# CORS Misconfiguration

when a server **improperly allows cross-origin requests**, enabling unauthorized third-party domains to access protected APIs or user data (like tokens, cookies, and PII).  
Improper CORS policy allows untrusted sites to read protected data.

# Key CORS Headers

| Header | Purpose |
| --- | --- |
| Access-Control-Allow-Origin | Specifies which origins are allowed |
| Access-Control-Allow-Credentials | Indicates if cookies/auth headers can be sent |
| Access-Control-Allow-Methods | Lists allowed HTTP methods (e.g., GET, POST) |
| Access-Control-Allow-Headers | Lists allowed headers (e.g., Authorization) |

# Example:

Access-Control-Allow-Origin: \*  
Access-Control-Allow-Credentials: true  
Wildcard Access-Control-Allow-Origin: \*  
This combo is **dangerous** because it:

* Allows all origins (\*)
* Allows sensitive credentials (cookies) to be sent

**Result**: Any malicious site can steal session data from authenticated users.

# Impact

| Area | Risk |
| --- | --- |
| Sensitive Data Exposure | Attackers can access user data (JSON, cookies) from victim sessions |
| API Abuse | Attacker-controlled sites can interact with protected internal APIs |
| Session Hijacking | Cookies sent with requests if withCredentials=true is enabled |
| Compliance Risk | Violates data protection standards (e.g., GDPR, HIPAA) |

# Types of CORS Misconfiguration

**1. Wildcard Origin with Credentials**

* **Definition**: Access-Control-Allow-Origin: \* + Allow-Credentials: true
* **Impact**: Any site can send credentialed requests (severe)
* **Fix**: Never use \* when credentials are allowed

**2. Reflection-Based Origin Whitelisting**

* **Definition**: Server reflects Origin header in Access-Control-Allow-Origin
* **Example**:   
  Origin: evil.com   
  → Access-Control-Allow-Origin: evil.com
* **Fix**: Only allow known, validated domains

**3. Unrestricted Subdomain Wildcard**

* **Definition**: Using \*.domain.com in CORS config without control
* **Risk**: Any subdomain takeover = full access
* **Fix**: Maintain strict subdomain allowlist

**4. Method/Headers Over-Exposed**

* **Definition**: Allowing unsafe methods or headers
* **Example**: Access-Control-Allow-Methods: GET, POST, PUT, DELETE
* **Fix**: Use only necessary methods for API functionality

# Exploitation Example (PoC)

1. Victim is logged into:  
   https://bank.com/api/account
2. Malicious site (evil.com) runs:

fetch("https://bank.com/api/account", {

method: "GET",

credentials: "include"

}).then(res => res.text())

.then(data => sendToAttacker(data));

If the bank has CORS misconfigured, the attack succeeds, leaking sensitive account data to evil.com.

# Testing Techniques

| Method | Tool | Description |
| --- | --- | --- |
| Manual Origin Injection | Burp Suite / Postman | Add Origin: evil.com, observe response |
| Test with curl | Simple testing |  |

curl -H "Origin: evil.com" <https://target.com/api>

| Chrome DevTools | Inspect CORS behavior |  
| CORS Scanner | Burp Extensions / CORS Misconfig tools |  
| JS Payloads | Create PoC HTML pages to test real browser behavior |

# Mitigation:

| Strategy | Description |
| --- | --- |
| Strict Allow-Origin Policy | Only allow specific trusted domains |
| Never Use Wildcard (\*) with Credentials | Violates CORS spec and leads to RCE |
| CORS Preflight Validation | Validate origin, headers, methods in OPTIONS request |
| Use Server-Side Whitelist Logic | Check Origin against a fixed allowlist server-side |
| Secure Subdomains | Avoid CNAME takeover or wildcard logic |
| Use Short-Lived Tokens Instead of Cookies | Safer against browser-based leakage |

# Real-World Breaches

| Company | Year | Description |
| --- | --- | --- |
| Facebook | 2018 | Reflected Origin header → CORS bypass |
| Uber | 2016 | CORS misconfig + token abuse exposed user info |
| Google Apps (Bug Bounty) | Multiple CORS bugs in internal tools |  |

# Points

“CORS is not a vulnerability by itself — it becomes one when **misconfigured with wildcard origins, credentials, and reflective headers**.”

“In audits, simulate **browser-like requests** to test how the app responds to unauthorized origins.”

“For high-risk APIs, Recommend **strict origin checks, credential separation, and using OAuth tokens over cookies**.”

# CORS Secure Configuration Template

Access-Control-Allow-Origin: https://yourapp.com

Access-Control-Allow-Credentials: true

Access-Control-Allow-Methods: GET, POST

Access-Control-Allow-Headers: Content-Type, Authorization