**Checklist:**

1. clickjacking
2. HTML injection
3. Reflected and Stored XSS
4. Missing security headers
5. Cache Control
6. Vulnerable software
7. Nmap scan
8. Testssl scan (Sslscan)
9. Host header injection
10. CORS
11. HTTPS methods (Trace, Debug)
12. Referrer
13. Origin
14. Verbose server version Disclosure in response
15. SSL/TLS certificate expiry
16. Deprecated protocols
17. Weak Ciphers
18. Information disclosure through error messages
19. robots.txt
20. Data semantics
21. Dirbuster
22. Anti automation/ No rate limit
23. X-Forwarded-host (Host header injection)
24. Internal IP disclosure
25. Privilege Escalation (vertical and Horizontal)
26. Insecure Direct Object Reference (IDOR)
27. Session Hijacking (Report only if XSS is present)
28. Session fixation
29. Cookie attributes set
30. Password change functionalities
31. User Enumeration (forgot and login page)
32. Unverified Input
33. CSRF
34. SSRF
35. Excessive session timeout

36. Concurrent login
37. File Uploads (.exe, aspx,. svg,. html,. py,.rb,. eicar,.bat)
38. Information disclosure in the GET query
39. Remote file Inclusion
40. Local file Inclusion
41. URL redirection
42. Exception & Error Handling
43. Anti-CSRF token
44. SQL Injection (and other injection methods)
45. Weak input validation
46. Sensitive data exposure in response
47. Remote code execution
48. ASP.NET related test cases
49. SQL map
50. No account lockout policy
51. No rate policy
52. Forced Browsing
53. Authentication bypass
54. SSTI
55. Session not validates after logout
56. Formula injection
57. Big IP cookie
58. Graphql
59. Path traversal
60. Business Logic-Parameter tamping
61. Weak password policy
62. Missing re-authentication upon email message
63. Missing session invalidation after password change
64. Cookie contains sensitive information
65. Hidden directory
66. No logout functionality
67. Unmasked PII
68. Prototype pollution
69. XXE injection
70. Source code disclosure

**1) Deprecated Protocols (SSL Labs → Test → website)**

- Not secure protocols

- Old / outdated versions of protocols → They have known vulnerabilities and performance issues.

**Common deprecated protocols:**

1. **SSL 2.0 & SSL 3.0**
2. **TLS 1.0 & TLS 1.1** → *(use 1.2 or 1.3)*
3. **HTTP 1.0 & HTTP 1.1** → *(use 2.0 or 3.0)*

**Use testing tools:**

1. nmap --Script: ssl-enum-ciphers
2. openssl s_client _connect yoursite.com:443
3. Check for **HTTP Strict Transport Security (HSTS)**

---

**2) Weak Ciphers**

- Outdated / insecure encryption algorithm
- Used in SSL/TLS communication between server & client for:
    1. Key exchange
    2. Authentication
    3. Encryption

→ Disable weak ciphers in the server's SSL/TLS configuration
→ Allow only strong ciphers

---

**3) Information Disclosure through Error Messages**

- Exposure of sensitive information
- Testing or trying how the backend handles special characters- to check errors in code or information leakage
- Try inputting special characters to test how backend handles them
    1. Helps check for errors in code
    2. Helps check for input leakage

→ Enter special characters in input fields
→ Observe any detailed error messages

---

**4) robots.txt**

- It's a file used by websites to give instructions to search engines or crawlers
- Tells which pages are allowed or disallowed for crawling
- Sometimes contains sensitive page paths like **admin or login**

→ Access via: <domain>/robots.txt
→ Might reveal structure or info of the site which is vulnerable.

**5) Data Semantics (curl, sqlmap, Nikto):**

Check input validation mechanisms- means the data entered by user follow expected format only.

- Checking integrity and security of application.

- Try SQL, xss, or changing values -if it bypasses the request then it will allow if not it is safe
- Try entering:
    1. Invalid data types or formats
    2. Script-based/XSS payloads to bypass validation

→ Check validation messages
→ Test with fields like **email, phone number, password**

---

**6) Dirb / DirBuster**

- Penetration testing tools
- Used to find **hidden files or folders** which reveals the sensitive information.

→ Helps to find:

1. Sensitive info
2. Missing validation
3. Loopholes in access control
4. Detect the misconfiguration

---

**7) Anti automation/no rate limit:**

- Stops scripts or bots from interacting with the app.

Definition:
When the server doesn't limit or restrict how many requests a user/client can send → then it leads to DoS or brute-force attacks.

Testing:

1. No limit on number of login or input attempts so limits for user per attempt(session)
2. Send many requests quickly to check if the app accepts them if the application receives that requests then it is vulnerable.
3. In Burp Suite, use Intruder to intercept request → set sequential numbers →then launch attack

---

**8) X-Forwarded-Host (Host Header Injection)**

- In **Burp**, there's a header line called X-Forwarded-Host
- Modify this line → Add another website name
- Check if the request gets redirected
  → If **yes**, it's vulnerable

→ External users should **not** be able to redirect
→ If header is missing, try manually adding it and test

→Check whether it redirects or not

**9) Internal IP Disclosure (Private IP Address)**

- Check if internal/private IPs are exposed
- Can be found by:

1. Viewing specific pages-source pages
2. Intercepting and inspecting requests

---

**10) Privilege Escalation (Vertical, Horizontal)**

- Exploiting flaws/vulnerabilities in a system to gain unauthorized access which is beyond the permission

**Types:**

1. **Vertical Escalation**: The normal user gains→Higher level access
    - Normal user gains **admin/root** level access
2. **Horizontal Escalation**: User access the resource/data of another user with the same level.
    - One user accesses **another user's data** (like emails, files, etc.)

→ If a user can **perform CRUD operations** after getting unauthorized access → it's privilege escalation→Can perform CRUD operation

**CRUD Operation→**

1. Create – Add something new
2. Read – View data
3. Update – Edit or change
4. Delete – Remove data

If a user gets unauthorized access and can do any of these actions (like viewing other users' data or changing settings), it is called privilege escalation.

---

**11)Insecure Direct Object Reference (IDOR)**

- Occurs when **URL or ID in the URL is changed**, and the app allows access after changing the URL.

→ If you can access data just by changing the ID → **it's IDOR**1
→ Similar to privilege escalation, but direct object access

---

**12) Session Hijacking (Report only if XSS is Present)**

- Attacker takes over **active user session** without login
- Hijacking the user account.

**How?**

1. Steals session ID or cookie/token
2. If XSS is present → attacker can capture session information
3. Without login or any credentials the attacker can gain access to user account.

→ Report only if **XSS vulnerability** is confirmed

**13) Session Fixation**

- Attacker **sets a session ID** and tricks the user into using it

→ In **Burp**, try to **change the session ID (cookie)** manually
→ If it works and gives access → it's session fixation

So basically, tricking user to use session id created by attacker.

---

**14) Cookie Attribute Set**

Cookies should have security attributes set:

1. **Secure** – only allows over HTTPS
2. **HTTP Only** – protects from XSS
3. **Samesite** – controls cross-site request
4. **Expires** – sets the cookie lifetime

→ If these are missing, cookies may be insecure. So, to manage or control cookies it will add some attributes.

---

**15) Password Change Functionalities:**

Test the security of password change feature: When user change their account password→

1. Allows weak password?
2. No authentication required?
3. No rate limiting?
4. Accepts reused/old passwords?

→ If any of these work → it's vulnerable

---

**16) User Enumeration (Forgot/Login Page)**

- Check if attacker can find out if a **username or email exists**

**Example:**

- If wrong login gives message like:
    - *"Password is incorrect"* (instead of generic message)
      → Then attacker knows the username is valid → which is **vulnerable**

---

**17) Unverified Input:**

- App doesn't validate/sanitize user input then it will lead to various attacks

→ Can lead to many attacks:

1. SQL Injection (SQLi)
2. Cross-site Scripting (XSS)
3. File Inclusion
4. Security Misconfiguration
5. Auth Bypass
6. Command Injection
7. SSRF
8. IDOR

9. Host Header Injection

---

**18) Excessive Session Timeout:**

- User session stays **active for too long**, even without activity. If user sessions are stay active for too long after login even after the user is inactive then it is vulnerable.

→ Should log out automatically after some time→ 15–45 minutes.
→ If session continues after long inactivity →then it is **vulnerable.**

---

**19)Clickjacking:**

**Code:**

```
<html>
<head>
  <title>Clickjacking Test</title>
</head>
<body>
  <h2>If you can see the website below, it may be vulnerable to clickjacking</h2>
  <iframe src="https://example.com" width="800" height="600"></iframe>
</body>
</html>
```

→If it is vulnerable it will open in the other website directly

→Also, the X-frame security headers are not there.

---

**20) HTML injection:**

→Injecting malicious HTML / JavaScript code into web page

→ if input is not properly sanitized →if it reflects back.
→ XSS can happens because of this
→ input HTML… into input field | try changing URL parameters or in elements (F12) tab search for html tags.

---

## 21) Concurrent login

→ when a same user account → logged into an application from multiple devices → at same time.
→ here login Firefox and as well as in browser → then check if it allows / no
→ from this Session Hijacking can happens.
→ so limit concurrent logins so that we can avoid session hijacking

**22)File upload (.exe, .php, .aspx, .svg, .htm, .py, .rb, .etc, .bat)**

Some websites or web applications allow users to **upload files** (like images, documents, etc.).

But if the website **doesn't check properly** what type of file is being uploaded, then it becomes **vulnerable**.

→ web application allows users → to upload files → so without proper security checks → (doesn't check file extensions)
→it will lead to security risks
→ lets upload any file → without checking →

The application must **validate the file** in 3 ways:
① file type
② file content
③ file size

1)Try uploading a normal file first (like .jpg, .pdf) → it should work safe file → (.jpg, .pdf)
2) Then try uploading a file like .exe or .php →
save file → try upload it → If the app allows it, then **it is vulnerable**

---

## 23. Information Disclosure in GET Query

Sensitive Information in URL (GET Request)
Sometimes, websites send user data through the URL. This is called a GET request.
If the URL shows important information like a username or password, it can be a security risk.
Anyone who sees or copies the URL can also see the sensitive data.
Hackers can use tools to intercept these URLs and steal the information.

Examples of sensitive data that should not appear in a URL:

- Username
- Password
- Session ID
- Token
- Cookies

If you try adding special characters or scripts in the URL and the website gives detailed error messages or exposes more information, it means the site may be vulnerable.

---

## 24. Remote File Inclusion & Local File Inclusion

Webserver / app → allow user to access / upload file
It should restrict the file access or upload
If input is not validated properly, it will allow it

**Local:**
→Try to hack or trick server → including a file that is stored on the same server
→In GET request or URL → try to change it as 127.0.0.1 or .../.../.../.../etc/passwd
→If it is vulnerable then it will reveal information of user id, username → also some system information, log files etc…

**Remote:**
Try to trick the server → including a file from a different server (remote) or from different host
Here they have to host the file and they add malicious script → and get a control over other web servers

## 25. URL Redirection

- Some websites automatically redirect users to another page or website. This is called URL redirection.
- This is normal — for example, after login, it may redirect to a dashboard.
- But if the website does not check where it's redirecting to, then hackers can misuse it.

- If a hacker can change the redirect link (like in a URL parameter), they can **trick users** into going to a **fake or malicious website** →This is called an **Open Redirect vulnerability**.
- Try to change parameter → to different URL → then If it redirects without proper validation→ then it is vulnerable

---

## 26. Exception / Error Handling

In an website where something goes wrong: such as
(i) missing file
(ii) incorrect input
(iii) server issue

→ happens → the server → should not display any technical errors
If it shows technical errors → it is vulnerable

Examples:

1. In URL www.example.com?id=101 something → remove the id / some part →
   If it is vulnerable → it will disclose the contents
2. In text field add → num / in num → email / … see the error
3. Try to type a very long message in form
4. In any text box → try to add special characters → check for errors
5. Visit other pages → try redirection → check errors

   So if it shows any technical errors then It is vulnerable

---

## 27. Anti CSRF – Token (Included in request forms)

- CSRF stands for Cross-Site Request Forgery.
- It's a type of attack where a hacker tricks a logged-in user (like you) into doing something you didn't want — like transferring money, changing password, etc.
- And you don't even realize it happened — it uses your active session!

**To stop this kind of attack, websites use something called an Anti-CSRF Token**.

This is a **random secret value (token)** added to:

- Forms
- API requests
- Login or transfer actions

When the user submits a form, the **token is sent with it**.

The server checks:

- "Does the token match the one I gave to this user?"
  - If yes → request is real
  - If no (or missing) → reject it

## 28. Missing Security Headers

These Protects user → server → from different types of attacks
(XSS, clickjacking, CSRF, sniffing …)

Go to target → select the testing URL →then Senqd it to scope → go to proxy → go to HTTP history
Click on → filters → select "show only scope" then it will show the filtered url and in response we can see the headers included

---

## 29). Cache Control

- When you visit a website, your **browser saves (caches)** some parts of the page (like images, pages, or data) so it loads faster next time.

Sometimes, **sensitive information** like:

- Bank details
- Account info
- Personal data

...can also be stored in the browser's cache.
If someone else uses the same browser or computer, they may see that sensitive data.

That's a **security risk**.

**So here the browser uses the cache control**

→To prevent this, websites should send **special HTTP headers** that tell the browser:

"Don't save or store this page."

These headers are called **Cache-Control** headers.

**It should show**

→Cache-Control: no-store, no-cache, must-revalidate Pragma: no-cache

So, these tell the browser:

"Do not save this page in cache at all."

**If the website uses:**

Cache-Control: max-age=0 this is vulnerable

It might still cache some parts. So, it's not fully safe.

---

**30)Vulnerable Software**:(Nessus/Qualys, nmap -sV <URL>)

→ Use wapalyzer extension to test the outdated or old software versions.

→it will show everything because it checks versions of every application.

---

**31)Nmap Scan:**

- Only ports 443 (HTTPS) and 80 (HTTP) should be open.
- If any other ports are open, it may be considered vulnerable.

Commands:

**nmap -sS <URL>**

**nmap --top-ports 5000 <URL>**

**32) SSL/TLS:**

**A) TestSSL Scan (SSL/TLS Security Check)**

**B) SSL/TLS Certificate Expiry**

- SSL certificate should clearly mention an expiry date.
- If the application is using the expired certificate, then that is vulnerable

→Go to the testssl.sh directory:

**cd testssl.sh**

**./testssl.sh <URL>**

- If it shows "vulnerable" in the output, the website is vulnerable.
- It also shows details about ciphers and cryptographic configurations.

**33) Host Header Injection**

- This involves manipulating the **Host** header in the request.

  **Host: <original-domain>**

  **To:**

  **Host: bing.com**

- Use **Burp Suite** to test.
- If it **redirects** or behaves unexpectedly, it is **vulnerable**.

**34) Origin Header (CORS)**

CORS is a security feature in web browsers that controls which websites (origins) are allowed to access resources (like APIs or data) on another domain.

- Used in **CORS (Cross-Origin Resource Sharing)** requests.
- It identifies the **origin (domain)** of the request.
- Only the **domain** is included (not the full URL).
- Security feature → controls → which website are allowed → to access → resource from another website →

  →[access-control-allow-origin]

  In **Burp Suite**, send a request and manually add this header:

  **Origin: bing.com**

  Check the response. If it includes:

  **Access-Control-Allow-Origin: bing.com**

  Then, the application is **vulnerable**, because it's allowing any origin.

**35)Verbose Server Version Disclosure**

- The server should **not disclose** its software version in response headers.
- This can sometimes be seen in **error messages** or **HTTP responses**.
- Test with different request types like **GET, POST** to check.

**43) SQL Injection (and Other Injections)**

- Targets the database by inserting **malicious SQL code** into user inputs.
- **Example payload:**

' OR 1=1 --

- This makes the condition **always true** and can bypass login or extract data.
- Occurs when app fails to sanitize SQL queries.

**Common Injection Types**

1. XSS (Cross-site Scripting)
2. CSRF (Cross-site Request Forgery)
3. SSRF (Server-side Request Forgery)
4. Session Hijacking
5. Command Injection
6. HTML Injection
7. Host Header Injection
8. LDAP Injection

---

**42) Weak Input Validation (Tools: Burp, ZAP, SQLMap)**

- The app **does not properly validate or sanitize** user input.
- Leads to different types of attacks.

**Examples of Weak Input Validation:**

- Accepting characters in number fields
- Allowing script code to bypass login
- Displaying detailed error messages
- No error handling

**Prevention Tips:**

- Limit input length
- Use **prepared statements**
- Sanitize all user inputs

---

**43) Sensitive Data Exposure**

(*input test, sensitive, HTTPS/SSL, sessions, data errors*)

- The web application sometimes **exposes sensitive data** through its environment or behavior.

  **Examples of data that can be exposed:**

  → Password
  → Email
  → Username
  → Server version and other system information

  **When does it happen?**

  → When passwords are stored in **plain text**
  → During **insecure data transmission** (i.e., not using HTTPS)
  → Through **error messages** that reveal sensitive backend details
  → Through **session-related vulnerabilities**

---

**45) HTTPS Methods**

- Allowed methods: **GET, POST**
- Dangerous methods: **DELETE, PUT, TRACE, OPTIONS, DEBUG**
- Send these methods using tools like Burp or curl.

  **If these are accepted**, it is a vulnerability.
- Example:
  TRACE method may echo back the request content.

---

**46) Referrer Header**

- The Referrer header indicates **where the request came from** (the previous page).
- Add:

  **Referrer: <some-other-page>**
- A secure response should include:

  **Referrer-Policy: no-referrer OR strict-origin**

---

**45) Authentication Bypass**

- In this case, an attacker tries to access restricted areas of the application without valid login credentials.

  **Why it happens:**

1. Token manipulation
2. SQL Injection
3. Insecure direct object access (IDOR)

**How to test:**

1)Try to access restricted pages without logging in
2) Use default credentials like admin, test, etc.
3) Manipulate URLs or cookies
4) Check for IDOR (Insecure Direct Object References)
5) Test with SQL Injection payloads to bypass authentication

---

**46) Session Not Invalidated After Logout**

- When a user logs in, a **session ID** is created by the server.
- After the user logs out, that session ID should be **deleted or disabled**.

User logs in session id was created when user logout from that session the id should be deleted/disabled after logout.

**How to test:**

→ Intercept the session ID using a proxy tool (like Burp Suite) means intercept the network note down the session id.
→ After logout, try to reuse the same **session ID means again send the request.**
→ If the session ID **still works**, the app is **vulnerable**
→ This means the token is still **valid** and can be used by an attacker to regain access.

---

**47) Weak Password Policy**

A weak password policy is a serious security concern in any web application. If the application accepts passwords that are simple, short, or commonly used, it exposes users to various threats.

**Issues that indicate a weak password policy include:**

1. The application allows users to create weak passwords (e.g., "123456", "password").
2. It does not enforce minimum password complexity (e.g., uppercase, numbers, symbols).
3. Password reuse is allowed, meaning users can reuse old passwords.
4. There is no protection against brute force attacks.
5. No limits on login attempts.
6. Lack of Multi-Factor Authentication (MFA) for additional security.

A secure password policy should enforce strong, unique passwords and additional measures like MFA

---

**48) Missing Session Invalidation After Password Change**

When a user is logged into their account on multiple devices and they change their password on one device, the session on all other devices should be immediately invalidated.

If the application fails to do so, an attacker who has access to the session on another device can continue using the account even after the password is changed.

To protect against this, the application must automatically log out all other active sessions once the password is updated.

---

**49) Cookies Contain Sensitive Information**

Cookies are often used to store user data and session tokens. However, storing **sensitive information** such as:

- Username
- Password
- Session tokens
- Email addresses
- User roles (e.g., admin)

…inside cookies **without proper protection** can lead to serious vulnerabilities.

**Best practices include:**

- Ensure **cookies are encrypted** or signed.
- Set the Secure flag to allow transmission only over **HTTPS.**
- Use the **HttpOnly flag** to prevent access from client-side scripts.
- Set the **SameSite** attribute to control cross-site requests.
- Do not store sensitive data like passwords directly in cookies.
  Attackers can tamper with cookie values to bypass login or trigger errors

---

**50) No Rate Limiting Policy**

Applications that do not implement **rate limiting** are at risk of abuse and attacks such as Denial of Service (DoS) or brute force login attempts.

Without restrictions, an attacker can send an unlimited number of requests, such as:

- Repeated login attempts to guess passwords
- Submitting forms repeatedly to flood the server

**Proper rate limiting measures include:**

- Locking the account after a certain number of failed login attempts (e.g., 3–5)
- **slowing down or blocking a user if they send too many requests too quickly from the same internet address** (IP), to prevent abuse or attacks.
- Implementing CAPTCHA after repeated actions

---

**51) No Logout Functionality**

If a website or app **does not give a logout option**, the user cannot **properly close their session**.

Even if there is a logout button, sometimes the **session is not actually closed** in the background. That means, even after logout, the user is still "logged in" secretly.

**Why is this dangerous?**

- Let's say you use a public computer or someone else's phone to log in.
- If the app doesn't **end the session** properly after logout, someone else can open the browser and **access your account without needing your password**.

**What should happen?**

- When the user clicks "logout", the app must **delete the session**.
- After that, if someone tries to use the old session ID, it should be **invalid**.
- The app should also send the user to the login page or homepage.

---

**52) Forced Browsing**

Forced browsing is an attack where users attempt to access unauthorized parts of the application by directly modifying the URL.

**Example:**
A regular user visits: www.example.com
But then tries to access:

- /adminpanel
- /user-records
- /hidden-files

If proper access control checks are not in place, the server may serve restricted or sensitive data. This is often due to poor implementation of authentication and authorization logic.

**Mitigation:**

- Validate user roles and permissions on every page and API endpoint.
- Use proper access control on the server side.

---

# Information Disclosure in Web Applications

---

## 1. What is Information Disclosure?

Information disclosure occurs when a web application reveals sensitive or private data due to weak protection or poor design.
Examples of leaked data:

- Usernames, emails, and passwords
- Credit card and personal details
- Internal IP addresses
- Server file paths
- Configuration files like .config, .env, .sql

Attackers may not use the information immediately but can collect it for future attacks like data theft, privilege escalation, or system compromise.

---

## 2. Why Does It Occur?

- **Misconfigured servers or applications**
- **Poor design or lack of access control**
- **Public exposure of private files or folders**
- **Developers accidentally leaving secrets in public code**

---

## 3. How Does It Happen?

Here are the common ways sensitive data gets exposed:

1. **Backup files in public folders**
   Example: .bak, .zip, .tar files available in browser — attackers can download them.
2. **Server allows directory browsing**
   Users can list and open all files in a directory like example.com/files/.
3. **Sensitive data in public scripts**
   Passwords, tokens, or database config in .js or .html files.
4. **Detailed error messages shown**
   Example: “SQL Error in /home/admin/config.php” — reveals server paths and queries.
5. **Developer comments in source code**
   HTML may include notes like: <!-- Admin panel at /admin/login.php -->.
6. **Debug mode enabled on live site**
   Shows detailed logs, system paths, and configurations.
7. **Misconfigured file permissions**
   Private files like .env, .git, or backups are exposed on the internet.

8. **Hardcoded credentials in front-end code**
   Credentials directly written into JavaScript or HTML, visible in browser tools.
9. **Sensitive info stored client-side**
   Important data like tokens or roles saved in cookies or localStorage.
10. **Google Dorking**
    Hackers use search queries like index of /config.php to find exposed files.
11. **Scanning tools**
    Tools like **Dirsearch**, **Burp Suite**, **Shodan**, **Censys** used to find exposed paths, headers, or technologies.

---

**4. Types of Information Disclosure**

- **Filename and Path Disclosure** – Reveals file paths like /var/www/html/index.php.
- **File Path Traversal (LFI/RFI)** – Attackers access restricted files using paths like ../../etc/passwd.
- **Sensitive Info in Code** – API keys, tokens, internal IPs, or passwords written in public files.

---

**5. Common Sources of Disclosure**

- Error and debug messages
- Developer comments in HTML or JavaScript
- Backup or old files stored on the web server
- Public configuration or database files

---

**6. Server-Side Information Disclosure**

- Server name and version (e.g., Apache/2.4.29)
- Stack traces with file and line numbers
- Web technologies (e.g., PHP, Node.js, Python)
- Database connections or environment variables
- Secrets or tokens printed in logs

---

**7. Impact of Information Leakage**

- Attackers gather useful internal info for further attacks
- Discovery of hidden admin panels, debug pages, or internal APIs
- Leaked API keys or credentials can lead to system access
- Can result in **Remote Code Execution (RCE)**
- Legal risks due to violation of standards like **GDPR**, **PCI-DSS**
- Loss of brand trust, revenue, and users

## 8. Information Gathering Tools and Examples

### A. Censys

- Scans internet for open services and certificates
- Example:
    - Search homelane.com
    - If 301 redirect, try Host Header Injection
    - Copy IP and open in browser to see if data is exposed

### B. Shodan

- Finds internet-connected devices and their info
- Example:
    - Search yahoo.com
    - Look for server software, PHP info, etc.

### C. Google Dorking

- Advanced Google search to find sensitive data
- Examples:
    - index of /config.php
    - site:github.com "api_key"
    - Netflix admin_passwd filetype:txt

### D. Investigator Tool

- Used to gather basic domain and server information.

### E. Bug Bounty Helper Tool

- Helps automate recon and discover sensitive endpoints.

### F. GitHub Dorking

- Find secrets in public code repositories.
  Steps:
    - Go to github.com
    - Search: "api_key" or "secret_key"
    - Click on "Code" tab and inspect exposed files

---

## 9. File and Directory Disclosure using Dirsearch

- **Dirsearch** brute-forces paths and files on web servers
- Example command:
  dirsearch -u https://buildexact.com -e php,asp,aspx --full-url
- Helps find directories like /admin/, /config/, /backup/

**10. Detecting Disclosure with Burp Suite**

Steps:

1. Open Burp Suite and intercept the website (e.g., uniraj.ac.in)
2. Add it to "Target" scope
3. Right-click → Engagement Tools → Discover Content
4. Review which files or directories are listed
5. Check for any sensitive exposure

---

**11. Viewing Web Page Source Code (Client-Side)**

- Press Ctrl + U in your browser to view the page source
- Look for:
    - Hidden comments
    - File paths
    - JavaScript variables
    - Sensitive data like tokens or links to admin panels

---

# HTML Injection

**HTML Injection?**

HTML Injection is a web security vulnerability where an attacker is able to inject HTML code into a web page due to improper input handling.

This usually happens when:

- User input is not properly sanitized.
- Output is not encoded before displaying it in the browser.

As a result, attackers can modify how the web page behaves or appears to users.

---

**2. Purpose and Use**

- HTML is used to create web pages and web applications.
- When attackers inject custom HTML, they can:
    - Display fake content (phishing)
    - Redirect users to malicious websites
    - Insert images, forms, or malicious scripts

**3. How HTML Injection Works**

- The attacker finds a form field or input area where HTML code can be submitted.
- The input is reflected back into the page without validation or encoding.
- The browser renders the injected HTML as part of the page.

Example:
Input: <h1>Hacked!</h1>
If the server reflects this back, the page will show a large heading “Hacked!”

---

**4. Types of HTML Injection**

**A. Stored HTML Injection**

- The injected code is saved on the server (e.g., in a database).
- Every time a user visits the page or triggers a function, the code executes.

**B. Reflected HTML Injection**

- The injected HTML is not stored.
- It only appears in the browser through a reflected request (like via URL, POST, or GET parameter).

Reflected Types:

- Reflected via **GET** method
- Reflected via **POST** method
- Reflected via **URL injection**

---

**5. Practical Demonstration Steps**

**Using Test Site:**

- Visit testvulnweb
- Enter HTML code in a form or search field
- If the code reflects back, the site is vulnerable

**Example Payloads:**

- Insert an HTML image: <img src=x onerror=alert(1)>
- Create a redirect link
- Inject fake login form

---

**6. Example Website Used**

- irisfloris.com – Use search field or form input
- Go to backend → Find the form ID → Copy and paste into search or input field
- Observe if HTML is reflected

**7. Using bWAPP for Stored Injection**

- Open bWAPP in browser
- Select **Stored HTML Injection** option
- Inject payload in message or comment box
- Submit and see if it displays on the page

---

**8. Capture Credentials (Example)**

Steps:

- Use nc -lvp 4444 (Netcat listener) on Kali Linux
- Enter fake form (username: tester, password: test12340)
- If vulnerable, attacker can capture the input in Netcat terminal

---

**9. Why This Happens (Causes)**

- Input fields do not validate user input properly
- No filtering or encoding of special HTML characters
- Back-end scripts are misconfigured
- Application does not sanitize user-supplied data

---

**10. Impact of HTML Injection**

- Attacker can modify webpage content
- Create fake forms to steal login credentials
- Redirect users to malicious websites
- Launch phishing attacks
- Display unwanted content

---

**11. Real-Life Use by Attacker**

- Attacker crafts custom malicious HTML code
- Embeds fake login form or redirects user to another website
- When victim enters credentials, attacker captures them

Example:
An attacker sends a link with injected HTML that shows a fake login form.
User enters their credentials → Attacker gets the username and password.

# Clickjacking

**Definition:**
Clickjacking is a malicious technique where a user is tricked into clicking something different from what they see. This can lead to revealing confidential information or allowing the attacker to take control of the user's actions.

**Example Tool:**

- Visit https://clickjacker.io
- Test target: admin.wickr.com
- It shows whether the site has protection against clickjacking.

**How to Check Manually:**

- Use SecurityHeaders.com
- Enter the URL and check for missing headers like:
    - X-Frame-Options
    - Content-Security-Policy

**Impact:**

- Submits unwanted actions on behalf of users
- May trick user into changing security settings or passwords

---

# Host Header Injection

**Definition:**
The attacker manipulates the Host header in HTTP requests to force the application to behave abnormally.

**Common Techniques:**

- Supplying arbitrary Host headers
- Injecting duplicate Host headers
- Line wrapping (e.g., Host:\rbing.com)
- Injecting alternative headers:
    - X-Forwarded-Host
    - X-Host
    - X-Forwarded-Server
    - X-HTTP-Host-Override

**Impact:**

- Web cache poisoning
- Password reset poisoning
- Open redirect and phishing
- Session hijacking

# Web Cache Poisoning

**Definition:**
An attacker poisons the cache of a web server by crafting a malicious request so that future users receive the malicious content from the cache.

**Method:**

- Manipulate headers like Host, X-Forwarded-Host, etc.
- Target websites that don't validate cache-related headers

---

# Password Reset Poisoning

**Steps to Perform (Practical):**

1. Register on a target site (e.g., utwente.nl) using:
    - Temporary email (like temp-mail.org)
    - Username: root, Password: simple
2. Log out from the account
3. Click on **Forgot Password**
4. Capture the request in **Burp Suite**
5. In Burp:
    - Change the Host header to a malicious one, e.g., evil.com
6. Forward the request
7. In the **Response**, you will see a **Location** header showing the reset link
8. Right-click → "Show response in browser"
    - Copy that URL and open in Chrome
    - It may redirect to the fake domain evil.com

**Purpose:**
Trick the system into generating password reset links with an attacker-controlled domain

---

# Parameter Tampering

**Definition:**
Altering parameters exchanged between client and server to manipulate application behavior.

**Common Areas:**

- URL query strings
- Cookies
- Hidden form fields
- HTTP headers (like Referrer)

**Example Target for Testing:**

- zero.webappsecurity.com

**Steps (Practical):**

1. Go to the site and try a payment or transfer action
2. Capture the request in **Burp Suite**
3. Change the **amount** value (e.g., from 100 to 1)
4. Forward the request and observe whether the backend validates it or not

---

## Cross-Site Scripting (XSS)

**Definition:**
XSS is a vulnerability where **malicious scripts are injected** into websites or web applications.
These scripts execute in a user's browser and can **compromise user interactions** with the application.

**Impact:**

- Session Hijacking
- Redirection to malicious sites
- CSRF attacks
- Access to sensitive user data
- Manipulation of website content

XSS happens **inside the user's browser**, but it can affect the **website or application** functionality and its users.

---

**Types of XSS Vulnerabilities:**

**1) Reflected XSS:**

- The attacker sends a **malicious link** to the victim.
- The user **clicks the link**.
- The browser **sends the malicious data** and it gets executed.
- The script does **not get stored** anywhere on the server.
- It works when the input gets **reflected back immediately** in the response.

**Example:**
Search fields, error messages, URL parameters.
If vulnerable, a simple script like <script>alert(1)</script> may trigger a popup.

---

**2) Stored XSS:**

- Occurs when the **application stores user input** (e.g., comments, feedback forms).
- The attacker **injects malicious script** into a field that gets saved in the backend.

- Whenever any user **visits the page**, the stored script gets executed.
- This can send the **user's cookies or data to the attacker**.

---

**3) DOM-Based XSS:**

- The application **writes data directly to the DOM** (Document Object Model) using JavaScript.
- If the data is not properly **sanitized or validated**, attackers can **manipulate the DOM** and inject malicious JavaScript.

**Example:**
document.write(), innerHTML, or URL fragments (#id=value) being used insecurely.

---

**4) Blind XSS:**

- Similar to **Stored XSS**, but the payload **does not immediately trigger** in your browser.
- The payload is triggered **later** when **an admin, developer, or another user** views the stored data in a different context (like the admin panel).
- No popup is seen, but tools like **XSS Hunter** will notify you via email when it gets triggered.

---

**Practical XSS Testing Steps (Blind XSS Example):**

1. **In Notepad:**
    - Paste a Blind XSS payload (e.g., from XSS Hunter).
    - Save it as .html.
    - Open it in the browser to test rendering.
2. **Using Burp Suite:**
    - In the vulnerable input (like search), type: hello.
    - Open **Burp Suite**, **reload the page**, and **capture the request**.
    - **Send to Intruder**.
    - Add payload position around hello (e.g., <hello>).
3. **Use XSS Validator Extension:**
    - Install the XSS Validator browser extension.
    - Select payload set from XSS Validator and **copy the payloads**.
    - In Intruder, go to **Payloads tab** and paste the payloads.
4. **Configure Grep-Match:**
    - In Intruder > Options > Grep-Match:
        - Remove all default values.
        - From XSS Validator, copy the **Grep Phrase**.
        - Paste it into Grep-Match.
        - **Uncheck "URL-encode"** option.

- Select **Payloads > First Payload Only**.

5. **Start Attack:**
    - Run the attack.
    - Check for the **highest-length response** — usually indicates XSS payload execution.
    - Take that payload.
    - **Show response in browser** to verify execution.

---

**Automation Tools for XSS Testing:**

- **ParamSpider** – For parameter discovery
- **GAUplus** – For gathering all URLs of a target
- **GF Tool** – For filtering XSS parameters
- **XSS Hunter** – For Blind XSS testing and notifications

---

# SQL Injection

SQL is a Structured Query Language used for storing, manipulating, and retrieving data in databases.

SQL Injection allows an attacker to interfere with the queries which belong to the database.

When the attacker is able to interfere with the queries related to the database, it allows the attacker to view the data. They can directly fetch information from the database such as application information, configurations used, user information, and login details.

The attacker can also easily modify or delete the data.

## Types of SQL Injection

In-band SQLi (error-based, union-based)
Inferential SQLi (blind: boolean-based, time-based)
Out-of-Band SQLi (authentication bypass techniques)

## In-band SQLi

Error message is thrown by the database server. When the attacker gets an error message from the database, they can try to manipulate or gain access to information or attempt further access.

## Union-based SQLi

The UNION SQL operator is used to combine two queries into one single query.

## Inferential SQLi (Blind)

The attacker can reconstruct the database structure. It takes a longer time compared to in-band SQLi.

## Out-of-Band SQLi

This occurs when the attacker cannot use the same channel to launch the attack and get the results. It relies on separate communication methods like DNS or HTTP for results.

**SQL Authentication Bypass**

Here, the attacker doesn't know the username and password but still tries to bypass authentication using SQL commands.
For example: 1=1 is always true, so it considers the condition as valid and may allow login.

**Impact**

SQL Injection affects the confidentiality and integrity of data. It can extract the complete contents of the database and compromise sensitive information.

**Error-based SQLi Practical**

In the URL, enter something like:
?id='1
If it throws an error, the application might be vulnerable.

**Boolean-based SQLi Practical**

First, check for error-based SQLi.
Try:
id=2-1 → It loads normally
id=1 and 0=1 → It may behave differently or not load

**Time-based SQLi Practical**

Try:
id=1-SLEEP (10)
If it takes a lot of time, then it is likely vulnerable.

Another example:
id=3'; WAITFOR DELAY '00:00:10

**SQLMap Usage – Practical Steps**

Install SQLMap in Kali Linux.

Then use the command:
sqlmap -u <url> --dbs --random-agent
(The --random-agent option helps bypass the firewall.)

To target a specific database:
sqlmap -u <url> -D information schema --tables --random-agent

Lastly, use:
--dump
(Do not use --dump while testing without permission.)

**Finding SQLi Using Wayback and GF Tool**

echo <website name> > target.txt
cd target.txt
cat target.txt | Wayback URLs | tee urls.txt
cat urls.txt | gf sqli

**Burp Suite and SQLMap Manual Method**

In testphp site, go to sign up and enter details.
Intercept the request using Burp Suite.
Copy the intercepted request.

**In Kali Linux:**
Create a file using the command:
nano sql.txt
Paste the copied request into the file and save it.

Run the command:
sqlmap -r sql.txt --dbs --random-agent

You can also inject using the session cookie by including the Cookie header in the request and checking for vulnerability.

---

## Directory Listing

**Objective of Directory Listing:**
When we create or build a website, it involves more than just what we see on the front-end.

There are also many important files on the back-end such as configuration files, backup files, and files containing usernames or passwords used to run admin panels or servers.

These files are usually stored in specific locations that are protected by index files (also called indexing files).

However, when these **index files are missing**, it leads to a vulnerability known as **Directory Listing** or **Open Directories**.

In the absence of index files, all the sensitive files stored in that directory (like backup files, configuration files, username/password files) become visible and accessible to anyone. If not properly authorized or protected, attackers can directly view and exploit these files.

**Note:**
Directory listing is a default **feature** in some web servers. But when it leads to **sensitive information disclosure**, it becomes a **vulnerability**.

---

## CORS (Cross-Origin Resource Sharing)

**SOP, which stands for Same Origin Policy,** is a security mechanism. It does not allow documents or scripts loaded from one origin to access resources from another origin. An origin is a combination of domain name, port number, and URL.

For example, when a user is trying to request some data from a web application, the developers may store some resources on different servers to reduce the load on a single server. So, to access these resources, the request has to move from one server to another.

If SOP is implemented in a web application, it makes sure that only the whitelisted domain or IP address is allowed to load particular resources from the server.

This prevents unauthorized access from other origins.

**CORS stands for Cross-Origin Resource Sharing.**

It is an HTTP header-based mechanism that allows a server to indicate which origins are permitted to access resources from it.

When a client requests the login page of a particular website, the server checks its database and immediately responds with the appropriate content sent by the client.

However, an attacker can craft an HTTP request using something like XMLHttpRequest. The XML can encode a malicious request and send it to the server. If the server trusts the request thinking it is genuine, it might respond with sensitive data.

If a user sends a request to a web application and the response contains the header "Access-Control-Allow-Origin" reflecting the same origin as the attacker's site, then the application might be vulnerable to a CORS attack.

There are three common scenarios of CORS misconfiguration:

1. The request is sent from origin: attacker.com
   The response contains: Access-Control-Allow-Origin: attacker.com and Access-Control-Allow-Credentials: true
2. The request is sent from origin: attacker.com
   The response contains: Access-Control-Allow-Origin: null and Access-Control-Allow-Credentials: true
3. The request is sent from origin: attacker.com
   The response contains: Access-Control-Allow-Origin: * and Access-Control-Allow-Credentials: true

In all of the above cases, if the Access-Control-Allow-Credentials is set to true and the origin is not properly restricted, it becomes vulnerable to CORS exploitation.

**To check for CORS vulnerability using the command prompt, use the following commands:**

curl "<url>/wp-json/" -I
curl "<url>/wp-json/" -I -H Origin: https://www.evil.com
curl "<url>/wp-json/" -I -H Origin: https://evilneushield.com

**You can also try in Burp Suite with Origin: neushield.com.evil.com**

As a Proof of Concept (PoC), if you get a JSON link in the response and it's a GET request, then copy that JSON link and add it into another GET request. Check if it leaks any sensitive data.

For exploitation, you can save the crafted attack code in an HTML file and run it to see if the server responds with data. If Access-Control-Allow-Origin and credentials are enabled improperly, the attack may succeed.

Metadata is also considered as a critical information. If it is revealed through such vulnerabilities, it can lead to sensitive information disclosure.

---

## Session hijacking:

Here attacker will try to steal the victim's session through the man in the middle attack or xss

If we want perform session hijacking the user session must be active

So here the attacker will use the session which was already used by the victim.

## Session fixation:

How secure application works is before user login there will be one session cookie or session id after login the session id or session cookie will be automatically changes.

It means before and after login the cookies must be different

In case if it is same then we can perform the session fixation

Attacker will fin one vulnerable website which doesn't change the id before and after login

So now the attacker will send that cookie to victim through social engineering.

So now attacker can login to the victims account without any credentials.

So, one's victim logs out attackers' session also logs out but inn rare cases the server will not invalidate or destroy the session so attacker can use that session

# What is a JWT (JSON Web Token)?

A **JWT (JSON Web Token)** is a type of token (a string of characters) used to identify the user identity and helps for secure communication.

It's used for **authentication** and **authorization** in web applications.

A JWT is typically used after a user logs in.

Once logged in, the server generates a token that proves the user is authenticated.

This token is sent to the client (browser or app) and can be used for future requests to verify the user's identity without having to log in again.

**How does a JWT work?**

1. **Header**: Contains information about how the token is signed.
2. **Payload**: Contains the user's **claims**, such as their **role** (e.g., admin, user).
3. **Signature**: Ensures that the token is **authentic** and has not been tampered with.

**What does it mean to manipulate?**

**They can alter the user role and id**

**Example of JWT manipulation:**

The attacker would get **admin privileges** even though they are not an admin.

**Why is this a problem?**

- **JWT tokens are often stored in the client-side** (like in a cookie or in local storage), and if an attacker can access and modify these tokens (using methods like **cross-site scripting** or **cookie theft**), they can **escalate their privileges**.
- The system might trust the data in the token without verifying it on the server, allowing unauthorized users to access restricted resources (like admin pages).

---

# What is CVE and why it helps?

**CVE (Common Vulnerabilities and Exposures)**

**It** is a public database that gives each known software flaw a unique ID and description.

It helps developers identify which components they use are **known to be vulnerable**

So they can fix or update them.

# What is NVD?

**NVD (National Vulnerability Database)**

**It** is maintained by the U.S. government and includes detailed security info about each CVE, like –

severity scores

impact

fix suggestions.

It helps teams **prioritize what to patch first** based on how risky the vulnerability is.

## 1)SQL injection (A3 category):

1) In Login page add SQL Payloads see it will bypass the Login page or any forms then it is vulnerable

2)You can try to login bypass in burp also (Blind sql, time based sql, Boolean based sql)

3)Using SQL map also u can try whether it is vulnerable or not

***Sqlmap run command-**

sqlmap -u "http://192.168.56.1:9090/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="security=low; PHPSESSID=abcdef1234567890" --batch –dbs

***To list all the databases-**

sqlmap -u "http://192.168.56.1:9090/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="security=low; PHPSESSID=abcdef1234567890" --batch –dbs

***To find tables inside database-**

sqlmap -u "http://192.168.56.1:9090/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="security=low; PHPSESSID=abcdef1234567890" -D dvwa --tables –batch

***List the columns in the –**

sqlmap -u "http://192.168.56.1:9090/vulnerabilities/sqli/?id=1&Submit=Submit" –cookie="security=low; PHPSESSID=abcdef1234567890" -D dvwa -T users --columns –batch

***Dump the data-**

sqlmap -u "http://192.168.56.1:9090/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="security=low; PHPSESSID=abcdef1234567890" -D dvwa -T users --dump –batch

## 2)Account Takeover by Brute Force (OTP) –(Category A1)

To test if an application is vulnerable to brute force attacks (e.g., OTP or password guessing) and if it has protections like **rate limiting**, **lockout mechanisms**, or **anti-automation defences**.

**Testing with Burp Suite Intruder:**

1. **Identify the Target Parameter:**

   Look for the field you want to brute force (e.g., **OTP**, **password**, or **username**).

   Highlight the parameter and set it using $ in Intruder.

2. **Choose the Attack Type:**

   - **Sniper**: For single payload testing (one field at a time).
   - **Cluster Bomb**: For multiple payloads sets in multiple positions (e.g., both username and password combinations).

→. **Add Payloads:**

   - Use a list of common passwords, usernames, or OTPs, You can load them manually or use Burp's built-in lists.

3. **Configure Payload Settings:** Adjust payload processing, number formatting, and payload sets as needed.
4. **Start the Attack:** Monitor **Status Code**, **Response Length**, and **Response Time** to detect successful logins or OTPs.
5. Here we have to check the length of the response because successful request has a different length**.**
6. According to the requirement we have to set the parameter ($$-if it is otp for last 2 digit set the parameter)

## 3)missing authentication:(Category A7)

- 1)While intercepting the request we will get the POST request with session ID or token then we can see the information which is used to test the authentication.
- similarly, when we remove the cookie or session id, we still see the response – Then it is an vulnerability.

## 4)unencrypted communication: (Category A2)

Runs over an unencrypted channel that is HTTP(SSL/TLS) should be used for secure communication.

## 5)forced browsing:(Category A1)

- The web application URL can be force browsed in any browsers and able to view and access the data without the authentication.
- Can use the directory listing method to check for information disclosure by forced browsing using DirBuster.

## 6)Clickjacking: (Category A5)

**Code:**

```
<!DOCTYPE html>
```

```
<html>
<head>
  <title>Clickjacking Test</title>
</head>
<body>
  <h2>If you can see the website below, it may be vulnerable to clickjacking</h2>
  <iframe src="https://example.com" width="800" height="600"></iframe>
</body>
</html>
```

→If it is vulnerable it will open in the other website directly

→Also, the X-frame security headers are not there.

## 7) Account takeover by-Response Manipulation (Category A1:)

Attempt Login with Wrong Credentials

Observe Response Structure (it will show a failed response)

Attempt Login with Correct Credentials

Go back to the original failed login request.

Click on do intercept and forward then delete the response of fake credentials and add the response of correct login.

Try injecting or replacing its **error response** with the previously saved **success response** using Burp's **Repeater** or **Proxy**.

## 8)Client-side CAPTCHA validation weakness:(Category A5:)

1)The captcha was sent over http in plain text- The captcha should not be there in plain text

2)And it is not validated in the server side (even though we add wrong captcha it will accept it)

## 9)Prototype Pollution:(Category A5)

- It's a security flaw in JavaScript where an attacker adds or changes properties on the base object called Object.prototype.
- JavaScript objects inherit properties from Object.prototype. If an attacker changes Object.prototype, all objects get those changes.
- Add the payload in console and watch whether it will allow or not.

**Because every object inherits these polluted properties, attackers can:**

- Break application logic
- Bypass security checks

- Cause unexpected bugs or crashes

## 10)Http Method Override (Category A5):

- Improper handling of HTTP request methods
- Which was a potential misconfiguration in application.
- After getting the POST request and response in that try to change the request method and view response whether it still gives 200Ok or not if so, it is a vulnerability.

## 11) Anti-Automation & Rate Limiting Testing (A5 category):

- Application allows to send multiple requests in a short time period and server receives the multiple requests.
- From this DOS, Bruteforce, bypass of credentials, business logic error can happen.

When we don't want to set any parameter we can choose the null payload for

**Rate Limiting Testing:**

1. **Test Without Rate Limits:**
    i. Set high request numbers (e.g., 100+) in Intruder.
    ii. If the application continues to respond normally, **rate limiting may not be implemented**. (it should limit the request).
2. **Anti Automation-** All methods combined to stop bots and automated attacks.
   **(Like CAPTCHAs, behavior checks, JavaScript tests, and rate limiting.)**
3. **Indicators of No Protection**
    a. Repeated OTP or password attempts without being blocked.
    b. New accounts or logins created without delay.
    c. No CAPTCHA or lockout mechanisms.

## 12)Session Fixation (Category A7)

Before and **after** login if the cookies are same then session fixation happens.

- Web application is there -before login the session will not be created but cookies will be created, and if cookies are same after login also session fixation happens.

**Attacker-Victim**

- One vulnerable we application is there where the cookies are same before and after login.
- Attacker doesn't login but he will send his cookie (which was created) he will send it to user or victim through phishing or any other methods.
- User will open the link and login but the application won't regenerate the new cookies after login but it uses the cookies which was created by attacker.
- Attacker will login using that same cookie in his browser as an attacker.

When you log into a website, the server needs a way to remember who you are on each request.

This is done using a session — usually identified by a session ID (like a long string of letters and numbers). This session ID is often stored in a cookie or URL.

**Session Fixation** happens when an attacker sets or predicts a session ID and then gets the victim to use it.

After the victim logs in using that session, the attacker can hijack the session — because both are using the same session ID.

**Prevention**:

- After login new session should be created
- samesite cookie should be strict
- http only should be enabled

**Practical:**

In cookie editor first see the session id before login

Then change the session id

Again, login now and check whether it changes or not if it does not change it is vulnerable

## 13)IDOR (Insecure Direct Object Reference) (Category A1)

User gets indirect access to the other user's data without any authorization but just by manipulating or tampering the URL id or any parameter. (Here they get access to other user's data).

## 14)Session not invalidated after logout:(Category A7)

- First login to the application and then capture the request
- Then do intercept and forward the request also send it to repeater
- Logout from the account
- Then change the any id in request then click send
- Then again login and watch whether it will allow the changes or not

## 15)Microsoft IIS TILDE Directory Enumeration:(Category A5)

It successfully retrieves the folder and sensitive files names which increases the risk of exploitation.

If it returns **404 Not Found** → the folder does **not exist**

If it returns **403 Forbidden** or **401 Unauthorized** → the folder **does exist**

## 16)Weak Input validation:

Weak input validation there are no proper validation checks on input fields for which malicious character are being accepted by application

- Weak input validation means inputs aren't properly checked for bad characters.
- This allows attacks like SQL injection or XSS.
- Use Burp Suite to intercept requests to a test site.
- Modify inputs with malicious payloads (e.g., admin' OR '1'='1).

- Forward the request and see if the app accepts or blocks it.

## 17)Missing Security Headers:

The security headers are not implemented properly

Missing security headers means a “website’s HTTP responses do not include important security-related headers that help protect users and the site from attacks”.

- **Content-Security-Policy (CSP):**
  This header tells the browser exactly which sources (websites or scripts) are allowed to load on the page. It helps stop attacks like Cross-Site Scripting (XSS) by blocking any malicious scripts that aren’t from trusted sources.

---

- **X-Content-Type-Options:**
  When this header is set to nosniff, it prevents browsers from guessing the type of a file (like treating a file as a script when it isn’t). This helps stop some attacks where browsers try to execute dangerous files in the wrong way.

---

- **X-Frame-Options:**
  This header prevents the website from being embedded inside a frame or iframe on another site. It protects users from clickjacking attacks, where attackers trick users into clicking hidden buttons by overlaying the site inside a frame.

---

- **Strict-Transport-Security (HSTS):**
  This header forces browsers to only connect to the site over HTTPS (secure connection) and never HTTP. It protects against attacks that try to intercept data by downgrading the connection to an insecure one.
- **Referrer-Policy:**
  This header controls how much information about the referring page (the page you came from) is sent to other websites. It helps protect user privacy by limiting sensitive data leakage.

---

- **Permissions-Policy:**
  It limits what features (like camera, microphone, geolocation) a website can access in the user’s browser. This helps reduce risks from malicious sites trying to misuse device features.

---

- **X-XSS-Protection:**
  This older header enabled the browser’s built-in XSS filter to block some simple attacks. However, most modern browsers rely more on CSP now because it is much stronger and flexible.

## 18)Vulnerable software version used:(Category A6)

- The application is running on outdated or unpatched software that has known security flaws.
- Attackers can exploit these weaknesses to gain unauthorized access or cause damage.
- Keeping software updated helps protect against such vulnerabilities.

## 19)Verbose Server Banner Disclosure:(Category A5)

The application is disclosing server version in the response headers.

**→namp -Pn -sV <Ip address>**

## 20)Concurrent user session logons:(Category A7)

Concurrent session logons" refers to multiple sessions being active at the same time using the same user account credentials.

What happens?

Detect session hijacking.

Mitigate account abuse or fraud.

Try to login from different accounts. If it allows login from different accounts then it is vulnerable.

# 21)Cross Site Scripting:(Category A3)

It is also one type of attack where the application doesn't validate or sanitize properly the user input instead it accepts the all the user input.

- Reflected XSS
- Stored XSS
- DOM based XSS

## 1)Reflected XSS:

Basically, it happens in server side and effects the client side.

- **Example**: Think there is one application of shopping there will be one search input field when we search or add any input it will reflect from backend. (check it and it will reflect from backend).
- Authenticated user- trying to access one any application but attacker wants to steal the session of a authenticated user
- The attacker only knows that the application is vulnerable to xss.
- The attacker inserts the payloads in search field which will be there in the backend.
- It was a onetime function.
- When we search it only that time it will reflect
- The attacker inserts payloads according to his needs for example stealing session id and also redirecting it (attacker will add other malicious links) etc…
- When authenticated user searches or do anything in input fields the attacker can get information's.

**Where and how it happens:**

1. **Search Fields:**
   Reflected XSS can happen in search boxes when the website displays your input on the page without filtering.
   Example: If you search for <script>alert (1) </script>, and the website shows that script in the results, it will run.
2. **URL Parameters:**
   When websites use values from the URL (like? name=value) and show them in the response.
   Example: https://example.com/welcome?name=<script>alert(1)</script> — if the site shows this input, the script will execute.
3. **Form Inputs (Login, Contact, Feedback):**
   If user input from a form is reflected in the response without sanitizing, it can trigger XSS.
   Example: Submitting <script>alert (1) </script> in a feedback form and seeing it appear on the thank-you page.
4. **Error Messages:**
   If an application shows user input in an error message, and it includes script tags, XSS can occur.
   Example: https://site.com/user?id=<script>alert(1)</script> shows: User <script>alert(1)</script> not found.
5. **HTTP Headers (User-Agent, Referrer):**
   Some websites reflect data from browser headers. If not filtered, malicious scripts in headers can execute.
   Example: If User-Agent: <script>alert(1)</script> is shown on a page, it can cause XSS.
6. **Query String (GET Parameters):**
   If GET request parameters (like ?id=123) are shown directly in the page, and scripts are not sanitized, XSS can happen.
   Example: https://site.com/page?id=<script>alert(1)</script>
7. **Login Redirects:**
   After login, if a redirect value is passed in the URL and reflected on the page, XSS can occur.
   Example: https://example.com/login?redirect=<script>alert(1)</script> — if the site shows this value, the script can run.

## 2)STORED XSS:

Example: In one web application think there was one feedback form. Then whatever the user adds in feedback it will store in the backend of the server.

The attacker will insert the stored xss payloads in the forms so this will be stored in the server

So, whenever the user click that stored forms the attacker will get the sensitive information's for example debit or credit card information.

It will not only for a single user because it was already stored permanently in the form so the attacker gets information of any user.

1. **Comment Sections:**
   An attacker adds a comment like <script>alert('XSS')</script>.
   If the website does not sanitize it, all users who view the comment will run the script.

2. **Feedback or Contact Forms**:
   If user feedback is saved and shown to admins or other users, a script inside the feedback can execute later.
   Example: Attacker submits feedback with <script>stealCookies()</script> → runs when admin reads it.

3. **User Profile Fields:**
   If users can update their name, bio, etc., and the app shows this data without cleaning it, a script can

run.
Example: A bio field with <script>alert('XSS')</script> affects anyone who views that profile.

4. **Forum Posts or Messages:**
   Attackers insert a malicious script in a forum post or private message. Other users who open it will unknowingly trigger the attack.

5. **Product Reviews:**
   Malicious scripts in reviews can steal data from other users visiting that product page.

Stored XSS affects every user who visits the infected page because the script is saved and sent from the server.

## 3)DOM based XSS (It happens in client side)

DOM-DOCUMENT OBJECT MODEL

It was one of tree structure of the JavaScript.

The user is using one website so he wants to change the colour or changing size or font of that so he uses DOM.

So, it was one of the easiest ways because if not in source code they have to change it which is very difficult.

Usually, it should modify in the user page only if it is not properly checked for example if DOM accepts every input which is given by the user DOM based xss happens.