VULNERABILITY ASSESSMENT AND PENETRATION TESTING(23CY701)

UNIT - 4 NOTES

Web Application Security Vulnerabilities

A **web application vulnerability** is a weakness or flaw in a website or web application that can be exploited by attackers to gain unauthorized access, steal data, modify information, or disrupt services.

Think of a web application as a house:

- **Users** are visitors.
- **Login page** is the front door.
- **Database** is the locker room containing valuable items.
- **Security controls** are locks and security guards.

If the locks are weak or a door is left open, an attacker can enter and cause damage. Such weaknesses are called **vulnerabilities**.

Overview of top web application security vulnerabilities

The most common and critical vulnerabilities are identified by the OWASP Top 10.

What is OWASP?

OWASP (Open Worldwide Application Security Project) is a **non-profit organization dedicated to improving software and web application security**.

1. Broken Access Control

Definition: Users can access resources or perform actions beyond their permitted privileges.

Example: A normal user accesses an administrator page by changing the URL.

Impact: Unauthorized access to sensitive data and functions.

---

2. Cryptographic Failures

Definition: Sensitive data is not properly protected through encryption.

Example: Passwords stored in plain text instead of being encrypted or hashed.

Impact: Exposure of passwords, personal information, and financial data.

---

3. Injection Vulnerabilities

Definition: Malicious input is interpreted as commands by a database or system.

Examples:

- SQL Injection
- Command Injection
- LDAP Injection

Impact: Data theft, unauthorized access, and system compromise.

---

4. Insecure Design

Definition: Security is not adequately considered during application design.

Example: Allowing unlimited login attempts without account lockout.

Impact: Increased risk of attacks and security breaches.

---

5. Security Misconfiguration

Definition: Improper security settings in servers, databases, or applications.

Examples:

- Default passwords
- Unnecessary services enabled
- Debug mode left active

Impact: Easy exploitation by attackers.

6. Vulnerable and Outdated Components

Definition: Using outdated software, libraries, or frameworks with known security flaws.

Example: Running an old version of a web framework with publicly known vulnerabilities.

Impact: Attackers exploit known weaknesses.

---

7. Identification and Authentication Failures

Definition: Weak authentication mechanisms allow unauthorized access.

Examples:

- Weak passwords
- Poor session management
- Lack of Multi-Factor Authentication (MFA)

Impact: Account compromise and unauthorized access.

---

8. Software and Data Integrity Failures

Definition: Applications trust software updates, plugins, or data without verification.

Example: Installing updates from untrusted sources.

Impact: Malware installation and system compromise.

---

9. Security Logging and Monitoring Failures

Definition: Security events are not properly logged or monitored.

Example: Failed login attempts are not recorded.

Impact: Attacks remain undetected for long periods.

---

10. Server-Side Request Forgery (SSRF)

Definition: Attackers trick a server into making unauthorized requests.

Example: A server is forced to access internal resources that should not be publicly accessible.

Impact: Internal network access and sensitive data exposure.

Important Vulnerabilities in Detail

SQL Injection (SQLi)

A type of Injection attack where malicious SQL commands are inserted into user input fields.

Example:
A hacker enters special SQL code in a login form to bypass authentication.

Impact:

- Database access
- Data theft
- Data modification

Prevention:

- Prepared statements
- Parameterized queries
- Input validation

Cross-Site Scripting (XSS)

A vulnerability where attackers inject malicious JavaScript into web pages.

Types of XSS

1. **Stored XSS** – Script is saved in the database.
2. **Reflected XSS** – Script is reflected through a URL or form.
3. **DOM-Based XSS** – Script executes through client-side JavaScript.

Impact:

- Cookie theft
- Session hijacking
- Account takeover

Prevention:

- Input validation
- Output encoding
- Content Security Policy (CSP)

Conclusion

Web application vulnerabilities are security weaknesses that can lead to data breaches, unauthorized access, and service disruption. Following OWASP guidelines, validating user input, implementing strong authentication, encrypting sensitive data, and regularly updating software are essential for building secure web applications.

NARSIMHA REDDY ENGINEERING COLLEGE

NRCM your roots to success. Estd.2007

An Autonomous Institution| Affiliated to JNTUH | Approved by AICTE
Accredited by NBA & NAAC with 'A' Grade

VULNERABILITY ASSESSMENT AND PENETRATION TESTING(23CY701)

OVERVIEW OF TOP WEB APPLICATION SECURITY VULNERABILITIES

Weaknesses in web applications that attackers can exploit to steal data, gain unauthorized access or disrupt services.

WHAT IS IT?

Web application vulnerabilities are security flaws in websites or applications that can be exploited by attackers.

WHY IT MATTERS

- Data Breach
- Unauthorized Access
- Financial Loss
- Reputation Damage

THE CIA TRIAD

OWASP TOP 10

1 BROKEN ACCESS CONTROL

Users can access resources or perform actions beyond their permissions.

Example: A user accesses an admin page by changing the URL.

2 CRYPTOGRAPHIC FAILURES

Sensitive data is not properly protected using encryption.

Example: Passwords stored in plain text.

3 INJECTION

Malicious input is sent to an interpreter as a command.

Example: SQL Injection in a login form.

4 INSECURE DESIGN

Security is not considered during application design.

Example: No limit on login attempts.

5 SECURITY MISCONFIGURATION

Improper configuration of servers, databases or applications.

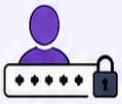
Example: Default passwords, debug mode enabled.

6 VULNERABLE & OUTDATED COMPONENTS

Using old software, libraries or frameworks with known flaws.

Example: Outdated plugins or libraries.

7 IDENTIFICATION & AUTHENTICATION FAILURES



Weak login and session management allows attackers to break in.

Example: Weak passwords, no multi-factor authentication.

8 SOFTWARE & DATA INTEGRITY FAILURES

Untrusted software updates or data can lead to compromise.

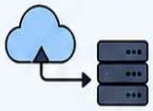
Example: Installing unverified plugins.

9 SECURITY LOGGING & MONITORING FAILURES

Insufficient logging and monitoring of security events.

Example: Failed login attempts not logged.

10 SERVER-SIDE REQUEST FORGERY (SSRF)



Attacker tricks the server into making unauthorized requests.

Example: Accessing internal systems through the server.

SQL INJECTION (SQLi)

HOW IT WORKS

Normal Query

```
SELECT * FROM users
WHERE username = 'john'
AND password = '1234';
```

Attacker Input

```
' OR '1'='1
```

Result

Bypasses login and gives access to the database.

Impact

Data theft • Login bypass • Data modification • Complete database access

CROSS-SITE SCRIPTING (XSS)

HOW IT WORKS

Attacker Input

```
<script>alert('Hacked');</script>
```

Stored / Reflected

Malicious script is stored or reflected on the page.

Victim

Other users open the page and the script runs in their browser.

Impact

Cookie theft • Session hijacking
Account takeover • Malware delivery

BEST PRACTICES

- Validate All Input
- Use Strong Authentication
- Secure Coding Practices
- Keep Software Updated
- Monitor & Log Security Events
- Follow OWASP Guidelines

KEY TAKEAWAY

Secure your applications today to protect data, users and your business tomorrow.

Injection Vulnerabilities

What is an Injection Vulnerability?

An **Injection Vulnerability** is a security weakness where a hacker enters malicious input into a website or application, causing it to execute unintended commands.

How Injection Works

Step 1: Application asks for input

Example:

Username: _____

Password: _____

Step 2: User enters information

Normal user:

Username: Rahul

Password: 12345

Step 3: Attacker enters malicious input

Username: admin

Password: ' OR '1'='1

Common Types of Injection

1. SQL Injection (SQLi)

Target: Database

What happens?
The attacker inserts SQL commands into input fields.

Example:
A login page is tricked into allowing access without a valid password.

Impact:

- Data theft
- Login bypass
- Database modification

---

2. Command Injection

Target: Operating System

What happens?
The attacker tricks the application into running system commands.

Impact:

- Server compromise
- File access
- System control

---

3. LDAP Injection

Target: Directory services used for authentication.

Impact:

- Unauthorized login
- Access to user information

---

4. XPath Injection

Target: XML databases.

Impact:

- Data leakage

- Unauthorized access

Why Injection Attacks Are Dangerous

Injection attacks can allow hackers to:

✓ Steal sensitive data

✓ Access accounts without passwords

✓ Modify or delete records

✓ Take control of servers

✓ Gain administrator privileges

How to Prevent Injection Attacks

1. Validate User Input

Accept only expected values.

2. Use Prepared Statements

Keep user data separate from commands.

3. Sanitize Input

Remove dangerous characters and unexpected input.

4. Follow Least Privilege Principle

Give applications and databases only the permissions they need.

5. Use Security Testing Tools

Regularly test applications for vulnerabilities.

EXAMPLE : Think of a web application as a restaurant.

- A customer should only place an order.
- A hacker tries to enter the kitchen and give instructions to the chef.

A secure restaurant accepts only food orders.

A vulnerable restaurant follows every instruction, including malicious ones.

That's exactly how an **Injection Vulnerability** works.

Injection Vulnerability is a security flaw in which an attacker inserts malicious input into an application, causing it to execute unintended commands or queries, leading to unauthorized access, data theft, or system compromise.

Cross-Site Scripting (XSS) Vulnerabilities

Cross-Site Scripting (XSS) is a web application security vulnerability that allows an attacker to inject malicious scripts, usually JavaScript code, into web pages viewed by other users.The injected script executes in the victim's browser and can perform actions on behalf of the user without their knowledge.

Web applications frequently accept user input through:

- Login forms
- Search boxes
- Comment sections
- Feedback forms
- Discussion forums

If the application fails to properly validate or sanitize this input, an attacker can insert malicious scripts into the webpage.

When another user visits the affected page, the browser interprets and executes the malicious script as if it were legitimate content from the trusted website.

The browser cannot distinguish between genuine website code and malicious code injected by an attacker.

How XSS Occurs

The basic concept behind XSS involves three stages:

1. Injection

The attacker inserts a malicious script into a vulnerable web application.

2. Delivery

The malicious script becomes part of a webpage delivered to users.

3. Execution

The victim's browser executes the script, believing it originated from the trusted website.

Why XSS is Dangerous

Web browsers trust content received from legitimate websites.

When malicious scripts are executed within this trusted environment, attackers can:

- Access user session information
- Steal cookies
- Capture user input
- Redirect users to malicious websites
- Modify webpage content
- Perform actions on behalf of users

Types of Cross-Site Scripting

1. Stored XSS (Persistent XSS)

Theory

In Stored XSS, the malicious script is permanently stored on the web server, usually in a database.

When users access the affected page, the stored script is retrieved and executed automatically.

Characteristics

- Script remains stored until removed.
- Affects multiple users.
- Considered highly dangerous because one attack can impact many visitors.

---

2. Reflected XSS (Non-Persistent XSS)

Theory

In Reflected XSS, the malicious script is embedded within a request and immediately reflected back to the user by the web application.

The script is not stored on the server.

Execution occurs only when the victim clicks a specially crafted link or submits manipulated input.

Characteristics

- Temporary attack.
- Requires user interaction.
- Commonly used in phishing attacks.

---

3. DOM-Based XSS

Theory

DOM-Based XSS occurs when client-side JavaScript modifies the Document Object Model (DOM) using untrusted input.

The vulnerability exists entirely within the browser and may not involve server-side processing.

Characteristics

- Occurs on the client side.
- Exploits insecure JavaScript code.
- Often difficult to detect using traditional security testing methods.

Impact of XSS Attacks

Cross-Site Scripting can result in:

Session Hijacking

Attackers steal session identifiers and impersonate users.

Cookie Theft

Sensitive browser cookies are captured and misused.

Account Takeover

Attackers gain unauthorized access to user accounts.

Website Defacement

Webpage content is altered or manipulated.

Phishing Attacks

Fake login forms are displayed to steal credentials.

Malware Distribution

Malicious software may be delivered to users.

Causes of XSS Vulnerabilities

XSS vulnerabilities typically arise because of:

- Lack of input validation
- Improper output encoding
- Unsafe handling of user-generated content
- Insecure client-side scripting
- Poor secure coding practices

Prevention Techniques

Input Validation

Validate and restrict user input before processing.

Output Encoding

Encode user data before displaying it in web pages.

Content Security Policy (CSP)

Implement browser security policies to restrict script execution.

Secure Coding Practices

Avoid unsafe methods that directly insert user content into webpages.

Security Testing

Regularly test applications for XSS vulnerabilities.

The Rest of the OWASP Top Vulnerabilities

1. Broken Access Control

Access control determines what users are allowed to do within an application. A vulnerability occurs when users can access resources or perform actions beyond their authorized permissions.

Example

A regular user gains access to administrator pages or other users' information.

Impact

- Unauthorized access
- Data theft
- Privilege escalation

---

2. Cryptographic Failures

Sensitive data such as passwords, credit card numbers, and personal information must be protected using encryption. Cryptographic failures occur when data is stored or transmitted without adequate protection.

Example

Passwords stored in plain text instead of encrypted or hashed form.

Impact

- Data exposure
- Identity theft
- Financial fraud

---

3. Insecure Design

Security should be considered during the design phase of an application. Insecure design occurs when security controls are missing or poorly planned.

Example

An application allows unlimited login attempts without any restrictions.

Impact

- Easier exploitation by attackers
- Increased security risks

---

4. Security Misconfiguration

Applications, servers, and databases must be configured securely. Incorrect settings can create security weaknesses.

Example

Using default passwords or leaving debug mode enabled.

Impact

- Unauthorized access
- Information disclosure
- System compromise

---

5. Vulnerable and Outdated Components

Modern applications rely on third-party libraries, frameworks, and software components. Using outdated versions with known vulnerabilities increases security risks.

Example

Using an old framework version that contains publicly known security flaws.

Impact

- Exploitation of known vulnerabilities
- Data breaches

---

6. Identification and Authentication Failures

Authentication verifies the identity of users. Weak authentication mechanisms allow attackers to gain unauthorized access.

Example

Weak passwords, poor session management, or lack of Multi-Factor Authentication (MFA).

Impact

- Account takeover
- Unauthorized access

---

7. Software and Data Integrity Failures

Applications should verify the integrity and authenticity of software updates and data. Trusting unverified sources can lead to compromise.

Example

Installing plugins or software updates from untrusted sources.

Impact

- Malware infection
- System compromise

---

8. Security Logging and Monitoring Failures

Applications should record security-related events and monitor suspicious activities. Without proper logging, attacks may remain undetected.

Example

Repeated failed login attempts are not recorded.

Impact

- Delayed detection of attacks
- Difficulty investigating security incidents

9. Server-Side Request Forgery (SSRF)

SSRF occurs when an attacker tricks a server into making requests to internal or external systems on their behalf.

Example

A server is manipulated into accessing internal resources that should not be publicly reachable.

Impact

- Access to internal systems
- Exposure of sensitive information
- Network compromise

Vulnerability Analysis

What is Vulnerability Analysis?

Vulnerability Analysis is the process of identifying security weaknesses (vulnerabilities) in a system, software, web application, or network before attackers can exploit them.

Real-Life Example

Think of a house security inspection.

Before a thief enters the house, the owner checks:

- Are doors locked?
- Are windows secure?
- Is the alarm system working?

Similarly, security experts examine software and systems to find weaknesses before hackers do.

Types of Vulnerability Analysis

1. Passive Analysis

Passive Analysis involves observing and collecting information about a system **without directly interacting with or attacking it**.

The goal is to gather information without affecting the normal operation of the system.

Example

Imagine a security guard watching a building from outside.

He observes:

- Number of entrances
- Security cameras
- Visitor movement

But he never enters the building.

This is Passive Analysis.

Characteristics

- No direct interaction
- No system modification
- Safe and non-intrusive
- Information gathering only

Information Collected

- IP addresses
- Domain names
- Server information
- Network traffic
- Publicly available data

Advantages

- Safe
- Does not disrupt operations
- Difficult to detect

Limitation

- Cannot discover all vulnerabilities

---

2. Source Code Analysis

Source Code Analysis is the process of examining the application's source code to identify security weaknesses and programming errors.

The code is reviewed to find vulnerabilities before the software is deployed.

Example

Imagine a teacher checking a student's answer sheet before submission.

The teacher finds mistakes and corrects them before the exam paper is submitted.

Similarly, developers inspect source code to find security flaws.

What is Checked?

- Coding errors

- SQL Injection vulnerabilities
- Cross-Site Scripting (XSS)
- Hardcoded passwords
- Improper authentication logic

Types

Static Analysis

The code is analyzed without running the program.

Manual Code Review

Security experts manually inspect the source code.

Advantages

- Detects vulnerabilities early
- Reduces development costs
- Improves code quality

Limitation

- Requires access to source code
- Time-consuming for large applications

---

3. Binary Analysis

Binary Analysis examines compiled software files (executables) when source code is unavailable.

Instead of reviewing source code, analysts study the machine code generated after compilation.

Example

Imagine a mechanic inspecting a completed car without seeing its design blueprint.

The mechanic studies the finished product to identify problems.

Similarly, Binary Analysis examines the final software product.

When is Binary Analysis Used?

- Source code is unavailable
- Third-party software testing
- Malware analysis
- Reverse engineering

What is Analyzed?

- Executable files
- Program behavior
- Security flaws
- Hidden functions
- Malware characteristics

Types

Static Binary Analysis

Examining executable files without running them.

Dynamic Binary Analysis

Running the software in a controlled environment and observing its behavior.

Advantages

- Useful when source code is unavailable
- Helps analyze third-party software
- Effective for malware analysis

Limitation

- More complex than source code analysis
- Time-consuming