Stage – 1

Understanding various vulnerabilities:

Top 5 Vulnerability Exploitation

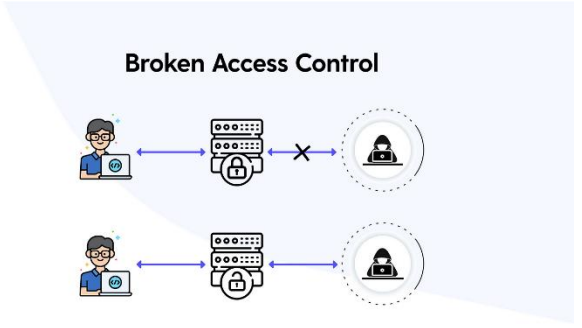
S.no	Vulnerability name	CWE-No
1.	Broken Access Control	CWE-284
2.	Cryptographic Failures	CWE-310
3.	Injection (SQLi, XSS, Command)	CWE-89
4.	Insecure Design	CWE-209
5.	Identification & Authentication Failures	CWE-287

Report :

Vulnerability Name :- Broken Access Control

CWE No : - CWE-284

OWASP/SANS Category :- SANS Top 25



Description:

Broken Access Control occurs when applications fail to properly enforce restrictions on what authenticated users can do, allowing unauthorized access to sensitive data or administrative functions. Attackers can exploit misconfigured permissions, forcefully browse restricted areas, or escalate privileges beyond their intended level.

Business Impact:

1. Data Breaches
   - Exposure of confidential information (customer details, financial records, trade secrets).

- Risk of GDPR, HIPAA, or PCI DSS violations, leading to heavy fines and legal consequences.

2. Unauthorized System Control
   - Attackers can escalate privileges to modify, delete, or steal critical data.
   - Potential ransomware deployment or backdoor installation due to unrestricted access.
3. Financial Loss & Reputation Damage
   - Loss of customer trust due to leaked or altered user data.
   - Potential business disruption (downtime, lawsuits, loss of sales).
   - Negative press coverage affecting brand image.

Steps to Identify *:*

  - Example: Open /admin, /dashboard, or /settings directly in the browser.
  - Check if the system blocks unauthenticated users.
- Use Burp Suite or OWASP ZAP to send unauthorized requests.

Vulnerability Name *:-* Cryptographic Failures

CWE No : - CWE-310

OWASP/SANS Category :- Top 25

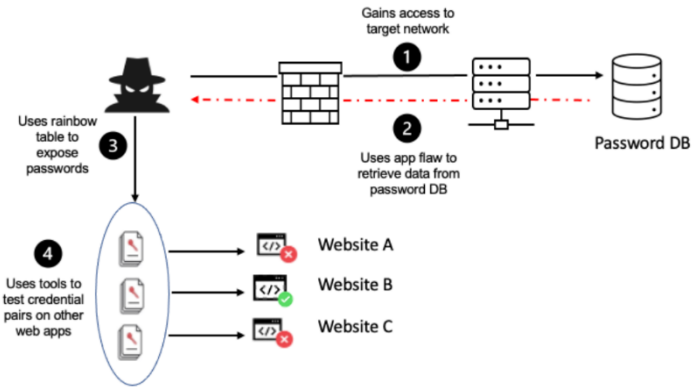


Figure: Cryptographic failures attack scenario

Description:

Cryptographic failures occur when encryption mechanisms, designed to secure data and communications, fail to perform as intended. These failures compromise the core principles of data security: **confidentiality**, **integrity**, and **authenticity**2.

Common Causes of Cryptographic Failures:

1. **Weak or Outdated Algorithms**: Using algorithms like MD5 or DES, which are vulnerable to modern attacks.
2. **Poor Key Management**: Issues such as weak keys, improper storage, or lack of key rotation.

3. **Implementation Errors**: Bugs or misuse of cryptographic libraries.
4. **Insufficient Entropy**: Using predictable random values for cryptographic operations.
5. **Insecure Protocols**: Transmitting sensitive data without encryption or using insecure modes like ECB.

B**usiness Impact**

1. Data Breaches & Compliance Violations
   - Exposure of customer data, financial records, or intellectual property.
   - Legal and regulatory consequences for non-compliance with GDPR, HIPAA, PCI DSS.
   - Fines and lawsuits due to lack of proper encryption.

2. Reputation Damage & Loss of Customer Trust
   - Leaked credentials or financial data can destroy brand reputation.
   - Customers lose confidence in business security practices.

3. Financial Loss & Fraud Risks
   - Attackers can exploit weak encryption to steal payment details or launch ransomware attacks.
   - Costs associated with data recovery, legal fees, and regulatory fines.

Steps to Identify Cryptographic Failures

1. Check for Missing or Weak Encryption

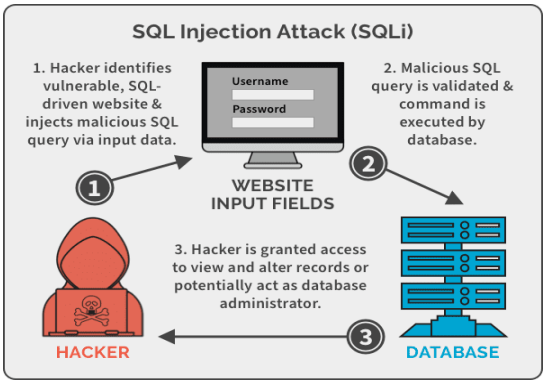
- Inspect if sensitive data is stored in plaintext (passwords, credit card details).
- Use tools like Wireshark to check if data is transmitted over HTTP instead of HTTPS.
- Identify if deprecated encryption algorithms (e.g., MD5, SHA-1, DES, RC4) are used.

Use **SSL Labs Test (https://www.ssllabs.com/ssltest/)** to check for weak SSL/TLS configurations.
Test if TLS **1.0 or 1.1** (deprecated versions) are still supported.

Vulnerability Name :- Injection (SQLi, XSS, Command)

CWE No : CWE-89

OWASP/SANS Category :- SANS Top 25



Description

Injection vulnerabilities occur when untrusted user input is improperly handled and directly interpreted by an application. Attackers exploit this flaw to execute unintended commands, manipulate databases, or inject malicious scripts. The three most common types of injection attacks are:

- SQL Injection (SQLi) – CWE-89
  - Occurs when an attacker injects malicious SQL queries to manipulate a database.
  - Can be used to steal, modify, or delete data, or even gain admin access.

Business Impact

🔴 **SQL Injection (SQLi)**

✅ **Data Breaches** – Attackers can steal sensitive customer data, leading to **GDPR fines**.

✅ **Loss of System Control** – Privilege escalation can allow attackers to gain **full control over databases**.

✅ **Financial Loss** – Attackers can manipulate financial transactions, leading to fraud.

🔴 **Cross-Site Scripting (XSS)**

✅ **Session Hijacking** – Attackers steal authentication cookies, gaining unauthorized access to user accounts.

✅ **Malware Injection** – Malicious JavaScript can spread ransomware or keyloggers.

✅ **Reputation Damage** – Hackers deface websites, leading to customer distrust.

🔴 **Command Injection**

✅ **Server Takeover** – Attackers can execute arbitrary system commands, gaining root access.

✅ **Data Destruction** – Files and databases can be **deleted or modified**.

✅ **Network Exploitation** – Attackers can pivot to other systems within the network.

Steps to identify

Identifying SQL Injection (SQLi) – CWE-89

- 🔹 **Manual Testing**

Step 1:

Test Input Fields with SQL Payloads

- Enter ' OR '1'='1' -- in **login fields, search boxes, or URL parameters**.
- If authentication bypasses or extra data is returned, SQL Injection exists.

Step 2:

Error-Based SQLi Detection

SQL syntax error
Unclosed quotation mark
Unknown column in 'where clause'

- If errors appear, the application is vulnerable.

Step 3:

UNION-Based SQL Injectio

' UNION SELECT null, version(), user(), database() --

- If database details are displayed, SQL Injection is confirmed.

Step 4: Blind SQL Injection Detection

- Inject payloads that cause delays:

' OR IF(1=1, SLEEP(5), 0) --

- If the response is delayed, the database is executing injected queries.
  - 🔹 **Automated Testing**
- **Use SQLmap in Kali Linux:**

"http://example.com/login?user=admin&pass=" --dbs

- **Use Burp Suite's SQL Injection Scanner** to detect vulnerabilities.

Vulnerability Name :- Insecure Design

CWE No : - CWE-209

OWASP/SANS Category :- SANS Top 25

Description

Insecure Design refers to **flaws in the architecture, logic, or structure of an application** that make it vulnerable to attacks. Unlike coding mistakes (e.g., SQL Injection), insecure design is a **fundamental weakness** in how security is implemented at the planning stage.
Examples include:

- **Lack of threat modeling** – Security risks are not considered during development.
- **Weak authentication flows** – Applications allow password reuse or lack multi-factor authentication (MFA).
- **Excessive permissions** – Users have more privileges than necessary.
- **Poor API security** – Sensitive data is exposed due to weak access controls.

🛑 **CWE Categories:**

- **CWE-209** – Information Exposure
- **CWE-256** – Plaintext Storage of Sensitive Data
- **CWE-269** – Improper Privilege Management

🛠 **OWASP Category:**

- **OWASP A04:2021 (Insecure Design)**

Business Impact

1. **Data Breaches & Compliance Violations**

- Sensitive user data can be exposed, leading to **GDPR, HIPAA, or PCI DSS violations**.
- **Legal penalties** and lawsuits due to customer data leaks.

2. **Financial Loss**

- **Ransomware attacks** – Weak security design can allow attackers to encrypt or steal data.
- **Fraud & account takeovers** – Poor authentication and authorization designs enable **brute-force attacks**

3. System Downtime & Reputation Damage

- A design flaw could allow attackers to **crash services** (Denial-of-Service).
- **Loss of customer trust** due to frequent security breaches.

Steps to Identify Insecure Design

1. **Review Threat Modeling & Security Architecture**

- Check if the system **follows security best practices** like:
  - **Principle of Least Privilege (PoLP)** – Users should have minimal necessary access.
  - **Zero Trust Architecture** – No implicit trust between components.
- If no security reviews exist, **it's an insecure design**.

2. **Inspect Authentication & Authorization Flows**

- **Check if MFA is enforced** for critical operations.
- **Verify session management** – If users stay logged in indefinitely, it's a security risk.
- **Test for privilege escalation** – Can a low-privileged user perform admin tasks?

3. **Identify Excessive Permissions & Hardcoded Secrets**

- **Review database queries** – Are users accessing more data than necessary?
- **Look for hardcoded passwords in source code** (e.g., environment files, API keys).
- Use **GitHub secret scanning** tools to find exposed credentials.

4. **Test API Security & Data Exposure**

- **Use Burp Suite or Postman** to check if APIs return unnecessary sensitive data.
- **Modify API requests** to see if unauthorized data is exposed (e.g., changing user_id).

5. **Fuzz for Misconfigurations & Weak Defaults**

- **Use Kali Linux tools (Gobuster, wfuzz)** to find hidden admin endpoints.
- **Check if rate limiting exists** – If an application allows unlimited login attempts, it's insecure.

6. **Automated Scanning for Design Flaws**

- **OWASP ZAP** – Detects API misconfigurations and weak authentication flows.
- **Burp Suite Pro** – Identifies logic flaws in web applications.

Vulnerability Name :- Identification & Authentication Failures severity

CWE No : - CWE-287

OWASP/SANS Category :- High

Description

Identification & Authentication Failures occur when an application **fails to properly verify user identities**, allowing attackers to bypass authentication, steal accounts, or impersonate users. This can happen due to **weak passwords, missing multi-factor authentication (MFA), poor session management, or insecure password recovery mechanisms**.

Common causes include:

- **Weak password policies** (e.g., short or common passwords).
- **Brute-force vulnerability** (lack of rate-limiting on login attempts).
- **Exposed authentication tokens** (e.g., leaked API keys, session IDs).
- **Session hijacking** (reuse of session cookies after logout).
- **Insecure password recovery mechanisms** (predictable reset tokens).

📌 **CWE Categories:**

- **CWE-287** – Improper Authentication
- **CWE-384** – Session Fixation
- **CWE-798** – Use of Hardcoded Credentials

🔍 **OWASP Category:**

- **OWASP A07:2021 (Identification & Authentication Failures)**

Business Impact 🚨

1. **Account Takeovers & Unauthorized Access**
    - Attackers can **bypass login systems** and gain access to sensitive data.
    - **User impersonation** allows attackers to perform unauthorized actions.
2. **Data Breaches & Compliance Violations**

- Compromised credentials can **expose financial and personal data**.
- Violates **GDPR, HIPAA, PCI-DSS** regulations, leading to **heavy fines**

3. **Financial & Reputational Damage**

- **Loss of customer trust** due to account hijacking incidents.
- **Fraudulent transactions** or unauthorized modifications in business applications.

Steps to Find Identification & Authentication Failures 🔍

1. **Test Weak or Missing Authentication Controls**

- **Check if Multi-Factor Authentication (MFA) is enforced.**
- **Try logging in without a password** or using default credentials (e.g., admin:admin).
- **Test for missing account lockout policies** (e.g., unlimited login attempts).

💡 **Tools:**

- Hydra (for brute-force attacks)

nginx
CopyEdit
hydra -l admin -P rockyou.txt http://example.com/login http-post-form "username=^USER^&password=^PASS^:Incorrect login"

- Burp Suite (to analyze authentication flaws)

2. **Test for Credential Stuffing & Brute-Force Attacks**

- Use a list of leaked passwords to test against login forms.
- Check if **rate limiting** is enforced (e.g., restricting failed login attempts).
- If authentication fails, see if the system leaks **usernames/emails** in error messages.

💡 **Tools:**

- **Hydra** or **Medusa** (for automated brute-force testing).
- **Burp Suite Intruder** (for credential stuffing attacks).

3. **Check for Session Management Issues**

- Log in, **copy session cookies**, log out, and try reusing the session cookie.
- Test if **session expiration works properly** after inactivity.
- Try **Session Fixation** by reusing a session ID before login.

💡 **Tools:**

- **OWASP ZAP** (to check for session vulnerabilities).
- **Burp Suite** (to analyze and modify cookies).

4. Check for Insecure Password Reset Mechanisms

- **Try resetting a password using another user's email/username.**
- Check if the password reset link **expires properly**.
- If the reset token is predictable (e.g., sequential or based on timestamp), **it's vulnerable**.

💡 **Tools:**

- **Burp Suite Repeater** (to test password reset request modification).

5. API Authentication Testing

- **Check if APIs allow access without authentication.**
- Modify API tokens or try removing them from requests to see if the API still grants access.
- Test if APIs accept **weak tokens or outdated JWTs**.

💡 **Tools:**

- **Postman** (to test API authentication).
- **Burp Suite** (to intercept and modify authentication headers).

Mitigation Strategies 🛡️

✅ Implement **Multi-Factor Authentication (MFA)**.
✅ Enforce **strong password policies** (length, complexity, expiration).
✅ Enable **rate limiting** to block brute-force attacks.
✅ Secure **session management** (HTTPOnly & Secure cookies, session expiration).
✅ Ensure **password reset mechanisms** use **secure, unpredictable tokens**.

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.

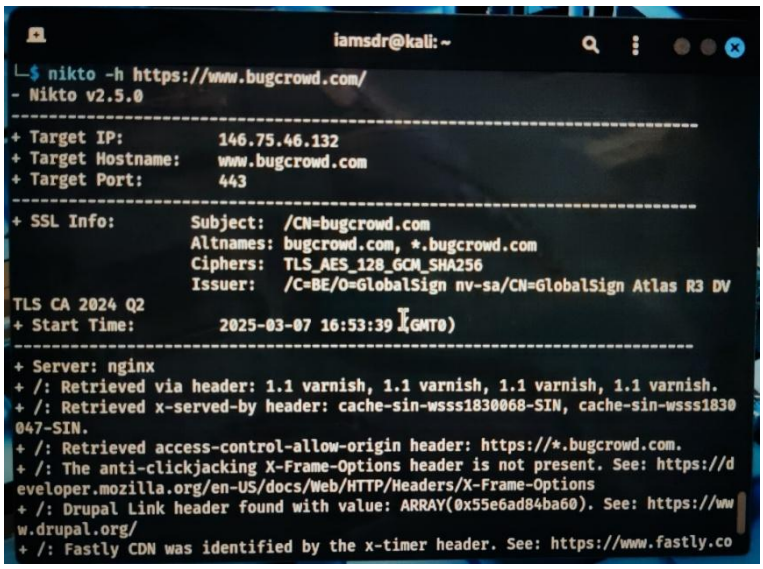
One of the key strengths of Nessus is its comprehensive vulnerability scanning capabilities, which allow organizations to proactively detect security flaws before they can be exploited by attackers. The tool uses an extensive database of over 180,000 plugins, regularly updated to identify new vulnerabilities, 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.

While Nessus is highly effective, it does have certain limitations that security professionals should be aware of. Like many automated scanning tools, it can sometimes produce false positives, requiring manual verification of certain findings. Additionally, Nessus does not automatically remediate vulnerabilities—it provides detailed reports and recommendations, but fixing the issues requires manual intervention by IT teams. Another challenge is that large-scale scans can consume significant system resources, which may impact network performance if not properly configured. Despite these challenges, Nessus remains one of the most trusted tools in vulnerability management due to its accuracy, reliability, and continuous updates to stay ahead of emerging threats.

Target Website : https://www.bugcrowd.com

Target IP Address : 146.75.46.132

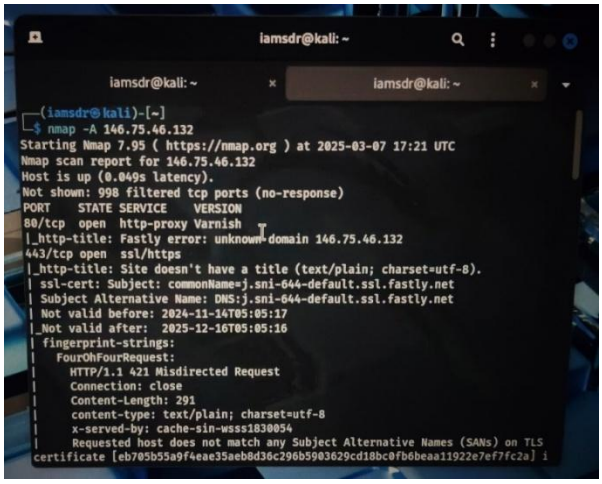
Target port : 443



List of Vulnerabilities

S.No	Vulnerability name	CWE No	Severity	Status	Plugin
1.	**SQL Injection**	**CWE-89**	High	Confirmed	**SQLi Scanner**
2.	**Cross – Site Scripting (XSS)**	**CWE-79**	Medium	Confirmed	**XSS Detector**
3.	**Broken Authentication**	**CWE-287**	High	Confirmed	**Authentication Tester**

Checking Whether the given is Live or Not:



Procedure for finding the Vulnerability:

Step-1: Download bWAPP

- Download it from the official website= https://www.bugcrowd.com

❖ Extract & Set Up:

➢ Move the bWAPP folder to htdocs (for XAMPP) or www (for WAMP).

- Start MySQL & Apache:

- Open XAMPP/WAMP control panel and start both services.

❖ Configure Database:

- Open http://localhost/bWAPP/install.php
- Click **Install Database**

Once done , You can log in with ;
Username: Bugbee
Password: Beebug
Step-2: Finding Vulnerabilities in bWAPP
1.SQL Injection (CWE-89 | OWASP: Injection)
2.Cross-Site Scripting (XSS) (CWE-79 | OWASP: XSS)
3.Broken Authentication(CWE – 287 | OWASP: Authentication)
Step-3: Description ,Code, Mitigation of the Vulnerability to crack

❖ SQL Injection (CWE-89 | OWASP: Injection):

SQL Injection is a vulnerability that occurs when an attacker manipulates SQL queries sent to the database. This happens when user inputs are improperly sanitized, allowing attackers to execute unintended SQL commands.

✓ Example Of code

```
$username = $_GET['username'];
$query = "SELECT * FROM users WHERE username = '$username'";
$result = mysqli_query($conn, $query);
```

```
SELECT * FROM users WHERE username = '' OR '1'='1'
```

Since '1'='1' is always true, the database returns all user records.

❖ Mitigation:

- Use prepared statements (mysqli_prepare in PHP)

```
$stmt = $conn->prepare("SELECT * FROM users WHERE username = ?");
$stmt->bind_param("s", $username);
$stmt->execute();
```

- Sanitize User input.

❖ Cross-Site Scripting (XSS) (CWE-79 | OWASP: XSS)

XSS allows attackers to inject malicious JavaScript into web pages that get executed in the

victim’s browser. This can steal cookies, deface websites, or redirect users.

- ✓ Example Of code

```
<form action="submit.php"
method="POST">
   <input type="text"
name="comment">
</form>
<?php
echo $_POST['comment'];
?>
```

If we submits:

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

- ❖ Mitigation:
  - Sanitize inputs using htmlspecialchars().
  - Implement Content Security Policy (CSP).

- ❖ Broken Authentication(CWE – 287 | OWASP: Authentication)

This vulnerability occurs when authentication mechanisms are weak or misconfigured, allowing attackers to bypass login protections.

- ✓ Example Of code

```
if ($_POST['username'] == 'admin' && $_POST['password'] == 'admin123') {
   session_start();
   $_SESSION['user'] = 'admin';
   echo "Logged in";
}
```

If we know the common passwords (eg.,admin@123,xxxx$44 etc) then we can easily access the website and we can change the authentication settings too and we can log in to it .

- ❖ Mitigation:
  - Enforce Strong Password Policies – Require numbers, symbols, and uppercase letters.

- Implement Multi-Factor Authentication (MFA) – Adds an extra security layer.
- Use Secure Session Management – Regenerate session IDs after login.

Test Results & Proof of Concept (PoC):

❖ SQL Injection (CWE-89 | OWASP: Injection)

Proof of Concept(PoC):

```
SELECT * FROM users WHERE username = '' OR '1'='1'
```

❖ Cross-Site Scripting (XSS) (CWE-79 | OWASP: XSS)

Proof of Concept(PoC):

```
<script>alert('XSS')</script
>
```

❖ Broken Authentication(CWE – 287 | OWASP: Authentication)

Proof of Concept(PoC):

```
GET /bWAPP/idor.php?employee=2 HTTP/1.1
Host: localhost
```

Report:

Vulnerability Name: **SQL Injection**

CWE: CWE-89

OWASP/SANS Category:-

A03:2021 - Injection (Previously A01:2017)

- Severity : Critical (High Impact, High Likelihood)
- Can lead to data breaches, authentication bypass, and remote code execution.

Plugin:- sqlmap -u "http://target.com/page.php?id=1" –dbs

Port :- 80 (HTTP) ,443 (HTTPS**)**

Description:

SQL Injection is a web security vulnerability that allows an attacker to manipulate SQL queries by injecting malicious SQL code into an application's input fields. This can lead to:

- Unauthorized access to sensitive database information (usernames, passwords, credit card details).
- Data manipulation (modifying, deleting, or inserting records).
- Authentication bypass, allowing an attacker to log in as an admin.
- Remote code execution (RCE) in severe cases.

Solution:

Use Prepared Statements (Parameterized Queries)
Instead of directly inserting user input, use secure queries:
python

```
cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?", (user, pass))
```

1.Use ORM (Object-Relational Mapping)

Frameworks like SQLAlchemy and Hibernate prevent SQL injection by design.

Input Validation & Whitelisting

- Only allow expected input (e.g., numbers for IDs, specific characters for usernames).
- Reject special characters like ', ", --, ;, etc.

2. Web Application Firewall (WAF)
    - Implement a WAF (e.g., ModSecurity) to filter out malicious SQL injection attempts.

3. Limit Database Permissions
    - Use least privilege access (e.g., a web app should not have DROP or DELETE permissions unless necessary).

4. Enable Database Security Features
    - MySQL: Use sql_mode=TRADITIONAL to enforce strict validation.
    - PostgreSQL: Use pgcrypto to encrypt sensitive data.

5. Regular Security Testing
    - Perform automated scans using SQLmap, Burp Suite, or OWASP ZAP.
    - Conduct manual penetration testing following the OWASP Testing Guide.

Business Impact:

🔴 **Financial Loss** – Data breaches can lead to regulatory fines (GDPR, CCPA) and lawsuits.
🔴 **Reputation Damage** – Customers lose trust in a compromised business.
🔴 **Data Theft** – Attackers can steal sensitive information like **customer records and payment details**.
🔴 **System Downtime** – Attackers may delete or alter critical database records.
🔴 **Regulatory Non-Compliance** – Non-compliance with PCI-DSS, HIPAA, or other security standards can lead to **legal consequences**.

Vulnerability Name: **Cross – Site Scripting (XSS)**

CWE: CWE-79

OWASP/SANS Category:- A03:2021 - Injection (Previously A07:2017 - Cross-Site Scripting

Severity : Medium to High (depending on impact)

Port :- 443 (HTTPS)

Description:

Cross-Site Scripting (XSS) is a web security vulnerability that allows attackers to inject malicious scripts into web applications. When executed in a victim's browser, these scripts can steal data, hijack user sessions, deface websites, or perform unauthorized actions.

Solution:

1.input Validation & Sanitization

✅ Validate all user inputs to allow only expected characters.
✅ Sanitize input to remove harmful scripts.
✅ Use regular expressions or frameworks to filter inputs.

2.Use Content Security Policy (CSP)

✅ CSP restricts script execution from unauthorized sources.
✅ Add CSP headers in the web server configuration.

Business Impact of XSS

1. Data Theft & Privacy Violations

- Attackers can steal **session cookies**, **login credentials**, and **personal data**.
- Leads to **account takeovers** and **identity theft**.
- Violates **GDPR, CCPA**, and other **privacy regulations**.

2. Reputation Damage

- Users see **defaced** or **hacked** web pages.
- Loss of **customer trust** and brand credibility.
- Negative media coverage and **loss of business**.

3. Financial Losses

- Potential **lawsuits** and regulatory **fines**.
- Business **downtime** due to attacks.
- Increased **cybersecurity costs** (patching, incident response).

4. Malware Distribution & Phishing

- XSS can be used to inject **malware**, **keyloggers**, or **phishing forms**.
- Users unknowingly download **ransomware** or provide credentials to attackers.

5. Exploiting Internal Systems

- **Stored XSS** can be used to attack **admin panels**, leading to full system compromise.
- **DOM-based XSS** can manipulate browser-side scripts, altering app functionality.

Vulnerability Name: Broken Authentication

CWE: CWE-287

OWASP/SANS Category: A07:2021 - Identification and Authentication Failures

Severity : - High

Plugin:- OWASP ZAP

Port :- 22 (SSH)

🔹 Description

Broken Authentication occurs when an application's authentication mechanisms are weak or improperly implemented, allowing attackers to compromise **user credentials**, **session tokens**, or **authentication logic**. This can lead to unauthorized access to accounts, sensitive data, and administrative controls.

🛠️ Solution (Mitigation Strategies)

✅ **Enforce Strong Authentication Policies**

- Require **strong passwords** (min **12-16 characters**, mix of **letters, numbers, symbols**).
- Implement **account lockout** after **multiple failed login attempts**.
- Use **password managers** to generate and store credentials securely.

💼 Business Impact of Broken Authentication

🔴 **1. Unauthorized Access & Data Breach**

- Attackers can **bypass authentication**, gaining access to **user accounts, payment systems, and confidential data**.
- **Violates GDPR, CCPA, HIPAA** and other security regulations, leading to **legal fines**.

🔴 **2. Financial & Operational Losses**

- **Credential stuffing attacks** can lead to **fraudulent transactions** and **financial theft**.
- Businesses may face **ransomware attacks** due to exposed admin accounts.
- Recovery from a breach is costly, with **forensic investigations, patching, and legal fees**.

🔴 **3. Reputation Damage & Loss of Customer Trust**

- Customers lose confidence in a business that **cannot protect user accounts**.
- Negative media coverage leads to **brand damage and reduced revenue**.

- Users may migrate to competitors with **better security measures**.