Vulnerabilities and Attacks

Objectives:

- 2.2: Explain common threat vectors and attack strategies
- 2.3: Explain various types of vulnerabilities
- 2.4: Given a scenario, you must be able to analyze indicators of malicious activity
- 2.5: Explain the purpose of mitigation techniques used to secure the enterprise
- 4.1: Given a scenario, you must be able to apply common security techniques to computing resources

- **Vulnerabilities and Attacks**
    - *Vulnerabilities*
        - Weaknesses or flaws in hardware, software, configurations, or processes
        - Consequences
            - Unauthorized Access
            - Data Breaches
            - System Disruptions
    - *Attacks*
        - Deliberate actions by threat actors to exploit vulnerabilities
        - Forms
            - Unauthorized Access
            - Data Theft
            - Malware Infections
            - DoS Attacks
            - Social Engineering

- *Hardware Vulnerabilities*
  - Focus
    - Firmware
    - End-of-life systems
    - Missing patches
    - Misconfigurations
  - Mitigation
    - Harden systems
    - Patch
    - Enforce baseline configurations
    - Decommission old assets
    - Isolation
- Bluetooth Vulnerabilities and Attacks
  - Vulnerabilities attacks like the following
    - Bluesnarfing
    - Bluejacking
    - Bluebugging
    - Bluesmark
    - Blueborne
- Mobile Vulnerabilities and Attacks
  - Topics
    - Sideload
    - Jailbreaking
    - Insecure connections
  - Mitigation
    - Patch Management

    - Mobile Device Management
    - Prevent sideloading
    - Rooting
- *Zero-Day Vulnerabilities*
  - Newly discovered and exploited vulnerabilities
  - Challenge
    - No known defenses or mitigations
- *Operating System Vulnerabilities*
  - Types
    - Unpatched systems
    - Zero-days
    - Misconfigurations
    - Data exfiltration
    - Malicious updates
  - Protection
    - Patching
    - Configuration management
    - Encryption
    - Endpoint protection
    - Firewalls
    - IPS
    - Access controls
- SQL and XML Injections
  - *SQL Injection*
    - Exploits web app or database vulnerabilities

  - *XML Injection*
    - Targets XML data processing
- Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) Attacks
  - *Cross-Site Scripting (XSS)*
    - Injects malicious scripts into web pages
  - *Cross-Site Request Forgery (CSRF)*
    - Triggers actions on different websites without user consent
- *Buffer Overflows*
  - Software vulnerability when more data is written to a memory buffer than it can hold
- *Race Conditions*
  - Multiple processes or threads accessing shared resources simultaneously
  - Key Terms
    - Time-of-Check (TOC)
    - Target-of-Evaluation (TOE)
    - Time-of-Use (TOU)

- **Hardware Vulnerabilities**
  - *Hardware Vulnerabilities*
    - Security flaws or weaknesses in a device's physical components or design that can be exploited to compromise system integrity, confidentiality, or availability
  - Types of Hardware Vulnerabilities
    - *Firmware Vulnerabilities*
      - Specialized software stored on hardware devices
      - Can grant attackers full control, leading to unauthorized access or takeover

        - Vulnerabilities due to insecure development, outdated practices, and overlooked updates
    - End-of-Life, Legacy, and Unsupported Systems
        - *End-of-life*
            - No updates or support from the manufacturer
        - *Legacy*
            - Outdated and superseded by newer alternatives
        - *Unsupported*
            - No official support, security updates, or patches
        - Vulnerable due to the lack of patching and updates
    - *Unpatched Systems*
        - Devices, applications, or software without the latest security patches
        - Exposed to known exploits and attacks
        - Risk from oversight, negligence, or challenges in updating
    - *Hardware Misconfigurations*
        - Incorrect device settings or options
        - May lead to vulnerabilities, performance issues, or unintended behavior
        - Caused by oversight, lack of understanding, or deployment errors
- Mitigation Strategies
    - *Hardening*
        - Tighten security by closing unnecessary ports, disabling services, and setting permissions
    - *Patching*
        - Regular updates to fix known vulnerabilities in software, firmware, and applications

        - *Configuration Enforcement*
            - Ensure devices adhere to secure configurations
        - *Decommissioning*
            - Retire end-of-life or legacy systems posing security risks
        - *Isolation*
            - Isolate vulnerable systems from the enterprise network
        - *Segmentation*
            - Divide the network into segments to limit the impact of breaches

- **Bluetooth Vulnerabilities and Attacks**
    - *Bluetooth*
        - Wireless technology for short-distance data exchange
        - It's commonly used for connecting devices but presents security challenges
        - Vulnerabilities include
            - *Insecure pairing*
                - Occurs when Bluetooth devices establish a connection without proper authentication
            - *Device spoofing*
                - Occurs when an attacker impersonates a device to trick a user into connecting
            - On-path attacks
                - Exploits Bluetooth protocol vulnerabilities to intercept and alter communications between devices without either party being aware

- Different Types of Bluetooth Attacks
  - *Bluejacking*
    - Sending unsolicited messages to a Bluetooth device
    - Often used for pranks or testing vulnerabilities
  - *Bluesnarfing*
    - Unauthorized access to a device to steal information like contacts, call logs, and text messages
  - *Bluebugging*
    - Allows attackers to take control of a device's Bluetooth functions
    - Can make calls, send messages, or access the internet
  - *Bluesmack*
    - Denial-of-service attack by overwhelming a device with data, causing it to crash or become unresponsive
  - *BlueBorne*
    - Spreads through the air to infect devices without user interaction
- Best Practices for Secure Bluetooth Usage
  - Turn off Bluetooth when not in use
    - Reduces the attack surface and exposure to threats
  - Set devices to "non-discoverable" mode by default
    - Prevents unsolicited connection attempts
  - Regularly update firmware
    - Ensures security is up-to-date with patches for vulnerabilities
  - Only pair with known and trusted devices
    - Mitigates the risk of connecting to malicious devices

        - Use a unique PIN or passkey during pairing
            - Adds security during the pairing process
        - Be cautious of unsolicited connection requests
            - Avoid accepting requests blindly
        - Use encryption for sensitive data transfers
            - Scrambles data to prevent unauthorized access

- **Mobile Vulnerabilities and Attacks**
    - Different Types of Mobile Vulnerabilities
        - *Sideloading*
            - Installing apps from unofficial sources bypassing the device's default app store
            - Can introduce malware; download apps from official sources with strict review processes
            - Mitigation techniques
                - always download apps from an official and trusted source
        - *Jailbreaking/Rooting*
            - Gives users escalated privileges but exposes devices to potential security breaches
            - Prevents installation of manufacturer updates, leaving devices vulnerable
        - Insecure Connection Methods
            - Using open Wi-Fi networks or pairing with unknown devices over Bluetooth exposes devices to attacks
            - Mitigation techniques
                - Use cellular data for more secure connections
                - Connect only to known devices and set devices to

        non-discoverable when not pairing
        - Use long, strong, complex passwords
        - Use 802.1x authentication methods
  - Mobile Device Management (MDM)
    - MDM solutions minimize mobile vulnerabilities by
      - Patching
        - Ensuring devices receive necessary security updates
      - Configuration Management
        - Enforcing standardized configurations for security
      - Best Practice Enforcement
        - Disabling sideloading, detecting jailbreaking/rooting, and enforcing VPN use

- **Zero-day Vulnerabilities**
  - *Zero-day Vulnerabilities*
    - Discovered or exploited before vendors issue patches
  - *Zero-day Exploits*
    - Attacks that target previously unknown vulnerabilities
  - *Zero-day*
    - Refer to the vulnerability, exploit, or malware that exploits the vulnerability
  - Zero-Day Exploits and Value
    - Zero-day exploits are significant in the cybersecurity world and can be lucrative
    - Bug bounty hunters can earn money by discovering zero-day vulnerabilities
    - Zero-days are also sold to government agencies, law enforcement, and criminals
    - Threat actors save zero-days for high-value targets, using generic malware for initial attempts

        - An up-to-date antivirus can detect known vulnerabilities' exploitation
        - Countries and nation states may stockpile zero-days for espionage and strategic operations

- **Operating System Vulnerabilities**
    - Unpatched Systems
        - Lack the latest security updates, making them vulnerable
        - Attackers exploit known vulnerabilities in unpatched systems
        - To mitigate unpatched system vulnerabilities, ensure regular system updates and patches, either automatically or manually
    - Zero-Day Vulnerabilities
        - *Zero-days*
            - Unknown vulnerabilities to developers and attackers
        - Security solutions like host-based intrusion prevention systems (IPS) can help detect and block suspicious activities
        - Frequent system and software updates provide additional defense against potential zero-day exploits
    - *Misconfigurations*
        - Occurs when system settings are improperly configured
        - Standardize and automate configuration processes with configuration management tools
        - Conduct periodic audits and reviews to identify and mitigate vulnerabilities due to misconfigurations
    - *Data Exfiltration*
        - Involves unauthorized data transfers from an organization to an external location
        - Protect against data exfiltration with encryption for data at rest and endpoint

        protection tools
    - Endpoint protection tools can monitor and restrict unauthorized data transfers
  - *Malicious Updates*
    - Appear as legitimate security updates but contain malware or exploits
    - Source updates from trusted vendors and official channels
    - Maintain application allow lists, verify update authenticity with digital signatures and hashes

- **SQL and XML Injections**
  - *Injection Attack*
    - Involves sending malicious data to a system for unintended consequences
    - SQL injection and XML injection share the goal of inserting code into systems
  - SQL (Structured Query Language) Injection
    - *SQL Data*
      - Used to interact with databases
      - Four main SQL actions
        - Select
          - Used to read data from the database
        - Insert
          - Used to write data into the database
        - Delete
          - Used to remove data from the database
        - Update
          - Overwrite some data in the database

        - Example statement
            - SELECT * FROM USERS WHERE userID = 'Jason' AND password = 'pass123';
    - *SQL Injection*
        - Involves inserting malicious SQL code into input fields
        - Attackers use URL parameters, form fields, cookies, POST data, or HTTP headers for SQL injection
        - Prevention
            - Input validation
            - Sanitize user data
            - Use a web application firewall
        - *SQL Injection Attempt*
            - Involve statements like " ‘ OR 1=1"
            - Example
                - Original SQL statement
                    - SELECT * FROM USERS WHERE userID = 'Jason' AND password = 'pass123';
                - Injected SQL statement
                    - SELECT * FROM Users WHERE userID = 'Jason' AND password = '' OR 1=1;
- *XML (Extensible Markup Language) Injection*
    - *XML Data*
        - Used for data exchange in web applications
        - Should be sent within an encrypted tunnel, like TLS
        - Input validation and sanitization are crucial for protection
        - Appears as tagged fields

  - Example

    -
      ```
      <?xml version="1.0" encoding="UTF-8"?>
      <question>
          <ID>SECURITY-002-0001</ID>
          <title>Is this an XML vulnerability?</title>
          <choice1>Option 1</choice1>
          <choice2>Option 2</choice2>
      </question>
      ```

- XML Exploits
  - *XML Bomb (Billion Laughs Attack)*
    - Consumes memory exponentially, acting like a denial-of-service attack
  - *XXE (XML External Entity) Attack*
    - Attempts to read local resources, like password hashes in the shadow file
    - Example

      -
        ```
        <?xml version="1.0" encoding="UTF-8"?>
        <!DOCTYPE foo [
        <!ELEMENT foo ANY>
        <!ENTITY xxe SYSTEM "file:///etc/shadow">
        ]>
        <foo>Some data</foo>
        ```
- Prevention
  - Implement proper input validation

- **XSS and XSRF**
  - *Cross-Site Scripting (XSS)*
    - Injects a malicious script into a trusted site to compromise the site's visitors
    - Goal
      - Have visitors run a malicious script so your system will process it, bypassing the normal security mechanisms
    - Mitigate the threat with proper input validation
    - Four steps to an XSS attack
      - The attacker identifies an input validation vulnerability within a trusted website
      - The attacker crafts a URL to perform a code injection against the trusted website
      - The trusted site will return a page containing the malicious code injected
      - The malicious code runs in the client's browser with permission level as the trusted site
    - Functions of a XSS Attack
      - Defacing the trusted website
      - Stealing the user's data
      - Intercepting data or communications
    - Types of XSS Attacks
      - *Non-Persistent XSS*
        - A XSS attack that only occurs when it is launched and only happens once
        - Server executes the attack (Server-side scripting attack)
      - *Persistent XSS*
        - Allows an attacker to insert code into a backend database used by

that trusted website

        - Server executes the attack (Server-side scripting attack)
      - *Document Object Model (DOM) XSS*
        - Exploits the client’s web browser using client-side scripts to modify the content and layout of the web page
        - Client’s device executes the attack (Client-side scripting attack)
        - Can be used to change the DOM environment
        - Runs using the logged in user’s privileges on the local system
  - *Session Management*
    - Enables web applications to uniquely identify a user across several different actions and requests
    - Fundamental security component in modern web applications
    - Cookie Tracking
      - *Cookie*
        - Text file used to store information about a user when they visit a website
        - *Non-persistent cookies*
          - Also known as a session cookie
          - Resides in memory and are used for a very short time period
          - Deleted at the end of the session
        - *Persistent cookies*
          - Stored in the browser cache until either deleted by a user or expire
    - *Session Hijacking*
      - Type of spoofing attack where the attacker disconnects a host and then

replaces it with his or her own machine by spoofing the original host IP

- *Session Prediction*
  - Type of spoofing attack where the attacker attempts to predict the session token in order to hijack the session
  - Prevent these attacks by using a non-predictable algorithm to generate session tokens

- *XSRF*
  - Malicious script is used to exploit a session started on another site within the same web browser
  - Can be disguised
    - Can use tags, images, and other HTML code
  - Doesn’t need victim to click on a link
  - Prevention
    - Use user-specific tokens in all form submissions
    - Add randomness and prompt for additional information whenever a user tries to reset their password
      - Require two-factor authentication
    - Require users to enter their current password when changing their password

- **Buffer Overflow**
  - *Buffer Overflow Attack*
    - Occurs when a process stores data outside the memory range allocated by the developer
    - Common initial attack vector in data breaches
      - 85% of data breaches used buffer overflow as the initial vector

        - Attackers exploit the excess data written beyond buffer boundaries to manipulate program execution
    - *Buffers*
        - Temporary storage areas used by programs to hold data
        - They have a defined memory capacity, just like a glass holding a limited amount of water
        - Overflowing a buffer results in data spilling into adjacent memory locations, causing unintended consequences
    - Technical Aspects
        - *Stack*
            - Programs have a reserved memory area called a stack to store data during processing
        - The stack uses a "first in, last out" organization
        - Stack contains return addresses when a function call instruction is received
        - Attackers aim to overwrite the return address with their malicious code's address
    - *Smashing the Stack*
        - Attackers aim to overwrite the return address with a pointer to their malicious code
        - When the non-malicious program hits the modified return address, it runs the attacker's code
        - This gives attackers a command prompt on the victim's system for remote code execution
    - *NOP Slide*
        - Attackers fill the buffer with NOP (No-Operation) instructions
        - The return address slides down the NOP instructions until it reaches the attacker's code

- Mitigations against Buffer Overflow Attack
  - *Address Space Layout Randomization (ASLR)*
    - Helps prevent attackers from guessing return pointer addresses
    - Randomizes memory addresses used by well-known programs, making it harder to predict the location of the attacker's code

- **Race Conditions**
  - *Race Conditions*
    - Software vulnerabilities related to the order and timing of events in concurrent processes
    - Exploiting race conditions allows attackers to disrupt intended program behavior and gain unauthorized access
  - Understanding Race Conditions
    - Race conditions occur when multiple threads or processes access and manipulate shared resources simultaneously
    - *Dereferencing*
      - Software vulnerability that occurs when the code attempts to remove the relationship between a pointer and the thing that the pointer was pointing to in the memory which allows changes to be made
    - Vulnerabilities stem from unexpected conflicts and synchronization issues
  - Exploiting Race Conditions
    - Attackers exploit race conditions by timing their actions to coincide with vulnerable code execution
    - Exploitation may lead to unauthorized access, data manipulation, or system crashes

- *Dirty COW Exploit*
    - A real-world example of race condition exploitation
    - Targeted Linux and Android systems, leveraging race conditions in the Copy On Write function
- Types of Race Conditions
    - *Time-of-Check (TOC)*
        - Attackers manipulate a resource's state after it is checked but before it is used
        - For example, overdrawing a bank account due to a time delay between checking and transferring funds
    - *Time-of-Use (TOU)*
        - Attackers alter a resource's state after it is checked but before it is used
        - Focuses on the time when the resource is utilized, rather than the time of the initial check
    - *Time-of-Evaluation (TOE)*
        - Attackers manipulate data or resources during the system's decision-making or evaluation process
        - Can lead to incorrect results or unexpected behavior
- Mitigating Race Conditions
    - Use locks and mutexes to synchronize access to shared resources
        - *Mutex*
            - Mutually exclusive flag that acts as a gatekeeper to a section of code so that only one thread can be processed at a time
            - Mutexes ensure only one thread or process can access a specific section of code at a time
    - Properly design and test locks to prevent deadlocks

- *Deadlock*
    - Occurs when a lock remains in place because the process it’s waiting for is terminated, crashes, or doesn’t finish properly, despite the processing being complete