STAGE - 1

List of Vulnerabilities

Top 5 Vulnerability Exploitation

S.no	Vulnerability name	CWE-No
1.	SQL Injection	CWE-89
2.	Cross-site Scripting (XSS)	CWE-79
3.	Insecure Direct Object Reference (IDOR)	CWE-639
4.	Security Misconfiguration	CWE-16
5.	XML External Entity	CWE-611

3.2 Report:

Vulnerability Name: SQL Injection

CWE No: CWE-89

OWASP/SANS Category: SANS Top 25

Description:

SQL Injection is a web application security vulnerability that allows attackers to inject malicious SQL code into a database, leading to unauthorized access, modification, or deletion of sensitive data. It occurs when user input is not properly validated or sanitized, enabling attackers to manipulate SQL queries and execute malicious actions. This can result in data breaches, tampering, elevation of privileges, and denial of service attacks. To prevent SQL Injection, it's essential to validate user input, use parameterized queries, implement stored procedures, grant minimal privileges, and conduct regular security audits.

Business Impact:

- Data Breach: Sensitive customer and business data can be stolen.
- Data Loss or Manipulation: Attackers can delete or alter records, leading to data integrity issues.
- Reputation Damage: Breaches erode customer trust and can lead to loss of business.
- Financial Loss: Organizations may face regulatory fines (e.g., GDPR, HIPAA violations) and lawsuits.
- System Compromise: Attackers can gain administrative database access, allowing them to control backend systems.
- Denial of Service (DoS): Large queries can overload and crash the database server.

Steps to identify:

1. Identify Input Points: Look for forms, search boxes, login fields, or any place where you can enter data into the web application.

2. Basic Injection Attempts: Try injecting simple SQL commands like ' OR 1=1-- or ' OR '1'='1 into the input fields. Observe if the application behaves differently, such as displaying unexpected results or errors.

3. Advanced Injection Techniques: If basic attempts work, try more complex payloads to extract data or modify the database.

4. Analyze Results: Pay attention to the application's responses. Errors, unexpected results, or changes in behavior can indicate a vulnerability.

Vulnerability Name: Cross-site Scripting (XSS)

CWE No: CWE-79

OWASP/SANS Category: SANS Top 5

Description:

Cross-site Scripting (XSS) is a web application security vulnerability that allows attackers to inject malicious JavaScript code into a website, leading to unauthorized access, modification, or theft of sensitive data. It occurs when user input is not properly validated or sanitized, enabling attackers to manipulate website content and execute malicious actions. This can result in data theft, session hijacking, malware distribution, and defacement. To prevent XSS, it's essential to validate user input, encode output, implement Content Security Policy (CSP), and conduct regular security audits.

Business Impact:

- Data Theft & Session Hijacking: Attackers can inject scripts to steal cookies, session tokens, or other sensitive user data, leading to unauthorized access to user accounts.
- Reputation Damage: Exploits can deface websites or redirect users to malicious sites, eroding trust and damaging the brand.
- Financial Loss: Compromised customer data and loss of trust can lead to decreased revenue, legal liabilities, and potential regulatory fines.

Steps to Identify:

1. Identify Input Points: Look for areas where users can input data, such as comment sections, search boxes, or user profile fields.

2. Inject Script Tags: Try injecting simple script tags into the input fields. For example, you can input <script>alert('XSS')</script>. If the application reflects this input back to the page without sanitization, it may be vulnerable.

3. Analyze Responses: Check if the injected script executes when the input is displayed. If you see an alert box or any other unexpected behavior, the application is likely vulnerable to XSS.

4. Test Different Contexts: XSS can occur in different contexts (HTML, JavaScript, URL, etc.), so try variations of your payload in different input fields to see how the application handles them.

5. Check for Stored XSS: If the input is stored (like in a database) and later displayed to users, test if the script executes when another user views the affected page.

6. Review Security Headers: Check if the application uses security headers like Content Security Policy (CSP) to mitigate XSS risks.

Vulnerability Name: Insecure Direct Object Reference

CWE No: CWE-639

OWASP/SANS Category: SANS Top 25

Description:

Insecure Direct Object Reference (IDOR) is a web application security vulnerability that allows attackers to access or manipulate sensitive data by manipulating object references. This occurs when a web application exposes internal object references, such as database keys or file names, and an attacker can modify these references to access unauthorized data. IDOR can lead to unauthorized data access, modification, or deletion, and can be exploited through techniques such as tampering with URL parameters or form data. To prevent IDOR, it's essential to use indirect object references, validate user input, and implement access controls to restrict access to sensitive data.

Business Impact:

- Unauthorized Access: Attackers may gain direct access to sensitive data, such as personal records or proprietary information, by manipulating object references.
- Data Breach: Exposure of confidential information can result in significant financial and reputational damage, including customer loss and legal consequences.
- Operational Disruption: Unauthorized data access might lead to alterations or deletions that affect business operations and data integrity.

Steps to Identify:

1. Identify Object Identifiers: Look for URLs, forms, or other inputs that contain unique identifiers like user IDs, product IDs, file names, or other object references.

2. Manipulate Identifiers: Try changing the identifier values in the URL or forms. For example, if you see a URL like /profile/123, try changing "123" to other values, including invalid ones.

3. Observe Responses: Analyze the application's responses to your manipulated requests. Look for:

    - Unexpected Data: If you access data that you shouldn't be able to, like another user's profile information, it's a sign of IDOR.

- Error Messages: Error messages revealing the structure of the application or database can be helpful.
- Unauthorized Actions: If you can perform actions you shouldn't be able to, like deleting other users' files, it's an IDOR vulnerability.

4. Test Different Objects: Try manipulating identifiers for different types of objects (e.g., users, products, files) to see if you can access or modify them.

5. Check for Authentication and Authorization: Make sure the application properly checks user authentication and authorization before allowing access to objects.

Vulnerability Name: Security Misconfiguration

CWE No: CWE-16

OWASP/SANS Category: SANS Top 25

Description:

Security Misconfiguration is a web application security vulnerability that occurs when a web application's configuration is not properly set up, leaving it exposed to attacks. This can include misconfigured file permissions, insecure default settings, or missing security patches. Security Misconfiguration can lead to unauthorized access, data breaches, and other security incidents. Common examples include misconfigured firewalls, insecure protocol versions, and exposed sensitive data. To prevent Security Misconfiguration, it's essential to implement secure configuration guidelines, regularly review and update configurations, and conduct security audits to identify vulnerabilities.

Business Impact:

- Increased Attack Surface: Incorrectly configured systems, unnecessary services, or default credentials can make it easier for attackers to exploit the system.
- System Compromise: Attackers might gain unauthorized access or escalate privileges, leading to data breaches, system downtime, or complete system takeover.
- Compliance Violations: Poor configuration can result in non-compliance with security standards and regulations, which may attract fines or legal action.

Steps to Identify:

1. Inventory and Discovery:

-Identify all assets: Make a comprehensive list of all software, hardware, and network devices within your environment. This includes servers, databases, applications, firewalls, routers, switches, etc.

-Gather configuration details: Collect information about the configuration of each asset. This includes operating system versions, installed software, network settings, firewall rules, user accounts, permissions, and any other relevant settings.

2. Benchmarking and Comparison:

- Use security benchmarks: Compare your asset configurations against industry-standard security benchmarks (e.g., CIS Benchmarks, NIST Cybersecurity Framework). These benchmarks provide best practices and recommended configurations for different technologies.
- Compare configurations: Compare configurations across similar assets. Look for inconsistencies or deviations that could indicate potential vulnerabilities.

3. Scanning and Testing:

- Use vulnerability scanners: Run vulnerability scanners to identify known vulnerabilities and misconfigurations in your systems.

- Perform penetration testing: Engage security professionals to conduct penetration tests to simulate real-world attacks and identify vulnerabilities that might be missed by scanners.

4. Reviewing Logs and Monitoring:

- Analyze logs: Regularly review security logs for suspicious activity, failed login attempts, unauthorized access, or unusual patterns.
- Implement monitoring tools: Use security information and event management (SIEM) tools to monitor network traffic, system activity, and security events for potential misconfigurations or attacks.

5. Regular Updates and Patches:

- Stay up-to-date: Ensure all software and operating systems are updated with the latest security patches and fixes.
- Automate updates: Implement automated patching processes to reduce the risk of unpatched vulnerabilities.

Vulnerability Name: XML External Entity

CWE No: CWE-611

OWASP/SANS Category: SANS Top 25

Description:

XML External Entity (XXE) is a web application security vulnerability that allows attackers to inject malicious XML code into a web application, potentially leading to unauthorized data access, denial of service, or remote code execution. This occurs when a web application parses XML input containing external entity references, which can be exploited to access sensitive data or execute system-level commands. To prevent XXE, it's essential to disable external entity parsing, validate XML input, and implement secure XML processing guidelines.

Business Impact:

- Sensitive Data Exposure: Improper handling of XML input may allow attackers to access internal files and sensitive configuration data.
- Denial of Service (DoS): Exploiting XXE vulnerabilities can lead to resource exhaustion or application crashes, disrupting business operations.
- Compliance and Reputational Risk: Breaches that expose sensitive internal data can result in regulatory fines and damage to the organization's reputation.

Steps to Identify:

1. Code Review:

- Look for XML parsing: Search your code for libraries or functions that parse XML data.
- Check for external entity processing: Identify if your code allows or disables external entity processing (e.g., DOCTYPE declarations).
- Review input validation: Ensure that all XML input is properly validated and sanitized to prevent injection of malicious entities.

2. Security Scanning Tools:

- Use specialized scanners: Utilize security scanners designed to detect XXE vulnerabilities. These scanners can analyze your code and identify potential risks.
- Web application firewalls (WAFs): Configure your WAF to block or filter XML requests that contain external entities.

3. Manual Testing:

- Craft malicious XML payloads: Construct XML documents with malicious external entities and attempt to process them through your application.
- Observe system behavior: Monitor your system for any unexpected behavior, data leaks, or code execution attempts.

4. Configuration Checks:

- Review XML parser settings: Ensure that your XML parser is configured to disable external entity processing by default.
- Check for secure defaults: Use secure defaults for XML parsers and avoid custom configurations that might introduce vulnerabilities.

5. Fix the Vulnerability:

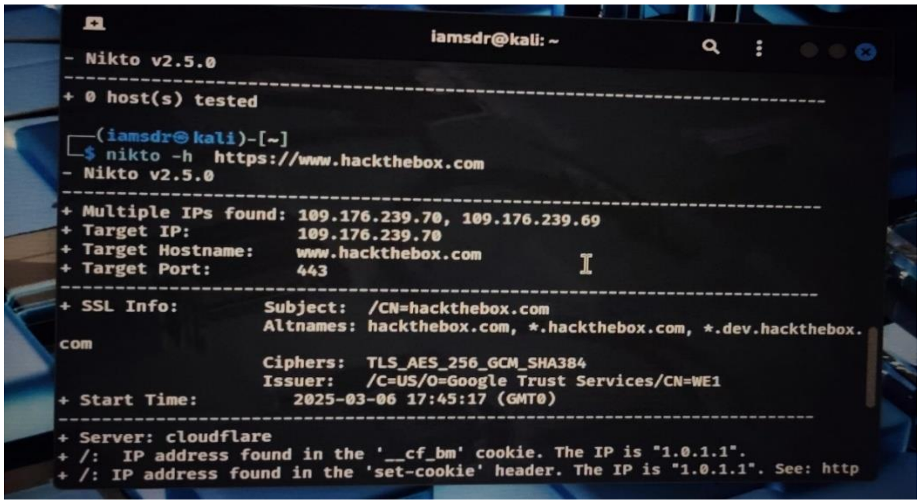
- Disable external entity processing: Configure your XML parser to disallow external entity references.
- Sanitize input: Validate and sanitize all XML input to prevent injection of malicious entities.
- Use secure libraries: Consider using secure XML parsing libraries that have built-in protections against XXE vulnerabilities.

STAGE – 2

NESSUS

Nessus is a powerful vulnerability assessment tool developed by Tenable, widely used by security professionals to detect vulnerabilities, misconfigurations, and compliance issues in IT systems. It helps organizations proactively identify security risks and remediate them before they can be exploited by attackers.

One of the key strengths of Nessus is its comprehensive vulnerability scanning capabilities, which allow organizations to proactively detect security flaws before they can be exploited by attackers. The tool uses an extensive database of over 180,000 plugins, regularly updated to identify new vulnerabilities, misconfigurations, and outdated software. Nessus scans devices for open ports, unpatched software, weak passwords, and dangerous configurations that could lead to security breaches. It also detects malware, backdoors, botnet activity, and ransomware-related vulnerabilities, ensuring that security teams can take immediate action to mitigate risks. In addition to standard vulnerability scanning, Nessus provides compliance auditing to help organizations adhere to regulatory standards such as PCI-DSS, HIPAA, ISO 27001, NIST, and CIS benchmarks. This makes it an essential tool for companies that must meet strict security requirements.



```
- Nikto v2.5.0
---------------------------------------------------------------------------
+ 0 host(s) tested

┌──(iamsdr㉿kali)-[~]
└─$ nikto -h  https://www.hackthebox.com
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Multiple IPs found: 109.176.239.70, 109.176.239.69
+ Target IP:          109.176.239.70
+ Target Hostname:    www.hackthebox.com
+ Target Port:        443
---------------------------------------------------------------------------
+ SSL Info:        Subject:  /CN=hackthebox.com
                   Altnames: hackthebox.com, *.hackthebox.com, *.dev.hackthebox.
com
                   Ciphers:  TLS_AES_256_GCM_SHA384
                   Issuer:   /C=US/O=Google Trust Services/CN=WE1
+ Start Time:         2025-03-06 17:45:17 (GMT0)
---------------------------------------------------------------------------
+ Server: cloudflare
+ /:  IP address found in the '__cf_bm' cookie. The IP is "1.0.1.1".
+ /: IP address found in the 'set-cookie' header. The IP is "1.0.1.1". See: http
```

Target website :- https://www.hackthebox.com

Target IP Address: 109.176.239.70

Target port: 443

Report:

Explanation of Vulnerabilities:

1.Vulnerability Name: Cross-Site Scripting (XSS)

CWE: CWE-79

OWASP/SANS Category: A07:2021 – Identification and Authentication Failures (OWASP Top 10)

Severity: - Medium

Plugin:- XSS Detector

Port:- 80 for HTTP

S.no	Vulnerability name	CWE No	Severity	Status	Plugin
1.	Cross-Site Scripting (XSS)	CWE-79	Medium	Confirmed	XSS Detector
2.	Insecure Direct Object Reference (IDOR)	CWE-639	Critical	Confirmed	Burp Suite, OWASP ZAP, Acunetix
3.	Broken Authentication	CWE-287	High	Confirmed	Authentication Tester

Description:

Cross-Site Scripting (XSS) is a security vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. This occurs when an application does not properly validate or escape user input before displaying it on a webpage. XSS attacks can be used to steal cookies, hijack user sessions, or redirect users to malicious websites.

XSS allows an attacker to inject malicious JavaScript into a web application. When a victim loads the page, the script executes in their browser, potentially leading to data theft, phishing, or unauthorized actions on behalf of the user. The vulnerability arises due to improper input validation and output encoding.

Solution:

- Implement input validation to filter out special characters like <script>.
- Use output encoding to prevent script execution in the browser.
- Implement Content Security Policy (CSP) to block unauthorized scripts.
- Sanitize user input using frameworks like OWASP ESAPI or built-in functions in modern web frameworks.
- Regularly test for XSS using automated scanners and manual penetration testing.

Business Impact :-

- ✓ User session hijacking can lead to unauthorized account access.
- ✓ Data theft and phishing attacks may compromise sensitive information.
- ✓ Website defacement and malware injection can damage reputation.
- ✓ Regulatory non-compliance (e.g., GDPR, PCI-DSS) leading to legal and financial penalties.

2. Vulnerability Name: Insecure Direct Object Reference (IDOR)

CWE No: CWE-639

OWASP Category: A01:2021 - Broken Access Control

SANS Category: Authorization Issues

Severity: Critical

Plugin: Burp Suite, OWASP ZAP, Acunetix, Nessus

Port: 80 for HTTP

Description:

IDOR occurs when a web application provides direct access to internal objects (such as database records, files, or user accounts) without proper authorization checks. This allows an attacker to manipulate object references (such as changing a user ID or file name) to access data they should not be able to view or modify.

Solution:

- Implement Proper Access Controls: Use Role-Based Access Control (RBAC) and Least Privilege Principle (PoLP) to restrict unauthorized access to resources.
- Use Indirect Object References (IORs): Avoid exposing direct database IDs in URLs; instead, use hashed values, UUIDs, or session-based tokens.
- Enforce Server-Side Authorization Checks: Always verify user permissions on the server-side before granting access to any resource.
- Secure API & Parameter Validation: Implement strong API security using OAuth2, JWT tokens, and strict validation of input parameters.

- Monitor & Audit Access Logs: Continuously track user activities using SIEM tools to detect and prevent unauthorized access attempts.

Business Impact:

- ✓ Regulatory Non-Compliance – Violates GDPR, HIPAA, PCI DSS.
- ✓ Financial Loss – Attackers could exploit pricing APIs or access sensitive financial data.
- ✓ Reputation Damage – Loss of user trust if personal data is exposed.

3. Vulnerability Name: Broken Authentication

CWE: CWE-287 (Improper Authentication)

OWASP/SANS Category: A07:2021 – Identification and Authentication Failures (OWASP Top 10)

Severity: - High

Plugin:- Authentication

Port :- 80 for HTTP

Description:

Broken Authentication occurs when an application improperly manages session IDs, passwords, or authentication mechanisms, allowing attackers to compromise user accounts. This vulnerability can arise from weak password policies, exposed session tokens, lack of multi-factor authentication (MFA), or session fixation attacks. An attacker could exploit these weaknesses to gain unauthorized access, escalate privileges, or impersonate legitimate users.

It occurs when an application fails to protect user credentials and session management. Attackers may exploit weak passwords, lack of session expiration, or credential reuse to gain unauthorized access. Common attack vectors include brute force attacks, session hijacking, and credential stuffing.

Solution:

- Enforce strong password policies (e.g., minimum length, complexity, and expiration rules).
- Implement Multi-Factor Authentication (MFA) for an extra layer of security.
- Use secure session management (e.g., regenerate session IDs upon login, enforce session timeouts).
- Store passwords securely using bcrypt, Argon2, or PBKDF2 instead of plaintext or weak hashing algorithms (e.g., MD5, SHA-1).
- Implement rate limiting and account lockout to prevent brute-force attacks.
- Ensure secure transmission of credentials using TLS (HTTPS) to prevent interception.

Business Impact:-

✓ Compromised user accounts leading to unauthorized system access.
✓ Financial and reputational damage from data breaches.
✓ Regulatory non-compliance penalties (GDPR, PCI-DSS, etc.).
✓ Potential for privilege escalation if admin accounts are compromised.