**CSRF and Hackers:**

Browser storage is bad

Session or Local storage is good

**1️⃣ Does Local Storage Belong to Every Website?**

Yes! **Local storage is separate for each website** (origin). This means:

* localStorage for https://mybank.com **CANNOT be accessed** by https://evil.com.
* This is enforced by the **Same-Origin Policy**.

📌 **Example: Isolation of Local Storage**

// On mybank.com

localStorage.setItem("jwtToken", "abc123");

Now, if a hacker runs JavaScript on **evil.com**, they **cannot** access this token:

console.log(localStorage.getItem("jwtToken")); // ❌ null (Different origin)

✅ **This makes JWTs safe from CSRF!**

**2️⃣ How Do Hackers Get the Session ID in a Cookie?**

Cookies are **automatically sent** with every request. This is where hackers exploit **CSRF (Cross-Site Request Forgery)**.

**🔥 How CSRF Works (Step-by-Step Attack Example)**

1️⃣ You **log into** https://mybank.com.

* The bank stores your session ID in a **cookie** (sent automatically).

Cookie: sessionId=xyz123

2️⃣ You visit a **malicious site** (https://evil.com) in another tab.

* The hacker’s website **contains a hidden form** that sends a request to your bank.

3️⃣ **Your browser automatically includes the session cookie!**

* Because **cookies are sent automatically**, the bank thinks it’s a real request.

📌 **Example: Malicious Request Sent Automatically**

<form action="https://mybank.com/transfer" method="POST">

<input type="hidden" name="toAccount" value="hackerAccount">

<input type="hidden" name="amount" value="1000">

<input type="submit">

</form>

<script>

document.forms[0].submit(); // Auto-submits form

</script>

🚨 **Result:** The bank sees this request:

POST https://mybank.com/transfer

Cookie: sessionId=xyz123 ✅ (Sent automatically)

Body:

toAccount=hackerAccount

amount=1000

✅ **The bank processes the request because it sees your session ID (cookie) and thinks it's you!**

**3️⃣ How Do We Prevent This?**

Since CSRF **only works because cookies are auto-sent**, there are **two main ways to stop it**:

✅ **Solution 1: Use CSRF Tokens (For Cookies-Based Auth)**

* The server generates a **CSRF token** and sends it to the client.
* The client must include this token in every request (as an extra field).
* Since a hacker **does not know the CSRF token**, their forged request fails.

✅ **Solution 2: Use JWT in Local Storage (No Automatic Sending)**

* If authentication is done with JWT **stored in localStorage**, **the hacker's request does NOT include it**.
* Since https://evil.com **cannot access mybank.com’s localStorage**, they cannot steal the JWT.
* The bank **rejects the request** because it **does not contain a valid token**.

**🔑 Summary**

| **Feature** | **Cookies (Session-Based)** | **JWT (LocalStorage-Based)** |
| --- | --- | --- |
| **Sent automatically?** | ✅ Yes | ❌ No |
| **Vulnerable to CSRF?** | ✅ Yes | ❌ No |
| **Needs CSRF protection?** | ✅ Yes | ❌ No |
| **Can be stolen by a hacker’s website?** | ✅ Yes (via CSRF) | ❌ No (Same-Origin Policy) |

✅ **That’s why JWTs (in localStorage) are safe from CSRF attacks!**

Let me know if you need more clarity! 🚀