Cheatography

Introduction to XSS Cheat Sheet

by binca via cheatography.com/44948/cs/13429/

Summary

The second most well-known web app flaw next to SQLi.

OWASP: "Most prevalent flaw in web application today"

More difficult to prevent than SQLi, RFI/LFI and Command Injection.

Likely victim is the end user not the vulnerable app resulting in less incentive to address the flaws.

HTML Injection

Provide maliciously crafted input that results in attacker-controlled HTML within the server's response.

The primary goal is arbitrary HTML rendering by the victim browser.

The lack of sanitization is the flaw but the victim is the rendering browser.

Script Injection

XSS often feels like manifestation of HTML injection with JavaScript able to be injected.

<script>alert("Hello!")</script>

Origin of Trust

Origin Server ➔ Same-Origin Policy (SOP)

Basic and critical security component of the web. THe goal is to make us feel safe.

SOP is only meant to prevent code from one site accessing content sent from another origin-server.

Requirements:
Port: Important when using tools to target internal apps that use nonstandard ports
Scheme/Protocol: HTTP/HTTPS most common difference encountered
Host: The main source of variance and the focus of SOP restrictions

Same-Origin determinations and SOP restrictions are governed by the browser.

A browser will allow an externally sourced script because it sees the origin-server as requesting we fetch the externally sourced script.

Can circumvent space limitations by externally sourcing scripts. This is not bypassing SOP but presenting scripts as if they came from the origin server.

Goal isn't just to get code execution but to get the content as coming from a particular origin server.

Popups and Alternatives

For initial testing and discovery of XSS simple popups work as POC. Not enough to demonstrate the risks of XSS.

Classic: alert("hello");

Classic look more style: confirm("hello");

Alternative: prompt("Pick a number from 1-10", "5");

Classes of XSS

1. **Reflected** (non-persistent, type 2)

2. **Stored** (persistent, type 1)

3. **DOM based** (type 0) iis newer

Self-XSS involves scammers tricking victims into copying commands into the address bar that leads to adversary controlled JavaScript execution

Universal-XSS is typically not a web app flaw but instea a method of injecting JavaScript by means of exploiting a separate tool, the most common targets being browser and associated plugins.

The goal is achieving JavaScript execution in the browser. The manner in which this is achieved differs by class.

Reflected XSS (Non-persistent)

Easiest to understand and simplest to discover

Most commonly used example of XSS.

Example:**<script>alert("Hello!");</script>**

Input is dynamically added to the HTML without any encoding.

Payloads are immediately delivered to the victim and will not persist for either the victim or other users.

URL Encoding (Percent Encoding)

Can be used for filter bypass but must be mindful of proper encoding

Burp Decoder will encode all characters even URLSafe characters, which can prove useful.

ZAP will encode characters that the tool considers to be unsafe.

Firefox Web Console is located inside the developer tools and will encode a parameter with the function **encodeURIComponent()** or if passed an entire URL **encodeURI()**.

Each technique produces a unique encoded output but all function the same.

Stored XSS (Persistent)

The other major type of XSS typically relevant to application assessments.

The adversary's input will persist across additional interactions with the site.

More difficult to discover and not reliant on social engineering.

Common app functions with increased likelihood include **blog comments, forum data, message functionality, log mechanisms, account profiles, support functionality**.

Any aspect of the app that facilitates communicating information to users or admins is important.

Out-of-Band Stored XSS

Indirectly supplying input that results in JavaScript executing within a web app. Interaction s not always possible.

Apps with more obvious OOB Stored XSS potential include **web--based email clients and security device consoles (IDS, SIEM, Firewalls)**

Any app where OOB information we control has a high likelihood of being rendered.

DOM Based XSS (Type 0)

Exploit still results in JavaScript execution.

Primary victim is the client.

Methods of discovery and exploitation similar to **Reflected XSS** because it most likely involves a dynamic request, non-persistent response, and social engineering.

Distinguishing feature is server does not deliver the attack to the client. Instead the client-side of the application allows for malicious JavaScript execution.

Server versus Client XSS

There can be non-persistent and persistent version of traditional and DOM based XSS.