Inhaltsverzeichnis

Abkürzungsverzeichnis

BB-Code	**B**ulletin **B**oard Code
CAPTCHA	**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part
CSRF	**C**ross-**S**ite-**R**equest-**F**orgery
CSS	**C**ascading **S**tyle **S**heets
DBMS	**D**ata**b**ase **M**anagement **S**ystem
DOM	**D**ocument **O**bject **M**odel
HTML	**H**ypertext **M**arkup **L**anguage
HTTP	**H**ypertext **T**ransfer **P**rotocol
IE6	Microsoft **I**nternet **E**xplorer in der Version **6**
IP	Kurzform für **I**nternet-**P**rotocol-Adresse
PHP	**P**ersonal **H**ome **P**age Tools
RSS	**R**eally **S**imple **S**yndication
SQL	**S**tructured **Q**uery **L**anguage
TAN	**T**rans**a**ktions**n**ummer
URL	**U**niform **R**esource **L**ocator
UTF8	**U**nicode **T**ransformation **F**ormat- **8**bit
VBScript	**V**isual **B**asic-**S**cript
W3C	**W**orld **W**ide **W**eb **C**onsortium
XSS	***C***ross-**S**ite-**S**cripting

Tabellenverzeichnis

1 Cross-Site-Scripting (XSS)

1.1 Beschreibung von XSS

1.1.1 Definition

„Cross Site Scripting (XSS) sind Angriffe auf Webservices über das HTT-Protokoll. Bei diesen Angriffen wird ein XSS-Loch bei der Eingabe von Benutzerdaten genutzt. Ein solches XSS-Loch entsteht, wenn der Benutzer beispielsweise HTML-Codes in das Eingabefeld eingeben könnte.“[1]

1.1.2 Gefahren von XSS

Die Gefahr von Cross-Site-Scripting entsteht bei dynamisch generierten Internetseiten – auch Webanwendungen oder Webapplikationen genannt –, welche Daten aus einer externen Quelle an den Nutzer einer Webapplikation weitergeben. Der Nutzer einer Webseite stuft die Webumgebung, in der er sich befindet, als sicher ein. In Zeiten des momentan vorherrschenden Web 2.0 interagiert er mit der Webanwendung, registriert sich und übergibt so private Informationen, wie beispielsweise eine Kontaktadresse oder ein Passwort, der Webanwendung. Werden diese Übergabewerte, auch Request genannt, gekonnt gewählt, so ist es einem Angreifer der Webanwendung möglich, schadhaften Code an die Webanwendung zu übergeben, der von dieser gespeichert und an den nächsten Nutzer, der die Seite aufruft, weitergegeben wird. Die Webseiten werden meist mit Hilfe von Programmiersprachen wie PHP oder Java in komplexer Art und Weise dynamisch zusammenbaut und reagieren interaktiv auf Nutzereingaben, während der Nutzer am Client technisch meist nicht versiert ist. Da das HTTP, über das Webseiten vom Server an den Client übergeben werden, die Verbindung zum Webserver nach jedem Aufruf wieder trennt, kann ein Nutzer über die Dauer des Besuches der Webseite nicht eindeutig identifiziert werden. Eine Identifizierung anhand der IP-Adresse reicht nicht aus, da viele User z.B. in Schulen oder Büros nur über einen einzigen Server und somit über eine IP-Adresse auf das Internet zugreifen. Eine übliche Möglichkeit

[1] [DAT07]

den Nutzer, der sich bei einer Webseite registriert hat und sich einloggt, zu identifizieren besteht darin, dass nutzerspezifische Informationen serverseitig in einer Datei, der so genannten Session, gespeichert werden. Der Browser, der beim Nutzer für die Darstellung der Webseite zuständig ist, wird angewiesen eine kleine Datei auf den Computer des Nutzers zu speichern. Diese Datei befindet sich beim Nutzer im gesicherten Umfeld und wird Cookie genannt. Hier können Webanwendungen begrenzt Informationen beim Nutzer speichern. Der Browser liest dieses Cookie bei jedem Aufruf der Seite aus und sendet die Informationen an den Webserver. Nun wird in dieser Datei eine ID, die so genannte Session-ID gespeichert. Anhand dieser ID, die für jeden Nutzer der Webanwendung eindeutig ist, kann der Nutzer nun validiert werden. Mit Hilfe von Cross-Site-Scripting ist es möglich, diese den Nutzer identifizierende Session-ID aus dem Cookie auszulesen und in einen nicht vertrauenswürdigen Kontext eines Angreifers zu übersenden. Eine Übernahme der virtuellen Identität des Nutzers der Seite ist nun möglich, wenn der Angreifer diese erspähte Session-ID in seine eigenen Cookies integriert. Dieser Vorgang ist natürlich auch möglich, wenn die Session-ID als Alternative zu Cookies mit an die aufgerufene URL als Parameter angehängt wird. In Versionen des Internet-Explorers ist es mitunter auch möglich, dass der Inhalt der Zwischenablage ausgelesen und weiter gesendet werden kann.

Webseiten sind intern in einer Baumstruktur, dem so genannten DOM organisiert. Mit Hilfe dieses DOM, kann von jeder beliebigen Stelle in dieser Baumstruktur auf alle Elemente zugegriffen werden, soweit sie sich im Browserfenster befinden. Somit ist es möglich, das Aussehen oder das Verhalten einer Seite, in die ein Cross-Site-Script eingefügt wurde, komplett zu verändern. Es ist somit durchaus denkbar, dass ein Nutzer seine Bankdaten in ein Webformular eingibt und diese dann wahlweise einer anderen Webanwendung geschickt werden oder dass der Nutzer beim Abschicken des Formulars direkt zur feindlichen Seite umgeleitet wird. Ist diese feindliche Seite dann eine optische Kopie der Originalseite, so denkt der Nutzer weiterhin, dass er sich in seinem vertrauten Umfeld befindet und gibt private Daten preis. Dieses Verfahren wird als Phishing bezeichnet. Werden bei der Seite Frames verwendet, so ändert sich die in der Adressleiste des Browsers angezeigte Adresse nicht. Eine Anwendung von XSS wäre die globale Manipulation des Aussehens der Seite. Hierfür kann beispielsweise das Design der Seite alternativ in einheitliches rot oder eine für den Nutzer unangenehme Kombination aus rosa und giftgrün mit gelb

abgeändert werden. Weiterhin können Links und Eingabefelder so ausgerichtet werden, dass eine Bedienung der Webapplikation unmöglich wird. Dieses ungewöhnliche Verhalten verunsichert den Nutzer der Seite langfristig. Die Schäden für das Image der Seite und Ihrer Betreiber sind inklusive und verbreiten sich durch die Medien.
Eine browserübergreifende XSS-Attacke auf das Betriebssystem ist in der Regel nicht möglich, da das DOM aller Browser den Zugriff durch Scripte, die auf der Webseite laufen, auf Objekte, die sich außerhalb der Webseite an sich befinden, verhindert. Eine Ausnahme bilden Scripte wie beispielsweise VB-Scripte, die direkt mit dem Betriebssystem operieren. Es empfiehlt sich diese abzuschalten, da wenig von ihnen Gebrauch gemacht wird. Allerdings können Cross-Site-Scripte für den Webmaster lange unentdeckt bleiben, da es bei großen Seiten mit User-generated-Content kaum möglich ist, diesen komplett zu überblicken. Zudem machen sich Cross-Site-Scripte meist nicht durch Fehlverhalten der ganzen Seite bemerkbar und da sie clientseitig ablaufen, sind ihre Auswirkungen und Spuren für den Webmaster meist nicht oder nur schwer ersichtlich.

1.1.3 Verbreitung XSS-anfälliger Seiten

Viele Webseiten mit Inhalten, welche von fremden Usern in die Seite eingebracht werden, weisen Schwachstellen auf. Auch bei manchen einfachen Mailclients ist dies der Fall. Oft sind sich Webentwickler der Gefahr gar nicht oder nur unzureichend bewusst. Andere interessiert diese Schwachstelle nicht, da sie sich primär nicht gegen die Seite an sich, sondern gegen ihre Nutzer richtet und damit für den Server keine Gefahr darstellt. Angriffspunkte für XSS sind alle Eingabefelder und Parameter die an Webanwendungen übermittelt werden, wie das beispielsweise bei Such- und Login-Formularen der Fall ist. XSS ist ein omnipräsentes Problem kleiner Seiten, aber auch großer Webapplikationen, wie häufig in den Medien nachgelesen werden kann.[2]
Eine zusätzliche Möglichkeit ist das Einbringen von XSS in den RSS-Feed einer Webapplikation, z.B. eines Blogs. Bei der Aufarbeitung des RSS-Feeds zu HTML kann dieser Schadcode den User erreichen. Meist sind Nutzer eines bestimmten

[2] Beispiel: [Hei07]

Blogs einer bestimmten Zielgruppe zuzuordnen, es können also zielgruppenspezifische Angriffe getätigt werden. Es ist für einen Blogbetreiber problemlos möglich XSS in seinem RSS-Feed unterzubringen. Er hat jederzeit die volle Kontrolle über seinen Schadcode und Seiten, die den RSS-Feed weiterverbreiten, wie z.B. Blogreader, prüfen den Inhalt des RSS-Feed selten.

1.1.4 Datenübermittlung von Nutzer zur Webanwendung

Es gibt vier Hauptelemente, dessen Werte vom Nutzer manipuliert an die Webanwendung übergeben werden und Schadcode enthalten können. Dies soll am Beispiel der Programmiersprache PHP erläutert werden.

1. GET-Parameter[3]
   Diese Parameter werden an eine URL angehängt und können vom Nutzer direkt in der Adresszeile seines Browsers manipuliert werden. Dieser Parameter ist wohl der unsicherste und sollte nur für Navigationsparameter oder ähnliches verwendet werden. In PHP ist das die superglobale Variable $_GET, d.h. sie ist von überall her ohne weiteres erreichbar.
2. POST-Parameter[4]
   Dieser Parameter wird meist durch ein Formular mit Daten gefüllt. Er wird im Hintergrund vom Browser an den Server gesendet und nicht mit in der URL übergeben. Um den Post-Parameter – in PHP die Superglobale $_POST – zu manipulieren sind Hilfsprogramme, zumindest ein Texteditor, notwendig. Die Webseite wird dafür abgespeichert und die Eingabefelder folgend nach eigenen Wünschen modifiziert.
3. COOKIE-Parameter[5]
   Mit jedem Aufruf einer Seite wird auch der Cookie-Parameter an den Server übergeben. In PHP kann auf ihn über $_COOKIE zugegriffen werden. Cookies werden grundsätzlich auf den PC des Nutzers gespeichert, obliegen somit vollständig seiner Kontrolle und sind als nicht vertrauenswürdig einzustufen.
4. META-Daten

---

[3] http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.3

[4] http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.5

[5] [Net00]

Die Meta-Daten der Verbindung setzen sich aus einer Reihe unterschiedlicher Quellen zusammen. Die meisten stammen direkt vom eigenen Server und werden dadurch des Öfteren als global vertrauenswürdig eingestuft. Dies ist jedoch ein fataler Fehler, da Elemente wie der Referrer, UserAgent, Host, Accept_Language, Accept_Charset mit Hilfe von Tools oder Proxies vom Nutzer geändert werden können. Bei manchen Anwendungen im Internet wird dies nicht berücksichtigt. Somit hatte vor einigen Versionen selbst die verbreitete Statistiksoftware Webalizer eine Anfälligkeit auf XSS, da sie den Referrer ohne vorherige Prüfung an den meist Webmaster ausgab[6]. In PHP kann auf die Meta-Daten über die Superglobale $_SERVER zugegriffen werden.

Die Manipulation der Daten und Formulare erfolgt entweder über manuelle Änderung durch Abspeichern und Anpassen des HTML-Dokumentes oder durch Tools. Gerade der Webbrowser Firefox ist mit seiner leistungsfähigen Pluginschnittstelle ein passabler Helfer. So kann ein HTML-Dokument bequem mit Plugins wie der Webdeveloper Toolbar, Firebug oder Modify Headers den Bedürfnissen des Angreifers angepasst werden.[7]

1.1.5 Funktion von XSS

Um eine Seite effektiv gegen XSS absichern zu können, muss zuerst einmal verstanden werden, wie XSS im Detail funktioniert. Wenn bei der Entwicklung folgende Regeln beachtet werden, kann XSS erfolgreich entgegengetreten werden:

1. Daten, die vom Nutzer kommen, sind grundsätzlich als gefährlich zu betrachten (dazu gehören auch Metainformationen wie Referrer oder UserAgent)
2. alle Daten aus externen Quellen müssen einer Plausibilitätsprüfung unterzogen und abgesichert werden

Cross-Site-Scripte werden schon wesentlich erschwert, wenn sämtliche Nutzereingaben auf die Zeichen < und > überprüft und diese durch < und > ersetzt werden. Dabei werden einfach die Zeichen, die ein HTML-Tag definieren, also auch die Tags zur Ausführung gefährlicher Scripte, so umgeschrieben, dass sie

---

[6] [CSP07], S. 103

[7] downloadbar unter [Moz07]

von den Browsern nicht mehr als HTML-Tag erkennbar sind. Dies ist natürlich nur eine unzureichende Weise sich zu schützen, da sie nicht alle Möglichkeiten, schädlichen Code in die Seite einzubringen, verhindert.
Die meisten Varianten von XSS werden von den Netscape-Browsern interpretiert, sicht gefolgt vom Internet Explorer in der Version 6 und dem Opera[8]. Es kann festgestellt werden, dass es keinen Browser gibt, in dem Cross-Site-Scripte nicht möglich sind, da einige Cross-Site-Scripte, wenn sie in den Kontext der Seite eingeführt werden, eine gültige HTML-Syntax aufweisen. Die Probleme entstehen vor allem, da die Browser nicht standardkonformen HTML-Code fehlertolerant interpretieren. Eine gewisse Variabilität ist zwar notwendig, da die meisten HTML-Dokumente Fehler enthalten, jedoch ist es so leichter z.B. Javascript in die Seite einzufügen. Beim IE6 und bei Opera können viele Angriffe über den HTML-Code *“<img src=javascript:;>“* realisiert werden. Dies ist nicht standardkonform.[9]
Zudem gibt es bei allen Browsern auch beim UTF8-Zeichensatz Probleme, da hier die im Quelltext der zum Browser gesendeten HTML-Datei stehenden UTF8-formatierten Zeichenketten, die meist durch einen Filter gehen, zuerst in normale Zeichen umgewandelt und dann bei Scripten entsprechend ausgeführt werden. Durch diese Umschreibung von Zeichen wird versucht vorhandene Filter zu umgehen. Dies wird auch versucht, indem in die übergebenen Zeichenketten Nullzeichen, Tabulatoren und Leerzeichen oder Zeilenumbrüche eingebracht werden.[10]

1.2 Schutz gegen XSS

1.2.1 Standardkonformität

Möchte man ein Cross-Site-Scripting in den Inhalt einer Webseite einpflegen, so beginnt dies am Anfang mit einer Analyse und darauf folgenden Angriffsversuchen. Meist wird versucht Javascript in die Seite einzubringen. Alternativ können auch andere Scriptarten wie VBScript oder Mocha eingebracht werden, welche aber nur vom Internet-Explorer interpretiert werden. Viele Cross-Site-Attacken können schon

[8] Entnommen der Auswertung von [RSn07]

[9] [W3C00] - Bereich Images

[10] Entnommen der Auswertung von [RSn07]

damit verhindert werden, dass immer auf die standardkonforme Auszeichnung des Quelltextes geachtet wird. Somit muss z.B. bei einem Link zuerst das angefangene Tag beendet werden, bevor eigener Code eingeschleust werden kann. Ob ein Quelltext standardkonform ist, kann mit Hilfe des Validators des W3C (http://validator.w3.org) überprüft werden. Erfolgt zusätzlich eine Filterung und Überprüfung der vom Nutzer kommenden Informationen, ist XSS praktisch ausgeschlossen.

1.2.2 HTML filtern am Beispiel von PHP

An den meisten Stellen wird keine Eingabe von HTML in Eingabefeldern durch den Nutzer benötigt. Daher können ohne Bedenken alle HTML-Tags mit Hilfe der PHP-Funktion strip_tags() entfernt werden. Danach sollten aber auch noch HTML-spezifische Zeichen wie ', ", < und > in ihre HTML-Entitäten ' " < und > umgewandelt werden. Dies geschieht mit der PHP-Funktion htmlentities($variable, ENT_QUOTES).
Soll zugelassen werden, dass der Nutzer in ein Eingabefeld HTML eingeben kann, so sollte restriktiv gefiltert werden. Es sollte alles aus dem Request herausgeschnitten werden, was nicht durch Ausnahmen genehmigt ist, d.h. es soll keine Blacklist, sondern eine Whitelist angewendet werden. Diese Ausnahmen können dann auch mittels der Funktion strip_tags(,<tag1><tag2>') realisiert werden. Eine andere Möglichkeit HTML zu erlauben, ist der so genannte BB-Code. Dieser ist eine an das HTML angelehnte Sprache mit eckigen Klammern. Dieser Code wird dann vor der Ausgabe an den Nutzer der Webanwendung geparst und in normales HTML umgewandelt. Diese Art der Absicherung kommt aus dem Bereich der Forenentwicklung. Fett geschriebener Text sieht dabei z.B. folgend aus: [b]fetter Text[/b]. Ein Vorteil ist, dass der Betreiber der Seite daraus komplexere Arten des Stylings implementieren kann.

1.2.3 Eingabedaten validieren

Wenn die Eingaben **immer** auf ihre logische Korrektheit geprüft werden, können fehlerhafte Einträge erst gar nicht in den Datenbestand der Webseite eingepflegt werden. An manchen Stellen wird immer eine Zahl benötigt, bei anderen ein Datum

mit dem Aufbau dd.mm.yyyy . Zahlen können einfach mit der Hilfe vom so genannten Casting umgewandelt werden:

```
$zahl = (int) $zahl;
$zahl = (float) $zahl;
```

Alternativ können auch die Funktionen intval() und floatval() genutzt werden. In Zeichenketten sollten nur eine beschränkte Anzahl Zeichen zugelassen sein, spezielle Zeichen sollten umgewandelt werden.
Eine sehr komfortable aber auch nach Ansicht des Autors komplexe und manchmal schwer verständliche Variante der Überprüfung sämtlicher Werte ist mit der Hilfe regulärer Ausdrücke möglich. Die Funktionen ereg(), eregi() und preg_match() geben einen booleschen Wert zurück, ob ein übergebener String mit dem regulären Ausdruck übereinstimmt oder nicht. Beispiele:
Ausdruck ist eine Zahl

```
preg_match("/^\d+$/", $num);
```

Ausdruck ist ein Preis

```
preg_match("/^[0-9]+[,.][0-9]{2}$/", $price);
```

Überprüfung, ob eine Kombination aus Straße und Hausnummer besteht

```
preg_match("/^[a-zäöüß. '\-\/]{4,}[0-9]{1,5}[a-z]?$/i", $address);
```

Überprüfung, ob Eingabe eine Postleitzahl sein kann (Deutschland)

```
preg_match("/^[0-9]{5}$/", $postcode);
```

Überprüfung einer URL (mit Umlautdomains bei http, https und ftp)

```
preg_match("#^(https?://|ftp://)[a-zäöü0-9-]+(\.[a-zäöü0-9-]+)*\.(([a-z]{2,3})|(aero|coop|info|jobs|museum|mobi|name|nato|travel))(/?$|/.+)#i", $url);
```

Überprüfung einer Mailadresse

```
preg_match("/^[a-z0-9_-]+(\.[a-z0-9_-]+)*@([a-äöü0-9-]+[a-zäöü0-9-]\.)+(([a-z]{2,3})|(aero|coop|info|jobs|museum|mobi|name|nato|travel))$/i);
```

Es empfiehlt sich auch die Überprüfung des Benutzernamens, indem nur bestimmte Zeichen zugelassen werden. Auch sollten alle Informationen, die aus den META-Daten und den Cookies eingelesen werden, entsprechend validiert werden.

1.2.4 Automatische Umwandlung

Will man ganz sicher gehen, dass für Cross-Site-Scripte keine Angriffspunkte existieren, so kann bei einem Neuaufbau der Programmstruktur eine Funktion

eingebaut werden, die automatisiert aufgerufen wird und aus den vier superglobalen Variablen, dessen Inhalt der Nutzer anpassen kann, schädliche HTML-Tags herausnimmt und Sonderzeichen umwandelt. Es kann auch eine automatisierte Typumwandlung vorgenommen werden, indem die Übergabeparameter mit der ungarischen Notation versehen werden. Dabei wird der Typ der Variable als Präfix an den Variablennamen angehangen (z.B. iZahl für eine Integerzahl). Sollten wirklich einmal HTML-Übergaben benötigt werden, so müssen diese als Ausnahme separat behandelt werden.

2 Cross-Site-Request-Forgery (CSRF)

2.1 Beschreibung von CSRF

2.1.1 Allgemeines

Cross-Site Request Forgery, welches als CSRF aber auch als XSRF abgekürzt wird, ist eine Vorgehensweise, bei der der Angreifer einer Webanwendung unberechtigt Daten verändert, indem er sich der Identität eines gegenüber der Webanwendung berechtigten Users bedient und diesen ohne sein Wissen einen modifizierten HTTP-Request unterschiebt, der eine Aktion im Namen des Users in der Webanwendung ausführt, ohne das der User der Webanwendung davon Kenntnis hat.

2.1.2 Wesen von CSRF

Diese Technik ist das Gegenstück zu Cross-Site-Scripting. Sie ist schwerer ausführbar als Cross-Site-Scripting und bedarf mitunter noch persönlicher Informationen des Opfers, die mittels Social Engineering erlangt werden müssen. Beim Social Engineering wird versucht Verhaltensweisen einer Person so genau wie möglich zu bestimmen um daraus Schlüsse für weitere Vorhaben zu ziehen. Während XSS direkt versucht schadhaften Code im Browser des Nutzers auszuführen, werden bei CSRF die vom Nutzer gespeicherten Daten, wie z.B. ein Session-Cookie dazu missbraucht, um im Namen des registrierten Nutzers eine Aktion in der Webanwendung auszuführen. Es wird eine legitime Session eines Fremden genutzt, der dann, meist durch die Hilfe von XSS, unfreiwillig ein HTTP-Request an den Server sendet. Dieses Vorgehen wird als Session-Riding bezeichnet. Die an den Server abgesendeten Daten werden vom Angreifer so gewählt, dass eine bestimmte Aktion in der Webanwendung ausführt wird. Dies kann alles sein, was mit der Berechtigung des Anwenders in der Webanwendung oder mit der Leistungsfähigkeit der Anwendung möglich ist. Meist erfolgt eine Modifikation von Inhalten. Kann jedoch jemand beim Administrator der Seite einen modifizierten HTTP-Request absetzen, kann dies die Bearbeitung oder Spionage sämtlicher Nutzerdaten zur Folge haben.

Ein anderes Szenario erläutert, welches die Sache gut verdeutlicht. Dabei weiß ein Angreifer, dass eine Person in einem bestimmten Forum liest und gleichzeitig Onlinebanking bei einer Bank betreibt. Ist die Bankseite nicht korrekt abgesichert, kann folgendes geschehen. Der Anwender surft in einer Seite noch in dem Forum, meldet sich in einem anderen Fenster schon bei der Bank an. Wird nun im Forum – z.B. durch automatischen Reload mit Hilfe eines durch XSS eingebrachten JavaScripts oder durch das direkte Laden einer Seite durch den Nutzer – ein Request an den Server der Bank geschickt, so wird dieser auf Seiten der Bank auch ausgeführt, da der Nutzer ja angemeldet ist und somit ein gütiges Session-Cookie aufweist.[11]
Meist geschieht das Absenden dieses HTTP-Requests durch die Modifikation der Quelle einer Bilddatei an die noch durch sinnfreie Parameter die Dateiendung angepasst wird. An der Stelle des Bildes erscheint dann nur ein unauffälliges Logo, dass das Bild nicht geladen werden konnte, wenn dieses nicht noch mittels CSS ausgeblendet wird. Aufgrund der Tatsache, dass es durch dieses Vorgehen möglich ist Überweisungen ohne die Zustimmung des Nutzers zu tätigen, wenn die Webanwendung der Bank unsicher programmiert ist und statt des POST-Request auch das GET-Request als Eingabedaten zulässt, sind Betreiber von Online-Banking dazu übergegangen weitere Sicherungsmechanismen wie die TAN einzuführen.
Mit der Hilfe von CSRF ist auch möglich eine Firewall zu umgehen und Aktionen in einem zum Internet abgesicherten Bereich auszuführen, wenn der Nutzer gleichzeitig auf einer mit CSRF versehenden Seite und in einen für die Außenwelt nicht zugänglichen Bereich surft. Dazu muss dem Angreifer natürlich die Arbeitsweise des internen Systems bekannt sein. Dies ist z.B. bei openSource-Software immer der Fall.
Es gibt keine einheitliche Vorgehensweise für den Schutz gegen CSRF, jedoch ist eine Anwendung ausreichend abgesichert, wenn bei der Programmierung aller Bereiche alle Übergabewerte validiert werden. Einen 100%igen Schutz kann es nicht geben, da aufgrund der Arbeitsweise des HTTP nicht sicher gesagt werden kann, von wo der Aufruf der entsprechenden Seite erfolgte, d.h. ob er aus der sicheren Seite oder von einer Fremdapplikation erfolgt.

---

[11] Szenario aus [CSP07]

2.2 Abwehrmöglichkeiten

Auch für das Cross-Site-Request-Forgery gibt es einige Abwehrmöglichkeiten, sowohl für die Verhinderung, dass ein schädlicher Request in die eigene Seite eingebracht werden kann, aber auch dafür, dass ein erfolgender Angriff keine nennenswerten Auswirkungen auf die eigene Seite hat.
Eine leicht umzusetzende und effektive Möglichkeit ist das automatische Ausloggen eines Nutzers, wenn dieser eine gewisse Zeit inaktiv war. Diese Zeitspanne erspannt sich je nach System von 30 Minuten bis 2 Stunden, ist aber frei wählbar. Somit ist der Anwender nicht mehr in die Anwendung eingeloggt und es können keine Aktionen mehr in seinem Namen gestartet werden.
Eine weitere Option ist, dass von der Webanwendung genau die Informationen aus dem Request entgegengenommen werden, die gebraucht und vom Nutzer abgesendet werden. Dies bedeutet, dass bei sicherheitsrelevanten Daten immer ein Formular per POST-Request übergeben wird, welches auch nur in der Webanwendung ausgewertet wird. Eine Auswertung des GET-Parameters darf in diesem Fall also nicht erfolgen, da Parameter, die mittels GET-Parameter übergeben werden, an dieser Stelle nicht definiert sind. Der POST-Parameter wird genommen, da dieser ein wenig schwerer zu manipulieren ist, als der GET-Parameter. In diesem Zusammenhang sollte auch möglichst darauf verzichtet werden, die PHP-Superglobale $_REQUEST, die eine Synthese der GET- und POST-Parameter bildet, zu nutzen.
Weiterhin kann auch eine Absicherungsmethode gegen Cross-Site-Scripting zugleich gegen Cross-Site-Request-Forgery helfen. Erfolgt eine korrekte und gute Überprüfung der Eingabedaten beispielsweise mittels Whitelists, kann verhindert werden, dass Schadcode in die eigene Seite eingebracht wird, der zur Ausführung eines Angriffes auf eine weitere Seite benötigt wird. Eine Option für CSRF ist die Möglichkeit des Einbringens externen Daten über Bilder, Audiodateien oder externen CSS-Dateien. Dies sollte aufwändig validiert oder ganz verboten werden.
Zudem ist eine elegante Möglichkeit, bei Veränderungen am Datenbestand bzw. bei sicherheitsrelevanten Aktionen, den Nutzer nach einer Bestätigung zu fragen. Dies kann mit Hilfe einer zwischengeschalteten Seite, aber auch über Captchas und ähnliches realisiert werden. CAPTCHAs sind Bilder, welche Anweisungen oder Zahlen abbilden, die den Nutzer als Mensch verifizieren sollen. In ein Eingabefeld

muss meist eine Zeichenkombination oder ähnliches eingegeben werden, das das Bild abbildet. Da Maschinen Bilder nur schwer auswerten können, kann so eine Maschine – ein so genannter Robot – von einem Menschen unterschieden werden. Eine weitere Möglichkeit ist die interne Mitführung von Keys, die in Hidden-Feldern der Formulare oder in Links mitgeführt werden und sich bei jedem Seitenaufruf ändern. Dieser Key wird in der Session des Nutzers gespeichert und bei jeder Aktion überprüft. Natürlich kann auch eine Bestätigungsmail oder das Passwort des Nutzers genutzt werden. Ein Referrer-Schutz, wie er manchmal angewendet wird, reicht nicht aus. Dabei wird der Referrer, der besagen soll von welcher Seite der Nutzer kommt, nach bestimmten Kriterien ausgewertet. Jedoch unterstützen nicht alle Systeme Referrer und sie sind einfach über das Firefox-Plugin ModifyHeaders manipulierbar.

3 SQL-Injection

3.1 Beschreibung von SQL-Injection

3.1.1 Allgemeines

Wird eine Sicherheitslücke in einer Webanwendung, die durch eine mangelnde Maskierung oder Überprüfung der Eingabedaten entsteht, dazu ausgenutzt Aktionen mit der Datenbank an bestimmten Stellen zu auszuführen, so wird dies SQL-Injection genannt. Dabei versucht der Angreifer eigene Datenbankbefehle einzuschleusen, Daten in seinem Sinne zu verändern oder Kontrolle über den Server zu erlangen.

3.1.2 Das Wesen von SQL-Injections

Bei SQL-Injections muss eine Query, die von der Webanwendung an die Datenbank geschickt wird, vom Angreifer erraten oder reengineert werden. Das Hauptproblem liegt an Nutzereingaben, die in PHP aus den Superglobalen $_GET, $_POST, $_COOKIE und $_SERVER kommen, welche nicht oder nur ungenügend validiert in eine SQL-Query eingebunden werden.
Anhand der Fehlermeldungen, die bei einem Angriff aufgrund des nicht gültigen SQL-Statements zurückgegeben werden, und durch die Analyse des Verhaltens der Webseite, kann der Angreifer nach und nach auf den Aufbau der internen SQL-Queries und des Datenbankschemas schließen. Das korrekte Übernehmen des richtigen Requests (GET bzw. POST) reicht nicht aus, da die Formulare Clientseitig manipuliert werden können.

3.1.3 Vorgehen des Angreifers bei einer SQL-Injection

Wird eine Nutzereingabe nicht genügend überprüft oder direkt in einen SQL-Query übernommen, kann dieses verändert werden. Ein paar Beispiele:
Aus *SELECT userdata FROM users WHERE user = '**egon**' AND pass='**passwd**'* wird dann SELECT userdata FROM users WHERE user='**nonsens' OR 1=1 --**' AND pass='**passwd**' . Wird diese Query zur Überprüfung der Nutzerdaten benutzt, birgt das die Gefahr, dass der Angreifer eingeloggt wäre als der erste User, der in der

Datenbank notiert ist – meist der Administrator. Eine weitere Möglichkeit der Querys wäre, dass daraus die Query SELECT userdata FROM users WHERE user='**nonsens'; UPDATE users SET pass='test' --** , AND pass='' . Durch diese Modifikation werden statt der einen nun 2 Querys ausgeführt. Die letzte setzt dabei die Passwörter aller Nutzer auf den Wert „test", ein enormer Schaden für die Seite würde entstehen. Durch die Fehlermeldungen, die beim Einfügen einiger falscher Querys entstehen, kann ein Angreifer auf seine Fehler bei der Querymanipulation schließen. Durch Kommentare am Ende des Querys werden alle Elemente nach der eingefügten Benutzereingabe auskommentiert. So kann die SQL-Query besser manipuliert werden.

3.2 Abwehrmöglichkeiten

3.2.1 Abwehr durch korrekte Einstellungen

Die Umgebung, in der die Webseite läuft, kann so eingestellt werden, dass keinerlei Fehlermeldungen an den Angreifer ausgegeben werden. Dies bietet zwar keinen absoluten Schutz, da manche z.B. bei openSource-Software den internen Aufbau kennen oder gut bzw. geübt im Erraten der richtigen Querys sind. Fehlermeldungen können optional in einer separaten Datei auf dem Server gespeichert werden.
Eine weitere sinnvolle Möglichkeit ist, dass keine Multi-Querys zugelassen werden. Dies schützt dann vor dem zweiten Beispiel, bei dem der Angreifer eine weitere Query einfügt.
Bei PHP können Fehlermeldungen in der Konfiguration der php.ini über **display_errors = 0** komplett deaktiviert werden. Diesen Aspekt der erhöhten Sicherheitsgewinnung durch das Verstecken bzw. zurückhalten von Informationen, wird Security by Obscurity genannt.[12]

3.2.2 Abwehr durch gute Programmierung

Bei der Programmierung einer Webanwendung gibt es im Groben zwei Möglichkeiten SQL-Injections zu verhindern. Die erste Möglichkeit ist durch eine erhöhte

[12] [CSP07] Seite 117

Komplexität der SQL-Querys. So ist es z.B. möglich, mehrere Klammern in die Query mit einzubauen, die z.B. das WHERE-Attribut umfasst. Bei dieser Konstellation muss der Angreifer diese überflüssige Klammer mit einfügen, um ein korrektes SQL-Statement zu bekommen. Eine weitere eigentlich schon selbstverständliche Möglichkeit ist richtiges Escapen der Eingabewerte, d. h. dass vor allen einfachen Anführungszeichen ein Backslash vorangestellt wird. Geschieht dies nicht, kann es auch im normalen laufenden Betrieb der Anwendung zu Fehlverhalten kommen. Dieser Vorgang kann kombiniert werden mit der Überprüfung der eingegebenen Daten, die wie weiter oben beschrieben das Cross-Site-Scripting verhindern sollen. Des Weiteren müssen einige Sonderzeichen und Escapesequenzen modifiziert werden. Dies sind z.B. \x1a (Strg+Z) zum Ausführen von Befehlen auf der Komandozeile, \ und % als Escapezeichen an sich und Platzhalter für Strings, \n und \r wirken sich nur auf den Log aus, da MySQL nur einen Datensatz in diesen schreiben kann. Ein Log ist eine Datei, wo Veränderungen und/oder Fehler für eine spätere Analyse protokolliert werden. Eine Absicherung gegen Zeilenumbrüche oder Escapesequenzen kann mit der PHP-Funktion mysql_real_escape_string() realisiert werden.

Eine weitere Möglichkeit, die sich gut mit der Absicherung der Seite gegen XSS und CSRF verbinden lässt, ist das genaue Überprüfen der Nutzereingaben. Es kann auch überprüft werden, ob die resultierenden Querys bestimmten Voraussetzungen genügen. Eine effektive Variante ist das Überprüfen der SQL-Statements auf einen Kommentar. Kommentare, die den Nutzen der Query beschreiben, sollten nicht in den Query geschrieben werden, da so mehr Daten zum Datenbankserver übertragen werden und die Sache ineffizienter wird. Eine Dokumentation empfiehlt sich hier im umliegenden Quelltext der Webanwendung. Auch könnten Querys, die ein Semikolon außerhalb eines Strings beherbergen, generell abgelehnt werden um Multiqueries zu verhindern.

Eine sichere und sehr beliebte Möglichkeit ist das Parameter Binding, auch Prepared Statements genannt. Diese Möglichkeit wird allgemein als eine effektive Handhabe gesehen, SQL-Injections komplett zu verhindern. Bei diesem Vorgehen wird zuerst ein Query-Template erstellt, womit das gewünschte SQL-Statement erzeugt wird. An der Stelle, wo eigentlich die Werte eingefügt würden, stehen nur Platzhalter. In einem zweiten Schritt werden die Benutzerwerte separat übergeben. Bei dieser Übergabe werden sie automatisch escaped und in das SQL-Statement eingefügt. Es findet

auch eine automatisierte Typumwandlung statt. Final wird das nun sichere SQL-Statement in einer dritten Funktion ausgeführt. Die Query-Templates sind wieder verwendbar, was den Vorteil hat, dass, wenn ein Query vielmals ausgeführt werden muss, nur die Daten an den SQL-Server geschickt werden und nicht immer wieder das SQL-Statement.

3.2.3 Weitere Gefahren durch SQL-Querys

Es gibt noch wenige SQL-Statements, die sich ziemlich leicht filtern lassen. Auch sind sie nur verfügbar, wenn der Nutzer, der gerade mit dem Query auf die Datenbank zugreift, root-Rechte hat. Der root-Nuter ist der Nutzer, der alle möglichen Berechtigungen hat. Er kann somit alle Funktionen des DBMS (Database-Management-System) wahrnehmen, auch wenn diese sicherheitskritisch sind, wie es beispielsweise das Anlegen neuer Nutzer oder die Löschung ganzer Datenbanken sind. Es empfiehlt sich einen Datenbank-Nutzer für jede einzelne Anwendung zu erstellen, der eingeschränkte Rechte aufweist.
Weiterhin kann es bei einer mangelnden Rechteverwaltung vorkommen, dass ein Angreifer, der eine Stelle für erfolgreich SQL-Injections gefunden hat, eine Denial of Service Attacke durch den Befehl SELECT BENCHMARK(20000000, MD5(‚nonsens')) einleitet. Bei dieser Angriffsart wird unnötig viel Last auf dem SQL-Server produziert und somit der Dienst für andere Anwender gestört.
Eine weitere Möglichkeit ist das Auslesen von Dateien auf dem Serverrechner. Dies erfolgt über das SQL-Statement SELECT LOAD_FILE(‚/home/gerhard/text.txt'). An einer günstigen Stelle kann sich ein Angreifer unter Umständen jede Datei vom Server ausgeben lassen.
Somit kann ein Angreifer sogar Befehle in der Konsole des Servers ausführen lassen. Dieser Sachverhalt kann realisiert werden z.B. über den Query GO EXEC cmdshell('format C') [13].

[13] Quelle: [Mik07]

4 Zusammenfassung

Die Verhinderung der in diesem Dokument beschriebenen Gefahren für Webseiten fangen mitunter schon bei der Planung des Projektes an. So ist es möglich, Abläufe der Validierung von Nutzereingaben zu automatisieren, damit Programmierfehlern schon im Keim entgegen gewirkt werden kann. Das Hauptaugenmerk sollte auf der Überprüfung der vom Nutzer eingegebenen Daten liegen. Eine interne Typkonvertierung in eine Zahl oder ein Datum empfiehlt sich sehr. Es ist auch sehr sinnvoll, Daten aus externen Quellen dahingehend zu überprüfen, dass nur bestimmte Zeichen erlaubt sind. Dies lässt sich komfortabel mittels Regulärer Ausdrücke lösen. Ein Nachteil dieser Technik ist eine erschwerte Wartbarkeit, da nicht alle Entwickler den Regulären Ausdrücken mächtig sind. Alle Nutzereingaben, bei denen nicht HTML-Elemente erwartet werden, sollten vorsorglich von HTML befreit werden. Zur Absicherung der Anwendung sollten immer Whitelists angewendet werden. Es müssen alle Formularfelder überprüft und abgesichert werden, also auch die für den Nutzer nicht sichtbaren Felder und Eingabefelder mit vordefinierten Inhalten wie Checkboxen, Radiobuttons, Comboboxen und Listen. Zudem müssen alle entgegen genommenen GET-, POST- und COOKIE-Parameter geprüft werden. Eine Gefahr geht auch von den META-Daten aus, die der Nutzer anpassen kann. Dieses sind mitunter UserAgent, AcceptedLanguage, Accept_Charset, Referrer. Mit Hilfe von einfachen Tools, die es meist als Plugin für den Firefox gibt (Webdeveloper Toolbar, Firebird, Modify Headers), kann der Client einfach die Übergabeparameter modifizieren und eventuelle Beschränkungen durch clientseitige Scripte oder Längenbeschränkungen umgehen.
Bei Verbindungen zu einer Datenbank mittels SQL muss zusätzlich beachtet werden, dass schädliche Zeichen und Phrasen entweder escaped oder aus dem Query entfernt werden. Weiterhin muss der Webserver so konfiguriert werden, dass dem Nutzer keinerlei technische Fehlermeldungen ausgegeben werden, sondern nur fachliche. Zudem sollte verhindert werden, dass mehrere Querys auf einmal ausgeführt werden können.

5 Quellen:

[CSP07] Christopher Kunz, Stefan Essner, Peter Prochaska: „PHP-Sicherheit“, 2.Auflage, dPunkt-Verlag, 2007

[Ove06] Ovelho, „SQL Injection“, 2006
http://www.youtube.com/watch?v=MJNJjh4jORY
Abruf: 02.09.2007

[RSn07] RSnake, „XSS Cheat Sheet”
http://ha.ckers.org/xss.html
Abruf: 01.09.2007

[DAT07] DATACOM Buchverlag GmbH, „XSS (cross site scripting), 2007
http://www.itwissen.info/definition/lexikon//_xssxss_xsscross%20site%20scriptingxss_xss.html
Abruf: 31.10.2007

[Hei07] Heise Zeitschriften Verlag, „XSS-Lücke in Ciscos Online-Hilfe“, 2007
http://www.heise.de/security/news/meldung/86994
Abruf: 31.10.2007

[W3C99] W3C, „RFC 2616“, 1999
http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
Abruf: 31.10.2007

[W3C00] W3C, “Extensible HTML version 1.0 Transitional DTD“, 2000
http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-transitional.dtd
Abruf: 31.10.2007

[Net00] Network Working Group, “HTTP State Management Mechanism”, 2000
http://www.ietf.org/rfc/rfc2965.txt
Abruf: 31.10.2007

[Moz07] Mozilla. “Firefox Add-ons“, 2007
https://addons.mozilla.org/de/firefox/
Abruf: 02.09.2007

[VFG07] Virtual Forge GmbH, „XSS“, 2007
http://www.virtualforge.de/vmovie/xss_selling_platform_v1.0.php
Abruf: 02.09.2007

[Mik07] Mikroökologie.de, “SQL-Injection”, 2007
http://www.mikroökonomie.de/SQL-Injektion.html
Abruf: 31.10.2007

[Wik07] Wikipedia-Community, “SQL-Injecition”, 2007
http://de.wikipedia.org/wiki/SQL-Injection
Abruf: 01.09.2007

6 Anhang

6.1 Schwachstellen von XSS, CSRF und SQL-Injection im Preisbock

Datei	Zeile	Modifikation	Art
File001.php	19,22 ,27 […]'$_SERVER[HTTP_REFERER]'[…]	Muss überprüft werden	SQL-Injection
File001.php	14, 19 $_REQUEST[PHPSESSID]	Muss validiert werden	SQL-Injection
File002.php	584,592,$_SERVER[HTTP_REFERER]	Validieren	SQL-Injection
File002.php	579,587,584$_REQUEST[PHPSESSID]	Validieren	SQL-Injection
File003.php	4,7 $_REQUEST[PHPSESSID]	Validieren	SQL-Injection
File004.php	124,141 $_REQUEST['public']	Verifizieren	SQL-Injection
File005.inc.php	39 user_email = '".$_GET['email']."'"	Query nur bei gültiger Mail ausführen	SQL-Injection
File005.inc.php	26 user_nicename = '$_GET[new_nick]'	Query nur bei gültigen Namen ausführen	SQL-Injection
In 3 Dateien wird der Referrer ungeprüft in $_SESSION['HTTP_REFERER'] geschrieben, aber nie ausgewertet			

Tabelle 1: Fehlerhafte Codesegmente mit nötiger Optimierung

6.2 Schon geänderte Elemente

Datei	Zeile	Modifikation	Art
File006.php	4 $id = $_GET['id'];	Casting nach int	SQL-Injection in line 10-12 Jeder kann jeden Bockmark löschen
File004.php	2 $bookmark_id = $_REQUEST['optional']; 100 $bookmark_id = $_REQUEST['ID'];	Casting nach int	File007.php (5x) Das schlägt durch bis zum Bildupload, wo es dann aufgrund eines ungültigen Bildes aber Fehler gibt.
File008.php	225 falscher RegEx Preis	^ hinzufügen	XSS
File008.php	230 falscher RegEx Adresse	^ hinzufügen	SQL-Injection

Tabelle 2: Fehlerhafte Codesegmente, die bereits berichtigt wurden

Es empfiehlt sich extra Variablen/Arrays zu nehmen, die validiert werden und nicht immer mit den Superglobalen zu arbeiten und diese zu validieren. Es wird ein assoziiertes Array $userdata empfohlen.

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich meine Praxisarbeit mit dem Thema

„XSS, CSRF, SQL-Injections“

ohne fremde Hilfe angefertigt habe,

dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und dass ich meine Praxisarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Ort, Datum	Unterschrift