Cross-Site Scripting

aus Wikipedia, der freien Enzyklopädie

Dieser Artikel oder Abschnitt bedarf einer Überarbeitung. Näheres ist auf der Diskussionsseite angegeben. Hilf bitte mit, ihn zu verbessern, und entferne anschließend diese Markierung.

Cross-Site Scripting (**XSS**) bezeichnet das Ausnutzen einer Computersicherheitslücke in Webanwendungen, indem Informationen aus einem Kontext, in dem sie nicht vertrauenswürdig sind, in einen anderen Kontext eingefügt werden, in dem sie als vertrauenswürdig eingestuft sind. Aus diesem vertrauenswürdigen Kontext kann dann ein Angriff gestartet werden.

Ziel ist es meist, an sensible Daten des Benutzers wie Cookies zu gelangen, um beispielsweise Benutzerkonten zu übernehmen (Identitätsdiebstahl).

Inhaltsverzeichnis

Terminologie

Die Bezeichnung „Cross-Site“ leitet sich von der Art ab, wie dieser Angriff Website-übergreifend ausgeführt wird.

Cross-Site Scripting wird manchmal auch „CSS“ abgekürzt, hat jedoch nichts mit der Cascading-Style-Sheet-Technologie zu tun, die weit häufiger CSS genannt wird. Um Verwechslungen zu vermeiden, sollte daher die Abkürzung XSS benutzt werden. (X steht im IT-Bereich des öfteren für ein Kreuz = Cross)

Funktionsweise

Beim *Cross-Site Scripting* wird Schadcode auf Seiten des Clients ausgeführt, etwa dem Webbrowser oder E-Mail-Programm. Dazu schiebt der Angreifer dem Opfer ein von ihm präpariertes HTML-Dokument unter, das beispielsweise per E-Mail verschickt wird. Oder der Angreifer lässt dem Opfer einen Hyperlink zukommen, der auf eine vom Angreifer präparierte Webseite weist oder selbst den Schadcode enthält. Häufig werden dazu auch URL-Spoofing-Techniken und andere Kodierungsverfahren eingesetzt, um den Link unauffällig oder vertrauenswürdig erscheinen zu lassen.

Ein klassisches Beispiel für *Cross-Site Scripting* ist die Übergabe von Parametern an ein CGI-Skript einer Website. So ist es unter Umständen möglich, manipulierte Daten an den Benutzer zu senden. Diese Daten sind oft Code einer clientseitigen Skriptsprache (meist JavaScript), die als Parameter an eine Website übergeben werden.

Gefährlich wird dies, wenn die Webseite, auf der der Schadcode untergebracht wurde, im lokalen Browser mit besonderen Sicherheitsrechten (Privilegien) ausgestattet ist. Der Schadcode kann dann in Abhängigkeit von der Mächtigkeit der Skriptsprache verschiedene Dinge tun, die mit den Rechten des lokalen Benutzers möglich sind.

Da aus Bequemlichkeitsgründen auf Microsoft-Windows-Systemen der lokale Benutzer häufig mit Administrator-Rechten ausgestattet ist, ist dies bereits eine potenziell sehr gefährliche Konstellation. Aber auch ohne Administrator-Rechte kann der Angreifer versuchen, durch Ausnutzung von Sicherheitslücken bei der Ausführung der betreffenden Skriptsprache diese Rechte zu erlangen.

Angriffsarten

Es gibt drei grundlegende Arten von Cross-Site-Scripting-Angriffen. (Diese werden in diesem Dokument allein der Einfachheit halber mit *Typ 0*, *Typ 1* und *Typ 2* bezeichnet.)

In den Beispielen wird zur Anschauung der einfache JavaScript-Code

```
alert("XSS")
```

verwendet, der mithilfe der `script`-Tags in ein HTML-Dokument eingebunden wird:

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

Dieser JavaScript-Code beschreibt zwar nur einen harmlosen Warnhinweis-Dialog mit dem Text „XSS“. Doch über dieses Schema kann auch jeder andere JavaScript-Code eingeschleust werden.

Typ 0

Diese Art der Sicherheitslücke wird *DOM-basiertes* oder *lokales* Cross-Site Scripting bezeichnet. Hierbei existiert die Sicherheitslücke in einem clientseitigen Skriptcode selbst. Dies kann beispielsweise ein JavaScript-Code sein, der einen URL-Argumentwert zur Ausgabe von HTML verwendet ohne diesen ausreichend zu prüfen. Ein solcher Angriff bedarf des gezielten Aufrufs einer kompromittierten URL.

Neuerdings werden auch Webspider missbraucht, um XSS- und SQL-Injection-Attacken auszuführen. Hierzu wird ein präparierter Link auf einer Webseite veröffentlicht. Sobald ein Webspider diesem Link folgt, löst er die Attacke aus. Dadurch taucht die IP-Adresse des Spiders und nicht die des eigentlichen Angreifers in den Protokollen des angegriffenen Systems auf. Der Angreifer kann somit anonym agieren.

Beispiel

Eine clientseitige Skriptsprache wie etwa JavaScript liest ein Argumentwert aus URL mit folgendem Schema aus:

```
http://example.com/foobar.html?arg=Argumentwert
```

Und fügt diesen nach folgendem Schema ungeprüft in das HTML-Dokument ein:

```
<p>Sie haben an die URL diesen String angehängt: Argumentwert<p>
```

Wird nun die aufrufende URL folgendermaßen manipuliert:

```
http://example.com/foobar.html?arg=<script>alert("XSS")</script>
```

würde der durch das serverseitige Skript erzeugte HTML-Code wie folgt aussehen:

```
<p>Sie haben an die URL diesen String angehängt: <script>alert("XSS")</script><p>
```

Somit würde obiges Skript im Kontext der aufrufenden Seite ausgeführt werden.

Typ 1

Diese Art wird als *nicht-persistentes* oder *reflektives* Cross-Site Scripting bezeichnet. Anders als beim Typ 0 liegt das anfällige Programm hier auf dem Server; ansonsten ist das Funktionsprinzip sehr ähnlich.

Hierbei wird der Umstand ausgenutzt, dass dynamische generierte Webseiten ihren Inhalt oft über URL (HTTP-GET-Methode) oder Formulare (HTTP-GET- oder HTTP-POST-Methode) übergebene Eingabewerte anpassen. *Nicht-persistent* heißt dieser Typ, da der Schadcode nur temporär bei der einzelnen Generierung der Webseite eingeschleust wird. Ruft man die Seite danach ohne die manipulierte URL oder das manipulierte Formular auf, ist der Schadcode nicht mehr enthalten.

Beispiel

Eine Suchfunktion soll das Durchsuchen sämtlicher Dokumente einer Website ermöglichen. Dabei wird auf der Ergebnisseite der Suchbegriff nochmals gesondert angezeigt. Der Suchfunktion kann der Suchbegriff entweder per URL-Argument oder über ein Formular übergeben werden, was in Anfragen nach folgendem Schema resultiert:

```
http://example.com/?suche=Suchbegriff
```

Die übergebenen Suchbegriffe werden nun von einer serverseitigen Skript- oder Programmiersprache ungeprüft auf der Ergebnisseite wieder ausgegeben:

```
<p>Sie suchten nach: Suchbegriff</p>
```

Würde nun hier, wie im Beispiel von Typ 0, folgender Suchbegriff verwendet:

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

würde dann folgender HTML-Code erzeugt:

```
<p>Sie suchten nach: <script>alert("XSS")</script></p>
```

Nach dem gleichen Prinzip kann auch über manipulierte Formulare oder Formulareingaben Schadcode eingeschleust werden. Dies ist schwieriger, da hierbei das Opfer zuerst auf eine Webseite mit manipuliertem Formular gelockt werden muss, das das Opfer dann auch nutzen muss.

Typ 2

Diese Art wird als *persistentes* Cross-Site Scripting bezeichnet. Hierbei wird der Schadcode auf dem Webserver gespeichert, wodurch er bei jeder Anfrage ausgeliefert wird. Dies ist theoretisch bei jeder Webanwendung möglich, die Benutzereingaben serverseitig speichert und diese später wieder ausliefert.

Beispiel

Eine Webseite bietet ein Gästebuch, in dem Besucher Nachrichten oder Grüße hinterlassen können. Die eingetragenen Daten werden serverseitig in einer Datenbank gespeichert und bei jedem Aufruf wieder ausgegeben.

Wird nun hier als Nachricht Folgendes eingegeben:

```
Eine wirklich sehr informative Website!<script>alert("XSS")</script>
```

Würde diese bei jedem Aufruf ausgeliefert.

Formen des Cross-Site Scripting

Es gibt viele Variationen, mit denen versucht wird, sensible Daten des Benutzers zu erlangen. Darunter beispielsweise:

- Cross-Site Authentication

- Cross-Site Cooking
- Cross-Site Tracing

Cross-Site Request Forgery (CSRF) ist *keine* Form von Cross-Site-Scripting.

Schutz

Schutzmaßnahmen für Webseitenbetreiber

Um einer Webanwendung keine Basis für XSS-Angriff zu bieten, müssen alle eingehenden Eingabewerte als unsicher betrachtet und vor der weiteren Verarbeitung auf der Serverseite geprüft werden. Dabei sollte man sich nicht darauf verlassen, „böse“ Eingaben zu definieren (Schwarze Liste), um diese herauszufiltern, da man nicht genau wissen kann, welche Angriffsmethoden[1] es gibt. Besser ist es daher, die „guten“ Eingaben exakt zu definieren (Weiße Liste) und nur solche Eingaben zuzulassen.

Hier treffen die Zitate eines Microsoft-Mitarbeiters zu, der ein Buch *Never trust the client* und sein nächstes bereits *All incoming data is EVIL* nannte. Jegliche Daten, die einmal beim Client, d.h. beim Besucher der Website, waren, sind demnach potenziell verseucht.

Als Schutz bei einer HTML-Ausgabe (Typ 1, Typ2, teilweise Typ 0) ist es nötig, die HTML-Metazeichen (insbesondere „<“, „>“, „&“, sowie Anführungszeichen „"“, „'“ in Attributwerten) durch entsprechende Zeichenreferenzen zu ersetzen, damit sie als normale Zeichen und nicht als Metazeichen behandelt werden. Erfolgt die Ausgabe in URLs („src“- und „href“-Attribut), dann muß URL-Kodierung für die Zeichen verwendet werden. Bei der Ausgabe in JavaScript-Code müssen die Zeichen mit „\“ "escaped" werden. Dabei ist zu beachten, dass an diesen 3 Stellen (HTML, URL, Skript) unterschiedliche Zeichen eine Metafunktion haben (z. B. ist „\“ für HTML kein Metazeichen), es muss also immer eine an das Ausgabemedium angepasste Kodierung verwendet werden.

Die meisten Prgrammier- sowie Skriptsprachen verfügen über vordefinierte Methoden zur Ersetzung der Metazeichen durch normale Zeichen.

Beispiel in PHP:

```php
<?php
  $boesercode = "<script>alert('XSS');</script>";
  echo "Sie haben ".htmlentities($boesercode,ENT_QUOTES)." eingegeben";

  //Ausgabe: Sie haben &lt;script&gt;alert(&#039;XSS&#039;);&lt;/script&gt; eingegeben
?>
```

Beispiel in Perl:

```perl
#!/usr/bin/perl
use HTML::Entities;

my $boesercode = "<script>alert('XSS');</script>";
print "Sie haben " . HTML::Entities::encode($boesercode) . " eingegeben";

#Ausgabe: Sie haben &lt;script&gt;alert(&#39;XSS&#39;);&lt;/script&gt; eingegeben
```

Zusätzlich können durch Einsatz von Web Application Firewalls (WAF) zumindest in der Theorie einfache (primitive) XSS-Attacken verhindert werden. Praktisch sind sichere Anwendungen jeder WAF vorzuziehen.

Schutzmaßnahmen für Webseitennutzer

Durch Ausschalten der JavaScript-Unterstützung (Active Scripting) im Browser kann man sich gegen clientseitige XSS-Angriffe schützen. Allerdings bieten einige Browser weitere Angriffsvektoren. Dies gilt allerdings nur für "echtes" XSS, also solches, welches mit JavaScript arbeitet. Wenn nur HTML-Injection (z. B. mit <IFRAME .../>) verwendet wird, dann hilft abschalten von Scripting im Browser nicht. Gleiches gilt wenn XSS in Styles (CSS) verwendet wird.

Für einige Browser gibt es auch Erweiterungen, mit denen gezielt nur mögliche XSS-Angriffe erkannt und verhindert werden.

Siehe auch

- SQL-Injektion
- Header-Injection
- Computersicherheit
- Webanwendung

Quellen

1. ↑ ha.ckers.org: XSS (Cross Site Scripting) Cheat Sheet (http://ha.ckers.org/xss.html) (englisch)

Weblinks

- Bundesamt für Sicherheit in der Informationstechnik (BSI): Maßnahmenkatalog und Best Practices für die Sicherheit von Webanwendungen (http://www.bsi.de/literat/studien/websec/WebSec.pdf)
- heise.de – Sicherheit von Webanwendungen (http://www.heise.de/security/artikel/84149)
- XSS-Anwendungs-/Wissenstest (http://blogged-on.de/xss/)
- XSS-Tutorial (http://su2.info/uni/sosi/xss_paper/node5.html)
- Maui Security Scanner – kommerzieller Web-Anwendungs-Scanner (http://www.elanize.com)
- Springenwerk Open-Source Cross-Site-Scripting-Scanner (http://springenwerk.org)
- Charset-De/Encoder für XSS-Tests (http://h4k.in/encoding)
- Die Funktion htmlentities auf PHP.net (http://de.php.net/manual/de/function.htmlentities.php)
- Michael Suttons Blog – *How Prevalent Are XSS Vulnerabilities?* (http://portal.spidynamics.com/blogs/msutton/archive/2007/01/31/How-Prevalent-Are-XSS-Vulnerabilities_3F00_.asp (englisch)
- Informationen der Apache Foundation zu CSS-Attacken (http://httpd.apache.org/info/css-security/) (englisch)
- *The Cross-site Scripting FAQ* (http://www.cgisecurity.com/articles/xss-faq.shtml) (englisch)
- *The Cross Site Scripting Threat* (http://www.virtualforge.de/cross_site_scripting_threat.php) (englisch)
- *The impact of Cross Site Scripting on your business* (http://www.virtualforge.de/cross_site_scripting_impact.php) (englisch)
- Cross Site Scripting – Animierter Kurzfilm (http://www.virtualforge.de/vmovie.php) (englisch)
- Web Security Threat Classification (deutsch) (http://www.webappsec.org/projects/threat/v1/WASC-TC-v1_0.de.pdf) PDF 432kb
- RSnake: *XSS Cheat Sheet, esp: for filter evasion* (http://ha.ckers.org/xss.html) (englisch)

Von „http://de.wikipedia.org/wiki/Cross-Site_Scripting“
Kategorien: World Wide Web | Sicherheitslücke
Wartungskategorie: Wikipedia:Überarbeiten