Ausarbeitung

XSS / Web Security

Nanak Tattyrek

5AHITT

30.04.2014

Inhaltsverzeichnis

1 Einleitung

Der Begriff "Web Security" hat in den letzten Jahren immer mehr an Bedeutung gewonnen. Die Entwicklung von "sicheren" Webapplikationen ist heute ein wichtiges und herausforderndes Thema für jeden, der irgendeine Art von Webauftritt oder Service über das Internet anbietet. Durch mehr oder weniger komplexe Attacken kann selbst in vermeintlich harmlose Webformulare eigener Code "injected" werden, was dazu führen kann dass möglicherweise sogar vertrauliche Daten wie Kreditkarteninformationen oder ähnliches in falsche Hände geraten.
Obwohl die Bedrohung durch Angriffe heute höher ist denn je, vernachlässigen viele Administratoren die ausreichende Sicherung ihrer Systeme. Laut einem Bericht der Firma "Cenzic" hatten 96% der von ihnen getesteten Web Applikationen massive und ernst zu nehmende Sicherheitslücken.[1] Diese Ausarbeitung soll einige der gängigen Angriffsarten, wie etwa Cross-Site-Scripting (XSS) oder SQL-Injections, inklusive ihrer Verwendung aufzeigen und darlegen wie Webapplikationen gegen derartige Angriffe geschützt werden können.

1.1 Angriffsarten

Es gibt viele verschiedene Arten von Angriffen um Schwachstellen auszunutzen. Mit 25% ist dabei XSS die verbreitetste Schwachstelle und daher auch das beliebteste Angriffsziel. Gefolgt von "Information Leakage" mit 23%, Fehler in Authentifizierung und Autorisierung mit 15%, Session Management Fehlern mit 13%, SQL Injections mit 7% und Cross-Site Request Forgery (CSRF) mit 6%.[1] In dieser Ausarbeitung wird insbesondere auf XSS und SQL Injections der Schwerpunkt gelegt, aber auch andere Arten werden angeführt und beschrieben.

2 Cross-Site Scripting (XSS)

Beim Cross-Site Scripting schleust der Angreifer Skriptcode in dynamisch erstellte Teile von Webseiten ein. Viele glauben dass damit nur nervige Nachrichtenboxen oder ähnliche Effekte hervorgerufen werden können. Dies entspricht jedoch absolut nicht der Realität. XSS hat wesentlich höheres Potential, welches bis zum Diebstahl von Daten oder das Verändern der Benutzeroberfäche. Dass praktisch jede, nicht explizit gesicherte Webseite angreifbar ist, trägt außerdem zu dem hohen Gefahrenpotential dieser Attacke bei.[2]

2.1 Typen

Es gibt drei grundlegende Typen von XSS. Diese werden im nachfolgenden kurz erläutert:

- Reflektiertes XSS

  Beim reflektierten XSS wird der zu verwendende Schadcode direkt mit dem Aufruf übergeben. Dies geschieht zum Beispiel durch einer präparierte URL, welche den Code enthält. Die Verbreitung dieser URL kann über beliebige Wege erfolgen, etwa per Email oder Foreneinträge. Das Opfer muss nun nur noch auf den präparierten Link klicken, dass die Website von dessen Browser inklusive dem eingeschleusten Script Code abgearbeitet wird. Dies ist die am weitesten verbreitete Art dieser Attacke.

- Persistentes XSS

  Bei dieser Art wird der Schadcode in der Datenbank der angegriffenen Website gespeichert. Dies geschieht beispielsweise indem ein Eintrag, welcher Scriptcode enthält in ein Forum gepostet wird. Dieses Forum sendet den Beitrag mit eingebettetem Script nun zu jedem Besucher, der dieses öffnet. Das Schadenspotential dieser Art ist deutlich höher, da kein Zutun des Opfers notwendig ist, allerdings auch nicht so weit verbreitet wie das reflektierte XSS.

- DOM XSS

  Diese Art ist die am wenigsten verbreitete, da sie von vielen Angreifern als auch potentiellen Opfer-Webseiten oft unterschätzt wird. Auch hier wird eine präparierte URL verwendet. Das in der URL angehängte Script wird nun vom Browser per JavaScript in das eigentlich aufgerufene HTML File eingebunden. Diese Attacke funktioniert aber nur in Browsern welche Sonderzeichen in URLs nicht verarbeiten. Deshalb ist diese Attacke z.B. in Firefox nicht möglich.

[2]

2.2 Effekte

Wie schon in der Einleitung zu XSS beschrieben, lässt sich wesentlich mehr damit anstellen als nur lästige Mitteilungsboxen darzustellen. Im folgenden werden die möglichen Effekte durch eine XSS Attacke erläutert. [2]

2.2.1 Auslesen von Cookies

Viele Websites speichern wichtige Daten wie zum Beispiel Session IDs in Cookies ab um später darauf zugreifen zu können und beispielsweise einen eingeloggten Benutzer wiederzuerkennen. Gelingt es, ein derartiges Session Cookie zu stehlen, ist es oft möglich die Sitzung des Besitzers dieses Cookies zu übernehmen. Nun kann mit sämtlichen Berechtigungen dieses Nutzers gearbeitet werden, ohne sein Passwort stehlen zu müssen und oft auch ohne dass dieser etwas davon bemerkt.
Heutzutage werden Session IDs in Cookies allerdings oft verschlüsselt, was ein Abhören sehr schwierig macht. XSS kann diese Schutzmechanismen allerdings umgehen, da der Code im Browser des Opfers ausgeführt wird und daher alle Daten im Klartext vorliegen. [2]

2.2.2 Auslesen von Inhalten der Web Application

Neben Cookies, kann natürlich auch die aufgerufene Website an sich ausgelesen werden. Dies ist beispielsweise in einem Online Shop relevant, wo am Ende der Bestellung nochmal alle vom Kunden eingegebenen Daten angezeigt werden. Dort angezeigte Kreditkartendaten oder ähnliches können nun vom Angreifer abgezweigt werden. [2]

2.2.3 Übertragen der Daten an den Angreifer

Gesammelte Daten müssen auch wieder zum Angreifer übertragen werden können. Dies kann durch einen Web Server geschehen, welcher sich im Besitz des Angreifers befindet. Die Per XSS ausgelesenen Daten werden nun in Form einer ungültigen Anfrage an den Server gesandt. Es wird also beispielsweise ein Bild mithilfe des "<img>" Tags angefragt, welches nicht existiert. In diese Anfrage werden die vom Opfer extrahierten Daten angehängt sodass sie über das "access_log" File des Webservers ausgelesen werden können. Praktischerweise enthält das Log File auch noch zusätzliche Informationen wie IP Adresse es Opfers und ähnliches. [2]

2.3 Vermeintliche Schutzmechanismen

Es gibt einige Gegenmaßnahmen, welche oft eine sehr einfache Struktur haben, aber nicht genug Schutz gegen derartige Angriffe bieten.

- Filtern von "<script></script>" Tags

Das Filtern von Script Tags scheint verlockend, da diese dem Browser den Einsatz einer Scriptsprache anzeigen.

- Begrenzen von erlaubten Sonderzeichen

  Auch diese Technik klingt vorerst sehr effektiv, da spezielle Sonderzeichen wie etwa spitze Klammern ("<" und ">") ausgefiltert werden und so auch unterschiedliche Schreibweisen wie "<sCrIpT>" unbrauchbar werden.

- Begrenzen der Zeichenanzahl

  Das Begrenzen der Zeichenanzahl hätte zusätzlich zur Folge, dass nur noch wesentlich simplere Scripte, welche weniger Schaden anrichten eingeschleust werden können.

Alle bisher genannten Schutzmechanismen sehen auf den ersten Blick nach brauchbarem Schutz mit wenig Aufwand aus. Leider können sie aber mit mehr oder weniger Aufwand umgangen werden. Es sind zum Beispiel in modernen Browsern oft keine "<script>" Tags mehr notwendig, dass JavaScript Code ausgeführt wird. Es gibt einige Ansätze um diese Tags zu umgehen. [2]

2.4 Escapen und listen-basierte Filter

2.4.1 Escapen von Sonderzeichen

Wenn die Möglichkeit, Code in bestehende Tags einzufügen nicht existiert, müssen eigene Tags geschaffen werden. Wie bereits vorher erwähnt, ist das simple Filtern von Tags nicht ausreichend. Mithilfe von Escapen kann man allerdings wesentlich geschickter an die Sache herangehen. Im folgenden werden die Sonderzeichen mit ihren Escape Sequenzen gegenübergestellt.

Zeichen	Escape Sequenz
<	<
>	>
=	=
&	&
#	#
+	+
-	‐
/	/
?	?
(	&lpar
)	&rpar
{	{
}	}
$	$
@	@
^	⁁
*	*
:	:
;	;

Tabelle 1: Escape-Sequenzen [2]

Während nun ein "<script src = script.js>" noch zum Ausführen des an dieser Stelle geladenen Scripts führt, wir ein escaptes "<script src = script.js>" nur noch als harmloser Text im Browser angezeigt. [2]

2.4.2 Erkennen von Scripten mithilfe von Listen

Eine derartige "Blacklist" von schädlichen Scripten ist ein sehr hoher Aufwand, da der Ersteller der Liste über sehr viel Fachwissen und verschiedenste Angriffsarten und -vektoren verfügen muss. Aus diesem Grund fertigt ein Web-Entwickler eine solche Liste nicht selbst, sondern beschafft diese von externen Personen oder Organisationen. Ein Beispiel wäre das Projekt "safehtml" welches in der Vergangenheit einen hohen Bekanntheitsgrad erreicht hat, aber leider seit einiger Zeit nicht mehr weitergeführt wird. Blacklisten haben aber naturgegebene Schwachstellen wie etwa die Unvollständigkeit, man kann nie wissen ob die verwendete Liste bereits die neuesten Sicherheitsrisiken beinhaltet oder hinterherhinkt. [2]

2.4.3 Escapen vs. Listenbasiert

Prinzipiell ist das Escapen von Inputs vorzuziehen, da es sich in der Praxis als sehr belastbar erwiesen hat und nicht über die bekannten Schächen von Blacklists verfügt. Allerdings gibt es Fälle wo das Escapen nicht zielführend ist. Wenn beispielsweise bei einer Webanwendung wie einem Webmail Client diverse HTML Tags gebraucht werden, kann man diese nicht einfach escapen, da sonst möglicherweise die Formatierung von Emails vernichtet würde. Auch wenn eine bestehende Web Applikation gegen XSS Attacken gesichert werden soll ist es manchmal

sinnvoller auf Listen zu setzten. Ein re-design von Input Elementen greift tief in die Struktur einer Applikation ein und Formatierungen von Texten werden möglicherweise unmöglich gemacht. Da dies sehr Zeit- und Kostenintensiv sein kann, ist es manchmal sinnvoller auf Listenbasierte Mechanismen zurückzugreifen. Diese reichen zwar nicht an die Sicherheitsstandards von Escapen heran, allerdings kann damit doch schnelle und bis zu einem gewissen Grad ausreichende Abhilfe geschafft werden. [2]

2.5 Beispiele?

3 SQL Injection

Als SQL Injection wird das Einschleusen von eigenen SQL Befehlen in die Datenbank einer Web Applikation bezeichnet. Ziel des Angriffs ist meist Zugriff auf die Datenbank zu erhalten um Daten zu stehlen, nach eigenem Interesse zu verändern, grösstmöglichen Schaden anrichten oder einfach die Verfügbarkeit der angegriffenen Anwendung zu behindern. [3]

4 Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery [3]

Literatur

[1] Application vulnerability trends report: 2014, 2014-04-19. `https://info.cenzic.com/2013-Application-Security-Trends-Report.html`.

[2] Paul Sebastian Ziegler. *Cross-Site Scripting*. O'Reilly, 2008.

[3] Mike Shema. *Hacking Web Apps*. Syngress, 2012.

Tabellenverzeichnis

Listings

Abbildungsverzeichnis