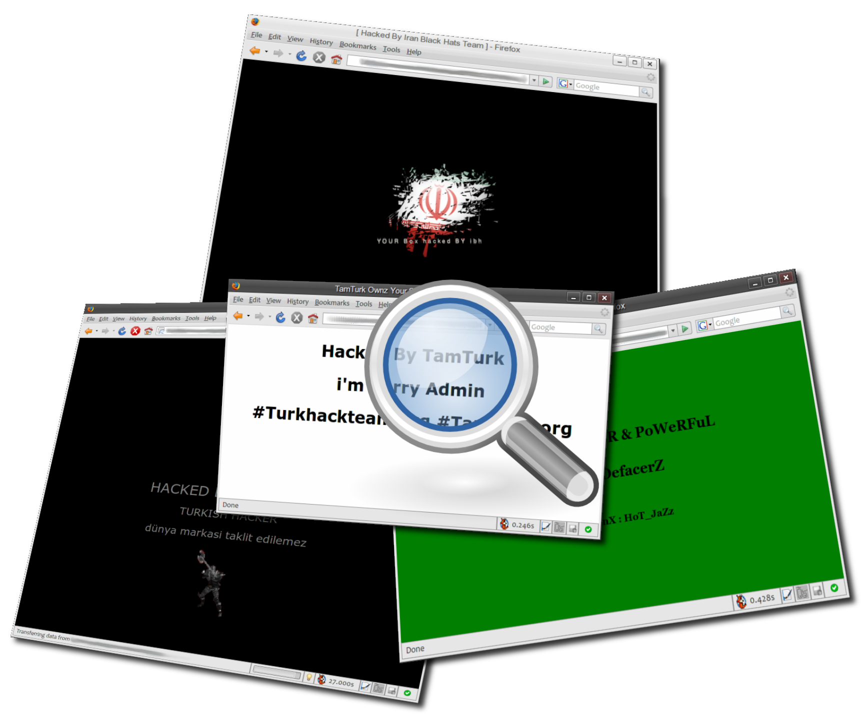
Bundes Oberstufen Realgymnasium Hermagor

Sicherheit in Webanwendungen

Armin Ronacher

Fachbereichsarbeit in Informatik



Inhaltsverzeichnis

I Allgemeines 4

1 Vorwort 5

2 Einführung 6
2.1 Die Programmiersprachen 6
2.1.1 PHP 7
2.1.2 Python 8
2.1.3 Ruby 9
2.1.4 JavaScript 9
2.2 Schematische Darstellung eines Zugriffes auf eine Webanwendung 9

II Arten von Sicherheitslücken 11

3 SQL-Injection 12
3.1 SQL-Injections in PHP Anwendungen 12
3.1.1 Die Sicherheitslücke demonstriert 12
3.1.2 Magic Quotes – ein gescheiterter Versuch 13
3.1.3 Lösung für die Datenbank MySQL 14
3.1.4 Lösung über MDB2 15
3.2 Situation unter Python 15
3.2.1 Lösung über die DBAPI 16
3.2.2 Lösung über SQLAlchemy 17
3.3 SQL-Injections unter Ruby 17

4 Cross-Site Scripting 18
4.1 Arten 18
4.1.1 Typ 0 - Lokales XSS 18
4.1.2 Typ 1 - Reflektiertes XSS 19
4.1.3 Typ 2 - Persistentes XSS 19
4.2 Fallbeispiele 20
4.2.1 Fallbeispiel 1: Forensoftware 20
4.2.2 Fallbeispiel 2: MySpace 23
4.2.3 Fallbeispiel 3: Phishing 23
4.3 Cross-Site Scripting verhindern 24
4.3.1 PHP 24
4.3.2 Python 25
4.3.3 Ruby 25

5 Cross-Site Request Forgery 26
5.1 Beispiel 26
5.2 Angriffsvektoren 26
5.2.1 Unterschieben einer URL 27

5.2.2 Cross-Site Scripting . . . 27
5.2.3 Lokaler Angriff . . . 27
5.3 Gegenmaßnahmen . . . 28
5.4 Alternative Lösungen . . . 28
5.5 Ungeeignete Lösungen . . . 29

6 Server Side Code Execution **30**
6.1 Funktionspointer unter PHP . . . 30
6.2 Funktionspointer unter Python . . . 31
6.3 Funktionspointer unter Ruby . . . 32

7 Session-Hijacking **33**
7.1 Session ID Streuung . . . 34

8 Directory Traversal **35**
8.1 PHP Nullbyte Problematik . . . 35
8.2 Situation unter Python . . . 36
8.3 Situation unter Ruby . . . 37

9 Konzeptionelle Lücken **38**
9.1 Hash URLs als Zugriffsschutz . . . 38
9.2 Passwortspeicherung . . . 39
9.3 Cookie Authentifizierung . . . 41

10 Information Disclosure **42**
10.1 Automatic Directory Indexing . . . 42
10.2 Information Leakage . . . 42
10.2.1 Entwicklungsmodus . . . 42
10.2.2 Testbetrieb . . . 43
10.2.3 Produktivbetrieb . . . 43

III Statistiken, Auswirkungen und Abschließendes **44**

11 Statistiken **45**
11.1 phpBB 2.x . . . 45
11.2 MediaWiki 1.x . . . 45
11.3 Online Portale . . . 46

12 Schlusswort **48**
12.1 Protokoll . . . 48
12.2 Lizenz . . . 48
12.3 Hinweis zu den Zitaten . . . 49
12.4 Erklärung . . . 50

Teil I

Allgemeines

1 Vorwort

Als ich Ende 2004 bei der Deutschen ubuntu Community ubuntuusers[1] als Portalentwickler tätig war, hatte ich das erste Mal mit Webanwendungen zu tun. Allerdings begann ich mich erst Anfang 2005 mit Sicherheit in Webanwendungen zu beschäftigen, als ich eine Nachfolgesoftware für das ubuntuusers Forensystem zu entwickeln begann.

Nachdem ich für die Matura die Möglichkeit hatte, eine Fachbereichsarbeit in Informatik zu schreiben packte ich die Gelegenheit beim Schopf und begann mich intensiv zu diesem Thema zu informieren. Rückwirkend bin ich sehr froh, dass ich dies noch vor der Fertigstellung meiner Webanwendung getan zu haben, ich hätte sicher über einen Teil der Risiken nichts gewusst.

Bedanken möchte ich mich bei meinen Eltern die mich bei der Auswahl des Themas meiner Fachbereichsarbeit beraten haben, den Personen von #python.de[2] – besonders Alexander Schremmer und Georg Brandl, die mich immer wieder auf Informationen aufmerksam gemacht haben, die ich wohl ansonsten übersehen hätte. Außerdem gilt mein Dank auch den übrigen Mitgliedern des Pocoo Teams[3] für deren Unterstützung.

Weiters möchte ich mich bei den Entwicklern der einzelnen Projekte (OpenOffice.org, LaTeX, phpBB, MediaWiki, GIMP, ubuntu, Python, PHP und Ruby) bedanken, dass sie diese quelloffen und kostenlos bereitstellen. Ohne diesen Umstand wäre meine Fachbereichsarbeit nicht möglich gewesen.

Mein besonderer Dank gilt aber meiner Informatik Professorin Frau, Elisabeth Jochum, die mich in den vier Jahren am BORG Hermagor begleitet und meine Leidenschaft an der Informatik immer unterstützt und gefördert hat und es mir ermöglichte diese Fachbereichsarbeit zu verfassen.

[1] ubuntuusers.de - `http://www.ubuntuusers.de/`
[2] IRC Channel am Freenode IRC Server, `irc.freenode.net`.
[3] Pocoo Team: `http://trac.pocoo.org/wiki/PocooTeam`

2 Einführung

Kaum eine Technologie wurde so wichtig für unser modernes Computerzeitalter wie das Internet. Mit Skriptsprachen wie PHP, Perl, Python oder Ruby wurde die Erstellung von dynamischen Webseiten[4] ein Leichtes und kaum noch eine Webseite ist komplett statisch aufgebaut. Allerdings wird hierbei immer wieder der Sicherheitsaspekt vergessen. Eine dynamische Webseite birgt nämlich enorme Sicherheitsrisiken, über die man sich bewusst sein sollte. Eine Sicherheitslücke in einer Webanwendung ist nicht nur ein Problem für den Anwendungsentwickler und/oder den Serverbetreiber, sondern für jeden Internet Benutzer.

Die Tragweite einer Sicherheitslücke ist unterschiedlich. Manche werden „nur“ zum Spam[5] Versand genutzt, andere können einem Angreifer helfen, einen kompletten Server zu „übernehmen“. Auch für den Besucher einer Webseite kann eine gehackte Webseite ein Problem werden. Wenn etwa der Server einer Bank unter die Kontrolle eines Angreifers gelangt, können Kontendaten gestohlen werden, ohne dass der Benutzer etwas davon bemerkt.

Dabei ist es gar nicht so schwierig, eine sichere Webanwendung zu erstellen, sofern man als Programmierer weiß, wo sich Sicherheitslücken verstecken können. Viele Entwickler wissen nicht, dass der Code, den sie geschrieben haben, große Sichheitsheitslöcher aufweisen.

Das Ziel dieser Fachbereichsarbeit ist es, Fehler zu analysieren und mögliche Lösungen aufzuzeigen. Sie beschränkt sich serverseitig auf die Programmiersprachen PHP, Python und Ruby, welche aus folgenden Gründen gewählt wurden:

- **PHP**, nicht nur weil es die verbreiteste Skriptsprache für Webanwendungen ist, sondern auch weil sie leicht zu erlernen ist. Sie läd allerdings zu unsauberer und unsicherer Programmierung ein.
- **Python**, weil momentan eine Unzahl an Webframeworks[6] in Entwicklung ist und sich diese Sprache in Zukunft warscheinlich mehr im Web verbreiten wird.
- **Ruby**, weil mit Ruby on Rails und Nitro zwei sehr einfach zu erlernende Frameworks existieren, die das erstellen von Webanwendungen zu einem Kinderspiel machen.

Sämtliche Beispiele in dieser Fachbereichsarbeit wurden unter „ubuntu[7] edgy eft 6.10„ getestet. Die Programmversionen verteilen sich wie folgt:

- *PHP*: PHP 4.4.2-1.1; built: Jun 20 2006 02:33:21); Zend Engine v1.3.0
- *Python*: Python 2.4.3 (#2, Aug 25 2006, 17:37:59); GCC 4.1.2 20060817 (prerelease) (Ubuntu 4.1.1-11ubuntu1) on linux2
- *Ruby*: ruby 1.8.4 (2005-12-24); i486-linux

2.1 Die Programmiersprachen

Im Folgenden werden die Programmiersprachen kurz beschrieben, die in dieser Arbeit verwendet werden. Zusätzlich zu den oben bereits erwähnten serverseitigen Sprachen kommt noch JavaScript

[4] Von einer dynamischen Webseite wird gesprochen, wenn auf der Serverseite ein Programm Benutzerdaten entgegennimmt und verarbeitet oder abhängig von Datenbank Daten Ausgaben erzeugt. Im Webserver ausgeführter JavaScript Code ist hiervon nicht betroffen.

[5] Als Spam bezeichnet man unterwünschte Werbemails.

[6] Ein Webframework stellt für den Programmierer Hilfsfunktionen und/oder Klassen zur Verfügung und vereinfacht die Entwicklung einer Webanwendung. Beispiele für Webframeworks sind „Ruby on Rails“, „django“ oder „struts“

[7] ubuntu ist eine freie und auch kostenlose Linuxversion

hinzu, die als einzige Clientseitige für die Webentwicklung eine tragende Rolle spielt.

2.1.1 PHP

Die PHP Programmiersprache wurde 1995 von Rasmus Lerdorf entwickelt. Das Acronym stand zum Zeitpunkt der Entwicklung für *Personal Home Page Tools* und stellte eine Sammlung von Perl-Skripten dar. Bald darauf entwickelte Lerdorf seine Skriptsammlung als eigene Programmiersprache in C neu und veröffentlichte sie als PHP/FI (*Form Interpreter*). Sie war Perl sehr ähnlich aber eingeschränkter und inkonsistent, dass heißt es gibt unter anderem keinen einheitlichen Weg, um Funktionen und Klassen zu bennennen.

1998 wurde PHP 3 von Andi Gutmans und Zeev Suraski neu geschrieben, unter der Begründung, dass PHP/FI für *eCommerce* ungeeignet sei. Zusammen mit der Neuumsetzung der Software wurde die Abkürzung „PHP" zu *PHP Hypertext Preprocessor* abgeändert.

In der Folge wurde von ihnen die Firma „Zend Technologies Ltd." gegründet, die die Entwicklung von PHP weiterführte. Für PHP 4 wurde dann die Zend Engine entwickelt die nun den Kern von PHP darstellt.

An PHP wurde vor allem in jüngerer Zeit häufig Kritik geübt. Nicht ganz unbegründet, immerhin haben sich in der Entwicklung der Sprache einige Inkonsistenzen und Probleme eingeschlichen. PHP bietet unter anderem keinen einheitlichen Weg um auf Datenbanken zuzugreifen. MySQL wird beispielsweise vollkommen anders angesteuert als SQLite, was es Programmierern schwierig macht, eine Anwendung für mehr als eine Datenbank zu entwerfen. Auch wurden den einzelnen Datenbankmodulen keine brauchbaren Funktionen zum Absichern vor SQL Injections bereitgestellt, weswegen alle großen PHP Anwendungen ihre eigenen Datenbank Zwischenschicht mitbringen, die dies übernimmt.

Bis PHP 5 gab es auch keine Möglichkeit, um gezielt Fehler auszulösen und abzufangen wie dies mit den Programmiersprachen Java, Python, Ruby, C++ und anderen schon lange möglich war.

Auch das *register_globals* Verhalten, das URL[8]- und Formularparameter[9] automatisch in den globalen Namensraum einbindet wurde heftig kritisiert und ist mittlerweile in der Standardkonfiguration deaktiviert. Doch da immer noch viele Anwendungen auf dieses Verhalten aufbauen, ist diese Option auf Webhosts sehr häufig aktiviert. Da in PHP sehr gerne mit nicht initialisierten Variablen gearbeitet wird treten sehr häufig Probleme mit *register_globals*.

Einiger dieser Kritikpunkte (Exceptions, fehlende einheitliche Datenbankschicht) werden entweder mit PHP 5 oder dem PEAR System behoben. PEAR stellt eine zentrale Platform für Erweiterungen dar, die in PHP geschrieben sind. Auf Grund der Tatsache, dass PHP keine Module oder Namespaces bereitstellt, kann es allerdings zu so genannten *Namespace Clashes* kommen. Das heißt, dass wenn zwei verschiedene PHP Dateien, die die selbe Funktion, Klasse, Konstante oder Variable im globalen Namensraum ablegen einen Fehler erzeugen oder den Ablauf der Anwendung stören.

Zusätzlich unterstützt PHP kein Unicode[10]. Beispielsweise funktionieren String Funktionen wie *strto-*

[8] Als URL Parameter werden Daten nach dem ersten Fragezeichen in einer URL angesehen. In der URL `http://www.example.com/?foo=bar` wäre „foo=bar" der URL Parameter

[9] Formularparameter werden für den Webserver ähnlich behandelt wie URL Parameter, die Übermittlung erfolgt aber im Hintergrund über die HTTP Transportschicht. Genauere Informationen betreffend des HTTP Protokolls finden sich im RFC 2068[GROUP 1997].

[10] Der Unicode Standard definiert eine Unzahl von Schriftzeichen (sogenannten Glyphen) und definiert, wie eine Anwendung mit diesen umzugehen hat. Unicode wurde als Antwort auf die unzähligen Zeichensätze entworfen die Datenaustausch fast unmöglich machten. Lange Zeit war es beispielsweise nicht möglich japanische und russische Text in einem Dokument

lower oder *strlen* bei nicht ASCII[11] Symbolen nicht wie erwartet.

2.1.2 Python

Python wurde Anfang 1990 von Guido van Rossum am „Centrum voor Wiskunde en Informatica“ in Amsterdam entwickelt. Ursprünglich für das verteilte Betriebssystem Amoeba entworfen, ist es mittlerweile für viele Plattformen, darunter Windows, Unix, OS2 und anderen Betriebssystemen erhältlich.

Das hoch gesteckte Ziel für Python war es, eine möglichst einfache und übersichtliche Programmiersprache zu entwickeln, die leicht zu verstehen ist, aber dennoch dem Fortgeschrittenen eine Vielzahl an Möglichkeiten zu geben.

Python erreicht dies durch die Reduktion des Sprachumfangs auf wenige Schlüsselwörter und Funktionen, sowie einer Syntax, die lögische Blöcke an der Einrückung erkennt, als auch einem leistungsstarken Modulsystem, das Module dynamisch nachladen kann. Auf Grunds der umfangreichen Standardbibliothek wird oft scherzhaft „Python comes with Batteries“ geschrieben, doch tatsächlich hat Python noch viel mehr zu bieten, als die mitgelieferte Standardbibliothek: Im Cheeseshop (das Python Paket Platform) finden sich zum aktuellen Zeitpunkt über 1500 Module.

Python unterstützt Unicode 4.0 (mit Python2.5) und Unicode 3.2. In den offiziellen Windows Builds werden Multibyte Chars[12] mit zwei Zeichen unterstützt, wer 4 Byte Unterstützung benötigt, kann diese beim Compilieren aktivieren.

Neben der originalen C-Implementierung gibt es auch Python Implementierungen die in Java (Jython), .net (IronPython) oder Python selbst (PyPy) geschrieben wurden. Letztere wurde in einem lose definierten Subset von Python programmiert welches problemlos von einem Translator in C, .net oder anderen Quellcode umgewandelt werden kann. So ist PyPy in der Lage sich selbst zu übersetzen. Das Ziel auf lange SIcht ist es, die schnellste Python Implementierung zu schreiben, die auch leichter zu warten ist. Als positiver Seiteneffekt ist es zum Beispiel möglich Python Quellcode in Javascript umzuwandeln.

Für die Webentwicklung mit Python, stehen neben großen Frameworks wie Django, TurboGears oder Pylons auch ganze Application Server wie Zope zur Verfügung. Aber auch kleine Lösungen wie Colubrid, CherryPy und RhubarbTart ermöglichen es schnell und einfach Webanwendungen zu entwickeln. Damit die Kommunikation zwischen Framework und Webserver funktioniert, bauen diese auf den WSGI[EBY 2004] Standard auf, welcher eindeutig definiert, wie eine Anwendung mit einem Gateway zu sprechen hat welches dann den Webserver ansteuert. Dies erlaubt einem Serveradministrator Python Programme nicht nur mit CGI oder mod_python auszuführen, sondern auch mit jeder anderen unterstützten Schnittstelle. Je nach Anwendungsbedarf kann so zwischen verschiedenen Systemen umgeschaltet werden ohne, dass etwas am Quellcode der Anwendung/Frameworks etwas verändert werden müsste.

abzuspeichern. Nachteil an Unicode ist die erhöhte Aufwand für einen Anwendungsprogrammierer und die Inkompatibilität zu älteren Programmen

[11] Zeichensatz der die lateinischen Buchstaben ohne Akzente oder andere diakretische Zeichen beinhaltet mit einigen Sonderzeichen und Zahlen beinhaltet.

[12] Auf Grund der hohen Zeichenanzahl im Unicode Zeichensatz werden mehrere Bytes zu einem Schriftzeichen kombiniert. Solche Zeichen nennt man Multibyte Chars.

2.1.3 Ruby

Ruby ist das von Yukihiro Matsumoto entwickelte Konglomerat aus Perl, Lisp und Python. Die Entwicklung startete 1993 und hat sich – wohl auch wegen der anfänglich nur japanischen Dokumentation – sehr schleppend verbreitet. Richtig bekannt wurde es erst 2005 als das Webframework „Ruby on Rails“ veröffentlicht wurde, das auf Ruby aufbaut.

Das Ziel von Ruby war es, objektorientierter zu sein, als es Python zum damaligen Zeitpunkt war und gleichzeitig der Perl Idee zu folgen, die mehr als einen Weg für die Lösung anbietet.

Auf den ersten Blick zeichnet sich Ruby durch das fast vollständige Fehlen von Klammern aus. Methoden können auch ohne Klammern aufgerufen werden, weswegen man sie auch nur sehr schwer einer anderen Methode als Parameter übergeben kann. Dies ist jedoch dank anonymer Blöcke kaum nötig. Da die Ruby Syntax sehr komplex ist und sehr schnell zu unübersichtlichen Code einläd, wird empfohlen, Klammern für Funktionen mit mehr als einem Parameter zu verwenden.

Zum aktuellen Zeitpunkt steht Ruby 1.9 bzw Ruby 2 vor der Fertigstellung, welches Unicode Unterstützung und eine Virtuelle Maschine[13] vorsieht, die es möglich macht, Bytecode[14] zu generieren.

Die Ruby Community selbst ist meist auf Blogs, IRC Channel und Mailinglisten beschränkt und zeichnet sich durch ihre ganz eigene Kultur und ihren angenehmen Umgangston aus.

2.1.4 JavaScript

JavaScript ist wie auch Python, Ruby oder PHP eine interpretierte Skriptsprache. Im Gegensatz zu den vorher genannten ist JavaScript hautsächlich auf Webbrowsern vorhanden, auch wenn es sich theoretisch als Serversoftware einsetzten lässt. Fast jeder moderne Browser bringt einen Interpreter für JavaScript 1.3 oder höher mit. Die aktuelleste Version (1.7) nimmt viele Anleihen von Python, wie etwa das Generator Konzept[15].

2.2 Schematische Darstellung eines Zugriffes auf eine Webanwendung

Um sich ein besseres Bild von Webanwendungen machen zu können soll im Voraus kurz erklärt werden, wie ein Zugriff auf eine Webanwendung erfolgt. Grundsätzlich fordert der Benutzer mit einem Webbrowser Daten von einem Server ab. Dieser Zugriff läuft über das HTTP Protokoll ab. Der Server überprüft die Anfrage des Benutzers und leitet diese über ein Webserver Interface wie CGI an den Interpreter der eingesetzten Programmiersprache weiter. Dieser ruft dann die Webanwendung auf und leitet die Ausgabe an den Webbrowser zurück. Die Abbildung 1 veranschaulicht dies.

[13] Eine Virtuelle Maschine übersetzt platformunabhängigen Bytecode in für die jeweilige Platform verständlichen Maschinencode. Bekannte Beispiele sind Java oder das .net System

[14] Bytecode ist eine Zusammenstellung von internen Befehlen für eine Virtuelle Maschine.

[15] Ein Python Generator ist eine Funktion die ein iterierbares Objekt zurückliefert

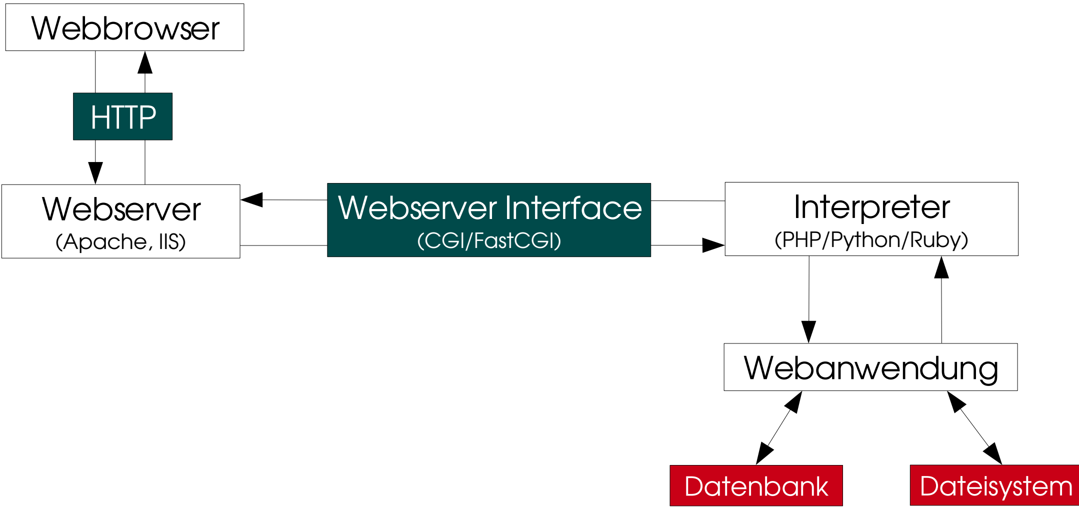


Abbildung 1: Schematische Darstellung eines HTTP Requestes

Teil II

Arten von Sicherheitslücken

3 SQL-Injection

Die SQL[16] Injection ist die warscheinlich am einfachsten vermeidbare Sicherheitslücke, sofern man die von der Datenbank bereitgestellten Escaping[17]-Werkzeuge auch wirklich nutzt.

Die Grundidee einer SQL Injection ist es, einen SQL Befehl so zu verändern, dass vom Angreifer injezierte oder veränderte Befehle von der Webanwendung im Kontext der Datenbank ausführt werden. Führt dies meist „nur“ zum Löschen, Manipulieren oder Hinzufügen von Daten, kann der Angreifer mit etwas Wissen über die verwendete Datenbanksoftware auch Systembefehle am Datenbankserver ausführen.

Grundsätzlich kann eine SQL Injection überall dort auftreten, wo vom Benutzer übermittelte Parameter ungefiltert in einen SQL Befehl eingebaut werden. Die Gefahren werden dabei oft unterschätzt oder sind erst gar nicht bekannt.

3.1 SQL-Injections in PHP Anwendungen

SQL-Injections in PHP Programmen sind außerordentlich häufig. Dies hat in erster Linie damit zu tun, dass es – wie bereits erwähnt – keine eigene Datenbank Schicht gibt. Zusätzlich sind in der offiziellen PHP Dokumentation in den Hilfeseiten zu den Datenbank Abfragefunktionen keine Hinweise angebracht, wie man Parameter in SQL Befehle einbauen kann.

Die logische Konsequenz daraus ist, dass viele Anwendungsentwickler einfach die Variablen direkt in die Strings einbauen, die dann an den Datenbankserver weitergeleitet werden — mit verherrenden Folgen, wie im nächsten Beispiel demonstriert wird.

3.1.1 Die Sicherheitslücke demonstriert

Dieses Beispiel beinhaltet eine sehr schwere Sicherheitslücke, die es einem Angreifer erlauben würde sich als Administrator anzumelden (Siehe Programmbeispiel 1).

Programmbeispiel 1: PHP SQL-Injection

```
<?php
/**
 * Diese Methode ueberprueft die Logindaten und gibt im
 * Erfolgsfall die Benutzer ID zurueck.
 * Im Fehlerfall wird -1 zurueckgegeben (zb wenn die
 * Benutzerdaten nicht uebereinstimmen.
 */

function check_login($username, $password) {
    global $db;
    $hash = md5($password);
    $sql = "select user_id from users
            where username = '$username' and password = '$hash';";
    if ($result = mysql_query($sql, $db))
```

[16]SQL ist eine Abfragesprache, die genutzt wird, um Daten aus einer Datenbank auszulesen oder in diese einzufügen.

[17]Mit Escaping bezeichnet man das Maskieren von unsicheren Zeichen. Wird Quellcode dynamisch zur Laufzeit erzeugt (zum Beispiel um SQL Befehle zu generieren) müssen beispielsweise die Anführungszeichen die Start- und Endposition einer Zeichenkette anzeigen durch Backslashes, oder bei ANSI SQL kompatiblen Datenbanken durch verdoppeln abgesichert werden.

```
        if (($row = mysql_fetch_assoc($result)) !== false)
            return $row['user_id'];
    return -1;
}
?>
```

Grundsätzlich erscheint diese Funktion korrekt. Sie nimmt Benutzername und Passwort entgegen und gibt entweder eine gültige Benutzernummer (User ID) oder -1 für den Fehlerfall zurück. In der Regel würde man dies mit einer zweiten Funktion koppeln, die die Session[18] für den Benuzter startet und die „Session ID“ als Cookie[19] zum Client senden würde.

Doch wenn nun die übermittelten Variablen „username“ und „password“ direkt an diese Funktion übergeben werden, kann sich nun ein Angreifer problemlos als Administrator anmelden, ohne dessen Passwort kennen zu müssen.

Der Fehler liegt nämlich daran, dass die Variablen ohne vorherige Absicherung in den SQL Befehl eingebaut werden. Wenn nun ein Angreifer einen Apostrophen einfügt, kann er aus der Zeichenkette „ausbrechen“ und den SQL Befehl für seine Zwecke verändern.

Der Angreifer gibt nun also folgende Werte in das Login Formular ein. (Siehe Programmbeispiel 2)

Programmbeispiel 2: Präparierte Werte

```
username=Administrator'--
password=
```

Da in der Funktion selber die Werte nicht überprüft werden und die Daten auch direkt ohne Prüfung an jene übergeben werden, hat nun der Angreifer leichtes Spiel. Er versucht sich nun als „Administrator“ anzumelden, aber anstatt ein Passwort einzugeben, schließt er nun die Zeichenkette und kommentiert mit zwei Bindestrichen die Passwortüberprüfung aus.

Der SQL Befehl der nun an die Datenbank übergeben wird, sieht so aus. (Siehe Programmbeispiel 3)

Programmbeispiel 3: Veränderter SQL Befehl

```
select user_id from users
 where username = 'Administrator'--' and password = 'd41d...427e';
```

Damit würde nun der Rückgabewert des SQL Befehls immer die Benutzer Nummer sein und nie eine leere Menge, die ein falsches Passwort signalisiert. Damit wurde die Passwortüberprüfung erfolgreich außer Kraft gesetzt.

3.1.2 Magic Quotes – ein gescheiterter Versuch

PHP kennt seit sehr frühen Versionen einen Schalter in der systemweiten Konfigurationsdatei „php.ini“, der SQL Injections verhindern sollte. Dies Methode wird „Magic Quotes“ genannt und sehr problematisch. Nicht nur hat sie ihr Ziel nicht erreicht, sondern war auch kontrakproduktiv, da sie die Entwicklung

[18] Mit „Session“ wird eine Benutzersitzung bezeichnet. Einem Benutzer, der sich auf einer Webanwendung anmeldet, wird in der Regel eine eindeutige Nummer (eine sogenannte Session ID, oder SID) zugewiesen, mit der er für die Anwendung zu einem späteren Zeitpunkt wieder erkennbar ist.

[19] Cookies können kleine Datenmengen von bis zu 4KB speichern, die bei jedem Abruf zum Webserver gesendet werden

von Webanwendungen stark erschwerte. Dies ist auch der Grund dafür, dass mit PHP 6 wird „Magic Quotes“ wieder aus dem Sprachumfang verschwinden.

3.1.3 Lösung für die Datenbank MySQL

Im Gegensatz zum gescheiterten Versuch von „Magic Quotes“ gibt es für jede Dankenbank geeignete Methoden um SQL-Injections zu verhindern. Hier wird MySQL als Beispiel angeführt. Für MySQL steht hierzu die Funktion *mysql_real_escape_string* zur Verfügung, die allerdings nur für Zeichenketten funktioniert. Da PHP aber schwach typisiert ist, werden Zahlen bei Bedarf in Zeichenketten umgewandelt. Dies stellt in den meisten Fällen kein Problem dar. Werden aber *NULL* Werte in den SQL Befehl eigebaut, verfehlt *mysql_real_escape_string* sein Ziel. Sollte die Absicht bestehen, *NULL* Werte an die Datenbank zu senden ist eine Sonderbehandlung notwendig.

Im vorigen Beispiel wurde MySQL direkt angesprochen, darauf aufbauend sieht nun der Quellcode folgend aus. (Siehe Programmbeispiel 4)

Programmbeispiel 4: Kein weiterer Angriffspunkt für die SQL-Injection

```
<?php
/**
 * Diese Methode ueberprueft die Logindaten und gibt im
 * Erfolgsfall die Benutzer ID zurueck.
 * Im Fehlerfall wird -1 zurueckgegeben (zb wenn die
 * Benutzerdaten nicht uebereinstimmen.
 */

function check_login($username, $password) {
    global $db;
    $username = mysql_real_escape_string($username, $db);
    $hash = md5($password);
    $sql = "select user_id from users
            where username = '$username' and password = '$hash';";
    if ($result = mysql_query($sql, $db))
        if (($row = mysql_fetch_assoc($result)) !== null)
            return $row['user_id'];
    return -1;
}
?>
```

Auffallend ist, dass nur der „$username“ Wert abgesichert wird. Zwar könnte man der Vollständigkeit halber auch „$hash„ maskieren; da dieser Wert aber der Rückgabewert der Funktion *md5*[20] ist, der per Definition nur aus Buchstaben von A bis F und Zahlen besteht ist dies nicht nötig.

Derselbe Angriff würde nun diesen SQL Befehl generieren. (Siehe Programmbeispiel 5)

Programmbeispiel 5: Gesicherter SQL Befehl

```
select user_id from users
 where username = 'Administrator''--' and password = 'd41d...427e';
```

[20]Hierbei sei darauf hingewiesen, dass dieses Beispiel eine potientelle Sicherheitslücke beinhaltet, weil das Password nicht „gesalzt“ wird. Auf diese Problematik wird im Kapitel „Konzeptionelle Lücken“ eingegangen.

Damit ist der Angriff wirkungslos, weil der Angreifer den Befehl nicht manipulieren konnte.

3.1.4 Lösung über MDB2

Der PHP Community sind die Probleme rund um Datenbanken mit PHP bekannt. Daher ist es nicht verwunderlich, dass es Datenbank Abstraktionsschichten[21] aller Art gibt. Eine der wichtigsten ist sicherlich „MDB2“.

Unter der Verwendung von MDB2 würde dieses Beispiel wie folgt aussehen:

Programmbeispiel 6: MDB2 Lösung

```
<?php
/**
 * Diese Methode ueberprueft die Logindaten und gibt im
 * Erfolgsfall die Benutzer ID zurueck.
 * Im Fehlerfall wird -1 zurueckgegeben (zb wenn die
 * Benutzerdaten nicht uebereinstimmen.
 */

function check_login($username, $password) {
  global $db;
  $query = $db->prepare("select user_id from users
        where username = ? and password = ?;",
        array('string', 'string'), MDB2_PREPARE_RESULT);
  $result = $query->execute($username, md5($password));
  if (!PEAR::isError($result)) {
    if ($row = $result->fetchRow(MDB2_FETCHMODE_ASSOC)) {
      return $row['user_id'];
    }
  }
  return -1;
}
?>
```

3.2 Situation unter Python

SQL-Injections unter Python lassen sich sehr leicht vermeiden da die *Python Database API Specification* Escaping als zentrales Element vorsieht.

Auch werden kaum noch SQL Befehle selbst geschrieben, sondern auf ORM Systeme wie „SQLObject“, „SQLAlchemy“ zurückgegriffen, die das Escaping von Parametern intern lösen. Auch das Webframework „Django“ bringt seine eigene Datenbankschicht mit ORM mit.

Zunächst wird hier ein Beispiel aufgezeigt, das die selbe Sicherheitslücke aufweist wie in der PHP Version. (Siehe Programmbeispiel 7)

Programmbeispiel 7: SQL-Injection in Python

[21] Datenbank Abstraktionsschichten verhindern solche Probleme und erlauben es dem Entwickler verschiedene Datenbanken auf die selbe Art anzusprechen.

```
import md5

def check_login(username, password):
    """
    Diese Methode ueberprueft die Logindaten und gibt im
    Erfolgsfall die Benutzer ID zurueck.
    Im Fehlerfall wird ``None`` zurueckgegeben (zb wenn die
    Benutzerdaten nicht uebereinstimmen.
    """
    hash = md5.new(password).hexdigest()
    cursor = db.cursor()
    cursor.execute("""
        select user_id from users
         where username = '%s' and password = '%s';
    """ % (username, hash))
    row = cursor.fetchone()
    if row:
        return row[0]
```

Auch hier das selbe Problem, die Daten werden unüberprüft in den String eingearbeitet.

3.2.1 Lösung über die DBAPI

Die Lösung ist allerdings sehr einfach, man muss lediglich ein Zeichen ändern und die Apostrophen entfernen. Anstatt die Werte mit Hilfe des Modulo Operators in den String einzufügen, übergibt man das Tuple mit den Werten als zweiten Parameter der „execute“ Funktion die dies an den Datenbanktreiber weiterleitet.

Zu beachten ist, dass manche Datenbanktreiber kein „%s“ als Platzhalter verwenden, sondern in vielen Fällen Fragezeichen oder andere Symbole. (Siehe Programmbeispiel 8)

Programmbeispiel 8: Geschlossene SQL-Injection Sicherheitslücke in Python

```
import md5

def check_login(username, password):
    """
    Diese Methode ueberprueft die Logindaten und gibt im
    Erfolgsfall die Benutzer ID zurueck.
    Im Fehlerfall wird ``None`` zurueckgegeben (zb wenn die
    Benutzerdaten nicht uebereinstimmen.
    """
    hash = md5.new(password).hexdigest()
    cursor = db.cursor()
    cursor.execute("""
        select user_id from users
         where username = %s and password = %s;
    """, (username, hash))
    row = cursor.fetchone()
    if row:
        return row[0]
```

Dies stellt auch sicher, dass Zahlen als solche der Datenbank übergeben werden und der spezielle

NULL-Wert nicht als String gesendet wird.

3.2.2 Lösung über SQLAlchemy

SQLAlchemy ist eine sehr gern genutzte Datenbankschicht in Python die SQL Befehle dynamisch erstellt. (Siehe Programmbeispiel 8)

Programmbeispiel 9: SQLAlchemy Lösung

```
import md5
from sqlalchemy import *
from mytables import users

def check_login(username, password):
    """
    Diese Methode ueberprueft die Logindaten und gibt im
    Erfolgsfall die Benutzer ID zurueck.
    Im Fehlerfall wird ``None`` zurueckgegeben (zb wenn die
    Benutzerdaten nicht uebereinstimmen.
    """
    result = engine.execute(sqlalchemy.select([users.c.user_id],
        (users.c.username == username) &
        (users.c.password == md5.new(password).hexdigest())
    ))
    row = result.fetchone()
    if row:
        return row[0]
```

3.3 SQL-Injections unter Ruby

Unter Ruby stellt sich die Situation etwas anders dar. Zwar kann auch dort das Problem einer SQL-Injection auftreten, allerdings weniger in Webanwendungen, wo fast ausschließlich ActiveRecord für Datenbankverbindungen genutzt wird, bei dem Tabellen und Relationen auf Klassen abgebildet werden.

4 Cross-Site Scripting

Das *Cross-Site Scripting* (XSS[22]) ist eine oft unterschätzte Sicherheitslücke. Die Idee dahinter ist, dass auf irgendeine Weise JavaScript Code auf einer fremden Webseite zur Ausführung gebracht wird. In der Regel passiert dies, wenn vom Benutzer übermittelte Daten einfach in den HTML[23] Quelltext eingefügt werden ohne, dass die speziellen Zeichen „&“, „<“ und „>“ durch ihre Enititäten „&“, „<“ und „>“ ersetzt werden. Ist erst einmal das Skript inkludiert, steht dem Angreifer der Weg offen, dies zu seinem Vorteil zu nutzen.

Von einfachem Stören des Besuchers eines Weblogs mit lästigen Hinweis Dialogen bis zum Programmieren einer DDoS-Attacke[24], die von den Besuchern einer Webseite losgetreten wird, ist alles möglich.
Der Name „Cross-Site Scripting“ klingt irreführend, das hat auch sein Namensgeber Marc Slemko[25] festgestellt:

> *This issue isn't just about scripting, and there isn't necessarily anything cross-site about it. So why the name? It was coined earlier on when the problem was less understood, and it stuck. Believe me, we have had more important things to do than think of a better name.*

Mit XSS ist gemeint, dass es möglich ist, eigenen HTML-Quelltext und eigenen JavaScript Quellcode in eine Webseite einzubetten. Häufig sind es URL Parameter, die nicht weiter überprüft und wieder in den HTML Text eingefügt werden.

Sehr häufig betrifft dies Suchmasken oder 404 Fehlerseiten, bei denen der Name der Seite angezeigt wird. Eher seltener sind XSS Lücken bei allgemeinen Formularen zu finden, wie Gästebüchern oder Ähnliches.

Der Angreifer hat zwei Methoden zur Verfügung: Entweder er manipuliert eine URL so, dass sie seinen Code ausführt, oder er speichert den Angriffscode auf der Webseite, beispielsweise in einem Kommentar zu einem Weblogeintrag, sofern die Kommentarfunktion anfällig für Sicherheitslücken ist.

4.1 Arten

Bei XSS unterscheidet man zwischen drei Haupttypen. Je nachdem, auf welche Art der Schadcode in die im Browser angezeigte Webseite gelangt, spricht man von Typ 0, 1 oder 2:

4.1.1 Typ 0 - Lokales XSS

Diese Art wird Lokales XSS (selten auch DOM[26] basierendes XSS) genannt, da der Schadcode nie auf den Server gelangt.

Das lokale XSS ist eine spezielle Art des Cross-Site Scripting und fällt deswegen aus der Nummerierung. Gemeint ist, dass sämtlicher Quellcode im Browser dies Benutzers ausgeführt wird. Damit dieser

[22] Die Abkürzung XSS wurde gewählt um Verwechslungen mit den Cascading Style Sheets und dem Content Scrambling System zu vermeiden, die beide ebenfalls CSS abgekürzt werden.
[23] Die Hypertext Markup Language (HTML) ist eine Auszeichnungssprache, die genutzt wird um Webdokumente zu beschrieben. Bis HTML 4 wurde SGML als Basis für die Sprache verwendet, mit den Nachfolgern XHTML und HTML 5 wurde auf XML umgestellt. XML ist um einiges restriktiver als XML und einfacher zu implementieren.
[24] Mit DDoS kürzt man Distributed Denial Of Service Angriffe deren Ziel es ist durch eine große Anzahl von Zugriffen einen oder mehrere Diense eines Computersystems außer Kraft zu setzen.
[25] entnommen aus [SLEMKO 2004].
[26] DOM: Document Object Model; Baumartiger Aufbau einer HTML oder XML Datei. Kann von Skripten genutzt werden, um Inhalte auf einer Seite dynamisch zu verändern.

Angriff funktioniert, muss die Originalseite bereits JavaScript Code mit einer Sicherheitslücke ausliefern. Der Umweg über den Server entfällt. Diese Art einer XSS Attacke ist relativ selten, wird aber mit zunehmender Verbreitung von AJAX[27] häufiger (Siehe Abb. 2).

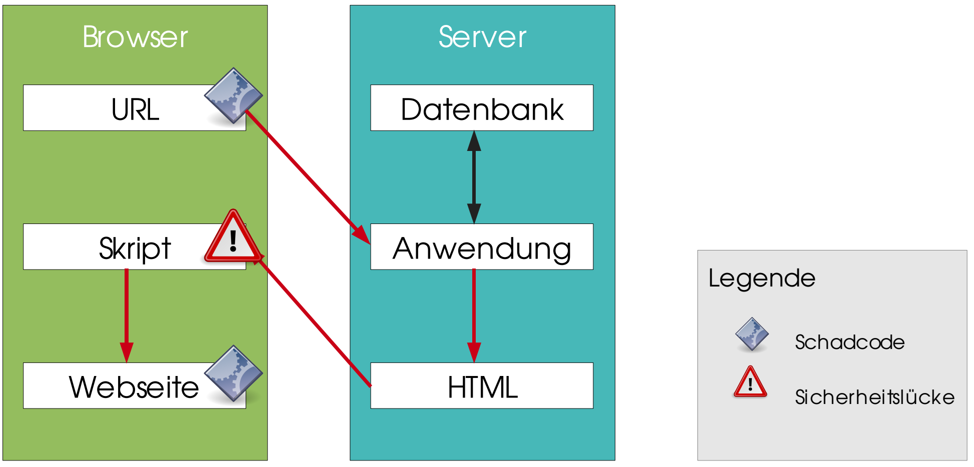


Abbildung 2: Schematische Darstellung eines Typ 0 XSS Angriffes

4.1.2 Typ 1 - Reflektiertes XSS

Die Bezeichnung wurde gewählt, da der Schadcode über den Umweg des Servers auf den Browser übertragen (reflektiert) wird.

Das Reflektierte XSS ist die verbreitetste Art einer XSS Sicherheitslücke. Der Angreifer übermittelt Schadcode wie auch bei allen anderen XSS Arten über einen manipulierbaren Parameter. Die Serveranwendung fügt diesen dann ungesichert in das ausgelieferte Dokument ein. Reflektierte XSS Lücken erfordern, dass der Angreifer eine präparierte URL einem Opfer übermittelt. Der Angriff hinterlässt keine Spuren am Server (Siehe Abb. 3).

Ein Bildschirmfoto eines solchen Reflektierten XSS Angriffes auf die Webseite fotocommunity.de findet sich in Abb. 4 [28] Dieser Angriff nutzte einen Fehler Suchfunktion aus um eigenen Inhalt (in diesem Fall das Wort „VULNERABLE“) anstatt des Suchformulares anzuzeigen. Genauso wäre es möglich gewesen Cookies und anderes auszulesen und auf externe Server zu übertragen.

4.1.3 Typ 2 - Persistentes XSS

Die Bezeichnung wurde gewählt, da der Schadcode persistent, das heißt dauerhaft am Server gespeichert bleibt.

Auch bei Persistentem XSS gibt es einen Angreifer und ein Opfer. Das Opfer muss keinem präparierten Link folgen, sondern der Angreifer hat den Schadcode bereits im Vorfeld auf dem Server gespeichert. (Siehe Abb. 5).

[27] JavaScript Anwendungen die Daten dynamisch vom Server nachladen.
[28] Die Veröffentlichung dieses Angriffes wurde mir schriftlich durch den Seitenbetreiber genehmigt.

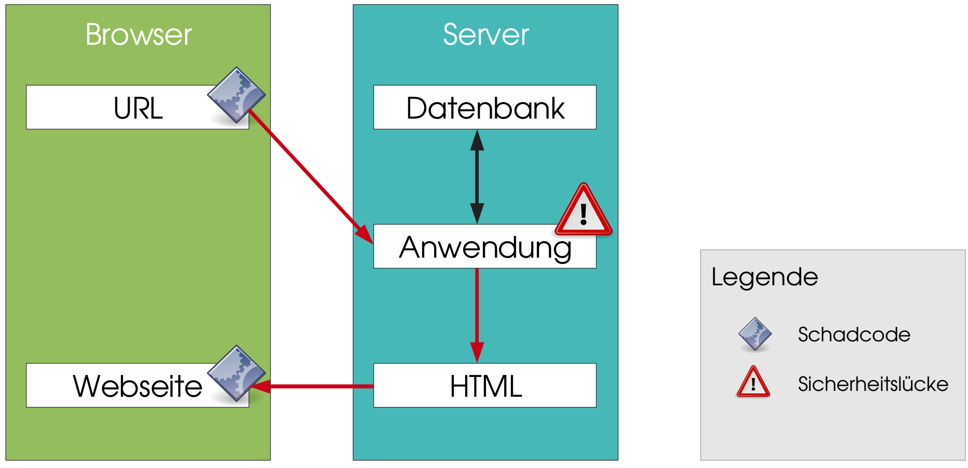


Abbildung 3: Schematische Darstellung eines Typ 1 XSS Angriffes

4.2 Fallbeispiele

4.2.1 Fallbeispiel 1: Forensoftware

Das erste Beispiel (für Persistentes XSS) stellt eine Forensoftware dar. Angenommen, eine solche Software wird auf einer Community mit rund 10.000 Besuchern eingesetzt. Administratoren haben besondere Rechte und können das komplette Forum verwalten, Moderatoren haben erweiterte Rechte, die das Verschieben und Löschen von einzelnen Beiträgen zulassen.

Damit die Forensoftware Benutzer wiedererkennen kann, bekommt jeder Client ein Cookie ausgehändigt, das seine Session ID speichert. Der Webserver schlägt daraufhin diese Session ID in der Datenbank nach und kann dem Benutzer seine persönlichen Rechte wieder zuordnen.

Sämtliche Aufgaben der Moderatoren und Administratoren die eine gefährliche Tätigkeit darstellen (Löschen von Foren/Beiträgen), erfordern eine Bestätigung. Die Bestätigung für die Forensoftware wird dadurch signalisiert, dass an die URL zum Löschen eines Beitrages die eindeutige Session ID des Benutzers angehängt wird (im folgenden Beispiel „sid“ genannt). Wenn diese mit der ID im Cookie übereinstimmt, wird der Löschvorgang durchgeführt:

Programmbeispiel 10: Die einzelnen URLs in der Übersicht

```
URL um ein Forum zu loeschen:
   http://www.example.com/forum/42?action=delete

bestaetigte URL:
   http://www.example.com/forum/42?action=delete&sid=1234...cdef
```

Dies erhöht die Sicherheit, weil ein Moderator damit nicht einem Link folgen kann, der einen Beitrag löscht[29]. Doch mit Hilfe von XSS kann auch diese Sperre umgangen werden, sofern ein Angreifer einen

[29]Dies wäre ein Beispiel für einen möglichen Cross-Site Request Forgery Angriffes, der im nächsten Kapitel ausführlich behandelt wird.

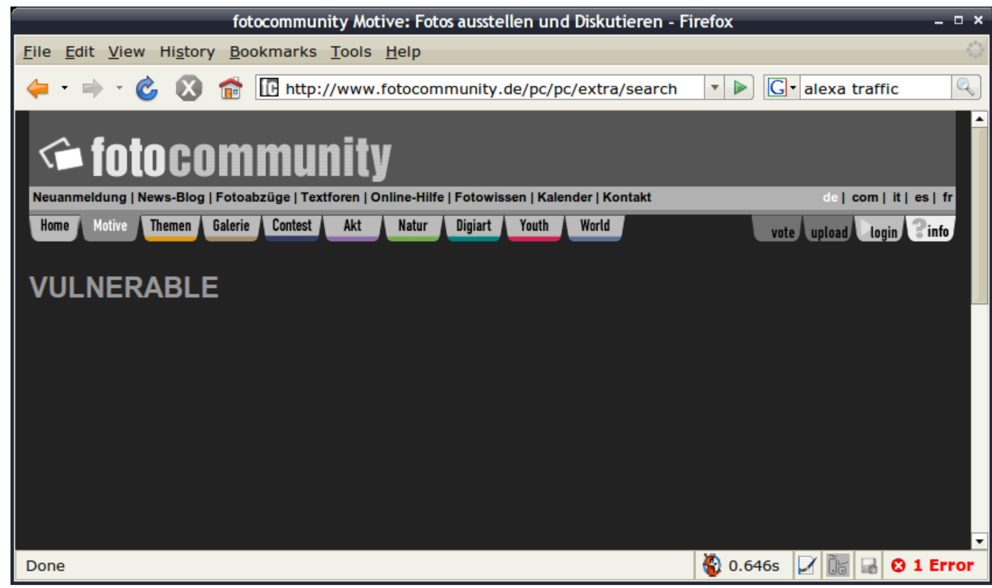


Abbildung 4: Erfolgreicher Angriff auf die Fotocommunity Webseite

wunden Punkt findet.

Der gedachte Angreifer, den wir hier Marvin nennen wollen, findet nun eine Lücke in der Forensoftware. Das Feld für Benutzeravatare in der Datenbank, das Benutzer normalerweise verwenden, um ihre eigenen Bilder neben Beiträgen darzustellen, wird nicht korrekt abgesichert und erlaubt ihm, eigenen JavaScript Code auszuführen. Marvin schreibt nun Quellcode:

Programmbeispiel 11: Gefährlicher JavaScript Quellcode im Avatarfeld

```
var parts = document.cookie.split('; ');
var sid = null;
for (var i = 0; i < parts.length; i++) {
    var pieces = parts[i].split('=', 1);
    if (pieces[0] == 'session_id') {
        sid = pieces[1];
        break;
    }
}
if (sid) {
    for (var i = 0; i <= 42; i++) {
        var el = document.createElement('img');
        el.src = 'http://www.example.com/forum/' + i +
            '?action=delete&sid=' + sid;
    }
}
```

Der Quellcode wird in folgenden HTML Quelltext eingebaut (der Plazhalter wird in diesem Beispiel "avatar_url,, bezeichnet):

Programmbeispiel 12: Eingefügter Avatar auf Serverseite

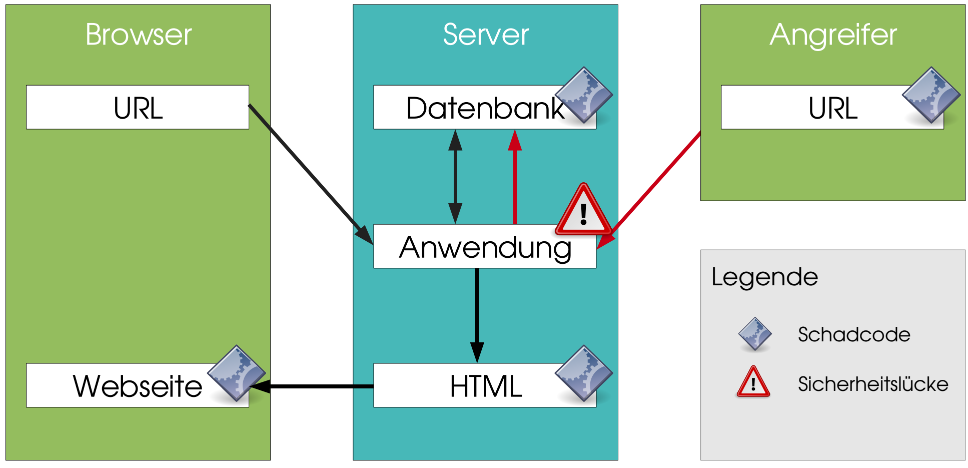


Abbildung 5: Schematische Darstellung einer Typ 2 XSS Angriffes

```
<img src="${avatar_url}" />
```

Der Angreifer kann seinen Quellcode nun sehr einfach einbauen, wenn er zunächst ein Anführungszeichen einfügt und dann seinen Quellcode injeziert, später dieses Anführungszeichen schließt und wieder ein leeres HTML Element wieder einfügt. Das folgende Beispiel veranschaulicht diesen Vorgang:

Programmbeispiel 13: Quellcodeschablone für XSS Angriff

```
"><script>...</script><foo a="
```

Der Angriff im Quelltext mit entfernten Leerzeichen, Zeilenumbrüchen, gekürzten Variablen und der Schablone sieht dann etwa so aus:

Programmbeispiel 14: Der fertige Angriff

```
"><script>p=document.cookie.split('; ');s=0;for(i=0;i<p.length;i++){P=p[i].
split('=',1);if(P[0]=='session_id'){s=P[1];break;}}if(s){for(i=0;i<=42;i++
){document.createElement('img').src='http://www.example.com/'+'forum/'+i+
'?action=delete&sid='+s;}}</script><foo a="
```

Dadurch wird der Quelltext so klein, dass er problemlos in das Avatar Datenbankfeld passt und seinen Schaden anrichten kann. Zwar wird nun der Avatar von Marvin nicht mehr angezeigt, doch sobald ein Administrator einen Beitrag von Marvin liest, startet er den Angriff und die Foren 1 bis 42 werdne ohne Abfrage gelöscht.

Dieser Angriff wird auch als eine Sonderform von „Session-Riding“ oder „Cross-Site Request Forgery“ bezeichnet. Dieses Thema wird im nächsten Kapitel noch genauer abgehandelt.

4.2.2 Fallbeispiel 2: MySpace

Folgendes Beispiel ist nicht fiktiver persistenter XSS Angriff und hat sich tatsächlich zugetragen.

MySpace[30] ist eine der größten Community Portale der Welt. Täglich tauschen dort Millionen von Mitgliedern Nachrichten aus. Nach Angaben der Betreiber melden sich 500.000(!) neue Mitglieder pro Woche an.

Anfang Oktober 2005 konnte ein Mitglied mit sehr trickreich sein Benutzerprofil so umgestalten, dass sich ein Skript, das er dort plaziert hatte, auf die Benutzerseiten von Besuchern, die sein Profil anklickten, kopierte. Das eigentliche Ziel des Skriptes war, dass jeder Besucher eines infizierten Profils den MySpace Account des Angreifers als Freund hinzufügte. Da die Verbreitung des Virus so gewaltig schnell voranschritt, waren nach knapp einem Tag über eine Million Besucherprofile infiziert. Als Folge davon mussten die Server von MySpace wegen der vielen Zugriffe (und um die Lücke zu finden) abgeschaltet werden.

4.2.3 Fallbeispiel 3: Phishing

Zu all den Spam Mails, die Produkte und zweifelhafte Verträge verteilen wollen, mischen sich mittlerweile auch immer wieder so genannte Phishing[31] Mails, deren Ziel es ist, die Bankdaten eines Benutzers abzufragen.

Das Vorgehen läuft in der Regel nach folgendem Schema ab:

1. Der Urheber einer Phishing-Mail schickt seinem Opfer ein offiziell wirkendes Schreiben als E-Mail. Das Ziel dieses Mails ist es, einem Link zu folgen und dort Benutzername, Passwörter oder PIN und TAN von Online-Banking-Zugängen dem Täter preiszugeben.
2. Dies erreicht der Angreifer, indem etwa den Benutzer bittet seine Bankdaten zu überprüfen, oder wegen eines (nie passierten) Abbuchungfehlers seinen PIN einzugeben um den Betrag zurückzuüberweisen.
3. Der Benutzer gibt seine Daten in das Formular ein und schickt es ab.
4. Der Angreifer nutzt die gewonnenen Daten zu seinem Vorteil.

Allerdings wissen Webbrowser mittlerweile die Benutzer vor solchen Angriffen zu warnen. Dabei macht dieser es sich zu Nutze die URL (die nicht die der Bank ist) mit einer Datenbank zu überprüfen, in der gemeldete Phishingserver eingetragen sind. Findet dieser einen Treffer, blendet er eine Warnung ein. Zusätzlich schauen Besucher auch sehr häufig auf die URL des Links und vergleichen diese mit der URL der Bank.

Um dieses Problem zu umgehen, suchen Angreifer häufig nach Lücken in Bankwebseites um diese zu manipulieren. Dazu wird ein Bereich der Webseite gesucht, in dem JavaScript Code eingefügt werden kann um das Ziel von Formularen umzuleiten oder neue Seiten temporär zu erstellen. Dies funktioniert analog mit zum fiktiven Angriff auf das vorher genannte Webforum, wo im Hintergrund unsichtbare Bilder erzeugt werden. Im Unterschied zu unsichtbaren Bildern werden hier neue Texte, Formulare oder Ähnliches auf einer Seite eingefügt, um den Benutzer in die Irre zu leiten.

[30] MySpace: `http://www.myspace.com/`

[31] Mit „Phishing“ wird das „Abfischen“ von Opfern bezeichnet. Hierbei wird via E-Mail versucht dem Empfänger Kontendaten oder TAN Nummern abzunehmen. Das Wort „Phishing“ geht auf das englische Wort „to fish“ zurück und wird – um die Hinterhältigkeit zu unterstreichen – mit „ph“ geschrieben.

Der Benutzer kann bei solch einer Seite nur noch sehr schwer feststellen, ob es sich hier um eine Seite der Bank oder eines Angreifers handelt. Besonders gefährlich sind auch sogenannte Framesets, bei denen einzelne Unterseiten zu einer größeren zusammengebaut werden. Mittels JavaScript kann hier ein Angreifer sehr einfach einzelne Frames gegen eigene austauschen.

Häufig lassen sich Such und Fehlerseiten für solche Dinge missbrauchen. Es kann beispielsweise vorkommen, dass ein URL Paramter ungesichert in die Fehlerseite eingebaut wird, um etwa den Namen der gesuchten Seite auf der Fehlerseite einzublenden. Der Angreifer kann dies nun dafür nutzen, um sein Script einzufügen. Auch Suchseiten, bei denen die eingegebenen Suchwörter wieder auf der Suchseite angezeigt werden sind häufig betroffen.

Auch sogenannte Redirect Seiten (Weiterleitungsseiten) können Gefahren beinhalten. Häufig erlauben Seitenm die dazu genützt werden einen Benutzer anzumelden, einen URL Parameter, der zu einer anderen Seite weiterleitet. Sehr oft wird dies seitenintern verwendet (Die Weiterleitungsziele gehen nicht über den Bankserver hinaus), doch in den allermeisten Fällen wird nicht nachgeprüft, ob es sich bei der im Parameter übergebenen URL nicht um eine externe, eventuell die eines Angreifers handelt. Sofern der Benutzer sich erfolgreich anmeldet, wird dann zur Seite des Angreifers weitergeleitet. Wenn der Benutzer nun nicht die URL kontrolliert, kann der Angreifer eine sichere Umgebung vorgaukeln und ihn um TAN oder PIN fragen. Sofern der Benutzer nichts von seinem Unglück ahnt, wird er diese dort eingeben. Und dies geschieht der Statistik nach sehr häufig. Schätzungen zur Folge sind mehr als neunhundert Millionen Doller durch Phishing in die Hände von Betrügern gekommen. Einer von 20 britischen Internet Benutzern wurden angeblich bereits Opfer. Tendenz steigend. Alleine von Mai 2004 bis Mai 2005 stieg diese Rate um 28%, wie Gartner berichtet[32].

In den meisten Fällen handelt es sich bei den ausgenützen Sicherheitslücken um reflektierte XSS Lücken.

4.3 Cross-Site Scripting verhindern

Cross-Site Scripting kann sehr einfach unterbunden werden. Dabei gilt: „Alle Daten die in einer HTML Seite ausgegeben werden, sind abzusichern“.

Sofern dies geschieht, hat ein Angreifer keine Möglichkeit, Schadcode in den HTML Quelltext einzufügen. (Eine Ausnahme sind Browser Sicherheitslücken auf die der Programmierer einer Websoftware allerdings keinen Einfluss hat)

Je nach Programmiersprache stehen hierzu mehrere Methoden zur Verfügung. Und auch wenn keine eingebaute Funktion bereitsteht, kann diese sehr einfach selbst geschrieben werden.

4.3.1 PHP

PHP bietet zum Absichern von Daten die Funktion *htmlspecialchars* an. Wenn dieser Funktion eine Zeichenkette mit HTML Daten übergeben wird, werden automatisch die Zeichen $<$, $>$ und & durch ihre HTML Codes `<`, `>` und `&` ersetzt.

Sofern die PHP Template Engine Smarty verwendet wird, kann man Variablen auch mit dem *|escape* Filter absichern. Details hierfür finden sich in der Dokumentation der Template Engine.

[32] vgl. „The growth of phishing“, „Business impact from phishing“ und „Consumer impact from phishing“ in [MICRO 2006].

4.3.2 Python

Python bietet Escaping Funktionen für HTML und XML in den Modulen *cgi* und *xml.sax.saxutils* unter den namen *escape* an. Diese funktionieren analog zur PHP Funktion *htmlspecialchars*.

Zudem bieten viele Template Engines und Frameworks alternative Möglichkeiten an. Template Engines wie kid, Genshi und simpleTAL übernehmen das Escaping sogar von selbst.

4.3.3 Ruby

Ruby bietet eine Escaping Funktion im Modul *cgi* unter dem Namen *CGI::escape*. Das Ruby on Rails Framework registriert auch eine globale Funktion *html_escape* die zusätzlich auch unter dem Alias *h* bekannt ist.

5 Cross-Site Request Forgery

Eine Cross-Site Request Forgery (Seitenübergreifende Manipulation einer HTTP Abfrage) – CSRF oder XSRF abgekürzt – ist ein Angriff auf eine Webanwendung die ausschließlich im oder über den Browser eines Benutzer abläuft. Dabei wird dem Benutzer eine gefälschte URL „untergejubelt", die mit seinen Rechten eine Abfrage am Server ansetzt.

CSRF ist ein sehr verbreitetes Problem bei Webanwendungen; kaum eine ist davor geschützt. Nicht zuletzt deswegen, weil es bis 2001 unbekannt war[Jesse Burns 2005].

Eine CSRF Attacke macht es sich zu Nutze, dass Webbrowser Sessions bis zum Ende einer Browsersitzung – oder bis ein Benutzer sich ausgeloggt hat – gespeichert werden. Sofern der Angreifer nun einem Benutzer eine URL zuschiebt, die eine Veränderung am Server auslösen würde, passiert dies im Benutzerkontext.

Das interessante an einer CSRF Attacke ist, dass sie rein zum manipulieren von Daten, nicht aber zum Auslesen selbiger geeignet ist. Grundsätzlich kann eine CSRF Attacke auch ausgeführt werden, wenn keine Benutzersession existiert. In diesem Fall wird aber in den seltensten Fällen Schaden angerichtet. Das Manipulieren von Daten durch CSRF Attacken durch Ausnutzung von laufenden Benutzersessions wird häufig auch „Session-Riding" genannt.

5.1 Beispiel

Ein einfaches und relativ harmloses Beispiel einer CSRF Attacke ist das Abmelden eines Benutzers. Viele Webanwendungen stellen dem Benutzer einen Link für das An- und Abmelden zur Verfügung. Beispielsweise `http://www.example.com/foo/logout.php`.

Sofern nun ein Angreifer diesen Link jemanden, der gerade auf der Seite angemeldet ist unterschiebt und dieser diesen anklickt würde er sofort abgemeldet werden.

Gefährlich wird es aber, wenn der Link beispielsweise auf eine Seite zeigt, die ein wichtiges Dokument löscht (zB `http://www.example.com/foo/delete.php?document=42`). Sofern nun ein Benutzer mit Löschrechten diesem Link folgt, wäre das Dokument verloren.

Das Beispiel mit einer per Mail oder Instant Messaging versendeten URL ist mag theoretisch erscheinen, da ein sorgfältiger Benutzer auf die URL sehen und den Löschbefehl erkennen könnte. Doch es existieren eine Reihe weiterer Angriffsvektoren.

5.2 Angriffsvektoren

Folgende Situationen ermöglichen einen CSRF Angriff.

Alle Angriffe haben eines gemeinsam: Das Opfer muss bereits am Server angemeldet sein. Wenn HTTP Auth verwendet wird, kann es auch vorkommen, dass das Opfer nach Zugangsdaten gefragt wird. Hier sollten natürlich die Alarmglocken schellen und die Warscheinlichkeit, dass dieser Angriff noch funktioniert ist sehr gering. Von einem unsichtbaren Request im Hintergrund bekommt ein Opfer allerdings selten etwas mit. Besonders bei häufig verwendeten Seiten speichern Benutzer gerne ihre Zugangsdaten im Browser oder aktivieren den automatischen Login, sofern vorhanden. Dies würde einen erfolgreichen Angriff bedeuten.

5.2.1 Unterschieben einer URL

Wie im vorigen Beispiel schon genannt besteht der einfachste Angriff darin, einem Benutzer einen präparierten Link zuzuschieben. Beispielsweise wird einem potentiellen Opfer ein Mail zugeschrieben in dem nebst des Links ein kurzer Text angegeben wird, warum das Opfer den Link anklicken sollte. Damit der Link nicht erkannt wird, kann könnte man ihn durch Services wie TinyURL[33] verstecken.

Wird nun beispielsweise der Administrator persönlich und mit Hilfe einer gefälschten Absenderadresse angesprochen und ihm klargemacht, dass der Link beispielsweise zu einem großen Problem zeigt, auf das man zufälligerweise gestoßen ist, besteht die reelle Gefahr, dass der Link angeklickt wird.

Oft wird angenommen, dass durch diese Methode nur GET Requests abgesetzt werden können, das heißt, dass URL Parameter die einzige Möglichkeit seien Daten, zu übermitteln. Das ist aber falsch. Der Angreifer könnte eine kleine HTML Seite mit einem unsichtbaren Formular erstellen die dann den Request zum Zielserver beim Betreten ausführt. Wird diese Seite auf einem Webserver hochgeladen, die URL dorthin via TinyURL verschleiert und dann an ein ahnungsloses Opfer versendet würde einen POST[34] Request am Zielserver ausführen. (Siehe Programmbeispiel 15)

Programmbeispiel 15: Verstecktes Formular für CSRF Attacke

```
<html>
  <body onload="document.forms[0].submit()">
    <form action="http://www.example.com/foo" method="post">
      <input type="hidden" name="action" value="delete"/>
      <input type="hidden" name="id" value="42"/>
    </form>
  </body>
</html>
```

5.2.2 Cross-Site Scripting

Die zweite Möglichkeit eines Angriffes ist XSS. Dies geschieht entweder direkt auf dem betreffenden Server selbst oder auf einem anderen, von dem man animmt, dass das Opfer ihn besuchen wird. Dies erfordert natürlich eine XSS Lücke. Doch sofern für den Angriff ein Request ohne POST Daten ausreicht, bietet sich noch eine dritte Möglichkeit an. Viele Webforen beispielsweise erlauben dem Benutzer zu einem externen Avatar zu verweisen. In diesem Fall würde dann von jedem, der eine Seite betrachtet, auf der dieser Avatar angezeigt wird, automatisch ein Request auf die Angriffsseite ausgeführt werden (nicht zu verwechseln ist dies mit dem XSS Angriff aus dem vorhergehenden Kapitel bei dem tatsächlich ein Script verwendet wurde. Hier hingegen reicht alleine der Zugriff auf ein „Bild“ aus.)

Wenn eine XSS Attacke am selben Server geschieht, hat der Angreifer natürlich viel mehr Möglichkeiten. Unter anderem ist es kann man das Cookie auslesen und XmlHTTPRequests abgesetzen.

5.2.3 Lokaler Angriff

Die dritte grundsätzliche Möglichkeit besteht darin, am Client installierte Spyware oder andere Schadsoftware einzusetzen. Diese könnte sämtliche Abfragen und Antworten mitschneiden, sofern es sich

[33]TinyURL Webseite: `http://www.tinyurl.com/`, verkürzt kryptische URLs zu kurzen unscheinbaren.
[34]Unter einem POST Request versteht man das Übertragen von Daten an den Webserver im Datenbereich des HTTP Protokolls.

tatsächlich ein ausführbares Programm handelt. Allerdings kommt hier ein zusätzlicher Angriffsvektor dazu: Eine lokale HTML Datei mit einem CSRF Angriff. Für sie gilt die selbe Umsetzbarkeit wie bei dem einfachen Unterschieben einer URL; Das Auslesen von Cookies ist allerdings nicht möglich.

5.3 Gegenmaßnahmen

Wenn sie nicht bereits bei der Entwicklung der Anwendung Teil des Konzeptes waren sind CSRF Attacken relativ schwer zu verhindern. Grundsätzlich sind CSRF Attacken nur auf Grund von konzeptionellen Fehlern in Javascript und Webbrowsern möglich, trotz alledem ist es Aufgabe des Entwicklers der Webanwendung diese Attacken zu vermeiden.

Der einzig wirksame Schutz sind zusätzliche, geheime Zeichenketten, die für jeden User und für jede Aktion am Server eindeutig und für einen Angreifer nicht berechenbar sind. Diese zusätzlichen Werte (Token) müssten nun für jede Operation am Server, die Daten verändert, mitgesendet werden. Der folgende Python Quellcode veranschaulicht dies. (Siehe Programmbeispiel 16)

Programmbeispiel 16: CSRF Token Generierung

```
import hmac
import sha

def calculate_token(secret, session_id, url):
    return hmac.new('%s|%s' % (url, secret),
                    session_id, sha).hexdigest()
```

Die Tokengenerierung verwendet die Ziel URL und einen anwendungsinternen und geheimen Schlüssel und verschränkt diese mit der aktuellen Session ID. Dieser Wert wird dann in einem versteckten Formular Feld mitgesand und nach dem Absenden mit dem internen Wert verglichen. Falls die beiden Schlüssel nicht übereinstimmen, sollte die Anwendung mit automatisch einem Fehler abbrechen und die aktuelle Session beenden. Wichtig ist, dass die URL zum Berechnen des Formulares korrekt ist. Wenn das Formular auf eine andere Adresse abgesendet wird muss natürlich dieser Wert verwendet werden, oder ein anderer, der für die Zieladresse berechenbar ist. Es ist auch möglich es jedem Formular einen internen Namen zu geben, der die URL in der Berechnungsfunktion ersetzt.

5.4 Alternative Lösungen

Eine alternative aber nicht vollkommen sichere Methode ist es die Session ID zusätzlich zum Cookie auch in der URL unterzubringen. Die Serveranwendung muss dann bei jedem Request die beiden übermittelten Session IDs auf Equivalenz überprüfen. Dies ist nur solange sicher, als der Angreifer keinen Zugriff auf das Cookie erhält. Einige ältere Webbrowser erlaubten unter bestimmten Umständen Zugriffe auf Cookies von nicht lokalen Webseiten; Auch ist diese Methode unwirksam gegenüber XSS Attacken, die in der Regel Zugriff auf das Cookie haben.

5.5 Ungeeignete Lösungen

Gerade gegen CSRF Attacken wurden immer wieder falsche Lösungen in Anwendungen integriert. Viele Entwickler nehmen offensichtlich an, dass Anwendungen sicher gegenüber CSRF Attacken sind, sofern sie POST anstatt URL Parameter für die Datenübertragung verwenden. Dies ist jedoch falsch: ein Angreifer kann mit minimalem Mehraufwand ein unsichtbares Formular absenden, das einen POST Request ausführt.

Auch die Überprüfung des Referers[sic][35] schützt nicht, denn viele Browser senden den Referer nicht korrekt oder gar nicht.

[35] Die fehlerhafte Schreibweise „Referer“ wurde in [GROUP 1997] erstmalig erwähnt und ging in die HTTP Spezifikation und deren Implementierungen ein.

6 Server Side Code Execution

Eine der gefährlichsten Formen von Sicherheitslücken ist es, wenn es einem Angreifer gelingt, eigenen Code am Server auszuführen. Durch Unachtsamkeit und Unwissenheit wurde am Server die Skriptsprachen-exklusive Funktion *eval* oder eine ihrer Permutationen verwendet, an die vom Benutzer übermittelte Daten ungesichert übergeben werden.

Durch die Natur von Skriptsprachen, dass diese Quellcode ohne Compilierung ausführen können, gibt es meist eine Funktion *eval* (manchmal auch *exec* genannt), die übergebenen Quellcode ausführt. Dies ist aber nicht sinnvoll und sicher, daher gilt hier die Faustregel:

Finger weg von eval!

Es gibt sicherlich auch Bereiche, in denen die Verwendung von *eval* beziehungszweise *exec* sinnvoll ist, allerdings wird in den allermeisten Fälle *eval* nur dazu verwendet, einen Funktionspointer nachzubauen. Das aber kann man in den Sprachen PHP, Python und Ruby auch eleganter und vor allem sicherer lösen.

6.1 Funktionspointer unter PHP

In folgendem Beispiel hat der Programmierer dieser Anwendungung eine Auswahlbox für die Operatoren „+“ und „−“ erstellt, mit zwei Textfeldern für zwei Zahlen. Am Server werden dazu dann die Funktionen *add* bzw *sub* mit den beiden Zahlen aufgerufen. Dies sieht so aus. (Siehe Programmbeispiel 17)

Programmbeispiel 17: Funktionspointer mit *eval* nachgebaut

```
<?php
function add($a, $b) { return $a + $b; }
function sub($a, $b) { return $a - $b; }

function call_function($name, $a, $b) {
    return eval($name . '(' . $a . ', ' . $b . ')');
}
?>
```

Aus dem Beispiel ist ersichtlich was alles schief gehen kann. Der Angreifer muss nur folgenden Code in eines der beiden Textfelder eingeben. (Siehe Programmbeispiel 18)

Programmbeispiel 18: Schadcode im Feld für die erste Zahl

```
, 0); system('rm -rf .'); //
```

Damit würde er den Befehl `rm -rf` auf den aktuellen Ordner ausführen und damit die komplette Webanwendung mit allen ihren Daten löschen. Wie gezeigt wird, kann man dieses Problem auch mit aufrufbaren Zeichenketten lösen, der PHP Version von Funktionspointern. (Siehe Programmbeispiel 19)

Programmbeispiel 19: aufrufbare Zeichenkette als Funktionspointer Ersatz

```
<?php
function add($a, $b) { return $a + $b; }
function sub($a, $b) { return $a - $b; }
$exported_functions = array('add', 'sub');

function call_function($name, $a, $b) {
    global $exported_functions;
    if (in_array($name, $exported_functions)) {
        return $name($a, $b);
    }
}
?>
```

Dabei kann man es sich zur Nutze machen, dass Zeichenketten in PHP aufrufbar sind. Das heißt PHP versucht den Wert der Zeichenkette im Funktionsnamensraum zu suchen und dann mit den übergebenen Parametern auszuführen. Zusammen mit dem eingeführten Array der erlaubten Funktionsnamen ist diese Funktion zu 100% sicher. Zwar werden nun die Parameter als Zeichenketten übergeben, da PHP eine Konvertierung zu einer Zahl allerdings dynamisch durchführt ist dies kein Problem.

Zusätzlich gibt es in PHP gibt es für die Funktion *preg_replace* einen speziellen Schalter *e*, der im Hintergrund eval nutzt, um eine Funktion aufzurufen, deren Ergebnis anschließend für die Ersetzung verwendet wird. Hierfür sollte generell die spezielle Funktion *preg_replace_callback* genommen werden, deren Verwendung in der offiziellen Dokumentation erklärt wird.

6.2 Funktionspointer unter Python

In Python sind Funktionen so genannte *First-Class* Funktionen, das heißt sie verhalten sich wie alle anderen Objekte unter Python auch. Man kann sie damit problemlos wie andere Objekte „umherreichen“ und Referenzen zu ihnen in „Wörterbüchern“ und Listen abspeichern. Weil die Funktionen sehr einfach aufgebaut sind, kann man sogar sogenannte lambda Ausdrücke verwenden. Folgendes kann damit als *eval* Ersatz verwendet werden. (Siehe Programmbeispiel 20)

Programmbeispiel 20: Funktionspointer unter Python

```
functions = {
    'add': lambda a, b: a + b,
    'sub': lambda a, b: a - b
}

def call_function(name, a, b):
    return functions[name](int(a), int(b))
```

Auffallend ist hier, dass eine Konvertierung zu *int* manuell durchgeführt werden muss, da Python stark typisiert ist. Zusätzlich wird keine Überprüfung gemacht, ob die Strings im *functions* „Wörterbuch“ vorkommen, da Python sowieso einen Fehler wirft, wenn der Name nicht im „Wörterbuch“ vorkommt. Auch eine nicht mögliche Umwandung in eine Ganzzahl würde mit einem Fehler enden der von einem Entwickler manuell gefangen werden müsste.

6.3 Funktionspointer unter Ruby

Im Gegensatz zu Python gibt es unter Ruby keine First-Class Funktionen, allerdings kann man sogenannte *Proc* Objekte dafür verwenden. Dies läuft dann analog zur Python Version ab, mit dem Unterschied, dass die Methode *call* des *Proc* Objektes aufgerufen wird. (Siehe Programmbeispiel 21)

Programmbeispiel 21: Proc Objekte als Ersatz für Funktionspointer in Ruby

```
@functions = {
  :add =>    Proc.new{|a, b| a + b},
  :sub =>    Proc.new{|a, b| a - b}
}

def call_function(name, a, b)
  @functions[name.to_sym].call(a.to_i, b.to_i)
end
```

Um Speicherplatz zu sparen wird hier zusätzlich anstatt eines Strings, ein sogenanntes Symbol als Schlüssel für den Hash verwendet.

7 Session-Hijacking

Beim Session-Hijacking übernimmt der Angreifer die Benutzersitzung eines Dritten. Beispielsweise des Seitenadministrators mit mehr Rechten. Der Angreifer versucht hier, die dem Benutzer übermittelte Session ID zu erhalten und dann seine Session ID mit dieser zu ersetzen. Sofern der Server dann keine weiteren Überprüfungen vornimmt, gilt die Sitzung als gehijackt und der Angreifer ist nun mit der Benutzer ID die der Session ID zugeordent war angemeldet.

Das *Hypertext Transport Protokoll* (HTTP) baut bei jedem Wechsel der URL eine neue Verbindung zum Server auf. Damit ist es einem Server nicht möglich, einen Benutzer wiederzuerkennen, da keine dauerhafte Verbindung besteht. Zwar wäre es möglich die IP Adresse dafür verwenden, doch könnte der Benutzer in einem Unternehmensnetzwerk, hinter einem Router oder einem Proxyserver stehen. Damit der Server trotzdem den Benutzer wiedererkennen kann, was die Grundvoraussetzung für Login/Warenkork Funktionalität ist, hat er zwei Möglichkeiten:

- **Cookies**: Bei einem Cookie handelt es sich um eine Textdatei mit einer maximalen Größe von 4KB, die mit allerlei Daten gefüllt werden kann. Diese werden bei jedem Besuchauf der Webseite zum Webserver übermittelt. Häufig wird darin eine sogenannte Session ID versteckt, also eine eindeutige Nummer die der Server zu Rate ziehen kann um den Benutzer wiederzuerkennen.
- **URL**: Die zweite Möglichkeit ist es, die Session ID in jeder URL zu verstecken. Diese Methode ist unsicher, weil die URLs im Server Logbuch inklusive Session ID auftauchen würden, Proxy Server diese eventuell mitprotokollieren und es einem Angreifer leichter machen, diese mit einem Sniffer-Tool mitzuschneiden. Zusätzlich besteht die Gefahr, dass ein Benutzer die URL jemandem anderen übergeben kann und dabei vergisst die Session ID aus dieser URL herauszulöschen.

Leider gibt es keine einfache Möglichkeit, ein Session-Hijacking zu unterbinden, man kann es dem Angreifer allerdings sehr schwer machen, wenn man folgende Punkte befolgt:

1. Session IDs ausschließlich im Cookie übertragen
2. Abgelaufene Sessions so schnell als möglich aus der Datenbank löschen, für den Fall, dass sich Benutzer abzumelden versuchen, sollte die Session ID nach einem gewissen Zeitraum automatisch gelöscht werden (zb, 15 Minuten)
3. Zusätzlich zur Session ID den Namen und die Version des Webbrowsers überprüfen. Dies ist zwar kein effektiver Schutz, aber ein zusätzlicher Faktor der gefahrlos abgefragt werden kann und einen Angreifer zusätzliche Schwierigkeiten bereitet.
4. keine durchgehenden, erratbaren Nummern als Session ID verwenden, besser sha oder md5 Hashes mit ausreichender Zufälligkeit erstellen.

Mit Hilfe dieser Vier-Punkte-Liste lässt sich das Ausnutzen von Session IDs deutlich erschweren. Der Angreifer müsste sich nu zwischen Client und Server schalten und die Pakete, die der Server mit dem Client austauscht, mitschneiden.

Der große Vorteil von Session IDs, die nicht Teil der URL sind, ist dass Benutzer auf der einen Seite nicht aus Versehen ihre Sessiondaten durch Veröffentlichen eines Logins preisgeben und Cookies von Proxies oder Server im Gegensatz zu URLs nicht protokolliert werden.

Ein vollständiger Schutz vor Session Hijacking wird nur durch die Verwendung von SSL Verbindungen möglich.

7.1 Session ID Streuung

Wichtig beim Erstellen einer Session ID ist, dass der Wert absolut zufällig ist. Zwar sind vom Computer generierte Zufallszahlen nur Pseudozufallszahlen, aber sie sind für die Session ID Generierung ausreichend zufällig. Da viele Session Informationen in Datenbanken gespeichert werden, ist eine fixe Länge in der Regel erwünscht. Um dies zu erreichen, kann man mit Hash Funktionen wie MD5, SHA1 oder komplexeren Algorithmen einen einheitlich langen hexadezimalen Hashwert erstellen.

8 Directory Traversal

Beim Directory Traversal wird auf eine Ressource zugegriffen, auf die der Anwender in der Regel keinen Zugriff haben dürfte. Dies kann zum Beispiel passieren, wenn ein URL Parameter direkt in einen Dateinamen eingebaut wird und der spezielle Dateiname „..“ nicht aus dem String entfernt wird. Dies kann ein Angreifer zum Beispiel nutzen, um Zugriff zu Passwortdateien und Ähnliches zu erlangen.

Sehr häufig werden Skriptsprachen dazu verwendet, Seiteinhalte in ein Seitenlayout einzufügen. Dazu wird zum Beispiel der URL Parameter *page* angefragt, die Datei geöffnet und der Inhalt gelesen.

8.1 PHP Nullbyte Problematik

In PHP kann dies zum Beispiel so aussehen, wie in Programmbeispiel 22 gezeigt.

Programmbeispiel 22: Inhalte in ein Layout einfügen

```
<?php
include 'includes/header.php';
$page = (isset($_GET['page'])) ? $_GET['page'] : 'index';
$filename = 'includes/' . $page . '.html';
if (file_exists($filename)) {
    $fp = fopen($filename, 'r');
    echo fread($fp, filesize($filename));
    fclose($fp);
}
else {
    include 'includes/error.php';
}
include 'includes/footer.php';
?>
```

Hier wird zunächst eine „Header“ PHP Datei inkludiert, dann der URL-Parameter „page“ ausgewertet und in einen Dateinamen umgewandelt. Sollte die Datei nicht gefunden werden, so wird die error.php geöffnet. Ansonsten wird ein Dateipointer erstellt, der Inhalt gelesen, ausgegeben und die Datei wieder geschlossen. Anschließend wird in jedem Fall der „Seitenfooter“ wieder eingefügt.

Wenn nun ein Benutzer die URL „index.php?page=foo“ öffnet wird die Datei „pages/foo.html“ inkludiert. Bei „index.php?page=category/blah“ analog dazu „pages/category/blah.html“. Allerdings kann ein Angreifer mit der speziellen Datei „..“ in ein höher liegendes Verzeichniss wechseln. Was er allerdings nicht kann, ist es eine Datei wie „/etc/passwd“ zu öffnen, weil die Funktion ja automatisch ein „.html“ anhängt.

Dies scheint allerdings nur so weil es noch das spezielle Null-Byte gibt. Wenn der Angreifer das spezielle Null-Byte (%00 in einer URL) im URL Parameter versteckt, wird dieser im PHP String mit abgespeichert. Sobald dieser jedoch an die C-Funktion *fopen* übergeben wird, passiert der Fehler. C nutzt nämlich das Null-Byte als Markierung für das eine einer Zeichenkette. Damit wird nun das „.html“ Suffix einfach abgeschnitten und der Angreifer kann mit der nötigen Anzahl von „..“ Zeichen einfach aus dem Verzeichnis der Webanwendung entschwinden:

Ein Aufruf von „index.php?page=../../../etc/passwd%00“ würde nun die Datei „/etc/passwd“ öffenen und deren Inhalt ausgeben. Unter UNIX Systemen wie Linux speichert diese zwar aus Sicherheitsgründen keine Passwörter mehr, allerdings die Benutzernamen. Für einen Angreifer könnten aber die Verzeichnisse „/dev“ und „/proc“ interessant sein, die spezielle Gerätedateien beinhalten. Über diese ist es möglich die Auslastung des Servers zu erfahren oder auch Details über die verwendete Hardware und dergleichen.

Die folgende Funktion sollte auf die Variable für den Dateinamen angewand werden. Sie überprüft einerseits den Speziellen Dateinamen „..“, andererseits auch werden Null-Bytes aus dem Dateinamen entfernt. Zu beachten ist, dass diese Funktion auch eine leeren Zeichenkette zurückliefen kann. Dann kann es vorkommen, dass die Datei „pages/.php“ an *fopen* übergeben wird. Solange diese Datei nicht existiert kann es hier zu keinem Fehlverhalten kommen. Eventuell sollte auf diesen Spezialfall reagiert werden; für dieses folgende Beispiel ist dies jedoch nicht von Bedeutung. (Siehe Programmbeispiel 23)

Programmbeispiel 23: Absichern von Dateinamen

```
<?php
function check_filename($filename) {
    $result = array();
    foreach (explode('/', $filename) as $part)
        if ($part != '..')
            array_push($result, str_replace("\0", '', $part));
    return implode('/', $result);
}
?>
```

8.2 Situation unter Python

Das Problem mit solchen „Includes“ existiert auch unter Python, allerdings fällt das Null-Byte Problem weg, da *file* eine Überprüfung durchführt:

Programmbeispiel 24: Null-Byte Überprüfung

```
>>> file('filename\0.html', 'r')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: file() argument 1 must be (encoded string without NULL bytes)
```

Die meisten Webframeworks würden im Produktivbetrieb an den Benutzer einen *500 INTERNAL SERVER ERROR* senden, was eventuell störend wirken könnte. Daher sollte der *open* Aufruf in einen try/except Block eingepackt werden. Eine mögliche Lösung ist in Programmbeispiel 25 gezeigt.

Programmbeispiel 25: Python Includes

```
try:
    f = file('pages/%s.html' % '/'.join(p for p in
              filename.split('/') if p != '..'), 'r')
except (TypeError, OSError):
    f = file('includes/error404.html', 'r')
```

Hier wird der Dateipfad (in der Variable „filename“ abgelegt) zuerst an jedem Slash getrennt, anschließend werden die speziellen „..“ Dateinamen aus dem Pfad entfernt und die Reste wieder zusammengesetzt. Zuletzt wird die Datei geöffnet.

Sollte dies Fehlschlagen, weil entweder die Datei nicht existiert oder ein Null-Byte im Dateinamen war, wird die Fehlerdatei geöffnet.

8.3 Situation unter Ruby

Im Gegensatz zu Python wird in Ruby keine Überprüfung auf ein Null-Byte im Konstructor des *File* Objektes durchgeführt.

Eine schnelle Lösung ist es, jedes Vorkommen des Null-Bytes durch einen Leerstring zu ersetzen. Alternativ könnte man auch einen Fehler werfen um im Anwendungscode auf diesen zu reagieren. (Siehe Programmbeispiel 26)

Programmbeispiel 26: Ruby Includes

```
begin
    f = File.new('pages/' + filename.split('/').
                    select{|x| x!= '..' }.
                    join('/').gsub(/\x00/, '') + '.html')
rescue
    f = File.new('include/error404.html')
end
```

9 Konzeptionelle Lücken

Besonders gefährlich sind Sicherheitsmängel die durch ein falsches oder zu wenig durchdachtes Konzept entstehen. Diese sind häufig nur durch eine Neuprogrammierung einer Anwendung zu verbessern.

Fehler im Grundaufbau einer Anwendung treten vor allem durch mangelndes Wissen auf. Wer sich nicht von Anfang an bewusst, wo Fehler auftreten können, hat schon verloren.

9.1 Hash URLs als Zugriffsschutz

Gerade bei größeren Anwendungen wird es über kurz oder lang nötig sein, mehr als einen Webserver einzusetzten. Mehrere Datenbankserver sind meistens kein Problem, weil moderne Datenbanksysteme selber Funktionalität zum verteilten Zugriff (Spiegelung, Master-Slave Konfiguration...) mitbringen, und auch bei mehreren vorgeschalteten Webservern die nur als Caching Proxies[36] arbeiten ist die Konfiguration kein Problem. Schwieriger wird es, wenn eine Anwendung auf mehrere Server verteilt wird.

Man stelle sich eine Webseite vor, die es dem Benutzer erlaubt, eigene Bilder hochzuladen und in privaten Bildergalerien zu speichern. In sehr vielen Fällen werden die Bilder dann auf einem anderen Server abgelagert und so umbenannt, dass das Erraten des tatsächlichen Bildnamens schwierig wird.

Die Webanwendung speichert anschließend in einer Datenbank, welcher Dateiname zu welcher Bildnummer gehört. Die Zugriffsverwaltung der Anwendung überprüft dann beim Öffnen einer Galerieseite, ob der Benutzer auf ein Bild zugreifen darf. Sollte dieser Test erfolgreich verlaufen, werden die einzelnen Dateinamen aus der Datenbank geladen und in URLs umgewandelt.

Von der URL *http://www.example.com/picture/42/* wird dann zum Beispiel eine Weiterleitung auf die statische Datei c033.jpg am Medienserver (*http://srv3.example.com/2/2d234cb204a/c033.jpg*) eingeleitet. Der Medienserver muss dann nicht mehr die Zugriffsrechte überprüfen weil angenommen wird, dass die Bild-URL so zufällig ist, dass niemand die URL errät.

Diese Vorgangsweise einen großen Vorteil: Sie ist schnell und Proxy Server können problemlos vorgeschaltet werden.

Ein größes Problem an dieser Konfiguration ist allerdings, dass es, sofern die URL einmal erraten oder „mitgesnifft“ wurde, nicht möglich ist einen Angreifer von einem regulären Benutzer zu unterscheiden und damit ausgespähte Bilder aufzuspüren und deren Dateiname und URL zu ändern.

Besonders dieses Problem ist sehr schwer zu lösen. Diese konzeptionelle Lücke entsteht aus mangelnder Rechenpower auf der Serverseite. Um dieses Problem lösen zu können, müsste man den Medienservern Zugriff auf die Benutzerverwaltung und das Session System geben. Dies würde zusätzliche Rechenkraft kosten und kann damit kaum nicht in Betracht gezogen werden.

Besonders bei externen Servern, die nicht über die selbe Domain zu erreichen sind ist dies ein Problem, da sie technisch gesehen keine Möglichkeit haben, auf das Cookie der Hauptseite zuzugreifen um den Benutzer erkennen zu können.

Eine wirkliche Lösung gibt es hier nicht. Sofern tatsächlich verschiedene Zugriffsregelungen existieren, die statische Dateien auf anderen Servern betreffen gibt es hier allerdings einige Punkte die verbessert

[36] Webserver die nur Daten zwischenspeichern. Dynamische Webseiten werden von einem vorgeschalteten Webserver vom Hauptserver angefordert und dort für einen kurzen Zeitraum gelagert um die Last vom Hauptserver zu nehmen.

werden können:

1. Die Server, die die statischen Dateien bereistellen, müssen so konfiguriert werden, dass sie unter keinen Umständen die Inhalte von Verzeichnissen darstellen.
2. Die URLs zu statischen Dateien müssen zufällig gestaltet sein. Unter keinen Umständen darf hierfür ein Wert verwendet werden, der bereits auf der Seite Teil der URL ist, die später auf die statische Datei verweist. Beispielsweise referenziert die Seite *http://www.example.com/file/42/* auf eine statische Datei am Medienserver *http://srv01.example.com/*. In diesem Fall darf der Pfad auf srv01 auf keinenfall die Bild ID 42 enthalten. Auch der Hashwert der Bild ID 42 wäre gefährlich, da leicht zu erraten. Besonders „ungesalzene" MD5 ist durch sogenannte Rainbow-Tables Angriffe schnell umkehrbar.
3. Die Pfade müssen ausreichend komplex sein. Als Dateiename ist zumindest ein 32bit hex Hash (möglicherweise verteilt in mehrere Verzeichnisse) zu empfehlen. Noch besser wäre das komplette Alphabet nebst einiger ASCII Sonderzeichen als gültige Zeichen für den Dateinamen zu verwenden.
4. Die Anwendung darf selbst nicht auf diese statischen URLs aufbauen. Sie muss so entwickelt sein, dass es möglich ist Daten nicht nur von einem Server zu einem anderen zu verschieben, sondern auch einzelnen Dateien neue URLs zu geben. Dies erleichtert später auch das Verschieben der Daten auf andere Server.
5. Um das unabsichtliche Copy/Paste von solchen Dateiadressen auf statischen Servern zu verhindern, sollten Permalinks (Adressen die sich nicht ändern) auf den Hauptserver zeigen, der dann eine Weiterleitung auf die betreffende Datei einleitet. Damit bleiben URLs nicht nur später gültig, auch haben die Benutzer keine Chance unabsichtlich eine URL zu veröffentlichen, ohne den Zugangsschutz zu umgehen.

9.2 Passwortspeicherung

Viele Anwendungen erlauben es dem Benutzer einen Account anzulegen, um sich eindeutig identifizieren zu können, einen gesicherten Namen zu haben (in Forensystemen oder Wikis) oder einfach nur persönliche Einstellungen zu speichern. Das Subsystem, das die Benutzeraccounts verwaltet, muss demnach zumindest Benutzername und Passwort speichern. Die einfachste Möglichkeit ist, dass das Passwort im Klartext am Server gespeichert wird. Wenn sich der Benutzer auf der Anmeldeseite authentifiziert, werden die Daten einfach mit der aus der Datenbank abgeglichen. Grundsätzlich spricht nichts dagegen, denn die Kommunikation zwischen Benutzer und Server läuft entweder verschlüsselt oder unverschlüsselt ab, die Art wie Passwörter am Server gespeichert werden, verbessern oder verschlechtern die Sicherheit dahingehend nicht. Gefährlich wird es dann, wenn ein Angreifer Lesezugriff auf die Datenbank bekommt oder Backupdaten entwendet. In diesem Fall hätte er Zugriff auf die Klartextpasswörter und könnte sich problemlos am System anmelden.

Eine erhöhte Sicherheit ließe sich durch sogenannte Hashing Algorithmen wie MD5, SHA1 oder komplexeren erreichen. Damit werden Passwörter sowohl vor dem Speichervorgang gehasht (Hashing ist eine mathematisch nicht umkehrbare Verschlüsselung) als auch nochmals vor der Prüfung. Wenn die beiden Hashes (der Hash in der Datenbank, und der Hash des vom Benutzer übermittelten Passwortes) übereinstimmen, hat der Benutzer Zugriff.

Durch einfaches Hashing des Passwortes durch MD5 wird allerdings das Problem nur minimal verringert. Durch preparierte Rainbow-Tables (Eine große Liste an Hashwerten wird erstellt und gegen die Datenbankdaten überprüft) kann sehr schnell eines der Passwörter geknackt werden. Auch Trivialkennwörter wie ”123456”, ”passwortöder häufige Vornamen, die öfter in der Datenbank vorkommen, können so schnell identifiziert werden.

Verbessern lässt sich das Verfahren dahingehend, dass bei der Erstellung der Hashes ein zufälliger Wert (ein sogenannter Salt) beigemengt wird. Dieser muss dann in einem extra Feld in der Datenbank gespeichert werden, um gegen Benutzerdaten testen zu können, doch die Warscheinlichkeit, mit Rainbow-Tables Passwörter herauszufinden, wird merklich kleiner.

Als Negativbeispiel sei hier phpBB 2 genannt. Die Passwörter werden nur mit Hilfe von MD5 und ohne Salt gehasht. Aus einer phpBB Datenbank mit 26795 Benutzern eines der Foren, die ich administriere, konnte ich mit Hilfe eines kleinen Ruby Skriptes, das Passwörter zwischen 1 und 6 Buchstaben Passwörter durch Brute-Force testet innerhalb von einer Stunde und zwei Minuten 5071 Passwörter wiederherstellen. Mit einer ausreichend großen Wordliste könnte dies sogar noch optimiert werden.

Dasselbe Problem betrifft auch eine Reihe weiterer Anwendungen, die auf bloßes MD5 Hashing vertrauen. Sofern man ausschließen kann, dass niemand Zugriff auf die Datenbank erlangt, ist dies auch kein Problem. Es sind aber immer wieder Fälle bekannt geworden, in denen Backup-Medien verloren gingen oder gestohlen wurden. Sofern nun Benutzer ihre Zugangsdaten auf mehreren Webseiten verwenden, kann der Angreifer diese Daten auch anderwertig nutzen.

In der Folge wird noch ein in Python implementiertes Beispiel dargestellt, welches eine sicherere Hashing und Prüffunktion bereitstellt, die einen zufälligen Salt verwendet um Brute Force Attacken nutzlos zu machen. Zwar verwendet sie kein aufwändiges Verfahren für die Verschränkung des Salts mit dem eigentlichen Passwort, doch ist es für die Überprüfung von Passwörtern ausreichend. Ein Brute-Force Angriff wäre damit zwar nach wie vor möglich, doch müsste der Angreifer nun pro Hash agieren, wärend sonst möglich ist, einen Hash des Prüfpassworts zu erstellen und gegen jeden Hash in der Datenbank zu testen. Es wäre auch möglich, den Benutzernamen in den Hash einfließen zu lassen. (Siehe Programmbeispiel 27)

Programmbeispiel 27: Passwort Hash- und Prüffunktion

```
import md5
from random import choice
from string import ascii_letters

def generate_hash(password, salt=None):
    salt = salt or ''.join(choice(ascii_letters) for _ in xrange(5))
    return '%s/%s' % (salt, md5.new('%s|%s' %
                    (salt, password)).hexdigest())

def check_hash(hash, password):
    salt = hash.split('/')[0]
    return hash == generate_hash(password, salt)
```

Wann immer nun ein Hash für die Datenbank erstellt wird, muss nur das neue Passwort an die *generate_hash* Funktion übergeben werden um einen neuen Hash in der Form „salt/password_hash“ zu erhalten. Die Überprüfung wird dann von der Funktion *check_hash* vorgenommen, die den Hash aus der

Datenbank als erstes Argument und das zu überprüfende Passwort an zweiter Stelle annimmt. Sofern das Passwort mit dem in der Datenbank übereinstimmt, gibt diese Funktion dann „True“ zurück.

9.3 Cookie Authentifizierung

Um Benutzersessions wiedererkennen zu können, muss die Webanwendung Informationen auf dem Client speichern können. Dies ist technisch bedingt nur durch ein Cookie möglich. (Session IDs in URLs ausgenommen die aber aus Sicherheitsgründen eine Gefahr darstellen)

Dabei muss beachtet werden, dass Cookies von Benutzern nach belieben verändert werden können. Ein bloßes Abspeichern des Benutzernamen im Cookie ist daher keine geeignete Lösung, ein Benutzer könnte ohne Probleme den Namen im Cookie durch den des Administrators ersetzen. Auch das zusätzliche Speichern des Passwortes im Cookie ist gefährlich wie im Kapitel „Cross-Site Scripting“ gezeigt wurde. Auch das Austauschen von Benutzername in eine interne Nummer (User ID) ist keine Hilfe, selbst dann nicht wenn die internen Benutzer Nummern zufällige Hashwerte sind. Sollte es einem Angreifer gelingen, eine solche ID auszuspähen ist es ihm möglich zu späteren Zeitpunkten sich immer am Authentifizierungssystem anzumelden. Die Standardauthentifizierung der populären MoinMoin Wikiengine hat zum Beispiel dieses Problem. Das Wiki speichert die interne MoinMoin User ID im Cookie, um einen Benutzer als angemeldet zu markieren.

Die geeignetere Methode ist es, ein Session System zu verwenden. Dem Benutzer wird beim Anmelden eine eindeutige Nummer zugeteilt, die nur von der Serversoftware in die Benutzer ID aufgelöst werden kann. Zusätzlich haben die Benutzer-Sessions eine Ablaufzeit, die sowohl am Server, als auch im Cookie definiert wird. Wenn das Cookie abläuft, muss der Benutzer sich neu anmelden, genauso wie, wenn der Server die Session ID aus der Zuordnungstabelle löscht.

Wie im Kapitel „Session-Hijacking“ schon erwähnt, ist auch diese Methode nicht absolut sicher, aber die problematische Situation wurde zumindest etwas entschärft.

10 Information Disclosure

Häufig unterschätzt werden Probleme, die dadurch entstehen können, dass Anwendungen zu viel von ihrem Innenleben verraten. Zwar widerspricht dies dem „Full Disclosure“[37] Konzept, doch einige Informationen sollten auf jedem Fall für den Anwender unsichbar bleiben.

10.1 Automatic Directory Indexing

Ein Großteil der heutigen Webserver ist standardmäßig so konfiguriert, dass er in Verzeichnissen, in denen sich kein Index Dokument (index.html oder je nach Konfiguration auch andere) befindet, ein Directory Listing, also eine Aufzählung aller Dateien im aktuellen Verzeichnis anbietet.

Dies ist in der Regel kein Problem, weil die Anwendung so entwickelt sein sollte, dass sie entweder den Zugriff auf Systemdateien verbietet oder zumindest mit einer Fehlermeldung abbricht, doch gerade für den im Kapitel „Konzeptionelle Lücken“ unter derm Punkt „Hash URLs als Zugriffsschutz“ angesprochenen Sonderfall kann dies ein Sicherheitsproblem darstellen.

10.2 Information Leakage

Je nach Größe einer Anwendung wird der Entwickler über kurz oder lang sogenannte Debugging-Ausgaben einarbeiten. Sie helfen auf der einen Seite beim Auftreten von Fehlern, sich schneller einen Überblick schaffen zu können, auf der anderen bestimmte Verhalten überprüfen zu können.

Nichts im Quellcode haben solche Ausgaben aber im Produktivbetrieb verloren. PHP hatte zum Beispiel in einigen früheren Versionen die Unart bei Fehlern im Verbindungsaufbau zu einer MySQL Datenbank die Zugangsdaten zur Datenbank in die Ausgabe, nebst einer Fehlermeldung abzusetzen.

Ideal für diesen Fall ist es, der Anwendung Informationen über die Umgebung mitzuteilen (lokaler Entwicklungsrechner, Testing-Server, Produktionsserver). Je nach aktueller Umgebung können dann Fehler im Browser, einem Logfile oder via Mail ausgegeben werden.

Hierfür gibt es einige Regeln:

10.2.1 Entwicklungsmodus

Die Anwendung sollte so häufig wie möglich Informationen über den internen Ablauf ausgeben, je nach Programmierumgebung (PHP, Python, Ruby) gibt es dazu mehrere Möglichkeiten. Wenn Fehler auftreten, sollte der komplette Weg zum Fehler (Traceback) dargestellt werden, mit Zeilennummer und Dateiname. Bei Python und Ruby werden Fehler je nach Framework verschieden behandelt, die meisten geben aber den kompletten Traceback entweder im Entwicklungsserver oder eine umfangreichen Informationsseite aus. In PHP hat man nicht allzu viele Möglichkeiten, aber durch das Ausführen der Funktion *error_reporting* mit dem Parameter *E_ALL* am Beginn der Codeausführung werden zumindest sämtliche Warnungen und Fehler ausgeben.

[37] Full Disclosure: Alle Informationen einer Anwendung sind frei Verfügbar. Wenn Sicherheitslücken gefunden werden, werden diese mit Detailinformationen veröffentlicht. Full Disclosure ist das absolute Gegenteil von „Security by Obscurity“, bei dem sämtliche Informationen über eine Anwendung nur dem Entwickler bekannt sind.

10.2.2 Testbetrieb

Bei größeren Anwendungen mit mehreren Entwicklern lohnt es sich, einen eigenen Server aufzustellen, der in regelmäßigen Abständen den Quellcode der Anwendung aus der Quellcodeverwaltung auscheckt und die Test Suite[38] ausführt. Diese sollte möglichst alle relevanten Teile des Programms unter einer produktionsnahen Umgebung ausführen und testen. Sofern die Anwendung klein und überschaubar ist, reicht es auch aus das Überprüfen auf dem Entwicklungsrechner durchzuführen.

10.2.3 Produktivbetrieb

Im Produktivbetrieb gilt die Regel: Der Benutzer darf keinerlei interne Informationen der Anwendung zu Gesicht bekommen, die Sicherheitsrelevant sind. Wenn eine SQL Query fehlschlägt, ist es von Vorteil diese dem Benutzer nicht auf der Seite zu präsentieren, eventuell befinden sich in dieser Abfrage Informationen, die er besser nicht hätte sehen sollen (Passwörter, interne Zuordnungsnummern usw.). Das WSGI Gateway flup (Verbindungsstück zwischen Python Webanwendung und FastCGI Webserver) ist zum Beispiel standardmäßig so eingestellt, dass es Fehler ausführlich dem Benutzer präsentiert, sofern die Webanwendung nicht selber die Fehler fängt.

Dies kann von Vorteil sein, doch wenn sich irgendwo im Stack Konfigurationsvariablen befinden, würde der Benutzer diese zu Gesicht bekommen.

[38]Große Anwendungen sollten sogenannte Unit Tests mitbringen, die ein Verhalten der Anwendung in bestimmten Situationen kontrolliert.

Teil III

Statistiken, Auswirkungen und Abschließendes

11 Statistiken

Genaue Statistiken über Sicherheitslücken in Webanwendungen lassen sich nur sehr schwer erstellen. Man muss hierfür nämlich zwei Typen von Anwendungen unterscheiden: Auf der einen Seite gibt es die traditionellen Webanwendungen, die speziell für eine Webseite entwickelt werden, in der Regel Closed Source[39] sind und aus diesem Grund auch nicht auf Vulnerability Listen auftauchen. Sicherheitslücken in solchen Anwendungen werden (sofern kein Schaden entstanden ist) ohne weitere Kommentare geschlossen. Auf der anderen Seite gibt es eine Unzahl an Webanwendungen die sich auf dem eigenen Webserver installieren lassen, sofern dieser die nötige Betriebssoftware installiert hat. Letztere haben ja nach Verbreitung eine verschieden große Anzahl an gefundenen Sicherheitslücken.

Um allerdings die Verbreitung von Sicherheitslücken demonstrieren zu können seien hier einige Programme genannt bei denen man nach der Veröffentlichung Sicherheitslücken gefunden hat.

11.1 phpBB 2.x

phpBB[40] ist eine Forensoftware die 2000 von James Atkinson für seine persönliche Webseite entwickelt wurde. Sehr schnell stiegen andere Entwickler in das Projekt ein und es wurden von 2000 bis 2006 die Versionen 1.x und 2.x entwickelt. Momentan wird am Dritten Major-Release (Codename Olympus) gearbeitet, dass eine komplette Neuentwicklung darstellt. phpBB wurde, wie der Name schon verrät in PHP implementiert und unter der GNU GPL veröffentlicht.[41]

Für phpBB 2.x sind 34 Security Advisories bekannt [42], die die Kernsoftware selbst betreffen. Zusammen mit phpBB Erweiterung sind dies um einige mehr. Alle 34 Advisories betreffen den Zeitraum 2003 bis 2006 und sich wie in Abb. 6 gezeigt. (zu beachten ist, dass einige Avisories mehrere Lücken in verschiendenen Bereiche zusammenfassen und daher im Diagramm mehrfach eingetragen sind):

phpBB erlangte vor allem durch die häufigen Sicherheitslücken eine zweifelhafte Berühmtheit. Besonders SQL-Injections und XSS Lücken lassen sich durch sorgfältige Programmierung und einer soliden internen Struktur verhindern. In die Kategorie „Andere“ fallen vor allem Fehler, bei denen durch das Absenden von falschen Werten an die Forensoftware ein Sicherheitscheck oder ein anderer Test übersprungen oder automatisch bestätigt wird.

phpBB weist zwar eine hohe Anzahl an Fehlern auf, allerdings wurden diese innerhalb kurzer Zeit behoben. Aktuell sind keine Sicherheitslücken bekannt.

11.2 MediaWiki 1.x

MediaWiki[43] ist eine Wiki-Engine, die der deutsche Biologe Magnus Manske für die Online-Enzyklopädie Wikipedia entwickelte und mittlerweile von der Wikimedia Foundation gepflegt wird. Sie erfordert eine MySQL Datenbank (Ab Version 1.5 ist eine experimentelle Postgres Unterstützung an Board) und PHP5. MediaWiki wird unter der GNU GPL veröffentlicht.[44]

[39] Closed Source: Der Quellcode ist nur dem Entwickler bekannt. Ist das Gegenteil von Open Source bei der komplette Quellcode für jedermann offen steht
[40] phpBB Projektwebseite: `http://www.phpbb.com/`
[41] vgl. Kurzbeschreibung in [Wikipedia 2006b].
[42] vgl. Secunia Datenbank, Stand 29. Dezember 2006 [Secunia 2006b].
[43] Projektwebseite unter `http://www.mediawiki.org/`
[44] vgl. Kurzzusammenfassung in [Wikipedia 2006a].

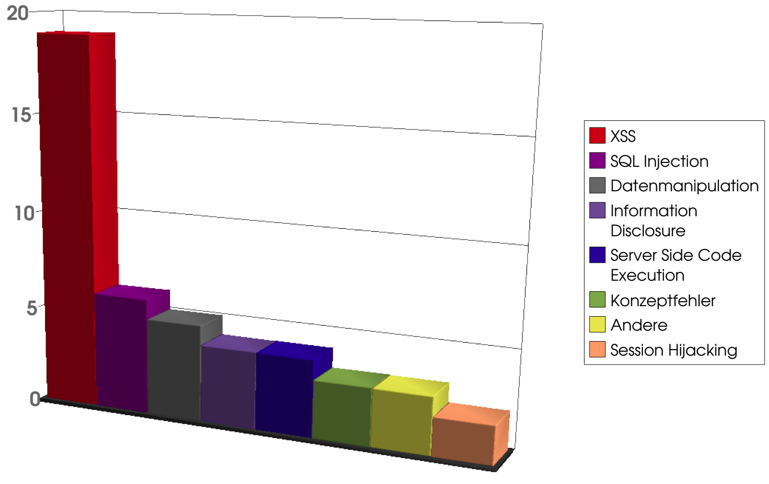


Abbildung 6: Statistik über die Sicherheitslücken in phpBB 2.x

In der Secunia Vulnerability Datenbank[45] werden für MediaWiki zum aktuellen Zeitpunkt 18 Advisories gemeldet – auch hier mit teilweise mehreren Lücken in einem Advisory. Im Vergleich zu phpBB fällt hier auf, dass keine einzige SQL-Injection und Konzeptionelle Lücke im Quellcode gefunden wurde. Wie auch bei phpBB sind XSS Lücken die Häufigsten. (Siehe Abb. 7)

Wie auch bei phpBB sind bei der MediaWiki Software die Reaktionszeiten des Entwicklungsteam auf Sicherheitslücken sehr schnell. Zum aktuellen Zeitpunkt sind keine offenen Sicherheitslücken bekannt.

11.3 Online Portale

Sicherheitslücken in Online Portalen muss man unter einem anderen Gesichtspunkt betrachten. Eine Sicherheitslücke in einer beliebten Community kann schnell ein Problem für alle angemeldeten Benutzer bedeuten, wenn persönliche Daten, die für die Registrierung nötig sind, ausgespäht werden können.

Selbst große Seiten sind vor Sicherheitslücken nicht gefeit. Immer wieder kann man im Heise Newsticker[46] von Sicherheitsproblemen in Webseiten wie ebay, Amazon oder Communityportalen wie zuletzt studiVZ lesen. Für das Kapitel über Cross-Site Scripting habe ich auf einigen der populärsten[47] deutschsprachigen Seiten einige XSS Lücken gefunden. Die Reaktion der Seitenbetreiber auf die gefundenen Sicherheitslücken war sehr unterschiedlich.

Lobend möchte ich hier das deutsche Fotoportal „fotocommunity“[48] erwähnen, dass die gefundene Lücke innerhalb von 48 Stunden geschlossen hat und mir auch die Erwähnung in dieser Fachbereitsarbeit genehmigte. Von den restlichen kontaktierten Seiten waren die Reaktionen eher unfreundlich bzw. automatisierte Mails. Auf Grund der Rechtsunsicherheit betreffend dieses Punktes werde ich keine

[45] vgl. Secunia Datenbank, Stand 29. Dezember 2006 [SECUNIA 2006a].
[46] Heise Newsticker: `http://www.heise.de/`
[47] Die Liste, die ich zur Rate gezogen habe war die Alexia Toplist für deutschsprachige Seiten: `http://www.alexa.com/site/ds/top_sites?ts_mode=lang&lang=de`.
[48] fotocommunity: `http://www.fotocommunity.com/`

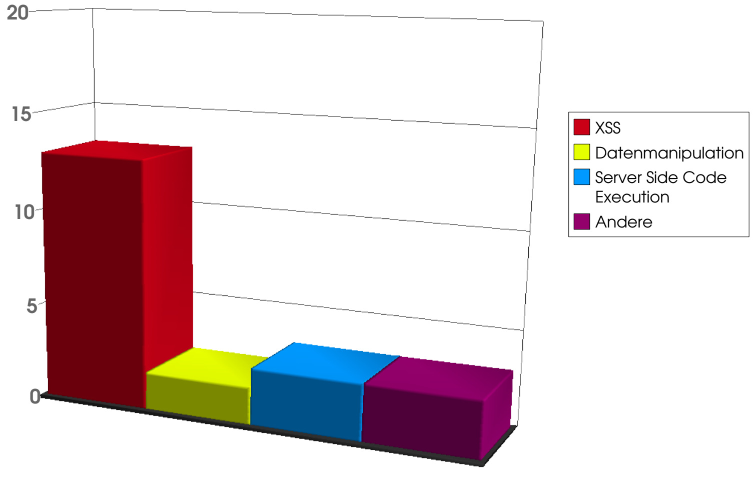


Abbildung 7: Statistik über die Sicherheitslücken in MediaWiki 1.x

dieser Seiten oder die Sicherheitslücken veröffentlichen.

12 Schlusswort

Das Thema „Sicherheit in Webanwendung“ ist durchaus ein heißes Eisen und auch, wenn ich zu Beginn der Arbeit gedacht habe, etwas davon zu verstehen, war ich doch überrascht wie naiv ich mit dem Thema Sicherheit bis zu diesem Zeitpunkt umgegangen bin. Das Ausarbeiten des Themas war spannend und erschreckend zugleich. Auf der einen Seite werden immer wieder neue Angriffsvektoren gefunden, an die man bisher nicht gedacht hat, andererseits sind viele der verbreiteten Lücken erstaunlich banal.

Mir war es ein Anliegen, die Arbeit nicht zu abstrakt zu halten, damit sie für den Interessierten leicht zu lesen und verständlich ist. Es stellte sich heraus, dass dies ein schwieriges Unterfangen ist da sich die drei verwendeten Sprachen in vielen Punkten unterscheiden und übersichtliche und praxisnahe Beispiele sehr schwer zu finden waren.

In einem Punkt war diese Arbeit für mich besonders interessant: Informationen zu diesen Themen sind erstaunlich schwer und ausschließlich im Internet zu finden. Zwar gibt es eine Unzahl an Büchern, die sich mit dem Thema Sicherheit beschäftigen aber lediglich in Hinblick auf den Computerbenutzer, nicht aber für den Anwendungsentwickler. Als wichtigste Informationsgrundlage kristallisierten sich schnell die Erfahrungswerte von anderen Entwicklern und diverse Veröffentlichungen von Sicherheitsunternehmen heraus, die rein online zur Verfügung stehen. Zwar war es für mich deswegen einfach an zu Informationen zu kommen, doch musste deren Wahrheitsgehalt durch Vergleichen mit anderen Informationen und durch Ausprobieren überprüft werden. Viele der erhaltenen Informationen waren auch schon nicht mehr auf dem aktuellen Stand und zeigten Sicherheitslöcher in den Programmiersprachen bzw. Webbrowsern selber auf.

Eine der interessantesten Erfahrungen, die ich durch das Schreiben dieser Arbeit gemacht habe, ist sicher die Kurzlebigkeit von Informationen im Internet. Es geschah mehrmals, dass sich Quellen zwischen den einzelnen Abrufen veränderten, andere verschwanden, oder die URL wechselte. Dies hat mir zumindest gezeigt, wie notwendig große Portale wie Wikipedia sind, die nicht nur gepflegt und gewartet werden, sondern auch genug Potential haben um nicht von einem Tag auf den anderen aus dem Netz zu verschwinden.

Insgesamt kann festgestellt werden, dass es ein enormes Potential an Sicherheitslücken gibt und die Entwickler aufgerufen sind, sich Gedanken darüber zu machen, wie diese besser vermieden werden können. Es würde mich freuen, wenn diese Arbeit für den einen oder anderen Entwickler eine Hilfestellung bietet.

12.1 Protokoll

kommt erst.

12.2 Lizenz

Diese Fachbereichsarbeit ist lizensiert unter der GNU-Lizenz für freie Dokumentation.

12.3 Hinweis zu den Zitaten

Technisch bedingt waren die brauchbaren Informationen zu diesem Thema ausschließlich online zu finden. Bei Informationen, die nicht von Institutionen in Form von PDF Dateien herausgegeben wurden, wird im Literaturverzeichnis der Zeitpunkt des Downloads angegeben. Sofern möglich wurden die Seiten als Permalinks angegeben.

12.4 Erklärung

Hiermit erkläre ich, dass ich meine Arbeit alleine und ohne fremde Hilfe und dass ich alle verwendete Quellen zitiert und kenntlich gemacht habe.

Datum 13. Feber 2007

Name Armin Ronacher

Unterschrift

Abbildungsverzeichnis

1 Schematische Darstellung eines HTTP Requestes 10
2 Schematische Darstellung eines Typ 0 XSS Angriffes 19
3 Schematische Darstellung eines Typ 1 XSS Angriffes 20
4 Erfolgreicher Angriff auf die Fotocommunity Webseite 21
5 Schematische Darstellung einer Typ 2 XSS Angriffes 22
6 Statistik über die Sicherheitslücken in phpBB 2.x 46
7 Statistik über die Sicherheitslücken in MediaWiki 1.x 47

Sämtliche Abbildungen sind, sofern nicht anderwertig gekennzeichnet, von mir selbst mit den Programmen Inkscape, OpenOffice.org und/oder GIMP erstellt. Alle Abbildungen sind wie die Fachbereichsarbeit selbst unter der der GNU-Lizenz für freie Dokumentation lizensiert.

Quelltext Beispiele

1 PHP SQL-Injection . . . 12
2 Präparierte Werte . . . 13
3 Veränderter SQL Befehl . . . 13
4 Kein weiterer Angriffspunkt für die SQL-Injection . . . 14
5 Gesicherter SQL Befehl . . . 14
6 MDB2 Lösung . . . 15
7 SQL-Injection in Python . . . 15
8 Geschlossene SQL-Injection Sicherheitslücke in Python . . . 16
9 SQLAlchemy Lösung . . . 17
10 Die einzelnen URLs in der Übersicht . . . 20
11 Gefährlicher JavaScript Quellcode im Avatarfeld . . . 21
12 Eingefügter Avatar auf Serverseite . . . 21
13 Quellcodeschablone für XSS Angriff . . . 22
14 Der fertige Angriff . . . 22
15 Verstecktes Formular für CSRF Attacke . . . 27
16 CSRF Token Generierung . . . 28
17 Funktionspointer mit *eval* nachgebaut . . . 30
18 Schadcode im Feld für die erste Zahl . . . 30
19 aufrufbare Zeichenkette als Funktionspointer Ersatz . . . 30
20 Funktionspointer unter Python . . . 31
21 Proc Objekte als Ersatz für Funktionspointer in Ruby . . . 32
22 Inhalte in ein Layout einfügen . . . 35
23 Absichern von Dateinamen . . . 36
24 Null-Byte Überprüfung . . . 36
25 Python Includes . . . 36
26 Ruby Includes . . . 37
27 Passwort Hash- und Prüffunktion . . . 40

Sämtliche Programmebeispiele sind, sofern nicht anders gekennzeichnet, von mir selbst erstellt und unter der GNU GPL lizensiert.

Literatur

[Eby 2004] Eby, Phillip J. (2004). *Python Web Server Gateway Interface v1.0*. http://www.python.org/dev/peps/pep-0333/. [Aufgerufen am 1. Jänner 2007].

[Group 1997] Group, Network Working (1997). *Hypertext Transfer Protocol — HTTP/1.1*. Technischer Bericht, MIT/LCS.

[Jesse Burns 2005] Jesse Burns (2005). *Cross Site Reference Forgery — An introduction to a common web application weakness*. https://www.isecpartners.com/documents/XSRF_Paper.pdf.

[Micro 2006] Micro, Trend (2006). *Botnet Threads and Solutions: Phishing*. Technischer Bericht, Trend Micro Incorporated. [Aufgerufen am 28. Dezember 2006].

[Secunia 2006a] Secunia (2006a). *Vulnerability Report: MediaWiki 1.x*. http://secunia.com/product/2546/?task=advisories. [Aufgerufen am 26. Dezember 2006].

[Secunia 2006b] Secunia (2006b). *Vulnerability Report: phpBB 2.x*. http://secunia.com/product/463/?task=advisories. [Aufgerufen am 26. Dezember 2006].

[Slemko 2004] Slemko, Marc (2004). *Cross Site Scripting Info*. http://httpd.apache.org/info/css-security/. [Aufgerufen am 30. Dezember 2006].

[Wikipedia 2006a] Wikipedia (2006a). *MediaWiki — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=MediaWiki&oldid=25680426. [Stand 30. Dezember 2006].

[Wikipedia 2006b] Wikipedia (2006b). *phpBB — Wikipedia, Die freie Enzyklopädie*. http://de.wikipedia.org/w/index.php?title=PhpBB&oldid=25691273. [Stand 30. Dezember 2006].