HAPTER

UNIT – I
Introduction to web technology and information Gathering

TCP, HTTP/S Protocol Basics, Encoding, Origin, Cookies, Sessions, Fingerprinting the web server, Subdomains enumeration, finding virtual hosts, fingerprinting custom applications, Enumerating resources, Relevant information through misconfigurations, Google hacking.

UNIT – II
Web Application Security Vulnerability Terminology

Introduction to Vulnerability Assessment, Life cycle of Vulnerability Assessment, Vulnerability Scanners, Unknown Vulnerability, False Positive, CVE, CWE, Common Vulnerability Scoring System (CVSS), STRIDE, DREAD, Secure Source Code Review.

UNIT – III
Proxy and Interception

Burp Suite / OWASP Zed Attack Proxy (ZAP): Logging and monitoring, learning tools to spider a website, analyzing website content, Brute forcing unlinked files and directories via ZAP and ffuf, Web authentication mechanisms, Fuzzing with Burp Intruder, Username harvesting and password guessing, Burp sequencer, Session management and attacks, Authentication and authorization bypass

UNIT – IV
Attack Landscape - Web Application Security 0786

OWASP 10 Ten – Injection, Broken Authentication, Sensitive Data Exposure, XML External Entities, Broken Access Control, Security Misconfiguration, Cross Site

Scripting, Insecure Deserialization, Using Components with Known Vulnerabilities, Insufficient Logging & Monitoring, Cross Site Request Forgery, File Inclusion, Click Jacking, File Inclusion, File Upload, Insecure Captcha, SSRF/XSPA.

UNIT – V
Advanced Web Security Pen-Testing

Web Service concepts, REST concepts, SQL Injection - Vulnerable code, Sensitive data in GET, Weak Auth tokens & IDOR, Leaky APIs and Insecure Data Storage, Automated Scanning with Fuzzy API / Astra / other industry standard tools, Introduction to CMS and Docker containers security.

Reference Books

1. Web Application Security, A Beginner's Guide, by Bryan Sullivan, Vincent Liu
2. The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws by Dafydd Stuttard, Marcus Pinto, Wiley Publication.
3. Burp Suite Cookbook, by Sunny Wear, Packt Publishing, McGraw Hill, Publication

1. What is VAPT and Importance of vulnerability assessments.

What is VAPT (Vulnerability Assessment and Penetration Testing)?

VAPT is a security testing approach that combines:

1. **Vulnerability Assessment (VA)**: Identifies and classifies potential vulnerabilities in systems, networks, or applications using automated tools.
2. **Penetration Testing (PT)**: Simulates real-world attacks by exploiting the identified vulnerabilities to understand their potential impact and risk.

Key Point: VAPT provides a comprehensive analysis of security weaknesses and helps prioritize remediation efforts.

Importance of Vulnerability Assessment:

1. **Early Detection**: Identifies vulnerabilities before attackers exploit them.
2. **Risk Mitigation**: Reduces the risk of security breaches by addressing weaknesses proactively.
3. **Compliance**: Helps organizations meet regulatory and compliance requirements (e.g., PCI-DSS, ISO 27001).
4. **Cost Reduction**: Prevents financial and reputational damage by minimizing security incidents.
5. **Continuous Security Improvement**: Regular assessments help maintain a secure environment as new threats emerge.

Key Point: Vulnerability assessment helps maintain security, reduce risks, and protect sensitive data from cyber threats.

2. Explain Email Security with Importance of Header Analysis.

What is Email Security?

Email security refers to the measures and practices used to protect email communications from threats like phishing, malware, and unauthorized access. It involves using techniques such as encryption, authentication, and filtering to ensure the confidentiality, integrity, and authenticity of email messages.

Key Point: Email security protects sensitive information, prevents unauthorized access, and mitigates cyber threats.

Importance of Email Header Analysis:

Email headers contain crucial metadata about the message, such as sender information, the route the email took, and authentication details. **Email header analysis** is essential for detecting suspicious or malicious activity.

Key Elements in Email Header Analysis:

1. **Sender Information**: Identifies the real sender by analyzing the "From" and "Return-Path" fields.
2. **IP Address**: Helps trace the origin of the email and detect if it's coming from a suspicious source.
3. **Authentication**: Checks protocols like **SPF**, **DKIM**, and **DMARC** to verify if the email is from a legitimate domain.
4. **Routing Information**: Shows the path the email took through various servers, which can reveal redirection through untrusted sources.

Key Point: Analyzing email headers helps detect phishing attacks, spoofing, and other malicious activities by verifying the sender's legitimacy and message integrity.

3. Explain Google dork with example and prevention against sensitive data leak with google dork.

What is Google Dork?

Google Dork refers to the use of advanced Google search operators to find sensitive or confidential information on the web. These search queries exploit specific Google search features to uncover data that might not be easily accessible through normal searches.

Key Point: Google Dorking can reveal sensitive information like exposed databases, login credentials, or confidential files that are inadvertently indexed by search engines.

Example of Google Dork:

1. Finding exposed files:
   - Search Query: `filetype:xls "sensitive data"`
   - Purpose: Finds Excel files containing "sensitive data" that are publicly accessible.
2. Locating admin login pages:
   - Search Query: `intitle:"admin login"`
   - Purpose: Identifies web pages with "admin login" in the title, which could indicate login portals.

Prevention Against Sensitive Data Leak with Google Dork:

1. Robust Access Controls:
   - Key Point: Implement strong authentication and authorization controls to restrict access to sensitive information.
   - Example: Use complex passwords, multi-factor authentication, and role-based access controls.
2. Proper File and Directory Permissions:
   - Key Point: Ensure that files containing sensitive information are not accessible publicly.
   - Example: Set file permissions correctly and use `.htaccess` or similar mechanisms to restrict directory access.
3. Avoid Indexing Sensitive Data:
   - Key Point: Prevent sensitive files from being indexed by search engines.
   - Example: Use `robots.txt` to disallow indexing of sensitive directories and add `noindex` meta tags to HTML files.

4. Regularly Review and Update Security Settings:
    - Key Point: Conduct periodic reviews to ensure sensitive data is not inadvertently exposed.
    - Example: Regularly audit web applications, file servers, and databases for exposure of sensitive information.
5. Educate and Train Employees:
    - Key Point: Raise awareness about best practices for data security.
    - Example: Train staff to avoid storing sensitive information in publicly accessible locations.

4. Explain NMAP and Its Scanning Techniques.

What is Nmap?

Nmap (Network Mapper) is an open-source tool used for network discovery and security auditing. It helps network administrators and security professionals scan networks to identify hosts, services, and vulnerabilities.

Key Point: Nmap is used to map out networks, discover devices, and identify open ports and running services.

Common Nmap Scanning Techniques:

1. TCP Connect Scan:
    - Command: `nmap -sT <target>`
    - Description: Completes the TCP handshake to determine open ports. It is reliable but more detectable as it establishes full connections.
    - Key Point: Ideal for scenarios where stealth is less of a concern.

2. SYN Scan (Half-Open Scan):
    - Command: `nmap -sS <target>`
    - Description: Sends SYN packets and analyzes responses without completing the handshake. It is faster and less detectable than TCP Connect Scan.
    - Key Point: Commonly used for stealthy scanning as it does not fully establish connections.
3. UDP Scan:
    - Command: `nmap -sU <target>`
    - Description: Scans for open UDP ports by sending UDP packets and checking for responses. UDP scanning is slower and less reliable due to the nature of UDP.
    - Key Point: Useful for discovering services that use UDP, though more challenging due to lack of acknowledgment in UDP.

4. Service Version Detection:
    - Command: `nmap -sV <target>`
    - Description: Detects the versions of services running on open ports by analyzing responses and banner information.
    - Key Point: Helps in identifying the exact software version, which can be crucial for vulnerability assessments.
5. OS Detection:
    - Command: `nmap -O <target>`
    - Description: Attempts to determine the operating system of the target by analyzing TCP/IP stack responses and other characteristics.
    - Key Point: Provides insights into the target system's OS, aiding in further security assessments.

6. Aggressive Scan:
    - Command: `nmap -A <target>`
    - Description: Combines multiple scans (including service detection, OS detection, and traceroute) for a comprehensive assessment.
    - Key Point: Provides extensive information but can be more intrusive and easily detected.
7. Stealth Scan:
    - Command: `nmap -sN <target>`
    - Description: Performs a FIN scan by sending FIN packets to detect open ports. Useful for bypassing some firewall rules.
    - Key Point: Less detectable but may be blocked by modern firewalls.

5. Define terms with Example: -
-Vulnerability
-Threat
-Attack
-Shellcode
-Reverse Shell

1. **Vulnerability**

- **Definition**: A vulnerability is a weakness or flaw in a system, application, or network that can be exploited by threats to gain unauthorized access or cause harm.
- **Example**: An outdated software version with unpatched security flaws.

2. **Threat**

- **Definition**: A threat is a potential danger or risk that could exploit a vulnerability to cause harm or damage to a system, organization, or individual.
- **Example**: A hacker attempting to exploit a software vulnerability to steal data.

3. **Attack**

- **Definition**: An attack is a deliberate attempt to exploit a vulnerability in a system to compromise its security, causing damage, unauthorized access, or disruption.
- **Example**: A SQL injection attack that manipulates a web application's database to retrieve sensitive information.

4. **Shellcode**

- **Definition**: Shellcode is a small piece of code used as the payload in an exploit to gain control over a system or execute commands. It often provides an attacker with a shell or command line interface on the compromised system.
- **Example**: A shellcode snippet that opens a reverse shell to a remote server.

5. **Reverse Shell**

- **Definition**: A reverse shell is a type of shell where the target system initiates a connection back to an attacker's machine, allowing the attacker to execute commands and gain control over the target system remotely.
- **Example**: An attacker sets up a listening server, and the compromised target system connects back to this server, providing the attacker with command-line access.

Overall Key Points:

- **Vulnerability**: Weakness that can be exploited.
- **Threat**: Potential source of harm.
- **Attack**: Action taken to exploit vulnerabilities.
- **Shellcode**: Code used to gain control or execute commands.
- **Reverse Shell**: Connection initiated by the target system to the attacker's machine.

7. Explain HTTP and its methods. 10

1. GET
   - **Description**: Requests data from a specified resource. It is used to retrieve information without making any changes to the resource.
   - **Key Point**: Used for fetching data; it should not modify the resource on the server.

   Example: Requesting a webpage.

   ```http
   GET /index.html HTTP/1.1
   Host: example.com
   ```

2. POST
   - **Description**: Submits data to be processed to a specified resource. Often used to submit form data or upload files.
   - **Key Point**: Used for sending data to the server to create or update resources.

   Example: Submitting a form.

```http
POST /submit-form HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
```

3. PUT

   - Description: Updates a resource or creates it if it does not exist. The request payload replaces the current representation of the resource.
   - Key Point: Used for updating or creating resources with a specific URL.

   Example: Updating user information.

```http
PUT /users/123 HTTP/1.1
Host: example.com
Content-Type: application/json
```

4. DELETE

   - Description: Requests the removal of a specified resource.
   - Key Point: Used to delete a resource identified by the URL.

   Example: Deleting a user.

```http
DELETE /users/123 HTTP/1.1
Host: example.com
```

5. HEAD

   - Description: Similar to GET, but retrieves only the headers of the resource, not the body. Useful for checking metadata.
   - Key Point: Used to obtain metadata without transferring the resource itself.

   Example: Checking resource headers.

```http
HEAD /index.html HTTP/1.1
Host: example.com
```

6. OPTIONS

- **Description**: Describes the communication options for the target resource. It can be used to determine the HTTP methods supported by the server.
- **Key Point**: Used to retrieve supported HTTP methods and other options.

Example: Querying allowed methods.

```http
OPTIONS / HTTP/1.1
Host: example.com
```

7. PATCH

- **Description**: Applies partial modifications to a resource. Unlike PUT, which replaces the entire resource, PATCH only updates the specified parts.
- **Key Point**: Used for making partial updates to a resource.

Example: Partially updating a user profile.

```http
PATCH /users/123 HTTP/1.1
Host: example.com
Content-Type: application/json
```

(a) Write a note on Types of Vulnerability Assessments

1. Network Vulnerability Assessment
    - **Description**: Evaluates the security of network infrastructure, including servers, routers, and switches.
    - **Key Point**: Identifies vulnerabilities such as open ports, misconfigurations, and weak network protocols.
    - **Example**: Scanning for unpatched network services or insecure configurations.
2. Web Application Vulnerability Assessment
    - **Description**: Focuses on web applications to identify security weaknesses like SQL injection, cross-site scripting (XSS), and insecure API endpoints.
    - **Key Point**: Detects vulnerabilities specific to web applications, including flaws in code and configuration.
    - **Example**: Testing input validation and session management vulnerabilities.

3. Database Vulnerability Assessment
    - **Description**: Assesses the security of databases to find weaknesses such as improper access controls, unpatched vulnerabilities, and insecure configurations.
    - **Key Point**: Identifies risks in database management systems, including exposed sensitive data and inadequate permissions.
    - **Example**: Scanning for default database accounts or weak encryption.
4. Host-Based Vulnerability Assessment
    - **Description**: Analyzes individual hosts (servers, workstations) for vulnerabilities such as outdated software, misconfigurations, and weak passwords.
    - **Key Point**: Focuses on securing individual devices within a network.
    - **Example**: Checking for missing patches or insecure configurations on a server.
5. Cloud Security Vulnerability Assessment
    - **Description**: Evaluates vulnerabilities in cloud environments and services, including misconfigured cloud resources and insecure API integrations.
    - **Key Point**: Addresses security issues specific to cloud infrastructures, such as improper

- **Example:** Assessing cloud storage permissions or misconfigured security groups.

6. **Wireless Network Vulnerability Assessment**
   - **Description:** Analyzes wireless networks to identify security issues like weak encryption protocols, unauthorized access points, and misconfigured settings.
   - **Key Point:** Ensures the security of wireless communications and prevents unauthorized access to the network.
   - **Example:** Scanning for rogue access points or weak Wi-Fi encryption.
7. **Penetration Testing**
   - **Description:** Involves simulating real-world attacks to identify vulnerabilities and test the effectiveness of security controls.
   - **Key Point:** Provides a hands-on approach to finding and exploiting vulnerabilities to understand their impact.
   - **Example:** Performing a simulated attack to assess the effectiveness of security measures and response capabilities.

(b)	Write a note on HTTP, why do we call *HTTP* a stateless protocol?

Note on HTTP: Stateless Protocol

HTTP (Hypertext Transfer Protocol) is the foundation for transferring data on the World Wide Web. It is a protocol used to request and deliver web resources, such as web pages, images, and files, between clients (e.g., web browsers) and servers.

Key Characteristics of HTTP:

- **Request-Response Model**: HTTP operates on a request-response model, where a client sends a request to a server, and the server responds with the requested resource or an appropriate status message.
- **Text-Based Protocol**: HTTP is a text-based protocol, which makes it simple and human-readable.

Why HTTP is Called a Stateless Protocol:

HTTP is referred to as a stateless protocol because it does not retain any information about previous interactions or requests. Each request from a client to a server is treated independently, without any knowledge of prior requests.

1. Independence of Requests:
    - **Description**: Each HTTP request is independent and does not rely on previous requests. The server does not maintain any memory of past interactions.
    - **Example**: When a user navigates from one page to another, each page request is processed separately without any context of previous page views.
2. No Persistent Session Information:
    - **Description**: HTTP does not inherently keep track of user sessions or any state information between requests.
    - **Example**: After logging into a website, the server does not remember the user's login status in future requests unless additional mechanisms (e.g., cookies, sessions) are used.
3. Scalability and Simplicity:
    - **Description**: Statelessness simplifies the protocol, making it easier to scale and manage since each request is self-contained and does not require server-side storage of state information.
    - **Example**: Servers can handle multiple requests from different clients simultaneously

4. State Management with Extensions:
    - Description: Although HTTP itself is stateless, additional mechanisms such as cookies, sessions, and tokens are used to manage state and provide a more interactive user experience.
    - Example: Cookies store user preferences and authentication tokens to simulate statefulness in a stateless protocol.

Overall Key Point: HTTP's stateless nature allows for a scalable and efficient protocol for web communication, where each request is independent, and additional mechanisms are used to manage user sessions and state.

(c)	How TCP is a problem in non-persistent HTTP protocol. Justify your answer with an example.	08

TCP and Non-Persistent HTTP

TCP (Transmission Control Protocol) establishes a connection between client and server to ensure reliable data transmission. In the context of non-persistent HTTP, each HTTP request/response pair is handled over a separate TCP connection.

Problem with TCP in Non-Persistent HTTP:

1. Connection Overhead:
    - Description: Each HTTP request requires a new TCP connection to be established and terminated. This results in significant overhead due to the time and resources needed for connection setup and teardown.
    - Example: Loading a web page with multiple resources (e.g., images, scripts) involves multiple TCP connections—one for each resource. Each connection requires a handshake (SYN, SYN-ACK, ACK) and teardown, which adds latency.
2. Increased Latency:
    - Description: The time taken for multiple TCP handshakes and teardowns increases the overall latency of loading a web page.

- **Example**: A webpage with 10 images would need 10 separate TCP connections. Each connection adds latency due to the handshake and teardown processes, leading to slower page load times.

3. **Resource Inefficiency**:
   - **Description**: Establishing and closing multiple TCP connections consumes server and client resources, including memory and CPU, which can impact performance.
   - **Example**: A website with numerous small resources could cause high resource consumption on both the client and server side due to multiple simultaneous TCP connections.

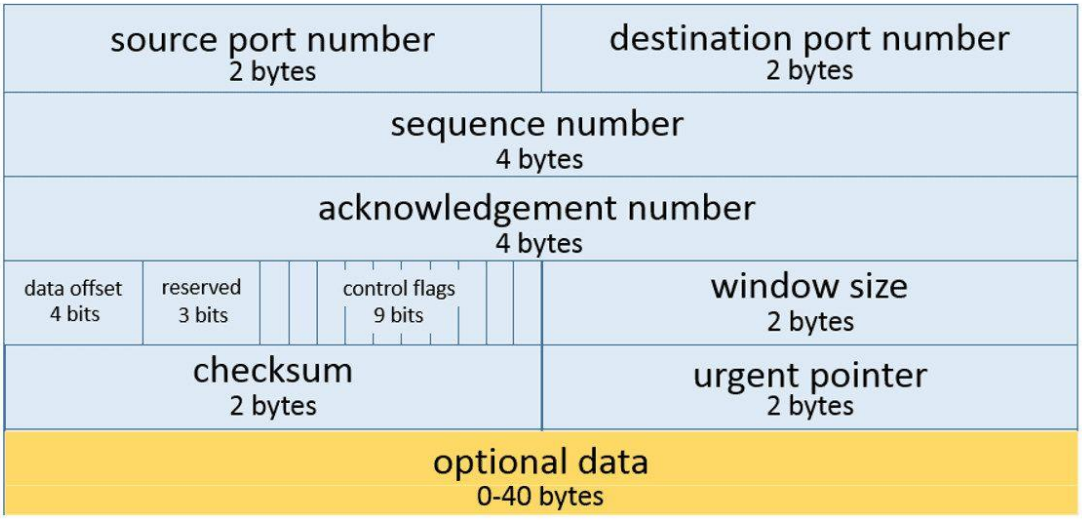
Justification with Example: In non-persistent HTTP, if a webpage requires 10 resources (e.g., HTML, CSS, images), the browser must open 10 separate TCP connections—one for each resource. Each connection incurs latency from connection setup and teardown, leading to slower load times and increased resource use compared to persistent HTTP, which uses a single TCP connection for multiple requests.

Overall Key Point: Non-persistent HTTP's use of separate TCP connections for each req introduces significant overhead, latency, and inefficiency, making it less suitable for performance-sensitive applications.

(d)	Explain TCP header and discuss the role of different flags in the TCP header.

Transmission Control Protocol (TCP) Header

20-60 bytes



1. URG (Urgent):
   - **Description**: Indicates that the Urgent Pointer field is significant and there is urgent data.
   - **Role**: Alerts the receiver to process urgent data immediately.
2. ACK (Acknowledgment):
   - **Description**: Indicates that the Acknowledgment Number field is valid.
   - **Role**: Used to acknowledge receipt of data segments.
3. PSH (Push):
   - **Description**: Instructs the receiver to push the data to the application immediately.
   - **Role**: Ensures data is delivered to the application layer without buffering.
4. RST (Reset):
   - **Description**: Resets the connection if there is an error or if a segment arrives and the connection is not valid.
   - **Role**: Used to abruptly terminate a connection.

5. SYN (Synchronize):
    - Description: Initiates a connection and synchronizes sequence numbers between sender and receiver.
    - Role: Used during the connection establishment phase (three-way handshake).
6. FIN (Finish):
    - Description: Indicates that the sender has finished sending data.
    - Role: Used to terminate a connection gracefully.

(a) Explain Google dorking in detail with five different examples

Google Dorking: Detailed Explanation

Google Dorking refers to the use of advanced Google search operators to find specific information or vulnerabilities that are not typically visible through standard search queries. This technique leverages Google's indexing capabilities to uncover sensitive or hidden data on the web.

Key Features of Google Dorking:

1. Advanced Search Operators: Use specific syntax to refine searches.
2. Hidden or Exposed Information: Finds data that might not be easily accessible otherwise.
3. Security Implications: Can reveal security weaknesses or sensitive information.

Examples of Google Dorking:

1. Finding Exposed Files
    - Query: `filetype:xls "confidential"`
    - Description: Searches for Excel files that contain the word "confidential." This can uncover spreadsheets with potentially sensitive information that is publicly accessible.

- Example: `filetype:xls "financial report"`

2. Locating Admin Login Pages
   - Query: `intitle:"admin login"`
   - Description: Finds web pages with "admin login" in the title, which could be administrative login pages that might be vulnerable or improperly secured.
   - Example: `intitle:"admin" inurl:login`
3. Identifying Open Directories
   - Query: `inurl:/backup/`
   - Description: Looks for URLs that include "/backup/", which might indicate directories containing backup files that could be exposed.
   - Example: `inurl:/admin/backup/`
4. Finding Devices with Web Interfaces
   - Query: `inurl:"8080" "default password"`
   - Description: Searches for web interfaces running on port 8080 with "default password" in the page content. This can help locate devices with default credentials that may be

- Example: `inurl:"8080" "login" "admin"`

5. Discovering Sensitive Information in Logs
   - Query: `filetype:log "password"`
   - Description: Searches for log files that contain the word "password," potentially revealing logs where sensitive information like passwords might be recorded.
   - Example: `filetype:log "error" "username"`

Overall Key Points:

- Google Dorking uses advanced search operators to locate specific types of information or vulnerabilities.
- Sensitive Data Exposure: Can reveal data or configurations that should be secured.
- Security Risks: Useful for both security professionals and malicious actors to uncover hidden or improperly secured information.

(b)	Write a note on cross-site scripting attacks and their types and prevention.

Cross-Site Scripting (XSS) Attacks: Brief Note

Cross-Site Scripting (XSS) is a type of security vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. These scripts can hijack user sessions, deface websites, or steal sensitive information such as cookies, session tokens, or credentials.

Types of XSS Attacks:

1. Stored XSS (Persistent XSS):
   - Description: The malicious script is stored on the server (e.g., in a database) and executed whenever a user accesses the affected page.
   - Example: A user submits a comment containing malicious code, and every time the page loads, the script executes.
2. Reflected XSS:
   - Description: The malicious script is reflected off a web server, typically in response to a user's input (e.g., in a search result), and executed immediately in the user's browser.
   - Example: An attacker crafts a URL with a script, and when a victim clicks on it, the script executes in their browser.

3. DOM-based XSS:
    - **Description:** The malicious script is executed directly in the browser through manipulation of the web page's Document Object Model (DOM), without involving the server.
    - **Example:** A vulnerable client-side script modifies the page content using data from the URL or user input.

Prevention of XSS Attacks:

1. Input Validation and Sanitization:
    - Ensure all user input is validated and sanitized, especially in forms, URLs, and parameters.
2. Escaping Output:
    - Escape special characters (like `<`, `>`, `&`) to prevent them from being interpreted as executable code.
3. Content Security Policy (CSP):
    - Use CSP to restrict the sources from which scripts can be loaded.
4. HTTPOnly and Secure Cookies:
    - Set cookies with the `HttpOnly` and `Secure` flags to prevent unauthorized access through XSS.

Key Point: XSS allows attackers to inject malicious code into web applications, posing a security risk to users and their data. Proper input sanitization, escaping, and security policies are essential to prevent XSS attacks.

(c)	Explain the different steps involved in vulnerability life cycle management.

Vulnerability management is the continuous process of identifying, evaluating, treating, and reporting security vulnerabilities in systems and software. The **vulnerability management life cycle** consists of several key steps:

1. **Discovery**:
   - **Description**: Identify vulnerabilities in systems, applications, or networks through tools like vulnerability scanners (e.g., Nessus, OpenVAS).
   - **Key Point**: Regular scanning helps uncover potential weaknesses.
2. **Prioritization**:
   - **Description**: Assess the risk level of discovered vulnerabilities based on factors like severity, exploitability, and the impact on business.
   - **Key Point**: High-risk vulnerabilities (e.g., those with critical CVSS scores) are addressed first.
3. **Remediation**:
   - **Description**: Take actions to fix the vulnerabilities, such as applying patches, updating software, or changing configurations.

- **Key Point**: Timely remediation reduces the window of exposure.

4. **Verification**:
   - **Description**: Verify that the remediation steps have successfully addressed the vulnerability by re-scanning or testing the system.
   - **Key Point**: Ensures the vulnerability is no longer present.
5. **Reporting**:
   - **Description**: Document the vulnerabilities, remediation efforts, and overall system security posture for stakeholders and auditors.
   - **Key Point**: Helps maintain security compliance and track improvements.
6. **Monitoring**:
   - **Description**: Continuously monitor systems for new vulnerabilities or changes in the environment that may introduce risks.
   - **Key Point**: Continuous monitoring ensures proactive management of security risks.

(d)	Write a short note on a proxy server. How does it help the security team?

Proxy Server: Short Note

A **proxy server** acts as an intermediary between a client and the internet. When a client requests a resource, the proxy server forwards the request to the destination server on behalf of the client and then returns the response back to the client. This process masks the client's identity and can enhance security.

How Proxy Servers Help the Security Team:

1. Anonymity:
    - Proxy servers hide the IP address of internal users, protecting their identity and preventing direct access to sensitive internal systems.
2. Traffic Filtering:
    - Security teams can use proxies to block access to malicious websites, filter harmful content, and enforce security policies.
3. Monitoring and Logging:
    - Proxy servers allow security teams to monitor and log all outbound and inbound traffic, helping detect suspicious activities and potential threats.
4. Access Control:
    - Proxies can restrict access to certain websites or services, ensuring only authorized users or systems can connect to external resources.

5. Data Caching:
    - Proxies can cache frequently requested resources, reducing load on the network and speeding up access while maintaining control over security policies.

How Secure Source Code Review helps to get stable products. Explain in detail.

Secure Source Code Review: Brief Overview

Secure Source Code Review involves systematically examining the source code of applications to identify and rectify security vulnerabilities and weaknesses. This process ensures that the code adheres to security best practices and helps in mitigating potential risks before deployment.

Key Points:

1. **Identification of Vulnerabilities:**
   - **Description:** Detect security flaws such as SQL injection, cross-site scripting (XSS), and buffer overflows within the source code.
   - **Key Point:** Helps in identifying vulnerabilities early in the development cycle.
2. **Adherence to Security Best Practices:**
   - **Description:** Ensure that the code follows established security standards and guidelines (e.g., OWASP Top Ten).
   - **Key Point:** Promotes the use of secure coding practices.
3. **Code Review Techniques:**
   - **Static Analysis:** Automated tools analyze the code without execution to find vulnerabilities.
   - **Manual Review:** Security experts manually review the code for more complex security issues and context-based vulnerabilities.
4. **Integration into Development Lifecycle:**
   - **Description:** Incorporate code reviews as part of the continuous integration/continuous deployment (CI/CD) pipeline.
   - **Key Point:** Ensures ongoing security assessment throughout development.
5. **Remediation of Issues:**
   - **Description:** Fix identified security issues and re-evaluate the code to ensure vulnerabilities are addressed.
   - **Key Point:** Essential for enhancing the security posture of the application.
6. **Documentation and Reporting:**
   - **Description:** Document findings and provide reports to development teams for understanding and addressing issues.
   - **Key Point:** Facilitates communication and tracking of security improvements.

(b)	Explain *STRIDE-based* threat modeling and how it is different from the *DREAD* model.

STRIDE-Based Threat Modeling

STRIDE helps identify threats based on their nature:

- **Spoofing**: Impersonating someone.
- **Tampering**: Modifying data.
- **Repudiation**: Denying actions.
- **Information Disclosure**: Exposing data.
- **Denial of Service (DoS)**: Disrupting service.
- **Elevation of Privilege**: Gaining unauthorized access.

DREAD-Based Threat Modeling

DREAD assesses threat risk using:

- **Damage Potential**: Harm caused if exploited.
- **Reproducibility**: Ease of replication.
- **Exploitability**: Ease of exploitation.
- **Affected Users**: Number of users impacted.

Differences

- **STRIDE**: Focuses on **identifying** threat types.
- **DREAD**: Focuses on **assessing** and **prioritizing** threat risks.

Overall: STRIDE helps categorize threats, while DREAD evaluates their risk level.

(c) What is Information Gathering? List out different types of information-gathering

Information Gathering

Information Gathering is the process of collecting data about a target system or organization to assess its security.

Types:

1. Passive Information Gathering:
   - **Description**: Collects data without interacting directly with the target.
   - **Example**: Searching public websites and social media.
2. Active Information Gathering:
   - **Description**: Directly interacts with the target to obtain information.
   - **Example**: Network scanning using tools like Nmap.
3. Network Scanning:
   - **Description**: Identifies devices and services on a network.
   - **Example**: Scanning for open ports.

4. Social Engineering:
    - Description: Manipulates people to gain sensitive information.
    - Example: Phishing emails to steal credentials.
5. Footprinting:
    - Description: Maps the target's IT infrastructure.
    - Example: WHOIS lookups for domain details.
6. Reconnaissance:
    - Description: Collects initial data about the target.
    - Example: Using public records and databases.

(b)	What is cookie, why it is required and how do vendors take advantage of it?

Cookie: A small piece of data sent from a website and stored on a user's device. Cookies help track and manage user sessions and preferences.

Why Cookies Are Required:

1. Session Management:
   - **Description**: Cookies maintain user sessions by storing session IDs, allowing users to stay logged in and navigate between pages.
   - **Example**: Keeping a user logged into an online account.
2. Personalization:
   - **Description**: Cookies store user preferences and settings to provide a customized browsing experience.
   - **Example**: Remembering language settings or themes.
3. Tracking and Analytics:
   - **Description**: Cookies help collect data on user behavior and interactions for improving website functionality and content.
   - **Example**: Tracking which pages users visit most frequently.

How Vendors Take Advantage of Cookies:

1. Targeted Advertising:
   - **Description**: Cookies track user behavior to serve personalized ads based on interests and browsing history.
   - **Example**: Showing ads for products recently viewed.
2. User Profiling:
   - **Description**: Vendors use cookies to build detailed user profiles for targeted marketing and customer segmentation.
   - **Example**: Creating profiles based on browsing habits to tailor offers and promotions.
3. Behavioral Analytics:
   - **Description**: Analyzing cookie data to understand user engagement and optimize website performance.
   - **Example**: Using data to improve website navigation and content relevance.

(c) Explain privilege Escalation and its type.

Privilege Escalation: Brief Overview

Privilege Escalation is a type of security vulnerability where an attacker gains higher access levels or permissions than initially granted. This can occur within an operating system, application, or network.

Types:

1. **Vertical Privilege Escalation:**
   - **Description:** An attacker elevates their access level from a lower-privileged account to a higher-privileged one.
   - **Example:** A regular user gaining administrative rights.
2. **Horizontal Privilege Escalation:**
   - **Description:** An attacker accesses resources or data belonging to other users with the same level of privilege.
   - **Example:** Accessing another user's data without authorization.

How It Happens:

1. Exploiting Vulnerabilities:
   - Description: Attackers exploit flaws in software or systems to gain higher privileges.
   - Example: Exploiting a software bug that allows regular users to execute privileged commands.
2. Misconfigurations:
   - Description: Incorrectly configured permissions or settings that grant unintended access.
   - Example: A file with incorrect permissions allowing unauthorized users to execute it.
3. Credential Theft:
   - Description: Using stolen credentials to access higher-privileged accounts.
   - Example: Capturing and using an admin's credentials to gain full access.

(d) Explain AAA.

1. Authentication:
    - Description: Verifies the identity of a user or system.
    - Key Point: Ensures that the entity accessing the system is who they claim to be.
    - Example: Logging in with a username and password.
2. Authorization:
    - Description: Determines what resources and actions an authenticated user or system is allowed to access.
    - Key Point: Controls access levels and permissions based on the authenticated identity.
    - Example: Granting or denying access to specific files or features based on user roles.
3. Accounting:
    - Description: Tracks and records user activities and resource usage.
    - Key Point: Provides audit trails and logs for monitoring and reviewing access and usage.
    - Example: Logging user login times, accessed resources, and actions performed.

(d)	Why sub-domain and virtual host enumeration is important in the pen-testing? Explain with example and also discuss various techniques to enumerate sub-domain and virtual host.

1. DNS Zone Transfer:
    - Description: Attempts to request a complete copy of the DNS records for a domain.
    - Example: Using tools like `dig` or `nslookup` to request a zone transfer.
    - Command: `dig axfr @ns.example.com example.com`
2. DNS Brute Forcing:
    - Description: Uses a list of potential subdomains to brute-force and identify valid subdomains.
    - Example: Using tools like `dnsenum` or `sublist3r` to test common subdomain names.
    - Command: `sublist3r -d example.com`
3. Reverse DNS Lookup:
    - Description: Queries the DNS records to find domains that resolve to the same IP address.
    - Example: Using tools like `host` or `dnsrecon` to perform reverse lookups.
    - Command: `host -a 192.168.1.1`

4. Google Dorking:
    - Description: Uses advanced Google search operators to find subdomains and virtual hosts.
    - Example: Searching for `site:example.com -www` to find subdomains.
    - Query: `site:example.com`
5. Virtual Host Enumeration:
    - Description: Identifies virtual hosts on a web server by sending requests with different host headers.
    - Example: Using tools like `Burp Suite` or `Nikto` to test different host headers.
    - Command: `curl -H "Host: subdomain.example.com" http://example.com`

(a)	Discuss CVE with example and how pen-tester will use it during the pen-testing of web application.

- CVE-2021-22986: A vulnerability in the F5 BIG-IP system that allows attackers to execute arbitrary code.
  - Description: An attacker can exploit this vulnerability to execute unauthorized commands on the affected system.
  - Reference: CVE-2021-22986 Details

How Pen Testers Use CVEs in Web Application Testing:

1. Identifying Vulnerabilities:
   - Description: Use CVE databases to identify known vulnerabilities related to the web application's components (e.g., libraries, frameworks).
   - Example: Checking if a web application uses a vulnerable version of a library listed in CVE.
2. Assessing Impact:
   - Description: Determine the impact of known vulnerabilities on the web application's security posture.

security posture.

   - Example: Evaluating how a specific CVE affects the application's authentication or data handling.
3. Guided Testing:
   - Description: Use CVE details to guide testing procedures, focusing on known exploit techniques or attack vectors.
   - Example: Testing for specific exploits or configurations described in a CVE entry.
4. Patch Verification:
   - Description: Verify if the vulnerabilities associated with CVEs have been patched or mitigated in the application.
   - Example: Ensuring that a patch addressing a CVE is correctly applied and functioning.

(b)	False positing is the biggest issue for the pen-tester while doing web application pen-testing, discuss the steps that pen-tester is advice to follow to tackle these issues.

Tackling False Positives in Web Application Penetration Testing: Brief Steps

False positives are incorrect indications of vulnerabilities that do not actually exist. Addressing them effectively is crucial for accurate assessments.

Steps to Tackle False Positives:

1. Validate Findings:
   - Description: Confirm suspected vulnerabilities through multiple testing methods.
   - Example: Re-test using different tools or techniques to ensure the issue is genuine.
2. Manual Verification:
   - Description: Perform manual testing to cross-check automated scan results.
   - Example: Manually verify reported issues by inspecting the code or reproducing the issue.
3. Consult Documentation:
   - Description: Refer to application documentation and specifications to understand the context of findings.

- **Example**: Check if the detected issue is a known and expected behavior.

4. **Use Reliable Tools**:
   - **Description**: Utilize well-established and updated security tools to minimize incorrect results.
   - **Example**: Employ tools with good reputations and regularly updated vulnerability databases.
5. **Communicate with Developers**:
   - **Description**: Discuss findings with the development team to get insights and verify if the reported issues are expected or intended.
   - **Example**: Collaborate with developers to understand whether an issue is a false positive or a genuine concern.
6. **Prioritize Issues**:
   - **Description**: Focus on high-confidence findings and address them before moving to lower-confidence results.

(c)	What is threat, threat modeling and threading modeling assessment? Discuss the thread model which focuses on identifying and rating potential attack vectors with example.
(d)	Discuss the PASTA

Threat, Threat Modeling, and Threat Modeling Assessment

Threat: A potential danger that could exploit a vulnerability to cause harm to a system or organization.

Threat Modeling: The process of identifying, evaluating, and prioritizing threats to a system or application, and understanding their potential impact.

Threat Modeling Assessment: An evaluation of a threat model to identify and rate potential attack vectors, ensuring that all significant threats are addressed.

Threat Model Example: STRIDE

STRIDE is a threat modeling framework focusing on identifying and rating potential attack vectors by categorizing threats:

- **Spoofing**: Impersonating another user.
- **Tampering**: Modifying data or code.
- **Repudiation**: Denying actions.
- **Information Disclosure**: Exposing sensitive data.

- **Denial of Service (DoS)**: Disrupting service availability.
- **Elevation of Privilege**: Gaining unauthorized access.

Example: In a web application, STRIDE could identify:

- **Spoofing**: An attacker impersonating an admin to gain access.
- **Tampering**: Modifying input data to exploit vulnerabilities.
- **Information Disclosure**: Accessing sensitive user data through weak authentication.

Overall Key Point: Threat modeling, such as using STRIDE, helps in identifying and rating attack vectors to address potential security threats effectively.

	Answer the following question (Attempt any two)
(a)	Discuss very preciously, as pen-tester when to use intruder, repeater and decoder with situations.

1. Intruder:

- **Purpose:** Automates attacks by sending a series of requests with varying inputs to find vulnerabilities.
- **When to Use:** Use for automated brute-force attacks, fuzzing, or testing for injection vulnerabilities.
- **Example:** Testing different passwords in a login form to identify weak authentication.

2. Repeater:

- **Purpose:** Manually modifies and re-sends individual HTTP requests to test for specific vulnerabilities.
- **When to Use:** Use for manual testing, analyzing responses, and fine-tuning payloads.
- **Example:** Modifying a request to test for SQL injection vulnerabilities by changing query parameters.

3. Decoder:

- **Purpose:** Decodes and encodes data to analyze or manipulate different encoding formats (e.g., Base64, URL encoding).
- **When to Use:** Use to decode encoded data in requests or responses to understand or exploit vulnerabilities.
- **Example:** Decoding a Base64 encoded payload to analyze its content for potential security issues.

Here's a quick rundown of the terms with examples:

1. **Target**: The system or application you are testing for vulnerabilities.
    - Example: A web server like "www.example.com" during a pentest.
2. **Proxy**: Intercepts and modifies traffic between a client and server.
    - Example: **Burp Suite's** Proxy tool intercepts HTTP requests to view or alter them.
3. **Spider**: Automatically crawls a website to map its structure and discover links.
    - Example: **OWASP ZAP Spider** crawls all the links on "www.example.com" to find hidden pages.
4. **Sniper**: A single-target attack that tests one input field with various payloads.
    - Example: Using **Burp Intruder** in "Sniper" mode to test a login form with different SQL injection payloads.

	(c)	Explain the HTTP protocol with example. Write three points related to HTTP protocol which pen-tester advice to consider for efficient web application security pen-testing.	08