of data produced by a cryptographic function that combines:

1. A secret key (shared between sender and receiver)
2. The message itself

The sender computes the MAC using the message and secret key, then sends both the message and MAC
The receiver uses the same secret key and received message to compute their own MAC
If the receiver's computed MAC matches the received MAC, the message is authentic
—How does it differ from a digital signature?
Key usage: MACs use the same symmetric key for creation and verification; digital signatures use asymmetric key pairs (private key to sign, public key to verify)
Computational efficiency: MACs are typically much faster than digital signatures
Size: MACs are usually smaller than digital signatures

• What properties does it provide?
—What property does it fail to provide that digital signatures do? Why?

Authentication: Verifies the sender possesses the correct secret key

Integrity: Confirms the message hasn't been altered in transit

Some degree of accountability: Shows the sender (or receiver) had the secret key

Property MACs fail to provide: MACs cannot provide true non-repudiation like digital signatures do

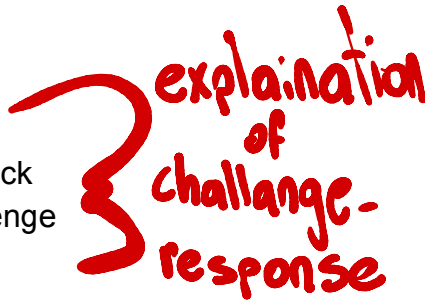
WHY?

Since both sender and receiver have the same secret key, either party could have created the MAC
In a dispute, the receiver could claim they received a legitimate message, but the sender could claim the receiver generated both the message and MAC themselves
A third party (like a judge) cannot determine which party actually created the MAC

Review Questions (4.1)
• If Alice and Bob share a key, how can they later verify
that they are talking to each other?

They can use their shared key in a challenge-response protocol:

- Alice sends Bob a random challenge (a random number)
- Bob encrypts this challenge with their shared key and sends it back
- Alice decrypts Bob's response and confirms it matches her challenge
- This proves Bob has the correct key and is therefore likely Bob



• What is a challenge-response protocol?
An authentication method where one party (verifier) sends an unpredictable challenge. The other party (prover) must provide a valid response that demonstrates knowledge of a secret. The response typically involves the challenge processed with a secret (password, key, etc.)

• What is unilateral entity authentication? Mutual entity authentication?

Unilateral Authentication: Only one party verifies the identity of the other

- Example: A website proves its identity to you via HTTPS, but you remain anonymous

Mutual Authentication: Both parties verify each other's identities

- Example: VPN connections where both the server and client authenticate to each other
- More secure but more complex to implement

• What is a zero-knowledge proof?
—Why is it needed?
A method allowing one party to prove they possess knowledge of a secret without revealing the secret itself. The verifier learns nothing about the secret, only that the prover knows it
The reasons needed: Prevents secret information from being exposed during authentication || Reduces risk of credential theft or impersonation

Review Questions (4.2–4.3)
• What are the three methods for establishing a shared key?
Pre-shared keys (manually distributed in advance)
Key transport protocols (one party generates and sends the key)
Key agreement protocols (both parties contribute to generating the key)

• How does a key transport protocol work?
—How can such a protocol be implemented using RSA?
One party generates the key and securely sends it to the other
RSA implementation:

- Alice generates a random session key
- Alice encrypts it with Bob's public RSA key
- Alice sends the encrypted key to Bob
- Only Bob can decrypt it using his private key

• How does a key agreement protocol work?
—What is the most common key agreement algorithm?
—What is unauthenticated key agreement vulnerable to?
Both parties contribute values that are mathematically combined to create a shared key
Most common algorithm: Diffie-Hellman key exchange

Vulnerability without authentication: Man-in-the-middle attacks

- An attacker can intercept and replace the key exchange messages
- The attacker establishes separate keys with both parties

• What is perfect forward secrecy?
—What protocol provides it?
Protection of past session keys even if long-term keys are compromised
Ensures that compromising current keys doesn't reveal past communications
Protocol that provides it: Ephemeral Diffie-Hellman (DHE or ECDHE)

Review Questions (4.4)
• What is an ephemeral key?
A temporary key used for just one session or transaction
Generated fresh for each session and discarded afterward

• What is implicit key authentication?
—How does it differ from explicit key authentication?
Implicit key authentication:

- Assurance that only the intended party could possibly know the shared key
- No proof they actually have computed the key yet

Explicit key authentication:

- Proof that the other party has actually computed the correct key
- Typically involves demonstrating knowledge of the key

—What is key-use confirmation?
Verification that the other party can successfully use the established key || Usually implemented by encrypting a known value with the new key|| Ensures both parties have derived the same key and can communicate securely Provides explicit key authentication

User Authentication

User Authentication

Review Questions (3.0–3.1)

• What is the difference between authentication and identification?

Identification: This is about claiming who you are || Example: Entering a username || It's simply stating "I am this person" without proving it

Authentication: This is about proving you are who you claim to be || Examples: Entering a password, using a fingerprint scanner, or providing a PIN

• What is an account? A username? A password?

An account consist of a username and a password

Username - Names to the account (public or secret)

Password - A string of enterable characters (does not have to be ASCII, though it usually is)

• What is a password composition policy?

PCPS specifies the min and max password length || specifies the character requirements and restrictions (lowercase, uppercase, digits, etc.) || however this often backfires

Passkeys: passwords that can be changed into cryptographic keys

• What are the five approaches to defeating password authentication?

1. Password Capture (Encrypt communication between user & server)
2. Phishing (attacker tricks users into giving attacker their username and password)
3. Online password guessing (tries to log into the server as the user)
4. Password Database Theft
5. Offline password guessing

Review Questions (3.1–3.2)

• Why is it important to encrypt passwords during transmission?

Protects against “man-in-the-middle” attacks might be monitoring network traffic. Ensures even if someone captures the data packs, they cant read the actual password

• Why is it important to hash passwords before storing them?

Transforms passwords into fixed-length strings of characters that can't be reversed || Example: When you enter your password, the system hashes it and compares it to the stored hash

• Why is it important to salt passwords before storing them?

—What is a salt?
Adds random data to each password before hashing || Ensures that two users with the same password will have different hash values || Makes mass password cracking significantly more difficult

• What are the three other best practices for password authentication?

1. Iterated Hashing
2. Specialized Functions
3. Keyed Hash Function:

Review Questions (3.3)

• Why is account recovery important?
People forget passwords all the time || accounts get compromised

• What are examples of account recovery schemes?
Emailed-based || SMS || Security questions

• How does the goldilocks principle apply to recovery?
—If it is too easy, what is the problem?
Attackers will “recover” victims account
—If it is too hard, what is the problem?
Users will permanently lose access

Review Questions (3.4–3.5)

• What are the three authentication factors?
—What are their strengths? Their weaknesses?
Something you know: Password, Pin ||
Something you have: Security Key, cell phone ||
Something you are: Fingerprint, Iris ||

• What is a one-time password?
—Which authentication factor is it related to?
A password that is valid for only one login session or transaction || Authentication under something you have

Access Control

Access Control

Review Questions (5.0–5.2)

Terminology: Subject - Entity that wants to take an action (read, write, exe) || Object (memory, services)

• What is a reference monitor?

—What is the definition of the following terms?

• Subject, action, object, policy

• Permission, capability-based access control, access control list (ACL)

• What are the requirements for a secure reference monitor?

The core component that enforces access control policies || It mediates all access attempts between subjects and objects || It cannot be bypassed, tampered with, or modified

Subject: The active entity requesting access (user, process, program)

Action: The operation a subject wants to perform (read, write, execute)

Object: The resource being accessed (file, memory location, device)

Policy: Rules determining whether a subject can perform an action on an object

Permission: A specific authorization granted to a subject for an object

Capability-based Access Control: Subject holds "tickets" (capabilities) that authorize access to objects

Access Control List (ACL): Object contains a list of subjects and their allowed actions

Review Questions (5.1)

• What is a memory segment?

A contiguous block of memory with specific attributes

• What permissions are associated with memory segments?

—Why are these permissions needed?

Read: Ability to view the content

Write: Ability to modify the content

Execute: Ability to run the content as code

These are needed to Prevent code injection attacks|| Stop programs from modifying other programs' data

Review Questions (5.3–5.4)

• What are possible subjects in filesystem access control?

User || group || process || everyone

• What are the possible actions in filesystem access control?

What do they do?

—Basic actions (read, write/modify, execute)

—Special actions ()

—Advanced actions (take ownership, change permissions, write attributes)

Review Questions (5.3–5.4)

• Filesystem access control on Linux

—What is the UGO model?

—How are permissions determined?

—How does inheritance work?

UGO stands for User, Group, other || Each entity (U, G, O) gets a combination of three basic permissions: Read (r): View file contents or list directory contents, Write (w): Modify file or create/delete files in directory, Execute (x): Run file as program or access directory contents

• Filesystem access control on Windows

—What access control model does Windows use?

—How does inheritance work?

Windows has true inheritance for access permissions

By default, new files/folders inherit permissions from their parent container

Inheritance can be:

- Enabled or disabled for specific containers
- Configured for specific permission types
- Set to "inherit only" or "inherit and apply to this object too"

Review Questions (5.5–5.7)

• What are hidden files?

—Why can’t they be used to secure files?

Hidden files are files that don’t appear in normal directory listings but still exist in the filesystem || They’re easily revealed with simple commands or setting changes

Software Security

Software Security:

Review Questions (6.1)

• What is time-of-check to time-of-use (TOCTOU) referring to?
Time-of-Check - When you check whether an action is allowed
Time-of-Use - When you execute the action
TOCTOU race: The potential that the action becomes disallowed between checking and executing the action || Occurs due to multi-threading

• What is a race condition?
• How does TOCTOU relate to race conditions?
TOCTOU is a specific type of race condition. The "race" happens in the window between the check and the use:

1. Your program checks if an action is allowed
2. Before your program completes the action, something changes the conditions
3. Your program performs the action based on the now-outdated verification

• What are the possible implication of TOCTOU races?
Privilege escalation: An attacker might gain unauthorized access to resources
Data corruption: Data might be modified in unexpected ways
Information disclosure: Unauthorized access to sensitive information
Denial of service: System crashes or resource exhaustion

Review Questions (6.2)

• Why is C more vulnerable to integer-based vulnerabilities?
C is more vulnerable to integer-based vulnerabilities because:

- It performs minimal runtime checking for integer operations
- It silently wraps around on overflow/underflow without generating errors or exceptions
- It allows implicit type conversions that can lead to unexpected results
- It gives programmers direct control over memory and low-level operations

• What is integer overflow? What is integer underflow?

• What is problematic with how C handles casting between

Integer overflow:

- Occurs when an arithmetic operation attempts to create a value that exceeds the maximum value representable by the data type
- Example: For an 8-bit unsigned integer (max 255), 255 + 1 = 0 (wraps around)

Integer underflow:

- Occurs when an arithmetic operation attempts to create a value that falls below the minimum value representable by the data type
- Example: For an 8-bit unsigned integer (min 0), 0 - 1 = 255 (wraps around)

primitive data types of different sizes? Different signs?

Different sizes:

- When casting from larger to smaller types, data can be truncated without warning
- Example: casting a 32-bit int to an 8-bit char will lose the high-order bits

Different signs:

- When casting between signed and unsigned types, the bit pattern may remain the same but the interpretation changes
- A negative number cast to unsigned becomes a large positive number
- Example: (unsigned int)(-1) becomes a very large positive number (typically 4,294,967,295 for 32-bit int)

• How does C handle type coercion?
—For operands
—For results
• What are potential impacts for integer-based vulnerabilities?

What are potential impacts for integer-based vulnerabilities?

- **Buffer overflows**: Integer overflows in buffer size calculations can lead to buffer overflows
- **Memory corruption**: Wrong size calculations can corrupt memory
- **Logic errors**: Security checks can be bypassed when integers wrap around
- **Denial of service**: Program crashes from unexpected values

- **Code execution**: In severe cases, arbitrary code execution by exploiting memory corruption
- **Resource exhaustion**: Allocating incorrect amounts of memory or resources

Review Questions (6.3)

• How is memory laid out in the x86-64 architecture

—Text, data, shared libraries, stack, heap

Text/Code segment: Contains executable instructions (read-only)

Data segment: Contains initialized global and static variables

BSS segment: Contains uninitialized global and static variables

Heap: Dynamic memory allocation (grows upward to higher addresses)

Shared libraries: Dynamically loaded libraries mapped into memory

Stack: Local variables and function call information (grows downward to lower addresses)

• What is the stack used for?

Storing local variables

Passing function parameters

Storing return addresses (where to return after a function completes)

• What is a stack frame?

—What data does it store?

A stack frame is a portion of the stack allocated for a function call.

• What is a buffer?

—How can it overflow?

—What happens when it overflow?

A buffer is a temporary storage area in memory used to hold data.

How can it overflow?

- When more data is written to the buffer than it was allocated to hold
- When input validation is missing or inadequate
- When using functions that don't check boundaries (like `strcpy`, `gets` in C)

What happens when it overflows?

- Adjacent memory is overwritten
- Stack frame information can be corrupted

What damage can be caused by a buffer overflow?
—What are the four different levels?
—What factors lead to vulnerabilities be placed in these four levels?
• Where can buffer overflows occur?
Denial of Service (DoS): Program crashes or becomes unusable
Information disclosure: Memory contents are leaked
Privilege escalation: Attacker gains higher privileges within the application
Arbitrary code execution: Attacker takes complete control by executing their own code

• Why are buffer overflows most common in C/C++?
No automatic bounds checking
Use of unsafe functions (`strcpy`, `gets`, `sprintf`)
Direct memory manipulation
Manual memory management

Review Questions (6.4)
• What are the following buffer overflow defenses? You should be able to explain how they work and their limitations.
—Non-executable stack and heap
—Stack canary
—Address space layout randomization (ASLR)
—Using bounds-checking functions
—Language features
Non-executable stack and heap (NX/DEP)
How it works: Marks memory regions as non-executable
Limitations: Cannot stop return-to-libc or ROP (Return-Oriented Programming) attacks which reuse existing executable code

Stack canary
How it works: Places a random value (canary) between buffer and control data; checks if it's modified before function returns
Limitations: Can be bypassed if the canary value is leaked or if the overflow occurs elsewhere

Address Space Layout Randomization (ASLR)
How it works: Randomly arranges memory addresses of process components (stack, heap, libraries)
Limitations: Can be defeated by information leakage or brute force on systems with low entropy

Using bounds-checking functions

How it works: Uses safer functions that specify buffer sizes (strncpy instead of strcpy, snprintf instead of sprintf)

Limitations: Still vulnerable if size parameters are calculated incorrectly; some functions still have subtle issues

Language features

How it works: Modern languages (Java, Python, C#, Rust) have built-in bounds checking, automatic memory management

Limitations: Performance overhead; legacy code remains an issue; interface with unsafe code can reintroduce vulnerabilities

Review Questions (6.5)

• What is privilege execution

—How does this relate to buffer overflow vulnerabilities?

The process where an attacker gains higher-level permissions or access rights than they should have

Moving from limited user access to administrative or system-level access

• How does lateral movement within an organization work?

Lateral movement refers to techniques that attackers use to move through a network after gaining initial access. It involves:

1. **Reconnaissance**: Mapping the internal network to identify potential targets
2. **Credential harvesting**:
    - Stealing passwords, hashes, or authentication tokens
    - Using techniques like memory scraping, keylogging, or accessing credential stores
3. **Exploitation methods**:
    - Pass-the-hash/pass-the-ticket attacks
    - Using stolen credentials to access other systems
    - Exploiting trust relationships between systems
    - Abusing administrative tools and protocols (like WMI, PowerShell, PsExec)
    - Exploiting vulnerabilities in internal systems

Malicious Software

Malicious Software:

Review Questions (7.1–7.2)

• Be able to define the following terms:

—Legitimate software, harmful software, malicious software, potentially unwanted software (PUS)

Legitimate software: Programs designed for useful purposes with no harmful intent

Harmful software: Software that causes damage, regardless of intent (includes buggy legitimate software)

Malicious software (malware): Programs specifically designed with malicious intent to cause harm

Potentially Unwanted Software (PUS): Software that isn't clearly malicious but may have unwanted effects (adware, spyware, etc.)

• What is a computer virus?

Legitimate software: Programs designed for useful purposes with no harmful intent

Harmful software: Software that causes damage, regardless of intent (includes buggy legitimate software)

Malicious software (malware): Programs specifically designed with malicious intent to cause harm

Potentially Unwanted Software (PUS): Software that isn't clearly malicious but may have unwanted effects (adware, spyware, etc.)

—What are the four parts of a computer virus's structure?

• What occurs in each part?

1. Dormancy period

— Executes automatically with no dormancy period

2. Propagation strategy

— Automatically and continuously attacks other systems

— Spreads between machines on the same computer network

— Does not require user interaction

• Exploits software vulnerabilities to avoid the need for user interaction

3. Trigger condition

— Same as a computer virus

4. Payload

— Same as a computer virus

• What is a computer worm?

—How does it differ from a computer virus?

Review Questions (7.1–7.2)

• How are computer viruses spread?

—What role does social engineering play?

—What role does phishing play?

—What role does email play?

—What role do data files play? (As opposed to executables.)

Role of social engineering:

Manipulates users into executing infected files

Creates convincing scenarios that bypass security awareness

Exploits trust and psychological vulnerabilities

Role of phishing:

Sends deceptive messages appearing to be from trusted sources

Tricks users into downloading/executing malware

Often includes malicious links or attachments

Role of email:

Serves as a primary vector for malware distribution

Allows easy transmission of infected attachments

Enables mass delivery to many potential victims

Can carry malicious macros in documents

Role of data files:

Exploit vulnerabilities in applications that process those files

Contain macros or scripts that execute malicious code

Leverage file format vulnerabilities (buffer overflows, etc.)

Can trigger exploits without appearing as executable files

• What is a zero-day exploit?

—How does it relate to malware?

A zero-day exploit targets a previously unknown vulnerability before developers can create and distribute a patch.

Relation to malware:

- Provides highly effective infection vectors for malware
- Allows malware to bypass security measures

Review Questions (7.1–7.2)

• How are signatures used to detect malware?
—What is a malware signature?
—What is a behavioral signature?
—How do they differ?
—What are their limitations?

Malware Signature

Unique patterns or sequences of bytes in malware code
Like a "fingerprint" that identifies specific malware
Can be based on specific strings, code segments, or file characteristics
Used by traditional antivirus software

Behavioral Signature

Patterns of suspicious activities or behaviors that malware exhibits when running
Focuses on what the malware does rather than how it looks
Examples: unusual registry changes, network connections, file operations
Used in behavioral detection systems

Key Differences

Malware signatures examine static code; behavioral signatures monitor runtime activity
Malware signatures target specific instances; behavioral signatures identify suspicious patterns

Limitations

Malware signatures:
Ineffective against new or modified malware
Easily defeated by obfuscation techniques
Requires constant updates to signature databases
Cannot detect zero-day threats

Behavioral signatures:
May generate false positives for legitimate activities
Resource-intensive monitoring
Clever malware can mimic legitimate behavior
Harder to develop accurate behavioral models

• How is integrity checking used to prevent malware?
—What is code signing? How does it stop malware?

Integrity Checking
Verifies that files haven't been modified from their original state
Uses checksums or cryptographic hashes to detect changes
Creates a baseline of legitimate files and monitors for unauthorized changes

Code Signing
Process of digitally signing executables and scripts with a certificate
Verifies both the identity of the software author and that the code hasn't been altered
Uses public key cryptography to validate authenticity

How code signing stops malware:
Operating systems can be configured to reject unsigned code
Warns users when running unsigned applications
Establishes a chain of trust for software distribution

Review Questions (7.3)

• What are four ways viruses modify their payload to avoid detection?
—How does each work?

Encryption:
Encrypts the virus body with different keys
Only the decryption routine remains constant
Makes signature detection difficult as the virus appears different in each infection

Polymorphism:
Changes both its encryption method and decryption routine
Uses different code sequences that perform the same function
Mutates its appearance while maintaining functionality

Metamorphism:
Completely rewrites its code when infecting new files
No constant parts between generations
Uses code substitution, insertion of junk code, and reordering of instructions

External Decryption Viruses
No encryption is used and entire payload is recompiled

• What are three techniques that computer worms use to spread?
—How does each work?
—What is a flash worm?

Review Questions (7.4)

• What are each of the following types of malware?
—Trojan horse
—Backdoor
—Key logger
—Rootkit

• How do the above types of malware work?

Types of Malware

Trojan Horse
Malware disguised as legitimate software
Appears useful but contains hidden malicious functionality
Relies on users willingly installing it
Does not self-replicate like viruses or worms

Backdoor
Creates an alternative way to access a system while bypassing normal authentication
Provides remote control capabilities to attackers
Can be installed by other malware or directly by attackers
Often maintains persistent access even after system reboots

Keylogger
Records keystrokes typed by users
Designed to capture sensitive information like passwords and credit card numbers
Can be software-based or hardware devices
May include screenshots, clipboard monitoring, and form grabbing

Rootkit
Designed to gain privileged access while hiding its presence
Modifies system core components to evade detection
Operates at a deep level of the operating system
Can hide files, processes, network connections, and registry entries
Extremely difficult to detect and remove once installed

Review Questions (7.5)

• How does system call hooking work?

—How does it differ from inline hooking?

—How are these techniques used by rootkits?

System Call Hooking

Intercepts calls made by applications to the operating system
Modifies system call tables to redirect to malicious code
Operates at the boundary between user space and kernel space

Inline Hooking

Directly modifies function code in memory
Replaces the beginning of a function with a jump to malicious code
Can target specific functions rather than entire system call interfaces

Rootkit Usage

Both techniques allow rootkits to hide their presence
Enable interception and modification of system information
Filter out information about malicious files, processes, or network connections
Allow rootkits to maintain persistence and avoid detection

• What are five methods for an attacker to install a rootkit?

Review Questions (7.6–7.7)

• What are drive-by downloads?

—How do they work?

Drive-by downloads are malware installations that occur without the user's knowledge or consent when visiting a website.

How they work:

Exploit browser, plugin, or operating system vulnerabilities
Execute malicious code automatically without user interaction
Often use multiple stages to deliver the final payload
May use obfuscation to evade detection

—What are three avenues for drive-by downloads?

Compromised legitimate websites: Attackers inject malicious code into trusted sites
Malicious advertisements (malvertising): Delivering exploits through ad networks
Malicious redirect chains: Series of redirects leading to exploit kits

• What is ransomware?
—How does encrypting ransomware work?
Ransomware is malware that restricts access to a computer system or data, demanding payment to restore access.

• How are cryptographic keys generated, stored, and used?
—What types of non-encrypting ransomware are there?
—Why is encrypting ransomware the most common type of ransomware
Encrypting Ransomware
Encrypts victim's files with a strong encryption algorithm
Demands payment (usually cryptocurrency) for the decryption key
Uses a combination of symmetric and asymmetric encryption

Cryptographic Key Management
Generation: Creates a unique encryption key for each victim
Storage: The attacker's server stores the decryption keys
Usage: Symmetric key encrypts files; asymmetric key protects the symmetric key

Non-encrypting Ransomware Types
Locker ransomware: Locks the system interface without encrypting files
Scareware: Uses intimidation and false claims to extract payment
Doxware/Leakware: Threatens to publish stolen sensitive data

Encrypting Ransomware Prevalence
More difficult to bypass than screen lockers
Creates unrecoverable damage without the decryption key

Review Questions (7.7)
• Be able to define the following terms:
—Botnet, bot master, botnet herder, bot, zombie
Botnet: Network of compromised computers controlled remotely
Bot Master/Botnet Herder: Person who controls the botnet
Bot/Zombie: Individual compromised computer in the botnet

• What does the attacker use to control their botnet?
• What are the four stages of a botnet?
—What happens in each stage?
—What role do zero-day exploits have in a botnet?
• For what purposes can a botnet be used?

Infection:
Initial compromise of vulnerable systems
Distribution through exploits, phishing, or drive-by downloads
Zero-day exploits play a critical role by targeting unknown vulnerabilities

Communication:
Establishing connection to C&C infrastructure
Registering with the botnet
Receiving initial instructions

Command and Control:
Receiving and executing commands from the botmaster
Updating malware components
Maintaining persistence

Attack/Monetization:
Executing the botnet's primary purpose
Generating revenue for the attackers
Expanding to new victims

Botnet Uses
Distributed Denial of Service (DDoS) attacks
Spam distribution
Cryptocurrency mining
Data theft
Credential harvesting
Proxy services for other attacks