
WEEK 2 – DAY 12

CROSS-SITE REQUEST FORGERY (CSRF)

---

1. WHAT IS CSRF? (CORE IDEA)

Simple Definition

CSRF is an attack where a **logged-in user is tricked into performing an unwanted action** on a trusted website **without their knowledge**.

Important:

- Victim is **authenticated**
- Attack abuses **browser trust**
- Server cannot distinguish forged request from real one

---

2. WHY CSRF WORKS (MENTAL MODEL)

Browsers:

- Automatically attach cookies
- Do not know user intent

Server:

- Sees valid session cookie
- Assumes request is legitimate

Result:

Action succeeds without user consent

---

3. CSRF ATTACK FLOW (STEP-BY-STEP)

Real-World Example: Change Email

1. User logs into bank.com
2. Session cookie is stored in browser
3. User visits attacker site evil.com
4. Hidden request is triggered:
5. <img src="https://bank.com/change-email?email=attacker@evil.com">
6. Browser sends request **with bank.com cookies**

7. Server processes request
8. Account is compromised

---

4. WHAT CSRF CAN DO

- Change password
- Transfer money
- Modify email
- Delete account

CSRF **cannot read responses**, but can **perform actions**.

---

5. WHEN CSRF IS POSSIBLE

CSRF exists if:

- Authentication uses cookies
- State-changing actions exist
- No CSRF protection is implemented

---

6. CSRF TOKENS (PRIMARY DEFENSE)

What Is a CSRF Token?

A random, unpredictable value generated by the server and verified for each sensitive request.

Key idea:

- Attacker **cannot guess or read** token
- Only legitimate forms contain valid token

CSRF-Protected Form

```
<form method="POST" action="/update">
  <input type="hidden" name="csrf_token" value="random_value">
  <button>Submit</button>
</form>
```

Server verifies:

- Token exists
- Token matches session

---

7. IMPLEMENTING CSRF PROTECTION (FLASK)

Using Flask-WTF

```
from flask_wtf import FlaskForm
from wtforms import StringField
from flask_wtf.csrf import CSRFProtect


csrf = CSRFProtect(app)


class UpdateForm(FlaskForm):
   email = StringField("Email")
```

Automatically:

- Generates token
- Validates token
- Rejects forged requests

---

8. SAME-SITE COOKIES (MODERN DEFENSE)

What Is SameSite?

Cookie attribute that controls whether cookies are sent on cross-site requests.

SameSite Options

Value	Behavior
Strict	Never sent cross-site
Lax	Sent on top-level navigation
None	Sent everywhere (requires HTTPS)

Secure Cookie Example

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

Why SameSite Helps

- Browser blocks cookies on forged requests
- Reduces CSRF attack surface

---

9. HANDS-ON: SIMULATING CSRF

Step 1: Victim App (Vulnerable)

```
@app.route("/transfer", methods=["POST"])
@login_required
def transfer():
    amount = request.form["amount"]
    process(amount)
```

Step 2: Attacker Page

```
<form action="https://victim.com/transfer" method="POST">
  <input type="hidden" name="amount" value="10000">
  <input type="submit">
</form>
<script>document.forms[0].submit()</script>
```

Victim visits page → money transferred.

---

10. FIXING CSRF CORRECTLY

Defense-in-Depth

- CSRF tokens
- SameSite cookies
- POST instead of GET
- Confirmations for critical actions

---

11. INTERVIEW QUESTIONS & STRONG ANSWERS

Q1: What is CSRF?

“CSRF is an attack where a victim’s authenticated browser is tricked into making unauthorized requests.”

Q2: Why does CSRF not work with JWT in headers?

“Because tokens in headers are not automatically sent by browsers.”

Q3: Is SameSite enough?

“No. It reduces risk but does not replace CSRF tokens.”

Q4: Difference between XSS and CSRF?

“XSS executes attacker code in victim’s browser; CSRF tricks browser into sending forged requests.”

---

12. ATTACKER VS DEFENDER THINKING

Attacker:

- Which actions change state?
- Are cookies sent automatically?
- Is token validated?

Defender:

- Is every state change protected?
- Are cookies properly configured?
- Are tokens unpredictable?